Solving Partially Observable Markov Decision Processes by Neural Networks.

S.A. Velázquez Lerma*, R. Alquézar†

*Institut de Organització i Control de Sistemes Industrials, Univ. Politécnica de Catalunya Barcelona,Spain
†Departament de Llenguatges i Sistemes Informátics, Univ. Politécnica de Catalunya,Barcelona, Spain
email: velazque@ioc.upc.es, alquezar@lsi.upc.es

Abstract

Partially Observable Markov Decision Processes (POMDPs) cope with sequential decision processes where an agent tries to maximize or minimize some reward without complete knowledge of the process. These models are of interest for quality control, machine maintenance, reinforcement learning, etc. More generally Monahan [9] has shown that many tasks in partially observable environments can be viewed as POMDPs.

A solution for the POMDP gives the best behaviour of the agent face to the environment. This gives a solution over all the state space, which is continuous and inside of an integral polytope. However, the approaches proposed until now use linear programming to solve the optimization problem in this type of processes. By other side, Neural Networks (NNs) have shown a promising potentiality for finding solutions to optimization problems; particularly, they have been used to solve quadratic 0-1 programming problems [4, 6].

In this paper, we use optimization neural networks as a different way to solve the optimization problem in the POMDP, which allows a parallel hardware implementation.

1 Introduction

Markov decision processes (MDPs) have been generally used in operations research (OR) and artificial intelligence (AI) to solve problems of decision-theoretic planning [11]. This paper addresses the problem of the optimization of a POMDP model for a sequence of observations. The computational effort in constructing the policy for POMDP is expensive and the main techniques are based on the assumption that the parameters of the task model are known [11, 3, 15], and use linear programming to solve the optimization problem. Also, the problem of learning the POMDP parameters has been studied in [16].

By other side, neural networks have shown a promising potentiality to find solutions for optimization problems. The first work was reported by Hopfield and Tank [6], in which neural networks were used as a means of approximately solving NP-complete combinatorial optimization problems. Some effort has been directed to find valid solutions, to escape from local minima and to force convergence to a hypercube corner [10]. Also it has been demostrated that a more adequate problem mapping can improve the quality of solutions [2],[5]. However, this technique has been only used to solve quadratic 0-1 programming problems. In the case of the POMDP the solution is inside the integral polytope and the NN technique is also applicable, but it is not necessary to force the convergence to a hypercube corner.

This paper is basically divided in three parts. In the first part, a brief review of both the POMDP problem and the optimization neural networks is presented. In the second part, the proposed approach including the POMDP optimization by NN is described and the corresponding algorithm outlined. The latest part includes some examples which show the effectiveness of this approach in solving the POMDP process.

2 POMDP models

The POMDP models represent sequential decision processes which are generated by the interaction between a system and a controller. In this paper, the words agent and controller have the same meaning and are used indistinctly. To solve a POMDP implies the controller's development optimization through a policy. This policy is obtained for a time interval called decision horizon. The dynamic of the process is used to build the policy. This dynamic can be described as follows: the agent takes an output value from the system and chooses an action based on this observation; the interaction with the environment will produce a system change, and finally the agent gets a reward or penalty as an index of his behaviour. The criterion used to improve the efficiency of the controller is to maximize the expected value of the reward (V) over time.

Formally the model that represents a POMDP is defined by an oc-tuple $(T, S, A, O, \mathbf{M_k}, \mathbf{B_k}, r_{jk}, \vec{\pi_0})$, where $T \subset \Re$ is the decision horizon, S is the set of states of the system, O is a set of observations that the agent can see, A is a set of possible actions and:

- $\mathbf{M_k}$ is a stochastic matrix whose elements represent the probability that exists of going from state s_i to state s_j through action a_k, *i.e.* $m^k_{ij} = p(s_j/s_i, a_k)$.
- $\mathbf{B_k}$ is a stochastic matrix whose elements represent the agent probability of seeing the output o_θ when the system is in state s_j, given the agent action a_k, *i.e.* $b^k_{j\theta} = p(o_\theta/s_j, a_k)$.
- $r_{ik} \in \Re$ determines the reward that gets the controller by taking the action a_k, seeing the output θ when the system is in s_i. Rewards are used to compute the expected value.
- $\vec{\pi_0}$ represents the probability density vector for the states at $t = 0$.

3 Algorithms to solve POMDP problems

The main algorithms to solve POMDPs were developed in the field of operation research. A good review can be found in [3, 15]. In the POMDPs the space of states is continuous and the policy will cover all this space. In the state space of the POMDPs each point is represented by a probability vector $\vec{p^t_s}$ and is updated, given the observation and the action performed, by the following equation:

$$p^{t+1}_i = f(\vec{p_s}^{\,t}, {a_k}^t, {o_\theta}^t) = \frac{\sum_j p^t_j m^k_{ji} b^k_{j\theta}}{\sum_{j,l} p^t_j m^k_{jl} b^k_{l\theta}} \tag{1}$$

For POMDPs, it has been shown [13] that the optimal expected value of the reward (V^*) is a piecewise linear and convex function in the state space. For each patch of V^* there is an associated action. The V for each iteration is given by

$$V^t_k(\vec{p_s}^{\,t}) = \sum_{i=1}^{ne} p^t_i \left(r_{ik} + \sum_{j,\theta}^{ne,no} m^k_{ij} b^k_{j\theta} V^{t+1*}(\vec{p_s}^{\,t+1}) \right)$$

where V^{t+1*} is the best expected value for $\vec{p_s}^{\,t+1}$. The above equation can be simplified as

$$V^t_k(\vec{p_s}^{\,t}) = < \vec{p_s}^{\,t}, \vec{\alpha}^t_k(\vec{p_s}^{\,t}) > \tag{2}$$

for some $\vec{\alpha}_k$ vectors, due to the fact that V^* is piecewise linear and convex.

The best policy given all the V^t_k is

$$d^*_t(\vec{p_s}^{\,t}) = argmax_{k \in A}\{V^t_k(\vec{p_s}^{\,t})\}$$

The algorithms to solve a POMDP are focused on finding the $\vec{\alpha}_k$ vectors. Each $\vec{\alpha}_k$ involved in V^* designates one region in the space of states and has an associated action. These $\vec{\alpha}_k$ vectors are given by

$$\alpha^t_{ik} = r_{ik} + \sum_{j,\theta} m^k_{ij} b^k_{j\theta} \alpha^{t+1}_{j(a^{t+1})^*} \tag{3}$$

where $(a^{t+1})^* = f(\vec{p_s}^{\,t}, {a_k}^t, {o_\theta}^t)$ is given by

$$(a^{t+1})^* = arg\ max_{k \in A}\{\sum_{ij} p^t_i m^k_{ij} b^k_{j\theta} \alpha^{t+1}_{jk}\} \tag{4}$$

The principal aim of these algorithms is to build the piecewise linear and convex function V^*. To that end, a value of $(a^t)^*$ is found given the optimal data at time $t+1$, which is used in computing a set of $\vec{\alpha_k}^t$. Finally the minimum set of $\vec{\alpha_k}^{\,t}$ which make up the expected value V^{t*} is found. The last step can be seen as a combinatorial optimization problem, which is usually solved by linear programming (LP).

4 Optimization networks

Many combinatorial optimization problems can be posed as the minimization of a single quadratic objetive function (E) over a set of 0-1 variables, but the problem of finding points which minimize E is typically NP-complete. The principal idea in using neural networks is to associate the problem's objective function with the network's Liapunov function by setting the connection weights and input biases appropriately. Therefore, the dynamic of the network can be used to minimize E [4].

According to [4], there are basically three optimization networks: the Hopfield network, the steepest descent network and the mean field annealing algorithm [6, 7, 1]. The Hopfield network comprises a set of artificial neurons connected together to form a parallel array. Each neuron i has input bias term i_i, net input u_i, output v_i and is connected to neuron j with a weight T_{ij}. Its dynamic is governed by

$$\begin{aligned} \dot{\vec{u}} &= -\eta\vec{u} + \mathbf{T}\vec{v} + \vec{i_i} \\ v_i &= g(u_i) \end{aligned}$$

where g is the transfer function of each neuron.

If matrix $\mathbf{T}$ is symmetric, the networks has a Liapunov function

$$\begin{aligned} E(\vec{v}) &= -\frac{1}{2}\vec{v}\,'\mathbf{T}\vec{v} - \vec{v}\,'\vec{i} + \varphi \\ E(\vec{v}) &\approx -\frac{1}{2}\vec{v}\,'\mathbf{T}\vec{v} - \vec{v}\,'\vec{i} \end{aligned}$$

where $v_i \in [0,1]$. Note that the term φ which measures the entropy of the system can be eliminated if the transfer function has a sufficient gain. The networks' dynamic drives $\vec{v}$ through some sort of bounded descent on the objective function E. Throughout the descent, v_i can take any value in the range 0-1 and some sort of annealing mechanism is used to free $\vec{v}$ from local minima and to drive it towards a hypercube corner.

5 The mapping process

The mathematical expresion of the quadratic 0-1 programming problem is

$$\begin{aligned} \text{minimize} \quad & E^{op}(\vec{v}) = -\tfrac{1}{2}\vec{v}\,'\mathbf{T}^{op}\vec{v} - \vec{v}\,'\vec{i^{op}} \\ \text{subject to} \quad & \mathbf{A}^{eq}\vec{v} = \vec{b}^{eq} \\ \text{and} \quad & \mathbf{A}^{in}\vec{v} \leq \vec{b}^{in} \\ \text{and} \quad & 0 \leq v_i \leq 1 \\ \text{and} \quad & \vec{v} \text{ integral} \end{aligned}$$

The aim of the mapping process is to construct an energy function E so that the network solutions not only minimizes E^{op}, but also satisfies the problem's constraints. The method consists in coding the constraints as terms in E which are minimized when the constraints are satisfied. In this case the Liapunov function takes the form:

$$E = E^{op} + cE^{cns}$$

where

$$E^{cns} = \frac{1}{2}\|\vec{v} - (\mathbf{T}^{val}\vec{v} + \vec{s})\|^2 \tag{5}$$

where $\mathbf{T}^{val}$ and $\vec{s}$ define an affine subspace of solution for the equality constraint [8]. The inequality constraints can be posed in the same form by introducing one slack variable for each inequality. Finally, the Liapunov function of the network is achieved by setting the network parameters as follows [4]

$$\begin{aligned} \mathbf{T} &= \mathbf{T}^{op} + c(\mathbf{T}^{val} - \mathbf{I}) \\ \vec{i} &= \vec{i}^{op} + c\vec{s} \end{aligned}$$

The dynamic of the network gives a continuous method of performing bounded descent minimization. It is bounded because the descent is limited to the unit hypercube. For the problem of solving POMDP we use the steepest descent dynamics [7]

$$\dot{v_i} = \begin{cases} 0 & \text{if } \left[\mathbf{T}\vec{v} + \vec{i}\right]_i < 0 \quad \text{and} \quad v_i = 0 \\ 0 & \text{if } \left[\mathbf{T}\vec{v} + \vec{i}\right]_i > 0 \quad \text{and} \quad v_i = 1 \\ \left[\mathbf{T}\vec{v} + \vec{i}\right]_i & \text{otherwise} \end{cases}$$

which share the same Liapunov function and has the advantage of a more efficient simulation on digital computers.

6 Proposed method to solve POMDP by optimization neural networks

The application of optimization networks was carried out using the easiest algorithm to solve POMDP, the Monahan's algorithm [12]. The NN technique can be used with any other of the POMDP algorithms if a suitable mapping is defined.

The principal idea in Monahan's method is that there are only a finite number of $\vec{\alpha}$ vectors at each step. This algorithm is splitted in two phases.

In the first one, an enumeration of all the posible $\vec{\alpha}^{\,t}$ is done by using the equation (3) – a summation over the observations and the choice of one of the previous $\vec{\alpha}^{\,t-1}$ for each observation –. It is easy to observe that there are as many vectors as possible combinations. If M is the number of vectors $\vec{\alpha}$ in the last step, there are at most $na * M^{no}$ ways to generate the $\vec{\alpha}^{\,t}$.

In the second part, a reduction of this set of vectors is needed to get the minimum set of $\vec{\alpha}^{\,t}$ which form the expected value V^* for that step. This reduction phase can be seen as a combinatorial optimization problem. In fact, Monahan solved it using linear programming. A linear program defines a region in some space through a set of constraints, generally inequalities. There is also an objective function, in this case a linear function, which will be optimized within the region.

The LP formulation for this problem is the following:

$$\begin{array}{l} \forall\, \vec{\alpha}_k \in V^t \quad \text{where } k = 1 : naM^{no} \\ max:\ \ \delta \\ \text{with constraints:} \\ <\vec{p_s}^{\,t}, (\vec{\alpha}_j - \vec{\alpha}_k) > +\ \delta \leq 0\ \forall\, \vec{\alpha}_j \in V^t \end{array}$$

If $\delta \leq 0$ then the vector $\vec{\alpha}_k$ is not in the minimal set needed to form V^*. The mapping from this LP formulation into an optimization neural network is as follows.

As the objetive function is linear, the problem is to find and minimize an energy function of the form

$$\begin{array}{rl} \text{minimize} & E(\vec{v})^{op} = -\frac{1}{2}\vec{v}'\,\vec{i^{op}} \\ \text{and} & \mathbf{A}^{in}\vec{v} <= \vec{0} \end{array}$$

For the POMDP problem the state vector $\vec{v}$ is composed by three elements: $\vec{v} = (\vec{p_s}, \delta, \vec{w})$. The first one is the probability state vector of the POMDP, the second one is the variable δ to optimize in the LP problem, and the last vector corresponds to the slack variables needed to map the inequalities. In order to maximize δ the vector $\vec{i}$ is a negative unitary vector; having in this way the same objective function than in the LP formulation. The matrix of the inequalities $\mathbf{A}^{in}$ is also composed by three parts as follows:

$$\begin{pmatrix} (\vec{\alpha}_1 - \vec{\alpha}_k)_1 & \dots & (\vec{\alpha}_1 - \vec{\alpha}_k)_{ne} & 1 & w_1 & 0 & 0 \\ \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ (\vec{\alpha}_i - \vec{\alpha}_k)_1 & \dots & (\vec{\alpha}_i - \vec{\alpha}_k)_{ne} & 1 & 0 & w_i & 0 \\ \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ (\vec{\alpha}_M - \vec{\alpha}_k)_1 & \dots & (\vec{\alpha}_M - \vec{\alpha}_k)_{ne} & 1 & 0 & 0 & w_M \end{pmatrix}$$

With this matrix the term in the energy which codes the constraints can be computed using the equation (5) where:

$$\mathbf{T^{val}} = \mathbf{I} - \mathbf{A}'(\mathbf{A}\mathbf{A}')^{-1}\mathbf{A}$$

and

$$\vec{s} = \mathbf{A}'(\mathbf{A}\mathbf{A}')^{-1}\vec{b^{in}}$$

Finally the network's parameters are set as

$$\begin{array}{rcl} \mathbf{T} & = & c(\mathbf{T}^{val} - \mathbf{I}) \\ \vec{i} & = & \vec{i^{op}} \end{array}$$

Note that $\mathbf{T^{val}} = [0]$ because the objective function is linear, and that the term $c\vec{s}$ is not in the equation due to the fact that $\vec{b^{in}} = 0$. In accordance with [4] and with the constraints of the problem, the first and the last elements of the state vector are within the interval [0,1], however the second one, δ, is not constrainted to be in this interval. In this case its magnitude is not so important because the dynamic of the neural networks will minimize it. Once the parameters of the network are set the steepest descent dynamic is used to lead the energy minimization. During the simulation, it is possible to stop the dynamic of the networks if the value of δ is less than zero because then the $\vec{\alpha}$ vector does not belong to the solution subset. Some experimental results are shown in the next section.

7 Experiments and results

To evaluate this new connectionist approach three different problems were tested. As a first example, the well known 1D paradigm was used. The model is depicted in the Fig. 1. It can be observed that the environment is composed by four states. The agent is able to perform two actions, moving west and east, and it can only recognize the final state. It gets a reward of 1 for any transition that carried it into the goal and 0 elsewhere.

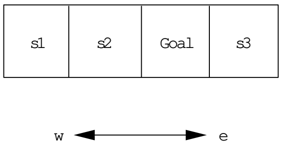

Figure 1: *1D maze model*

The second example consists of an environment of 16 states, four actions and two possible observations. This is known as the 4×4 maze without obstacles (Figure 2). Finally, the problem used by Sondik in [13] was tested.

Simulations were carried out in a Sun Ultra2 with a UltraSparc 200 MHz processor. To compare the proccesing time required, these problems were also solved by the Monahan's method and the witness algorithm [3]. For these methods, the LP problems were solved using lp-solve freeware software. The results obtained are displayed in Table 1, where the total CPU time, the number of iterations, the number of support lines (optimal $\vec{\alpha}$ vectors) and the discount parameter γ used are shown. The ϵ value used to stop the time iterative process was 1×10^{-9} in all cases. The error for policy improvemente for a given t was 0.1 for the NN approach and 1×10^{-9} for the other methods. All the simulations ran at infinity horizon.

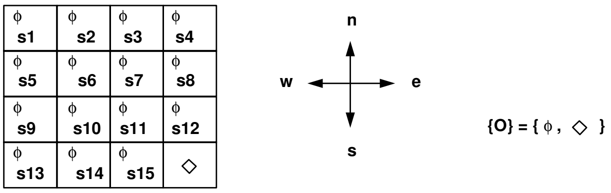

Figure 2: 4×4 *maze diagram*

1D maze gamma = 0.8			
method	CPU secs.	num. it.	num. sup.
Monahan's	0.40	70	4
witness	1.54	70	4
NN	0.67	70	4

4x4 maze gamma = 0.95			
method	CPU	num. it.	num. sup.
Monahan's	73.7	374	20
witness	2203.4	374	20
NN	444.54	374	20

Sondik maze gamma = 0.8			
method	CPU	num. it.	num. sup.
Monahan's	0.08	97	2
witness	0.28	97	2
NN	0.41	97	2
gamma = 0.3			
Monahan's	0.02	19	2
witness	0.10	19	2
NN	0.08	18	2

Table 1: *Comparison results*

The Monahan's algorithm was shown to be the most efficient for the tested problems. The NN approach to solve the LP problems, in a sequential implementation, was slower than Monahan's method but faster than the witness algorithm, especially in the 4×4 maze which is the most time consuming problem. We must said, however, that the code of our NN approach is not yet optimized while the other algorithms are indeed. A decrease in processing time is expected for the NN technique once the code be optimized. In addition, NN offers an inherent parallelism which may lead to efficient hardware implementations.

8 Conclusions and future work

We have presented a connectionist approach to solve the POMDP problem using optimization neural networks. The approach is based on the Monahan's algorithm and its effectiveness has been shown empirically in several experiments.

To the contrary of previous works that have used optimization neural networks, in this work it has been shown that this NN technique can also be applied to solve problems in which the variables are not binary ($\{0,1\}$) but with values in the interval [0,1]. As far as we know, it is the first time that optimization neural networks have been used to solve this type of problems. Furthermore, the restriction of the variable values to the interval [0,1] can be eliminated if a mean field annealing network is used together with an activation function whose range is not limited within this interval (*e.g.* an antisymmetric logarithm function [14]).

Optimization networks appear to offer a good compromise between solution quality and speed of operation, since their inherent parallelism may lead to hardware implementations which could solve problems of reasonable size efficiently. It must be emphasized that the approach presented here provides an optimal solution for the POMDP and therefore is computationally untractable for large problems. An interesting topic in which we are currently working is to find other neural techniques for computing good suboptimal solutions for the POMDP with less computational effort. Also there is the possibility of applying neural networks to the task of learning the parameters of the POMDP model as an alternative to the present method, which is based on the Baum-Welch algorithm [16].

References

[1] S. V. B. Aiyer. Solving combinatorial optimization problems using neural networks. *Technical report CUED/F-INFENG/TR89, Cambridge University*, 1991.

[2] S. V. B. Aiyer, M. Niranjan, and F. Fallside. A theoretical investigation into the performance of the hopfield model. *IEEE Transaction on Neural Networks, 1(2)*, pages 204–215, 1990.

[3] A.R. Cassandra. Optimal policies for partially observable markov decision processes. Technical report, CS Brown University, 1994.

[4] A. H. Gee, V.B. Aiyer, and R. W. Prager. An analytical framework for optimizing neural networks. *Neural Networks 6(1)*, pages 79–97, 1993.

[5] A. H. Gee and R. W. Prager. Polyhedral combinatorics and neural networks. *Tech. Rep. CUED/F-INFENG-100, Engineering Department, Cambridge University*, 1992.

[6] J. J. Hopfield and D. W. Tank. Neural computation of decision in optimization problems. *Biological Cybernetics, 52*, pages 141–152, 1985.

[7] S. Kirkpatrick, C.D. Gellatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science, 220*, pages 671–680, 1983.

[8] T. Kohonen. *Self-Organization and Associative Memory.* Springer Series in Information Sciences, 1989.

[9] George E. Monahan. A survey of partially observable markov decision processes: theory, models, and algorithms. *Management Science, 28*, pages 1–16, 1982.

[10] C. Peterson and B. Söderberg. A new method for mapping optimization problems onto neural networks. *International Journal of Neural Systems 1(1)*, pages 3–22, 1989.

[11] M.L. Puterman. *Markov Decision Processes.* John Wiley and Sons, Inc., 1994.

[12] Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable markov processes over a finite horizont. *Operations Research, 21*, pages 1071–1088, 1973.

[13] Edward J. Sondik. The optimal control of partially observable markov processes over the infinite horizon:discounted costs. *Operations Research, 26(2)*, pages 282–304, 1978.

[14] J. M. Sopena and R. Alquézar. Improvement of learning in recurrent networks by substituting the sigmoid activation function. *Proc. International Conference on Artificial Neural Networks, ICANN'94, Sorrento, Italy, vol. 1*, pages 417–420, 1994.

[15] S.A. Velázquez-Lerma. Decision task in visual search. Technical Report IC-DT-9608, Institute of Cybernetics, UPC, 1996.

[16] S.A. Velázquez-Lerma. Learning pomdp models in unknown environments. *Proc. Second International ICSC Symposia on Soft Computing and Intelligent Industrial Automation, Nimes*, pages 129–135, 1997.