From Continuous Behaviour to Discrete Knowledge

Agapito Ledezma, Fernando Fernández and Ricardo Aler

Universidad Carlos III de Madrid. Avda. de la Universidad, 30, 28911, Leganés, Madrid. Spain
{ledezma, ffernand, aler}@inf.uc3m.es

Abstract. Neural networks have proven to be very powerful techniques for solving a wide range of tasks. However, the learned concepts are unreadable for humans. Some works try to obtain symbolic models from the networks, once these networks have been trained, allowing to understand the model by means of decision trees or rules that are closer to human understanding. The main problem of this approach is that neural networks output a continuous range of values, so even though a symbolic technique could be used to work with continuous classes, this output would still be hard to understand for humans. In this work, we present a system that is able to model a neural network behaviour by discretizing its outputs with a vector quantization approach, allowing to apply the symbolic method.

1 Introduction

Neural networks (NN) are very useful to solve a wide range of tasks, such as classification, prediction, optimization, etc. In this work, we begin with a NN that is able to control a robot. There are many ways to learn such a NN, from typical algorithms, like backpropagation [1], to reinforcement learning approaches [2] or evolutionary computation [3]. However, it is very difficult to interpret the results in order to extract general conclusions on the correctness of the learned knowledge, its possible drawbacks, or to improve it.

The ability to transform a procedural description of the reasoning process of, for instance, a given control skill into a declarative representation allows to more easily share knowledge, or to reason about other control behaviors. Specifically, one of our goals is the study of automatic ways of extracting knowledge (models) from non-symbolic representations, such as NN's. This has been already studied by some authors by analyzing the internal structure of the NN [4]. We propose an alternative that consists on modeling their behavior by observing how they "solve problems": what output they generate in response to some inputs. So, the modelling task is mapped to a symbolic classification task, where a classifier must learn to assign the right action (output of the neural network) to any state (input of the network) [5].

Nevertheless, NN output is continuous, while symbolic supervised learning algorithms, like C4.5 [6], typically work with discrete classes. Thus, a mapping from the initial set of continuous values to a finite (and preferable small) set is required. In this work, we apply a vector quantization technique, the Generalized Lloyd Algorithm (GLA) [7], to do this map and to automatize this process, that in other case would have to be done by hand.

In this paper, Section 2 presents our learning approach to symbolic modeling. Section 3 describes the GLA algorithm. Section 4 describes the way in which experiments were defined, and presents the obtained results. Finally, Section 5 discusses the main conclusions achieved and further research.

2 Automatic acquisition of models

The behaviour of a reactive robot can be characterized by its response, (outputs), to the sensorial information that it receives, (inputs). So this behaviour can be considered as a relationship between inputs and outputs. In terms of a classification task, this allows to define a class for every possible output. Then, the task of modeling (generating a declarative representation of a robot behavior) can be translated into a classification task using symbolic methods, where the relationship between the inputs and the outputs must be learnt. In a previous work we presented results for discrete tasks in which the outputs belonged to a finite set of values[5]. However, the assumption of having a finite set of actions is false in many domains. Thus, the work above was extended to apply two different approaches. First, to discretize the output by hand and to use a typical symbolic classifier (like C4.5 [8]). And second, to use a symbolic algorithm that is able to deal with continuous outputs (like regression trees [9,10]). Both approaches obtain very good results, preseted in [11]. However, the first approach has as main problem of the discretization step, that must be done by hand. The problem of the second approach is, given that the output of the regression tree is continuous, it is harder to interpret it, so the goal of the work, to model and to understand the behaviour, may not be achieved. In this work we follow the first approach, but we look for an automatical way to discretize the output. This approach is based on vector quantization methods [12], particulary the GLA.

In this work, we use a NN that controls a robot, and a technique that generates rules, such as C4.5, to model the network, i.e. to model the robot behaviour. The general framework is described in Figure 1 which shows the interrelation between the robot r_1, the modeler r_2 that tries to learn and reason about a model of r_1, the classification technique c used for modeling its behavior, and the obtained classifier m (model of r_1). This classifier m should model the behavior of robot r_1, in such a way that if one presents the same set of input patterns (sensory data) to both r_1 and m the error between the output provided by r_1 and m should be minimal. Furthermore, a vector quantization (VQ) step is introduced to discretize the r_1 outputs. Next section describes briefly this technique.

3 Generalized Lloyd Algorithm (GLA)

The Generalized Lloyd Algorithm [7] is a clustering technique that consists of a number of iterations, each one recomputing the set of more appropriate partitions

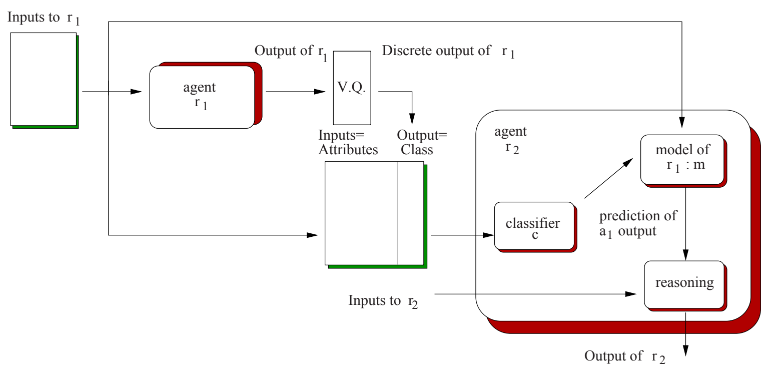


Fig. 1. Architecture of the modeling of robots behavior.

of the input states (vectors), and their centroids. The algorithm is shown in Figure 2. It takes as input a set T of M input states, and generates as output the set C of N new states (*quantization levels*).

There are two design decisions to be made when using such a technique. The first one is how to choose the initial set of clusters, given that the solution is highly dependent on it. The second one is what to do with empty cells obtained

```
Generalized Lloyd Algorithm $(T, N)$
1. Begin with an initial codebook $C_1$.
2. Repeat:
   (a) Given a codebook (set of clusters defined by their centroids) $C_m = \{y_i; i = 1, \ldots, N\}$, redistribute each vector (state) $x \in T$ into one of the clusters in $C_m$ by selecting the one whose centroid is closer to $x$ (nearest neighbour rule).
   (b) Recompute the centroids for each cluster just created, R, to obtain the new codebook $C_{m+1}$, using equation (1):
```

$$cent(R)[i] = \frac{1}{\|R\|} \sum_{j=1}^{\|R\|} x_j[i] \tag{1}$$

```
       where $x_j \in R$, $x_j[i]$ is the value of component (attribute) $i$ of vector $x_j$, and $\|R\|$ is the cardinality of $R$.
   (c) If an empty cell (cluster) was generated in the previous step, an alternative code vector assignment is made (instead of the centroid computation).
   (d) Compute the average distortion for $C_{m+1}$, $D_{m+1}$.
   Until the distortion has only changed by a small enough amount since last iteration.
```

Fig. 2. The Generalized Lloyd Algorithm.

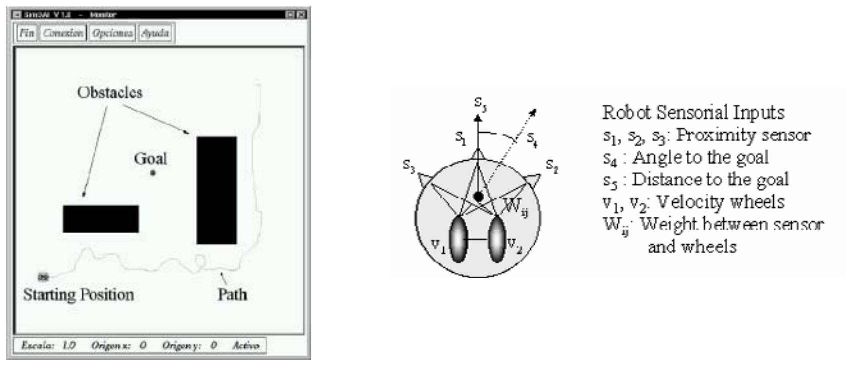
in step 2(c), given that empty cells are useless. Both topics are solved by using only one cell initial prototype, and increasing this value in different iterations. Empty cells are eliminated and replaced by other ones that result from splitting non-empty ones. All these mechanisms are explained in depth in [13].

4 Experimental Setup and Results

This section describes the experimental sequence needed to obtain a symbolic model m from an agent r_1 (which is considered as a black box) that can be used by an agent r_2. To do so, we have carried out three phases: a robot training phase for r_1, a training phase for obtaining a model m of r_1, to be used by r_2 and the test of model m in a robot simulator.

4.1 Robot Trainning

During robot training (r_1), Uniform Coevolution was used to obtain a NN that controls the movement of the robot based on a Braitenberg scheme [14]. The robot moves in a bi-dimensional world, where there are obstacles with different shapes. Its goal is to move to a goal position in an efficient way, as shown in Figure 4.1(a). The robot has five sensors shown in figure 4.1(b). Three of them (the proximity sensors) inform the robot how close obstacles are. The next two measure how far the target location is, and what the angle to that location is. The robot has two wheels that can move at different speeds v_1 and v_2, so the robot can turn. However, for experimental purpose, the speed of wheel v_1 is fixed, therefore the NN can only control the wheel v_2. The aim of the neural network learned by Uniform Coevolution is to control r_1, that is, to map its sensors into v_2 speed. We refer the reader to [3] for details about Uniform Coevolution.


Fin
Conexion
Opciones
Ayuda
Obstacles
Goal
Starting Position
Path
Escala: 1.0 Origen x: 0 Origen y: 0 Activo
s_1
s_2
s_3
s_4
s_5
W_{ij}
v_1
v_2
Robot Sensorial Inputs
s_1, s_2, s_3: Proximity sensor
s_4 : Angle to the goal
s_5 : Distance to the goal
v_1, v_2: Velocity wheels
W_{ij}: Weight between sensor and wheels

(a)Environment View (b) Robot Architecture

Fig. 3. Robot and Environment

4.2 The Modelling Task

Once the controller for r_1 has been learned, the knowledge that tries to model the behaviour of r_1 is obtained by a rule modeler after applying the Lloyd algorithm to discretize the outputs. The detailed steps for training r_2 are as follows:

1. The robot r_1, being controlled by the (co)evolved neural network is run several times. At every instant, the readings of the five sensors (inputs) $r(t_i)$ and the wheel velocities (outputs), $c(t_i)$ are logged to produce a trace of the reactive behaviour of the robot. From this trace it is straightforward to obtain a set of examples T so that r_2 can learn and model r_1.
2. Each instance t_i of the set T is composed by the input, $r(t_i)$, and the output, $c(t_i)$. From the set T, we extract all the $c(t_i)$ values, generating the set T_c of possible outputs. This set is used as input of the generalized Lloyd algorithm to obtain a reduced set of values $\hat{T}_c$. Last all the $c(t_i)$ values in T are discretized based on $\hat{T}_c$, obtaining $\hat{T}$.
3. The Set $\hat{T}$ is used to train r_2, which generates modeling knowledge. This knowledge will be obtained using C4.5 [15], a decision tree generator, that will generate a set of rules that models the robot behaviour. That is, r_2 knowledge should predict the output of r_1, no matter whether it is the right or the wrong output. Of course, we are also interested in modeling r_1's mistakes, if any.

In this domain, the learning task is the prediction among a range of classes, based on 5 attributes (sensors data). The number of instances are 976 corresponding to six simulations of r_1. The classes have been obtained "ad-hoc" for the first experiment, visualizing the distribution of the data, and selecting 11 different values.

The GLA has been also used to obtain the classes using different number of discretization levels (2, 4, 8, 16, 32 and 64). To determine how closely r_2 knowledge models r_1 behaviour, we carry out ten-fold cross-validation. The closeness of the performance of both r_1 and r_2 is measured as the number of examples in which the predictions of r_2 differ to the behaviour of r_1 for the same sensorial input, taking into account that, in each case, both outputs are discretized following the "ad-hoc" classes or the ones generated by GLA. Classification results are summarized in table 4.2 for the "ad-hoc" solution, and the seven solutions obtained by different executions of GLA, for different number of discretization levels or classes.

Approach	"ad-hoc"	GLA 2	GLA 4	GLA 8	GLA 16	GLA 32	GLA 64
Accuracy	89.9%	98.90%	97.30%	93.30%	88.80%	81.80%	70.50%

Table 1. Classification results

While the number of classes is increasing, the accuracy decreases. This is because while the number of classes to differentiate increases, the classification problem is harder to solve. However we will show later that this is not a problem, and that a high accuracy does not imply a good model.

4.3 Testing the Obtained Models

Last, and once the 7 models m_i (the hand made solution, and the 6 models with different number of outputs generated with GLA) have been generated, we tested them in a robot simulator, SimDAI [16], and we compared the performance of our model against the robot controlled by the NN in terms of average time consumed and average distance covered to reach the goal. In total there were 50 trials (robot motion) with r_1 and the 7 models m_i. Each trial begins from different places in a bi-dimensional world and consists in reaching the goal in an efficient way.

Figure 4 shows average distance covered by each model (including the original one) to achieve the goal for the 50 trials, while Figure 5 shows the time spent in running such a distance. The original model and the model using "ad-hoc" classes obtain similar results, showing that the last approach is successful.

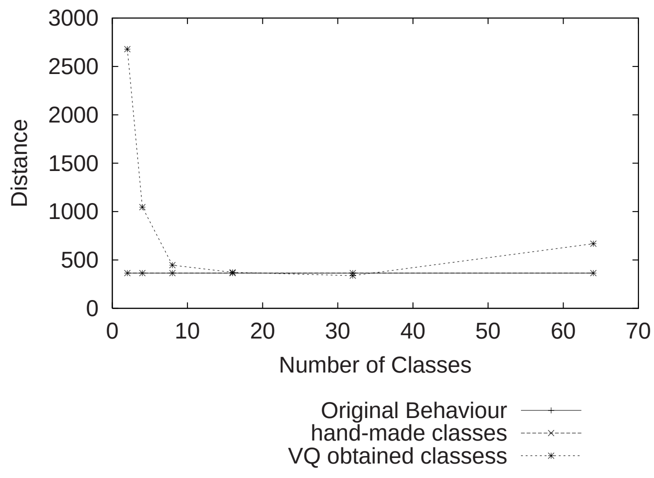


Fig. 4. Covered distance.

When we use the models obtained by discretizing the classes with GLA, we observe that with only 2 and 4 different classes, results are worse than previous solutions, as well as when the number of classes is very high (32 and 64 classes). In the first case, it is because the number of different classes or actions that the robot can execute is very low, so although the modelling task was very succesful (as shown in Table 4.2), the learned model is not good. In the second case, despite the fact that the model with 32 classes displays similar results for average distance with original model, the average time is much worse. This is duel.

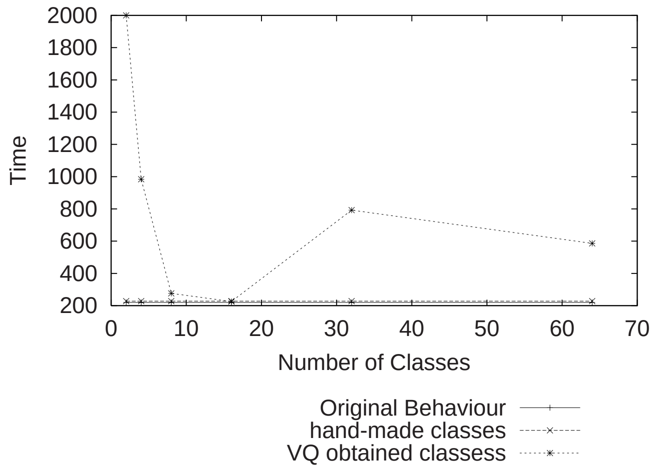


Fig. 5. Covered time.

to this model sometimes generates trials where the robot is blocked and does not move, covering a low distance, but without achieving the goal in the maximum time set ($time < 2000$). When 8 and 16 classes are used to discretize the network output, the model obtained behaves very similarly to the ones obtained by the original model and by the model with classes discretized by hand.

5 Conclusions

Results show that our approach is quite good when modeling simple neural agents evolved by means of coevolution in a goal-seeking obstacle-avoiding robot domain. In this paper we have used a machine learning technique, c4.5, to obtain the modeling knowledge after applied the Lloyd algorithm to automatically obtain a small discretized set of classes.

To discretize the data is needed when using classification techniques that require a finite set of classes and when requiring the outputs are easily understandable by humnas. In this work, we have used two approaches to do this quantification. The first one is to do it by hand, so the set of classes is obtained after an extensive analysis of the data, locating the quantification levels in those places where clusters were visually identified. However, this process can not be always done by hand. For instance, if we had a NN with several outputs, multidimensional data should have to be quantified, requiring an automatic method. In this work, we have used a vector quantization method that computes the quantization levels, allowing to design a model that obtains results very similar to the results obtained by the original model and the model designed with classes computed by hand (which is very good because of the extensive analysis required to generate this approach).

In the future we want to extend our approach with the following ideas. First, to test our approach with other obstacle configurations. Second, to make some agents learn models of other agents so that they can cooperate together. Last, in this experiments, the acquisition of models of other agents is off-line. However, it would be very interesting that an agent could learn about other agents by observing them on-line.

References

1. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back–propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.
2. Leslie P. Kaelbling, Michael L. Littman, and Andrew W. Moore, "Reinforcement learning: A survey," *Int. J. of Artificial Intelligence Research*, pp. 237–285, 1996.
3. Antonio Berlanga, Araceli Sanchis, Pedro Isasi, and José M. Molina, "A general coevolution method to generalize autonomous robot navigation behavior," in *Proceedings of the Congress on Evolutionary Computation*, La Jolla, San Diego (CA) USA, July 2000, pp. 769–776, IEEE Press.
4. Jude W. Shavlik and Geoffrey G. Towell, *Machine Learning. A Multistrategy Approach.*, vol. IV, chapter Refining Symbolic Knowledge using Neural Networks, pp. 405–429, Morgan Kaufmann, 1994.
5. Ricardo Aler, Daniel Borrajo, Inés Galván, , and Agapito Ledezma, "Learning models of other agents," in *Proceedings of the Agents-00/ECML-00 Workshop on Learning Agents,*, Barcelona, Spain, June 2000, pp. 1–5.
6. J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.
7. S. P. Lloyd, "Least squares quantization in *pcm*," Unpublished Bell Laboratories Technical Note. Portions presented at the Institute of Mathematical Statistics Meeting Atlantic City, New Jersey, September 1957. Published in the March 1982 special issue on quantization of the IEEE Transactions on Information Theory, 1957.
8. J. Ross Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, CA, 1993.
9. L. Breiman, J.H. Friedman, K.A. Olshen, and C.J. Stone, *Classification and Regression Tress*, Wadsworth & Brooks, Monterey, CA (USA), 1984.
10. J. Ross Quinlan, "Combining instance-based and model-based learning," in *Proceedings of the Tenth International Conference on Machine Learning*, Amherst, MA, June 1993, pp. 236–243, Morgan Kaufmann.
11. Antonio Berlanga Agapito Ledezma and Ricardo Aler, "Extracting knowledge from reactive robot behaviour," in *Proceedings of the Agents-01/Workshop on Learning Agents,*, Montreal, Canada, 2001, pp. 7–12.
12. Allen Gersho and Robert M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, 1992.
13. Fernando Fernández and Daniel Borrajo, "VQQL. Applying vector quantization to reinforcement learning," in *RoboCup-99: Robot Soccer World Cup III*, number 1856 in Lecture Notes in Artificial Intelligence, pp. 292–303. Springer Verlag, 2000.
14. V. Braitenberg, *Vehicles: experiments on synthetic psychology*, MIT Press, Massachusets, 1984.
15. J. Ross Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.
16. L. Sommaruga, I. Merino, V. Matellán, and J.M. Molina, "A distributed simulator for intelligent autonomous robots," in *In Proccedings of Fourth International Symposium on Intelligent Robotic Systems*, 1996, pp. 393–399.