ESANN'2005 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 27-29 April 2005, d-side publi., ISBN 2-930307-05-6.

Adaptive robot learning in a non-stationary environment

Kary Främling

Helsinki University of Technology, Department of Computer Science, FI-02015 HUT, Finland
Kary.Framling@hut.fi

Abstract. Adaptive control is challenging in real-world applications such as robotics. Learning has to be rapid enough to be performed in real time and to avoid damage to the robot. Models using linear function approximation are interesting in such tasks because they offer rapid learning and have small memory and processing requirements. This makes them suitable as adaptive controllers in non-stationary environments, especially when the controller needs to be an embedded system. Experiments with a light-seeking robot illustrate how the robot adapts to the environment by Reinforcement Learning where the robot collects training samples by exploring the environment.

1 Introduction

The use of machine learning in real-world control applications is challenging. Real-world tasks, such as those using real robots, involve noise coming from sensors, non-deterministic actions and uncontrollable changes in the environment. In robotics, learning must be relatively rapid and possible to perform without causing damage to the robot. Only information that is available from robot sensors can be used for learning. This means that the learning methods have to be able to handle partially missing information and sensor noise, which may be difficult to take into account in simulated environments.

Artificial neural networks (ANN) are a well-known technique for machine learning in noisy environments. In real robotics applications, however, ANN learning may become too slow to be practical. One-layer linear function approximation ANNs (often called Adalines [7]) offer faster training than non-linear ANNs and their convergence to an optimal solution can usually be guaranteed. These are properties that are particularly useful in non-stationary environments that require rapid adaptation, especially if the robot has to explore the environment and collect training samples by itself. Learning by autonomous exploration of the environment is often performed using reinforcement learning (RL) methods. Finally, the limited memory- and computing power needs of Adalines make them easy to use in embedded systems.

The structure of this paper is as follows. Section 2 gives background information about gradient descent learning and RL, followed by experimental results in Section 3. Related work is studied in Section 4, followed by conclusions.

2 Gradient descent reinforcement learning

In gradient descent learning, the free parameters of a model are gradually modified so that the difference between the output given by a model and the corresponding "correct" or target value becomes as small as possible for all training samples available. In such supervised learning, each training sample consists of input values and the corresponding target values. Real-world training samples typically involve noise, which means that it is not possible to obtain a model that would give the exact target value for all training samples. The goal of learning is rather to minimize a statistical error measure, e.g. the Root Mean Square Error (RMSE)

$$RMSE = \sqrt{\frac{1}{M}\sum_{k=1}^{M}\left(t_j^k - a_j^k\right)^2} \quad (1)$$

where M is the number of training examples, t^k_j is the target value for output j and training sample k and a^k_j is the model output for output j and training sample k. In RL tasks, each output a_j typically corresponds to one possible action.

RL differs from supervised learning in several ways. The agent has to collect the training samples by exploring the environment, which forces it to keep a balance between exploring the environment for new training samples and exploiting what it has learned from the existing ones. In supervised learning, all training samples are usually pre-collected into a training set, so learning can be performed off-line. In RL, target values are only available for used actions whereas in supervised learning target values are typically provided for all outputs (actions). Finally, in RL the target value may not be available directly; it may be available only after the agent has performed several actions. Then we speak about a delayed reward learning task.

RL methods usually model the environment as a Markov Decision Process (MDP), where every state of the environment needs to be uniquely identifiable. This is why the model used for RL learning is often a "lookup-table", where each environment state corresponds to one row (or column) in the table and the columns (or rows) correspond to possible actions. The values of the table express how "good" each action is in the given state, usually called the action-value. Lookup-tables are not suitable for tasks involving sensorial noise or other reasons for the agent not being able to uniquely identify the current state of the environment. Therefore state generalization techniques, e.g. ANNs, are often used instead of lookup-tables. Generalization assumes that an action that is good in some state is probably good also in "similar" states. ANNs can handle any state descriptions (not only discrete ones) so they are well adapted for problems involving continuous-valued state variables and noise, which is usually the case in robotics applications.

The simplest ANN is the linear Adaline [7], where neurons calculate their output value as a weighted sum of their input values

$$a_j(s) = \sum_{i=1}^{N} s_i w_{i,j} \quad (2)$$

where $w_{i,j}$ is the weight of neuron j associated with the neuron's input i, $a_j(s)$ is the output value of neuron j, s_i is the value of input i and N is the number of inputs. They are trained using the Widrow-Hoff training rule [7]

$$w_{i,j}^{new} = w_{i,j} + \alpha(t_j - a_j)s_i \quad (3)$$

where α is a learning rate. It can easily be shown that there is only one optimal solution for the error as a function of the Adaline weights, so gradient descent is guaranteed to converge if the learning rate is selected sufficiently small. In this paper, the well-known *normalized least mean square* (NLMS) method is used, where α in (3) is replaced by α_{norm}:

$$\alpha_{norm} = \alpha \Big/ \sum_{i=1}^{N} s_i^2 \qquad (4)$$

, which guarantees that the output value a_j is adjusted towards the target value t_j with the same ratio independently of the input values s_j, which simplifies determining a suitable learning rate and avoids differences in learning speed for input values of different magnitude as shown in [2].

In RL tasks, convergence of gradient descent cannot always be guaranteed. If the action selection policy π does not make the agent collect representative training samples, then learning may fail to converge to a good solution. Therefore, the action selection policy must provide sufficient exploration of the environment to ensure that "good" training samples are collected. At the same time, the goal of training is to improve the performance of the agent, i.e. the action selection policy. A commonly used method for balancing exploration and exploitation is to use *ε-greedy exploration*, where the greedy action is selected with probability (1-ε) and an arbitrary action is selected with probability ε using a uniform probability distribution. Another commonly used exploration method is *Softmax* that selects actions according to Boltzmann-distributed probabilities, where the randomness of action selection is adjusted by the so-called temperature parameter [7].

3 Experimental results

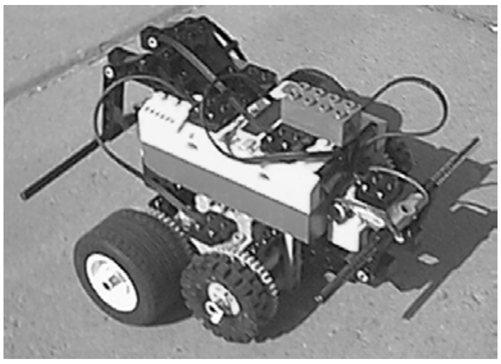

Fig. 1. Lego Mindstorms robot. Light sensor is at the top in the front, directed forwards (right in picture). Touch sensors are installed in the front and the rear.

Experiments were performed using a robot built with the Lego Mindstorms Robotics Invention System (RIS) programmed with the Java programming language. The robot has one motor on each side; touch sensors in the front and in the back and a light sensor directed straight forward mounted in the front (Fig. 1). Robots usually have more than one light sensor, which were simulated by turning the robot around and getting light readings from three different directions. One light reading was from the direction straight forward and the two others about 15 degrees left/right, obtained by letting one motor go forward and the other motor backward for 250 milliseconds and then inversing the operation.

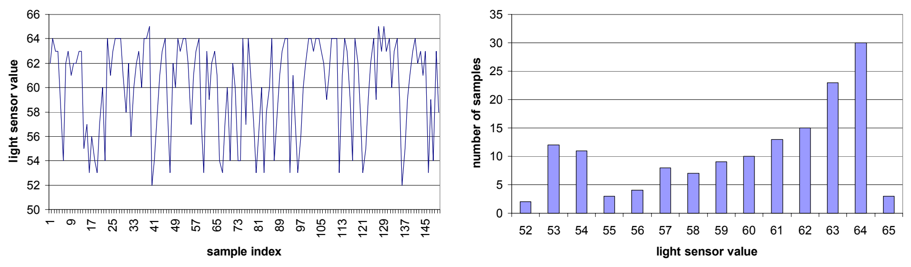

Fig. 2. 150 light samples for given light conditions, taken with 500 millisecond intervals. Average value is 60.0. Left: raw values; right: value distribution.

Performing the following motor commands for 450 milliseconds gives five possible actions: 1) both forward (advances about 5 cm), 2/3) one forward, other stopped (goes forward about 2 cm and turns about 15 degrees), 4/5) one forward, other backward (turns about 40 degrees). The robot starts directed straight towards the lamp at a distance of about 110 centimeters. A light value of 80 out of 100 signifies that the goal is reached, i.e. one to fifteen centimeters from the lamp depending on direction and sensor noise. Noise comes both from the environment (floor reflections, different colors etc.) and the robot itself (imprecise motor movements, battery charge etc.). The light sensor is the biggest source of noise as shown in Fig. 2.

There is one ANN output per action, where each output value corresponds to an action-value estimate $a_j(s)$ in equation (2). With five actions and three state variables, a 5x3 weight matrix is sufficient (no bias input used). The NLMS method is used with $\alpha = 0.5$ and ε-greedy exploration with $\varepsilon = 0.2$. The target value t_j in equation (3) comes from the light sensor reading in the forward direction after performing an action. Therefore the ANN learns to predict the new light value after taking an action, so the action with the highest $a_j(s)$-value is the one that should allow the robot to approach the light source the most. ANN weights are initialized with random values in the range [0, 1), which tends to give higher action-value estimates than the true ones. This technique is called *optimistic initial values* and favors exploring unused actions.

Multiplying the value of the left light sensor by 0.1 simulates a faulty sensor, which illustrates how an adaptive system can be used for calibrating the controller. Two cases were tested: 1) having a defective sensor from the beginning of training and 2) sensor becoming faulty after three episodes. Fig. 3 shows that the smaller values of the faulty sensor compared to the others tend to slow down gradient descent but it still converges to a good policy that continues improving after the tenth episode. When learning is started without faulty sensors and introducing the faulty sensor in the 4th episode, the agent adapts rapidly after some right-bound circuits or collisions. Starting with non-faulty sensors apparently makes final learning quicker compared to having a faulty sensor from the start. The up going "wave" observed in the graphs around the fourth and fifth episodes is due to exploration of new actions, which destabilizes already identified (but usually sub-optimal) solutions.

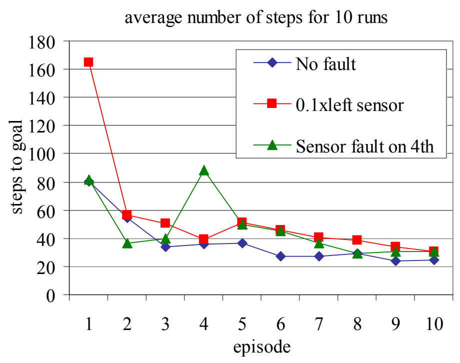

Fig. 3. Results for 1) no sensor fault 2) sensor fault from beginning 3) sensor fault starting from fourth episode.

Fig. 4 shows the number of manual resets that occur when the robot drives against a wall or passes behind the lamp. The closest wall was only about 40 centimeters away from the straight line from start to goal, so collisions sometimes occur due to ε-greedy action selection even with a perfect controller. The number of collisions is great for the third agent on the fourth episode when the sensor signal has changed, but such collisions are avoided already on the next episode.

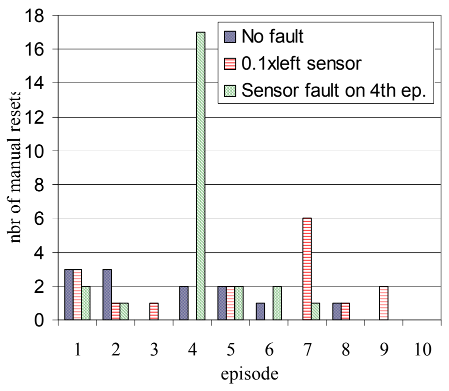

Fig. 4. Number of manual resets per episode.

4 Related work

Due to the challenges in real-world robotics applications, most experimental RL work has been done in simulated environments. RL has still been used with real robots by some researchers. Tasks include wall following, going through a door, docking into a charger [3], box finding, box pushing, unwedging from stalled states [4] and selecting the appropriate pre-programmed behaviour to use [5]. Millán [6] and Dorigo and Colombetti [1] have tested similar tasks as the one used here, but using much more complex learning methods than those described here. Both sonar and light sensors have been used, but state variables are usually converted into binary values. The number of actions varies from five to sixteen. What is common to these approaches is that they tend to use complex function approximators. They also require a lot of pre-existing knowledge to be able to learn successfully. As in the experiments performed in this paper, they also all use immediate reward to guide learning efficiently. The work presented in this paper differs from these approaches in several ways: 1) Pre-existing knowledge: the only assumption used here is that a linear function approximator is sufficient; 2) learning is adaptive to changes in the environment (e.g. moving the goal) or in the agent itself (e.g. faulty sensors) and 3) simplicity that makes learning robust and rapid.

5 Conclusions

As shown by the results of this paper, the elementary behavior of moving towards a light source can be learned using a simple linear model. It is also shown that such a learning system can rapidly adapt to situations occurring in real-world non-stationary environments, where sensor calibration and noise are issues to take into account. The simplicity of the learning system makes it interesting compared to related work, taken that moving towards a light source is a task performed successfully even by simple organisms in nature.

Using an Adaline avoids discretization of the state variables, which is usually done in RL applications. Discretization tends to make hidden state problems worse unless using a very fine discretization, but then the size of the state space grows exponentially, necessitating the use of discounted reward and more exploration. Future work will consider delayed reward tasks, which may also necessitate using non-linear ANNs.

References

[1] M. Dorigo and M. Colombetti, *Robot shaping: an experiment in behavior engineering*, A Bradford book, USA, 1998.

[2] K. Främling, Scaled gradient descent learning rate - Reinforcement learning with light-seeking robot. In *proceedings of ICINCO'2004 conf.*, pages 3-11, 25-28 August 2004, Setubal (Spain), 2004.

[3] L.-J. Lin, Programming robots using reinforcement learning and teaching. In *proceedings of the Ninth National Conference on Artificial Intelligence (AAAI)*, pages 781-786, 1991.

[4] S. Mahadevan and J. Connell, Automatic Programming of Behavior-based Robots using Reinforcement Learning, *Artificial Intelligence*, 55(2-3):311-365, 1992.

[5] M.J. Mataric, Reward Functions for Accelerated Learning. In Cohen, W. W., Hirsch, H., editors, *Machine Learning: Proceedings of the 11th International Conference*, Morgan-Kaufmann, 1994.

[6] J.R. Millán, Rapid, Safe, and Incremental Learning of Navigation Strategies. *IEEE Trans. Systems, Man, and Cybernetics - Part B*, 26(3):408-420, 1996.

[7] S.B. Thrun, The role of exploration in learning control. In DA White & DA Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*. New York, NY: Van Nostrand Reinhold, 1992.

[8] B. Widrow and M.E. Hoff, Adaptive switching circuits. In *1960 WESCON Convention record Part IV*, Institute of Radio Engineers, New York, pages 96-104, 1960.