

Reinforcement Learning and Neural Reinforcement Learning

Samira Sehad and Claude Touzet

LERI - EERIE, Parc Scientifique Georges Besse, 30 000 Nîmes, France.
Tel: +33 66 38 70 25 Fax: +33 66 84 05 06
Email: samira@eerie.eerie.fr touzet@eerie.eerie.fr

Abstract. In this paper, we address an under-represented class of learning algorithms in the study of connectionism: reinforcement learning. We first introduce these classic methods in a new formalism which highlights the particularities of implementations such as Q-Learning, Q-Learning with Hamming distance, Q-Learning with statistical clustering and Dyna-Q. We then present in this formalism a neural implementation of reinforcement which clearly points out the advantages and the disadvantages of each approach.

1. Introduction

Reinforcement learning (RL) was initially designed by the psychologists and has been studied for almost a century [8]. It was then reused by the Machine Learning community [2, 8, 11]. Up until today, a clear synthetic view of the approach allowing each variation of the implementation to be positioned has not been available. By decomposing existing RL methods into functions and elements, we are able to propose a general model of the reinforcement approach. Instantiations of this general model on the widely used Q-Learning [11, 5] and its refinements [6, 8] allow us to easily understand a neural implementation of reinforcement and to point out the advantages and disadvantages of this approach. Furthermore, we can determine where our efforts should be made so as to improve the performance of a neural implementation of reinforcement.

2. Reinforcement Learning

RL is the learning of a mapping from situations to actions so as to maximize a scalar reward or reinforcement signal [8]. An agent learns a given behavior by being told how well or how badly it is performing an action which starts from a situation. It then receives a returned feedback as a single information item from the world. By successive trials and/or errors, the system determines a function G which is adapted through learning. For this purpose, numerous RL algorithms are available [8, 11].

3. Methods of Reinforcement Learning

We propose to view RL as a composition of (Fig. 1.):

- an internal state S ,
- an evaluation function V ,
- an update function U ,
- a heuristic function H .

The following elements propagate information among functions:

- a set $I = \{i_1, \dots, i_n\}$ of world situations. By definition, I is so large that it cannot be exhaustively explored during learning.
- a set $A = \{a_1, \dots, a_m\}$ of actions,
- a reinforcement signal labelled r .

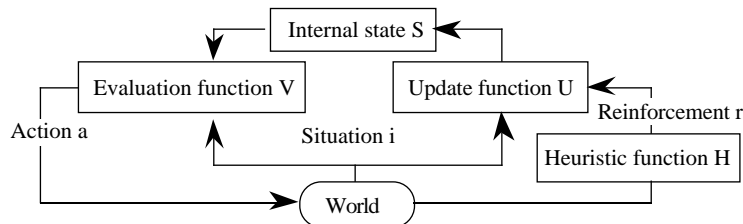


Fig. 1. General decomposition of reinforcement methods

A general algorithm for reinforcement methods is then [6]:

1. Initialization of the internal state S to S_{initial} .
 2. Repeat :
 - 1 - Let i be a world situation.
 - 2 - Select the action a to be performed, by the evaluation function V :
 $a = V(i, S)$.
 - 3 - Execute the action a in the world. Let r be the immediate reward (if it is available) associated with the execution of the action a in the world.
 - 4 - Update the internal state S by the update function U :
 $S_{\text{new}} = U (S_{\text{old}}, i, a, r)$.
- Knowledge about returned rewards is stored in S .

The heuristic H is a function given by a human expert. It is a formula which leads the agent to adopt the behavior G . It is the only function dependent of the application. The reinforcement r is a qualitative signal (i.e., 1: good; -1: bad; 0: unknow) returned by the heuristic function H .

We propose a description in the formalism given above for each of the following reinforcement algorithms: Q-Learning and its refinements (Q-Learning with Hamming distance, Q-Learning with statistical clustering and Dyna-Q).

3.1 Q-Learning Algorithm

The Q-Learning algorithm builds a Q function that maps situation-action pairs (i, a) into expected returns r . $Q(i, a)$ is the system's estimate of the return it expects to receive given the fact that it executes action a in situation i .

a/ The internal state S :

The algorithm uses a lookup table to store the estimated cumulative evaluation Q . All these values represent the internal state S .

$$S = \{ Q(i, a), i \in I, a \in A \}$$

Row indices in the lookup table represent situations, column indices represent actions. The representation of the internal state does not need to be only in the form of tables, but can also be more compact representations like neural networks, decision trees or symbolic rules[8].

b/ Evaluation function V:

Using the Q values, the behavior of the system, G, is determined by the rule:

$$G(i) = a \quad \text{such that} \quad Q(i, a) = \text{Max}\{Q(i, b), b \in A\}$$

In a definite number of cases, the choice of an action with the evaluation function V is given by: $a = V(i, Q)$ such that V is the maximum function.

For the other cases (typically 20%):

$a = \text{Random}(A)$ such that V is a random function (for example, a Boltzman distribution [11]).

c/ Update function U:

The internal state Q is updated by the following function :

$$U: Q_{\text{new}} = U(Q_{\text{old}}, i, a, i', r) \quad \text{where } i' \text{ is the situation after executing a in } i.$$

$$Q(i, a)_{\text{new}} = Q(i, a)_{\text{old}} + \beta(r + g \cdot \text{Max}Q(i', a) - Q(i, a)_{\text{old}}).$$

where β and g are constant coefficients, between $0 < \beta, g < 1$.

The reinforcement at the present time should be equal to the expected returned rewards. The error between the expected value $r + g \cdot \text{Max}Q(i', a)$ and the current value $Q(i, a)$ must then be minimized.

This updating rule has the effect of propagating a reward associated with a given situation-action to previous pairs of situation-action. It is, in fact, a way to backpropagate delayed rewards (Fig. 2).

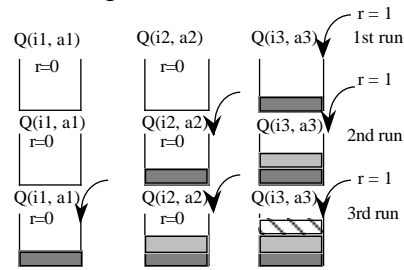


Fig. 2. Example of backward propagation of a delayed reinforcement on 3 runs of the same sequence of situation-action.

3.2 Q-Learning with Weighted Hamming Distance

The main idea of this refinement is to learn faster. With this end in mind, Mahadevan [6] proposes computing a Hamming distance between the real situation i and its similar situations in order to apply the update function on all of them and this for the same reward.

a/ Update function U:

The update function U is the same as the one used in the Q-Learning. However, all similar situations are updated at the same time using the same reward. The Hamming distance between any two situations is simply the number of bits that are different between them. Bits can be of different weights. Two situations are distinct if the Hamming distance between them is greater than a fixed threshold (r) (Fig. 3).

Example: $i_2=(0001)$, $i_3=(0011)$, $r=2$.

Hamming distance(i_2, i_3) = $1 < 2$.

	a	a1	a2	a3	a4	a5	a6
i	0000	i1					
0001	i2						
0011	i3						
0111	i4						
1111	i5						
1110	i6						

Fig. 3. Updated Q values for similar situations (in the sense of Hamming distance) after executing a_2 in i_3 .

■ All the Q values are updated using the same reward.

3.3 Q-Learning with Statistical Clustering

The goal here is the same as in the previous approach, Mahadevan [6] uses statistical clustering to propagate returned rewards across situations.

a/ Update function U:

In order to propagate a returned reward for a situation i , the same function U that was used in Q-Learning is used here.

Each action is associated with a set of clusters giving information concerning the usefulness of performing the action in a particular class of situations. Clusters are a set of "similar" situation instances which use a given similarity metric. Each situation i is updated, if it belongs to a cluster in which i appears (Fig. 4.).

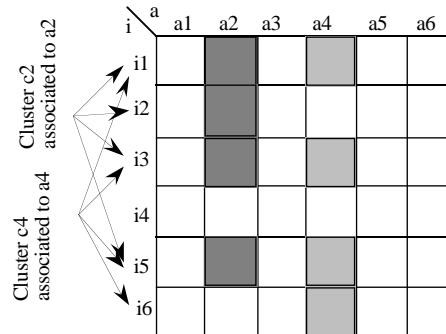


Fig. 4. Updating of the Q values associated with 2 clusters.
 After executing a2 in i3.
 After executing a4 in i3.

3.4 Dyna-Q

It may be difficult to rerun the same sequence of situations-actions so as to backpropagate delayed rewards. As a result, Sutton [8] imagined adding a model of the world in which pairs of situation-action are randomly carried out again (Fig. 2.).

a/ Heuristic function H:

The H function is modified in order to deal with the modelled world. In this case, only previously seen pairs of situation-action (in the real world) will lead to a non-zero reinforcement reward. The returned reward r' is the same as in the real world.

b/ Evaluation function V:

When the experience is performed in the real world, V is the maximum function. Otherwise (i. e., for an experience in the modelled world) V is a random function.

c/ Update function U:

The updating rule uses rewards r or r' indifferently.

As we have seen, RL methods are faced with two important problems. Generalization is limited to syntactic criteria. Methods like those proposed by Mahadevan and Connell [6] and Sutton [8], can speed up learning but they are nevertheless subject to a memory requirement for storing all possible situation-action utility values. Neural networks are another kind of approach for RL methods.

4. Neural reinforcement learning

Numerous authors ([2, 4, 7,1], among others) propose a neural implementation of reinforcement learning, but it is not clear what the differences are nor where lie compared with a classic implementation. In our formalism, a neural network

implementation implicates the following modifications:

a/ The internal state S:

The internal state is constituted by the weight set of the network (W). The memory size required by the system to store the knowledge is then defined, a priori, by the number of connections in the network. It is independent of the number of explored situation-action.

b/ Evaluation function V:

The evaluation is the result of the processing by the network of the input situation i plus a random component b . This component decreases during the learning process. At the same time, the network generalization becomes better : the system learns. The selection of an action is not based on deterministic quantities, like Q values. The neural network proposes the "best" action to perform in the given situation, but it gives no information about the expected reinforcement or other actions having the same usefulness. It should be remembered that in classic RL methods, each utility value associated with a situation-action pair is stored and remains available for computation. The architecture of the neural network can be multilayer if mapping from situations to actions is believed complex.

c/ Update function U:

The update function U works on the internal state. When applied to a neural network, it is a weight modification algorithm. Qualitative reinforcement gives information about how well the system behaves. In the case of a positive reinforcement signal, it is particularly easy to determine the output error. This error is equal to the added random value. Therefore, a gradient error descent algorithms are good choices for updating functions. As regards a negative reinforcement signal, an easy definition of the output error is restricted to simple cases where only two actions are possible. In this case, the desired output is the other action. If actions are binary coded, the desired output is then the inverse of the network proposal. As we can see, the definition of an error in the case of negative reinforcement is difficult. It is important to note that this update rule is not the same as for Q-Learning.

Among the advantages of neural reinforcement learning are a limitation in the memory requirement and more intelligent exploration of the situation-action space. The generalization achieved by the neural network cannot be characterized as easily as for the Hamming distance or clustering techniques. Nevertheless, all connectionist applications used in industry serve to highlight all the more the interest one can find in using artificial neural networks. The main limitation which arises when using neural reinforcement learning is that the reward value associated with a situation-action is not available, however, this is the price to be paid if one wants to obtain compactness and generalisation. It should be noted that coding on the output layer of the actions is of great importance and that proposals have been made to extend neural reinforcement to multiple possible actions [10].

5. Conclusion

Following the description of a formal model and a general algorithm for RL, we have decomposed Q-Learning, Q-Learning with Hamming distance, Q-Learning with statistical clustering and Dyna-Q, into functions and elements. This decomposition

allows us to highlight the evaluation, update and heuristic functions of each method and then to point out the different particularities of each function. We have used this model for the analysis of a neural implementation of reinforcement. In a case such as this, the memory requirement is limited by the number of connections and generalization by neural networks is, a priori, more interesting. Experiments undertaken with a real robot underline the interest generated by a neural network implementation of RL [10].

It will be particularly interesting to develop different evaluation functions not only using maximum or random functions [9] or neural networks, but also an evaluation function that proposes several actions to be performed. It will also be worthwhile to develop different update functions which will allow several utility values to be updated at the same time. This will lead to faster learning. The internal state represented as a lookup table can be characterized by its easy implementation and use. However, as we have seen, the implementation of neural networks is interesting in term of generalization. Classifier systems which use genetic algorithms [3] have a similar objective but demand further study within the framework of this formalization.

References

1. D. Ackley, M. Litman: Interactions Between Learning and Evolution. *Artificial Life II, SFI Studies in the Sciences of Complexity*, vol. X, C. G. Langton & Co. Eds, Addison-Wesley, 487-509 (1991).
2. A. G. Barto, P. Anandan: Pattern Recognizing Stochastic Learning Automata. *IEEE Transactions on Systems, SMC-15*, 360-375 (1985).
3. J. Biondi, P. Collard: Cooperation Between Reactive Agents Reinforcement And Hybridization. *Second Congress of System Science, AFCET, Prague, October, Vol. II*, 599-608 (1993).
4. J. Hertz, A. Krogh, R.G. Palmer: *Introduction to the Theory of Neural Computation*. SFI Studies in the Sciences of Complexity, Addison-Wesley, Redwood City (1991).
5. L-J. Lin: Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. *Machine Learning*, 8, 293-321 (1992).
6. S. Mahadevan, J. Connell: Automatic Programming of Behaviour-based Robots using Reinforcement Learning. *AI journal*, July 25 (1991).
7. J. Millan, C. Torras: A Reinforcement Connectionist Approach to Robot Path Finding in Non-Maze-Like Environments, *Machine Learning* 8, 363-395 (1992).
8. R.S. Sutton: Reinforcement Learning Architectures for Animats. *Proceedings of the First International Conference on Simulation of Adaptive Behaviour. From Animals to Animats*. Edited by J-A Meyer and S.W. Wilson, 288-296 (1991).
9. S.B. Thrun: Efficient Exploration In Reinforcement Learning. Technical report CMU-CS-92-102, School of Computer Science, Carnegie-Mellon University, Pittsburg, Pennsylvania 15213-3890, January (1992).
10. C. Touzet: Extending Immediate Neural Reinforcement Learning to Multiple Actions. *ESANN* (1994).
11. C.J.C.H. Watkins, P. Dayan: Technical note: Q-Learning. *Machine Learning* 8, 279-292 (1992).
12. S.D. Whitehead: a Complexity Analysis of Cooperative Mechanisms in Reinforcement Learning. In *Proceedings of AAAI-91*: 607-613 (1991).