Housebreaking Your Robot: The Observe-Train-Perform Algorithm

Madeleine Abromowitz

May 13, 2009

Abstract

Observe-Train-Perform (OTP) is an algorithm designed to allow a robot to learn new commands at any time. Using a modified the Growing Neural Gas algorithm, OTP parses continuous input, such as the shape of a hand, into discrete categories. Each category is assigned a neural network that is trained to perform the desired action using reinforcement learning, implemented in a variation on Complementary Reinforcement Back-Propagation. OTP is tested in on a dog-shaped Sony Aibo robot, which is taught to recognize a set of hand shapes and respond by barking, wagging its tail or sitting still. Results, though positive, show shortcomings in the algorithm that will have to be addressed in further experimentation.

1 Introduction

Robots are often used in settings where they can become more useful by learning new tasks. Household robots, in particular, can benefit from having the flexibility to learn new abilities according to the needs of a human being. However, the user may not have the skill or the inclination to reprogram the robot himself. Another, more universal approach is to create a robot that can be trained the way one might train a dog: by teaching it to associate a certain action with an arbitrary hand signal, word, or whistle. Furthermore, it would be useful if the robot could learn new cue/action pairs at any time, so that the user does not have to anticipate at one time all tasks the robot might ever need to perform.

This paper describes an algorithm, called 'Observe-Train-Perform' (OTP), which is designed to accomplish these two goals: arbitrary association and unlimited cue learning capacity. OTP relies heavily on two existing algorithms. One is Growing Neural Gas, which allows the robot to distinguish between and identify visual cues.[1] The other is Complementary Reinforcement Back-Propaagation, which is a method of reinforcement learning for neural networks.[2] As proof-of-concept, OTP was implemented on a Sony Aibo robot, which was taught to recognize several hand shapes and to bark, wag its tail, or be still, according to the hand signal it was shown. Results indicate the viability of the algorithm, although there are difficulties that must be addressed before OTP is applied to more complex problems.

2 Related Work

2.1 Dogged Learning

Daniel H Grollman and Odest Chadwicke Jenkins at Brown University developed a different solution to the same problem in an algorithm they call 'dogged learning.'[3] Dogged learning allows a robot to operate independently while a human teacher may intervene and correct the robot's behavior whenever necessary. The algorithm, which is task- and platform-independent, consists in arbitrating between output from a teacher (the human) and output generated by the robot, based on measures of confidence calculated for each source. Grollman and Jenkins implemented Dogged Learning on an Aibo, teaching it to move its head in the direction of its tail and to follow a ball. OTP does not have a mechanism arbitrate between student and teacher-produced output. Whenever human reinforcement is provided, the system accepts it. However, in the Dogged Learning experiment, a hand-coded controller served as the teacher, which had a confidence measure of 100 percent whenever activated, and therefore always superceded output from the learning robot. The arbitration was thus effectively performed by the human, as with OTP. Unlike OTP, the two experiments performed with Dogged Learning were specific to the type of input (position of a ball or angle the tail). OTP builds on this, offering a general approach to dealing with visual cues of any kind.

2.2 Incremental Topology Preserving Map

Another similar idea, called an 'incremental topology preserving map,' or ITPM, was designed in 2002 by Jose del R. Millan, Daniele Posenato, and Eric Dedieu at the European Commission's Joint Research Centre in Italy.[5] Like OTP, ITPM consists of a self-organizing network that represents the input space with a unit graph, wherein each unit contains motor-action information that is updated using reinforcement learning. ITPM is based on Growing Neural Gas, as is OTP, but uses a variation on Q-learning instead of back-propagation on neural networks, and is designed to support online training, while with OTP training must be done offline. A wall-following experiment showed this 'continuous' Q-learning algorithm to outperform standard 'discrete' Q-learning. ITPM is similar in concept to OTP, but a robot running ITPM does not accept human correction during learning.

2.3 Face recognition

A third related study, done in 2006 by Shireen M. Zaki and Hujun Yin at the University of Manchester, applied the a variation on Growing Neural Gas algorithm to the task of face recognition.[4] Zaki and Yin found that classification of facial images in a GNG-based classifier could be improved with some supervision, by training the classifier on both labeled (pre-classified) and unlabeled images. This idea has not yet been incorporated into the OTP algorithm, but would be an avenue for future work, as will be discussed.

3 Supporting Algorithms

3.1 Growing Neural Gas

Growing Neural Gas (GNG), created by Bernd Fritzke, is an algorithm for learning the topology of an input space and for classifying points in that space. A Growing Neural Gas is a self-organizing structure that stores a set of vectors, called 'model vectors,' which are roughly evenly distributed throughout the input space. A GNG grows in real time as a new vector from the input space is sampled at each timestep, and there is no limit on the number of model vectors that may be added. The algorithm depends works as follows:[1]

- Initialize two model vectors. Usually, these are random vectors of the appropriate length.
- At each step:
  - Sample the input space
  - Find the two model vectors closest and second-closest (in Euclidean distance) to the input vector
  - Update the ages of all edges that have the closest model vector as a vertex
  - Increment the error of the closest model vector by its distance from the input vector

- Adjust the closest model vector to be closer to the input vector, according to the parameter winnerLearnRate; also adjust its neighbors (the model vectors connected by an edge to the best model vector), according to the parameter neighborLearnRate
- If the closest and second-closest model vectors have an edge between them, set its age to zero; otherwise, create an edge between them.
- Remove all edges with age larger than maxAge, and remove any model vectors left without edges.
- If the number of input vectors seen since a model vector was last added goes above a certain threshold, add a new model vector between the two model vectors with highest error, and update their errors.
- Decrement the errors of all model vectors according to the parameter reduceError

3.2 Complementary Reinforcement Back-Propagation

Complementary Reinforcement Back-Propagation, designed by David Ackley and Michael Littman in 1990, is a form of reinforcement learning designed to train a neural network. In normal reinforcement learning on a neural network, the network is fed inputs; it yields outputs; and a teacher gives the network positive, negative, or no reinforcement based on the input-output combination. If the reinforcement is positive, the network is trained through back-propagation, using the outputs as target output. If the reinforcement is negative, the network is trained on a random target. If there is no reinforcement, nothing is done. The distinguishing feature of Complementary Reinforcement Back-Propagation is that when there is negative reinforcement, the network is trained on the complement of the outputs that the network produced. The original output is converted into a binary list, in which each bit is complemented (0 becomes 1 and 1 becomes 0), and the resulting list is used as the target output. This way, the network is more likely to produce outputs different from the negatively-reinforced output than it would be if negative reinforcement produced a random target.

Another feature of Complementary Reinforcement Back-Propagation is that after positive reinforcement, the network is trained until its discretized outputs match the target output. After negative reinforcement, the network is trained until the outputs no longer match the original outputs.[2]

4 Algorithm

Observe-Train-Perform a three-stage algorithm combining adaptations on Growing Neural Gas and Complementary Reinforcement Back-Propagation. In the 'observation' stage, the robot learns to recognize and distinguish between the cues that will be used for learning. In the 'training' stage, the robot is taught to associate actions with cues. In the 'performance' stage, no learning takes place; the robot is given cues and performs the appropriate actions.

4.1 Observation

The robot observes visual cues and creates a group of representative images in real time. These representative images are the model vectors in a modified version of a Growing Neural Gas. In this experiment, the GNG's first two model vectors are the first two images received from the camera, rather than two random vectors. It was determined experimentally that this method produced better model vectors, especially when the observation stage was short (making it more likely that the two random vectors would remain close to where they started). In another alteration on the original GNG, new model vectors were added when the average error over all model vectors surpassed a constant, errorToInsert. This change resulted in new model vectors appearing less frequently when the robot was shown one cue for a long period of time, and resulting in a better distribution of model vectors around the input space when the observation period was short.

The observation stage ends when the human is finished showing the robot cues and indicates (by pushing a button or any other means) that the training stage should begin.

4.2 Training

The robot is taught what the cues mean. During this stage, the model vectors are each assigned a neural network which may be trained, using reinforcement learning, to produce output according to the desired action associated with that vector. Assigning each model vector its own neural network, rather than creating one universal neural network with model vector numbers as inputs, was found to speed up training and to reduce the likelihood of catastrophic forgetting, wherein old learned behavior is replaced by new learned behavior.

It is likely that some cues will map to more than one model vector. If there are three vectors representing a hand in a fist, and one of these already has a neural network that has been trained to produce barking, then it would be redundant to train a new neural network for each fist model vector and train each network to bark. Instead, this algorithm uses the Growing Neural Gas to find existing neural networks that have already been trained to perform the correct action.

At each timestep, the robot takes in a processed image from the camera, converts the image into a vector, and queries the Growing Neural Gas for the model vector most similar to that image. If that 'best-fit' model vector already has a neural network attached to it, that network is used to produce motor outputs, which the robot performs. The robot then receives reinforcement (reward or punishment) from the human being, and the network is trained accordingly. If the model vector most similar to the input vector has no network associated with it, then the algorithm examines the next closest model vector and so on, until it arrives at a model vector with a network. (Upon entering the training stage, one of the model vectors is assigned a network with randomized weights and activations so that this process must end.) If the outputs of that network result in positive reinforcement, then a copy of that network is attached to the 'best-fit' model vector. If the robot receives no reinforcement or negative reinforcement, then it is presumed that there is no existing neural network that has been trained to perform the action associated with the cue given. The algorithm stops looking for an existing neural network that performs the right action, and instantiates a new neural network with random weights and activations that is assigned to the 'best-fit' model vector. The outputs of this network are converted to actions that the robot performs, and the system awaits reinforcement.

The neural network is trained using a variation on Complementary Reinforcement Back-Propagation. In the original CRBP, outputs are discretized (turned into binary values) stochastically, so that new actions are occasionally introduced. This randomness is desirable, but in OTP, it is introduced differently. When searching for a network to copy to a particular model vector, the algorithm must be sure without extensive testing that the network it is looking at will yield the right output all the time. In the version of CRBP used in this algorithm, outputs are discretized deterministically, so that when networks are being copied from one model vector to another, it is sufficient to see that the network produces the right output once to know that it will always produce the same output (given the same input). The element of randomness is instead introduced in the creation of a target output after negative reinforcement, similar to regular reinforcement back-propagation. Given a vector of floating-point outputs in the range [0, 1], all values less than 4.95 are converted to 0, and all greater than 5.05 are converted to 1. Values within the range [4.95, 5.05] are converted stochastically to 0 or 1 depending on their value; for instance, the value 5.01 (4.05 plus 60 percent of .1) has a 60 percent chance of becoming a 1. This randomness ensures that a neural network is capable of producing any output given any initial input. If, for instance, the network's initial output for a certain input were [1, 1], there might be no way to produce the output [0, 1], because negative reinforcement would always train the network on the target [0, 0].

In this algorithm, the repeated training feature of CRBP is omitted, due to the nature of the task. When training a robot in real-time, it is possible to accidentally give the wrong reinforcement, or to give the right reinforcement too late, so that the robot interprets it as pertaining to its next action.

The learning algorithm must therefore be forgiving. In the experiment described in this paper, it was found that training the network just once after either positive or negative reinforcement still resulted in successful training that was sufficiently fast for this task. However, the learning rate (epsilon value) of the network does change depending on the type of reinforcement: positive reinforcement is trained at a higher epsilon value (0.2) than negative reinforcement (0.1). This is to ensure that a positive target – which is necessarily an action that the user wants – gets more weight than a negative target, which may or may not represent the desired action.

4.3 Performance

Finally, when the human is satisfied that the robot has learned to associate the correct actions with the cues, the robot is set to performance mode, where it stops learning entirely and merely responds to input. The robot can be set back to observation or training mode and resume learning cues or actions at any time.

5 Experiment

In this experiment, a Sony Aibo robot was taught to respond to cue cards depicting various hand signals. The Aibo was controlled wirelessly from a computer using Pyrobot and Tekkotsu software. In response to a cue card, the robot could bark, wag its tail, do neither, or do both. Actions were limited to barking and tail-wagging because these allowed the robot to remain stationary throughout the experiment. The cue cards showed solid blue hand silhouettes on a white background and were placed on a white wall approximately 10 cm from the Aibo's camera, so that the Aibo could not see anything except a white hand on a blue background. The input to the robot at each timestep was a vector containing a color value for each pixel in the robot's vision at that instant. The data from the camera was preprocessed with a color threshold filter and rendered monochromatic (black or white), so that each vector was composed of entries of 255 (black) and 0 (white). The light in the room was controlled so that the color filter picked up only the shape of the hand on the cue card; the rest of the field of view was white, with some small black shadows occasionally appearing close to the edge of the field of view. Given this input, it was found that a GNG error threshold of 60000000 and learning rates of 0.2 (for the best model vector) and 0.006 (for its neighbors) yielded a good distribution of model vectors during this stage. (For more parameters, see Appendix.)

During the observation stage, three cue cards were shown to the robot for a period of approximately 25 steps each. These cue cards respectively depicted a hand in a fist, a hand with two fingers raised, and a hand with all five fingers raised.

At the end of the observation period, a pressing a button on the robot's head initiated the training stage. During this stage, the robot would perform an action and wait three seconds for reinforcement. Reinforcement was administered through buttons on the robot's back – the front button indicating reward, the rear button indicating punishment.

Reinforcement was not weighted in any way; pressing either button always resulted in one unit of reward or one unit of punishment.The goal was to teach the robot the following cue-action pairs:

- Fist: wag tail
- Two fingers: do nothing
- Open hand: bark

All neural networks attached to model vectors in this experiment contained only three nodes: one input and two outputs, with no hidden layer. Because output did not depend on input, input was always [1]. The two output nodes represented Booleans controlling barking and wagging.

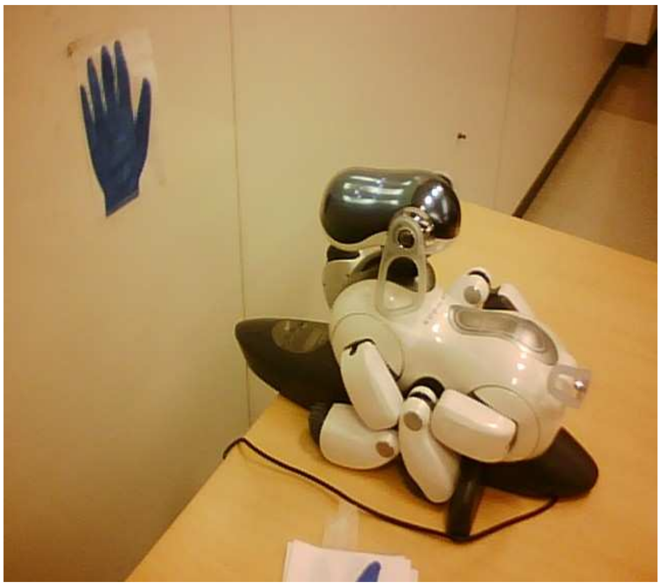

Figure 1: The Aibo in observation mode.

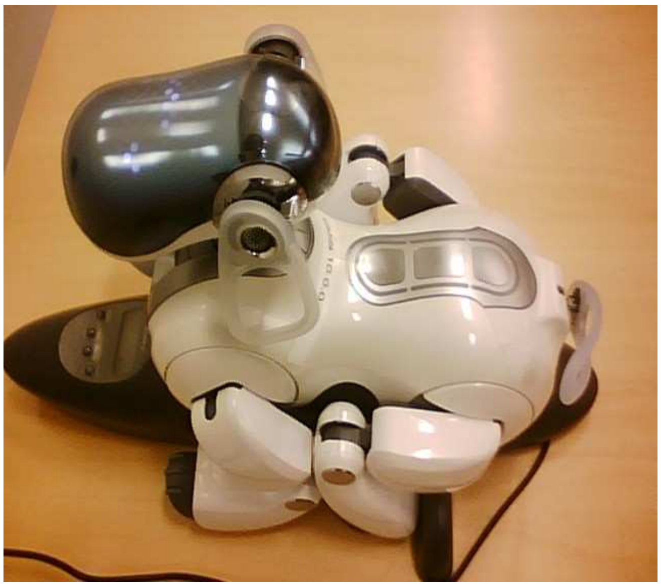

Figure 2: Buttons on the Aibo's back were used to administer reward and punishment.

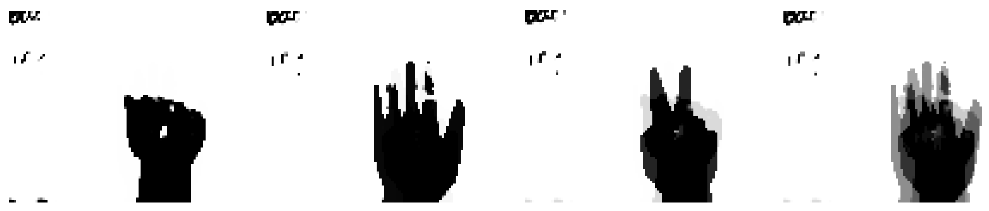
Figure 3: The four model vectors after seventy-five steps of observation.

When the training stage was finished, the robot was put in performance mode, and then back into observation mode to learn a fourth cue, a flattened hand in profile (figure). During this second observation stage, the neighborLearnRate in the Growing Neural Gas was reduced from 0.006 to 0.001. This helped keep the model vectors in the pre-existing Growing Neural Gas mostly in place, so that a model vector that had already been trained to perform a certain action did not become too similar to a new cue with a different action associated with it. If further observation periods had been run later in this experiment, they would also have begun with a neighbor learning rate of 0.001.

6 Results

The results of this experiment were positive but show the need for certain improvements. One representative run will be examined here. After the observation stage, the Growing Neural Gas contained four model vectors, as shown in (Figure). Three were clearly representations of the three hand shapes. The fourth resembled the open hand, overlaid with the faint images of the other two hands. It was therefore expected that in the training stage the Growing Neural Gas would identify the fist with model vector 1, the two fingers with model vector 3, and the open hand with model vectors 2 and 4.

The robot was first shown the fist, for which model vector 1 was selected as the best match, as expected. Since model vector 1 had no network attached to it, the algorithm looked next at model vector 4, the second closest fit, and seeing that it also had no network, looked at model vector 3, which did have a network. The output of this network happened to be [0, 1], meaning 'don't bark, do wag tail,' which received positive reinforcement. A copy of that network was accordingly made and attached to model vector 1. The fist cue was shown continuously and model vector 1 was continuously identified as the best fit, and the robot was trained to wag its tail. This took only one step of positive reinforcement, as the network happened to be randomly configured to produce the right output, and was considered finished when the network produced the right output three times in a row (chances of this happening randomly were one in 64).

Next, the robot was shown the hand with two fingers raised. As expected, model vector 3 was identified as the closest, and since model vector 3 already had a network attached to it, training began right away. Since the initial behavior (don't bark, do wag) was different from the desired behavior for this cue (do nothing), and since mistakes were made in administering reinforcement (some reinforcement was not administered in time), training this network took 27 steps, or approximately 80 seconds.

Finally, the robot was shown the third cue, the open hand. Unexpectedly, model vector 4 was selected as the best fit, rather than model vector 2. Though model vector 4 resembled the open hand more closely than it resembled any of the other cues, model vector 2 appeared to resemble the open hand better than any of the other model vectors. The selection of 4 rather than 2 may have been caused by the fact that the cue card was removed after the observation period and replaced during training, and may have been slightly displaced from its original position. Model vector 4 may have been a closer match to the cue in its second position because the edges of the hand in vector 4 were less clearly defined than in model 2. Because model vector 4 had no network, model vector 2 was

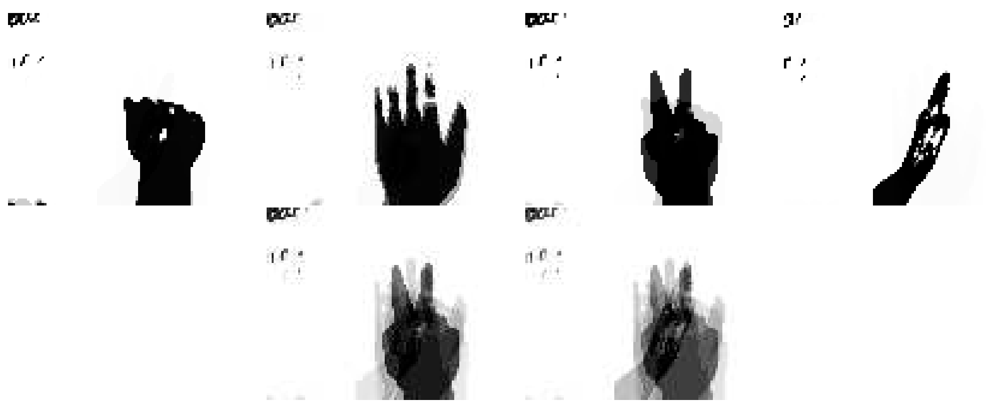

Figure 4: The six model vectors after a second round of observation.

looked at next; because 2 also had no network, output was taken from the network attached to model vector 3. Since this output (wag) resulted in punishment, a new network was created for model vector 4 and training began to teach the robot to bark. Partway through training, model vector 2 replaced model vector 4 as the closest fit to the input, probably due to fluctuations in the input due to noise. Since model vector 2 did not have a network attacked to it, the algorithm checked model vector 4, as expected. Although the network attached to model vector 4 had not finished its training, and the output was wrong (wag tail, rather than bark), reward was administered accidentally, and the network was copied to model vector 2. After that step, model vector 4 was once again identified as closest to the cue, and training on model vector 4 continued for a few steps. However, this was interrupted when model vector 3 was unexpectedly identified as the best match for the input, although the image from the camera did not appear to have changed in any major way. This misidentification resulted in model vector 3 being retrained on the wrong input, and model vector 4 not being trained.

When the system was set back to observation mode and presented with the fourth cue – the flat hand – two new model vectors were created, as shown. (Figure) The existing model vectors were not visibly affected by the addition of more, and the 4th model vector clearly represents the new cue.

7 Discussion and Future Work

This experiment, though mostly successful, but points to several aspects of OTP that call for improvement. First, better image processing tools would allow for more practical cues than cue cards and actions more complex than barking and wagging. In this experiment, camera output was converted to a vector of pixel values, which meant that a fist in the lower right-hand corner of the robot's field of view would produce input entirely different from a fist in the upper left-hand corner, and that the same fist viewed from closer or further away would produce different input altogether. The robot was therefore kept stationary throughout the experiment. In addition, the Aibo's camera was very sensitive to changes in ambient light, and moving the robot towards or away from the cue cards could cause the image to become washed-out or so dark that the shape on the card was not visible. Furthermore, if there had been any moving blue objects in the background, they would have passed through the color filter and affected the model vectors. More advanced image processing algorithms, capable of recognizing shapes despite changes in position, distance, and lighting, could make it possible to use real hands instead of cue cards, and to allow the robot to move its body. Zaki and Yin's findings on guiding the Growing Neural Gas by training it on labeled and unlabeled cues might also provide a way to make categorization more robust, although this would also demand more human involvement in the learning process.

Second, no two different signals should ever point to the same model vector, as they did in this experiment. This problem arises when a model vector resembles both cues and the cues given are ambiguous enough that they both map to the same vector. This problem will never be completely eliminated, since cues can always be ambiguous, but with better image pre-processing, it may be reduced. In this experiment, small changes in position – a cue card being removed and put back in a slightly different place – resulted in large changes in the input vectors, making it more likely that even precisely the same shape would be misinterpreted.

8 Conclusion

In a qualitative analysis of a simple experiment, Observe-Train-Perform was successful at categorizing semi-discrete visual cues from continuous input; at learning to associate actions with each of these cues; and at adding new cues after training has been performed. It also dealt gracefully with mistakes in the administering of reinforcement, which is important for any system interacting with humans. However, the problem of multiple cues mapping to the same model vector, and therefore the same behavior, remains to be solved. Better image processing algorithms will be needed if the algorithm is to be applied to more complex problems.

9 Appendix

Growing Neural Gas Parameters:

- winnerLearnRate = 0.2
- neighborLearnRate = 0.006 for initial training; 0.001 thereafter
- maxAge = 50
- reduceError = 0.995
- errorToInsert = 60000000
- insertError = 0.5

Neural Network Parameters:

- Epsilon (positive reinforcement): 0.2
- Epsilon (negative reinforcement): 0.1
- Momentum: 0

References

[1] Bernd Fritzke. A growing neural gas learns topologies. *Advances in Neural Information Processing Systems 7*, 1995.

[2] David H. Ackley, Michael L. Littman Generalization and Scaling in Reinforcement Learning In *Advances in Neural Information Processing Systems 2*, 1990.

[3] Daniel H. Grollman. Odest Chadwicke Jenkins. Dogged Learning for Robots IEEE International Conference on Robotics and Automation, 2007.

[4] Shireen M. Zaki. Hujun Yin. A Semi-Supervised Learning Algorithm for Growing Neural Gas in Face Recognition In *Journal of Mathematical Modelling and Algorithms*, Vol 7, no 4, pages 425-435, 2008.

[5] Jos del R. Milln. Daniele Posenato. Eric Dedieu. Continuous Action Q-Learning. In *Machine Learning*, Vol 49, no 2-3, pages 247–265, 2002.