Fuzzy Neural Network Implementation of Q(λ) for Mobile Robots

AMIR MASSOUD FARAHMAND[1], CARO LUCAS[2]
(1) Department of Electrical & Computer Engineering
University of Tehran
North Karegar Street, Tehran
IRAN

Abstract: Programming mobile robots can be long and difficult task. The idea of having a robot learn how to accomplish a task, rather than being told explicitly is appealing. $TD(\lambda)$ implementation of Reinforcement Learning (RL) using Fuzzy Neural Network (FNN) is suggested as plausible approach for this task, while Q-Learning is shown to be inadequate. Although there is no formal proof of its superiority, but it did better in a simple simulation.

Keywords: Mobile robots, Reinforcement learning, Fuzzy neural network, $TD(\lambda)$

1 Introduction

The need for using robots is increasing. They can help us in a lot of jobs which are not suitable for humans, e.g. they can manipulate heavy materials, explore dangerous places or simply helping us clean the room. We will focus on one of the main problems, which prevents us to get maximum benefit from them. The problem is that, in general we do not know how a robot should do its job! In order to do so, we should design suitable controller for the robot but this job can be very time-consuming or even impossible due to the ill-known dynamics of robots and environment or more importantly, time-varying dynamics of the whole system. Our robot might be placed in some priori unknown environment and our desire is that it learns how to comply with what it is supposed to accomplish without knowing exactly how it should be done. Thus, it is necessary to have some learning mechanism that enable the robot to confront changing dynamics of environment and ideally learn all the controlling mechanism from scratch [1].
There are several learning methods in machine learning domain that are more proper in robotics [2]. On one side, we can use supervised-learning methods to enable the robot to imitate some prescribed behavior and on the other side, we can use reinforcement learning to enable it to change its behavior according to the reward it takes from some critic. The first method is well suited for the cases that a human operator (or some other experts, which might also be another robot) can do the robot's job well enough and the robot's controller changes in a way to imitate that behavior. On the other hand, RL is suitable whenever we do not know (or do not want to specify) how the robot should do a task, but know what is desirable to do and what is not, so we can reward or punish it accordingly. It is possible to mix these two methods and have something between them; the robot learns its general behavior by imitating an operator and fine tunes by RL[1].
Reinforcement learning is a well-known psychological phenomenon which many complex animals use it to adapt to their environment. Suppose an agent who perceives S state from its environment and act A action accordingly. Depending on properness of its action, it may receive reward or punishment. If the agent changes its behavior in order to increase the chance of receiving rewards in future, it is called that it uses RL as a learning paradigm. More formally, the agent can observe $s_t \in S$ from state space and act $a_t \in A$ from its action space at time t and accordingly it receives r_t from the environment.
The goal of RL is to maximize some optimality criterion that might be finite-horizon, average-reward, infinite-horizon discounted or some other model [3]. The infinite-horizon discounted model which is stated bellow is what most of RL literature is based on

$$E\left[\sum_{t=0}^{\infty}\gamma^t r_t\right] \qquad (1)$$

that E[.] is expectation operator and γ is discount factor which is $0 \le \gamma < 1$.

There are some method for learning proper action for a given state in order to maximize this criterion. One of most famous method that is often used in RL is Q-learning [4]. Q-learning is a good framework because it can be proved that whenever an agent is situated in a Markovian environment, the agent can learn optimal strategy if it visits each state infinitely. Although the convergence ensuring property is very good, but it is not all one wants from a machine learning method. There are two kinds of disadvantages in Q-learning. One of them is that it works in discrete spaces which is not of the kind of problems we face in real world and the next is it uses of $TD(0)$ ([5]) kind of temporal credit assignment which do not credit previous actions (except the just before one) whenever the agent receives reward or punishment. The first property forces us to discretize state-space. This can be severe problem when the dimension of that space is rather high (well-known curse of dimensionality) [3]. Beside that, selecting appropriate discretization is not so easy job. The later property slows down learning in situations that the agent should act some long chain of well-selected actions in order to receive reward. Noting that most real-world problems do not obey Markov process, we are encouraged to dismiss the original version of Q-learning (that has convergence proof) and extend it in order to be more suitable for our robotics problems, though we will loss some aspects of that well-defined theoretical basis. So we have extended Q-learning to have both these desirable features: dealing directly with continuous spaces and use $TD(\lambda)$ method for temporal credit assignment. We will discuss our method in section 2, experimental results will be given in section 3, the conclusion will be presented in section 4 and finally in section 5 we will state the references.

2 Neurofuzzy-based $Q(\lambda)$

In order to overcome the mentioned problems with ordinary Q-learning, we have extended it to have two previous properties. We have used Peng's $TD(\lambda)$ [6] extension of Q-learning that is an incremental multi-step credit assignment version of Q-learning to enable our agent to learn faster. Then we use some implementation of Fuzzy Neural Network (FNN) as Q-Table [7]. FNN can be a general function approximator with some special properties. First, it has linguistic interpretation that simplifies working with it [8]. We can easily convert it to some set of fuzzy rules. It has a good intuition to change a priori knowledge expressed with some fuzzy rules into this FNN framework too. Therefore, we can represent a rough control mechanism in our system and then let RL fine-tunes it. Beside that, simply replacing lookup tables with general function approximators has been shown that might cause learning to fail [9], [10], [11]. Without going to details, one cause of this problem is hidden extrapolation – the approximator should not extrapolate whenever it has not seen any input data in that part of space. FNN with compact membership functions is a local approximator and do not suffer from this problem or at least suffers less in comparison with MLP. Now we introduce $Q(\lambda)$ and then FNN will be discussed.

Suppose that we have a Markov Decision Process (MDP), which is defined as follow [3], [12]:

- a set of states S,
- a set of actions A,
- a reward function $R : S \times A \to \Re$
- a state transition function $T : S \times A \to P(S)$ in which $P(S)$ is the probability distribution of being in some specific state. We write $T(s,a,s')$ for the probability of making a transition from state s to state s' using action a.

The model is said to be Markovian if the state transition depends only on current state and action and not the previous ones. We want to seek a policy that maximizes some optimality criterion. Here we use infinite-horizon discounted model as introduced before. We will speak of the optimal value of a state, which is the expected reward gained by an agent if it starts at that state and executes the optimal policy. Using π as a policy, optimal value is written

$$V^*(s) = \max_{\pi} E\left(\sum_{t=0}^{\infty} \gamma^t r_t \right) \qquad (2)$$

Now we define $Q(s,a)$ which is a value of taking action a in state s. Note that $V^*(s)$ is a value of being in state s and taking best action, so $V^*(s) = \max_a Q^*(s,a)$. We can update our estimate of $Q^*(s,a)$ using

$$\hat{Q}(s_t,a_t) = \hat{Q}(s_t,a_t) + \alpha\left(r_t + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1},a_{t+1}) - \hat{Q}(s_t,a_t) \right) \qquad (3)$$

```
1. $\hat{Q}(s,a)=0$ and $Tr(s,a)=0$ for all s and a. We do this by zeroing every weight of FNN (we
   will discuss it more later).
2. Do Forever
    a. $s_t \leftarrow$ current state
    b. Choose an action $a_t$
    c. Carry out action $a_t$ in the world. Let the short term reward be $r_t$, and the new state be
       $s_{t+1}$
    d. $e_t = r_t + \gamma\hat{V}(s_{t+1}) - \hat{V}(s_t)$
    e. $e'_t = r_t + \gamma\hat{V}(s_{t+1}) - \hat{Q}(s_t,a_t)$
    f. For each state-action pair (s,a) do
        • $Tr(s,a) = \gamma\lambda Tr(s,a)$
        • $\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha Tr(s,a)$
    g. $\hat{Q}(s_t,a_t) \leftarrow \hat{Q}(s_t,a_t) + \alpha e'_t$
    h. $Tr(s_t,a_t) \leftarrow Tr(s_t,a_t) + 1$
```

Table 1- $Q(\lambda)$ algorithms

Without going into any details, we state that if each action is executed in each state an infinite number of times on an infinite run and α is decayed appropriately, the $\hat{Q}$ value will converge with probability of 1 to Q^* if the state-action space is discrete.
Now we are ready to introduce the $TD(\lambda)$ formulation of Q-learning as proposed by [6]. We define two prediction errors as follow, one for difference between predicted state value function ($r_t + \gamma\hat{V}(s_{t+1})$) and current estimate of value function ($\hat{V}(s_t)$) and the other for difference between predicted state value function (as before) and estimated state-action value $\hat{Q}(x_t,a_t)$ function.

$$e_t = r_t + \gamma\hat{V}(s_{t+1}) - \hat{V}(s_t) \tag{4}$$
$$e'_t = r_t + \gamma\hat{V}(s_{t+1}) - \hat{Q}(s_t,a_t) \tag{5}$$

where $\hat{V}(s_t) = \arg\max_{a_t} Q(s_t,a_t)$. In order to assign credit for previous visited state-actions with $TD(\lambda)$, we should know when they were visited before and thus they must be stored in some table which indicates when this particular state-action pair have been visited. But as for $\lambda \neq 0$, this credit assigning task extends to every previously seen steps, this table enlarges tremendously. One way of overcoming this problem is using finite-time window, which stores only some previous steps and not all of them. The other method is using some structure that inherently discounts the effect of previous state-actions. This structure is known as eligibility trace as described in [13]. This table can be a look-up table for discrete problems or some kind of general function approximator for continuous ones. We have implemented eligibility trace with the similar FNN to that we use for Q-Table. The algorithms of $Q(\lambda)$ is written in Table 1.
Before discussing FNN, it is necessary to mention some notes. First is the way we store these tables which as stated before is a FNN that will be discussed soon. The second is the way we choose action (part b of algorithm). One way is greedy action-selection in which the action that maximizes action-state value function will be selected ($a_t \leftarrow \arg\max_{a_t} \hat{Q}(s_t,a_t)$). However, in order to increase the exploration, we should have some amount of randomness. There are different techniques such as ε-greedy, choosing according to Boltzman probability distribution or simply adding some random noise to selected action. We have chosen ε-greedy because of its simple implementation. Although it might be not as good as Boltzman method, but it works. The third note to mention is experimentation-sensitiveness of $Q(\lambda)$ when non-greedy action selection is used. This is a big disadvantageous of this method [6]. Now we are ready to introduce our FNN.
Suppose that $s_t \in \Re^n$ and $a_t \in \Re$. We may have following simple Mamdani fuzzy rule-base in order to determine how much a given state-action is valuable

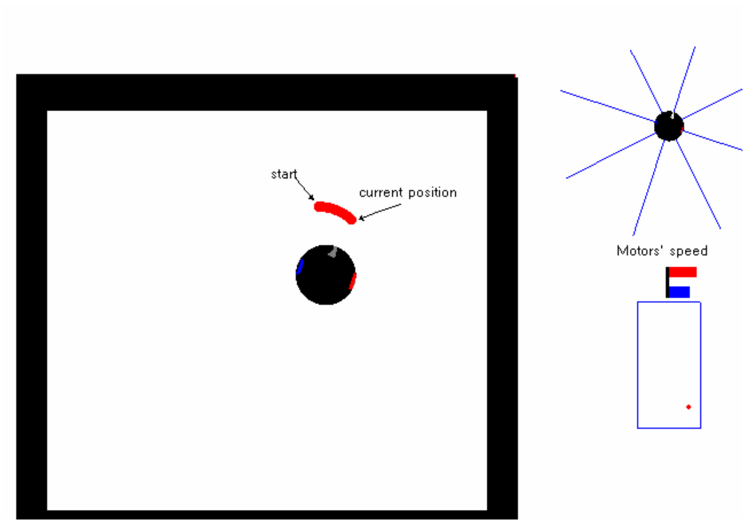

Fig 1. Simulator environment

Rule_1: If s_t^1 is S^{11} and … and s_t^n is S^{1n} and a_t is A^1 then value is q^1
Rule_2: If s_t^1 is S^{21} and … and s_t^n is S^{2n} and a_t is A^2 then value is q^2
…
Rule_m: If s_t^1 is S^{m1} and … and s_t^n is S^{mn} and a_t is A^m then value is q^m

If all of the state and action labels (S^{kl} and A^k) were crisp set, and the value is crisp too, it is exactly the same as look-up form of Q-Table. However, by having fuzzy labels, it becomes Fuzzy version of that. It is important to mention that the consequent part (value part) is a crisp value. In order to make this structure suitable for Q-Learning, we must make it flexible. In this naïve form, it is a rigid structure but it can simply be modified to acquire the ability of learning. Suppose that membership function of S^{ij} is $\mu_{ij}(x^j)$. Therefore, truth-value of antecedent part of i^{th} rule is

$$\sigma_i = T(\mu_{i1}(s^1), \mu_{i1}(s^2), ..., \mu_{in}(s^n), \mu_{in+1}(a)) \qquad (6)$$

If we use product as t-norm, we have

$$\sigma_i = \mu_{in+1}(a) \cdot \prod_{j=1}^{n} \mu_{ij}(s^i) \qquad (7)$$

Considering consequent part is a crisp value and using Center of Area (COA) method as defuzzifier, we have

$$Q(s,a) = \frac{\sum_{i=1}^{m} w_i . \sigma_i(s,a)}{\sum_{i=1}^{m} \sigma_i(s,a)} \qquad (8)$$

Thus, we could express the output of a defuzzified Fuzzy inference engine in a neural network-like structure. Learning is easy now. Having supervisor which provides the correct value of output (which is available in Q-learning formulation), we can express necessary changes of output weights this way

$$E = \frac{1}{2}\left(Q - \hat{Q}\right)^2 \overset{\text{steepest descent}}{\Rightarrow}$$
$$\frac{\partial E}{\partial w_i} = -\eta (Q - \hat{Q}) \frac{\sigma_i(s,a)}{\sum_{i=1}^{m} \sigma_i(s,a)}; \eta > 0 \qquad (9)$$

Although we can also change membership functions in a similar way (though much more involved formulas would be obtained), we avoid doing so because it is not much necessary and beside that, doing this job using some clustering method would give better results. It might be interesting to note that FNN is structurally similar to Radial Basis Function (RBF) Neural Networks. So although we look at them from different perspectives, some similar conclusion can be made. For instance by comparing FNN and RBF, it is evident that by choosing appropriate membership functions, we can make FNN a universal function approximator. Beside that, as mentioned before, we can use clustering method for finding parameters of membership functions as one can do for RBF [14]. In addition, by investigating the general form of FNN, we can interpret w_i as a crisp

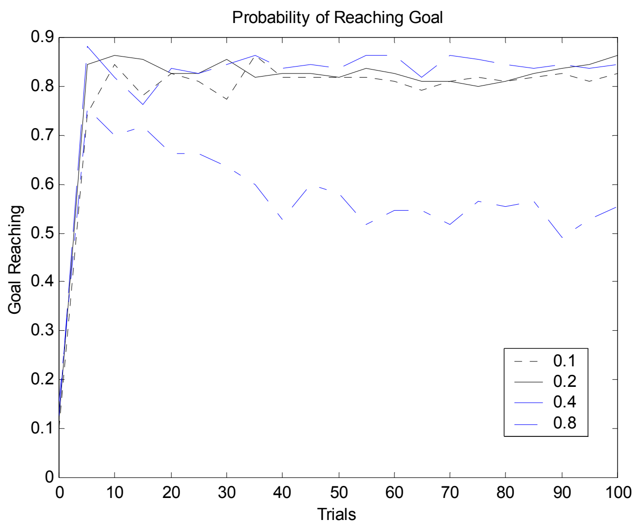

Fig 2. Comparison of results with different λ

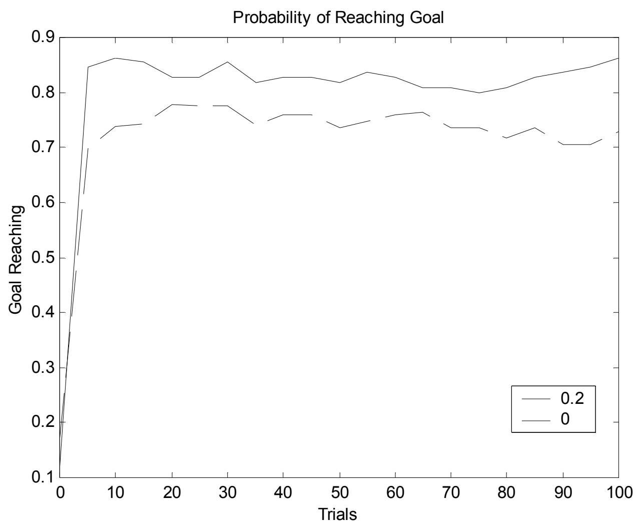

Fig 3. Comparison of $TD(0.2)$ and $TD(0)$ (simple Q-Learning)

consequent part, which is weighted by truth of antecedent, and learning can be thought as adjusting this crisp membership function (which is actually a singleton). We could consider some more general form of consequents (e.g. Fuzzy consequents), but it increases the computational complexity too much.

3 Experiments

For testing the proposed ideas, we have made a mobile robot simulator (Figure 1). This simulator enables the robot to have sonar sensors with beamwidth and distance precision uncertainty and also a simple vision system that can detect some predefined (e.g. balls) objects in the field. In this experiment, we want the robot to learn how to follow a moving object in the field. The robot uses its vision system to detect the relative angular distance of the object with its head. Then, it determines the relative speed of its two wheels. Therefore, the state space is $\Delta\theta$ (relative angular distance) and its output is Δv and the speed of each wheel is determined using following equations

$$V_{left} = V_0(1+\Delta v) \tag{10}$$

$$V_{left} = V_0(1-\Delta v) \tag{11}$$

Now it should be decided how to divide the state and action space and what kind of membership function to use. Gaussian membership function is used here. The centers and variances of membership functions are as follow

$\Delta\theta$: (-0.5,0.4), (-0.1,0.1), (0,0.05), (0.1,0.1), (0.5,0.4) (in radian) (12)

Δv: (-1,0.4), (-0.4,0.3), (0.4,0.3), (1,0.4) (13)

It is evident that this partitioning is rather small and results in only 20 rules (weights in FNN framework) if we use fully connected network (relating every label of an input to the others of other inputs) which we do so. During training, selected object moves across the field in a circular path with random speed and initial location. Whenever the robot reaches it, a one point ($r=1$) reward is given to it and if it cannot do so, it will be punished half a point ($r=-0.5$). Learning parameters were set as

$$\alpha = 0.4 \tag{14}$$

$$\gamma = 0.95 \tag{15}$$

$$\varepsilon = 0.1 \tag{16}$$

and λ was set different values in order to investigate the effect of λ in $Q(\lambda)$, more precisely we set it 0 (simple Q-Learning), 0.1, 0.4, and 0.8. Beside that, we decreased α whenever the agent got reward. In order to measure the effectiveness of our method, we run an evaluation test after every five randomly chosen learning situation (which results in a reward or punishment). The test is consist of running the robot in five predefined situations and counting how many times it can catch that object (similar to learning phase the objects moves in a circular path, but with prescribed speed). As learning is depend on randomly generated learning situations, its convergence to optimal solution is not the same in every run, so we run each test for 22 times and average over it. The comparison of results for $TD(\lambda)$ cases is depicted in Figure 2 and comparison of $TD(0.2)$ with simple Q-Learning ($TD(0)$) is presented in Figure 3. For small λs all of them are the same, although $\lambda = 0.2$ seems to be a little better. However, when λ is not small, the results degrades significantly. Comparing with simple Q-

Learning (although implemented in the same FNN framework), it is evident that these $Q(\lambda)$ extension results better which is intuitively correct, because by using $TD(\lambda)$, previous states-actions which leads into goal (or hell) can be rewarded or punished.

4 Conclusion

We have proposed a framework for RL. It is based on implementing $Q(\lambda)$ with FNN that enhances RL's generalization capability. Using $Q(\lambda)$, we can back-propagate rewards and punishments into previously seen states. This improves learning speed considerably, especially in tasks that exact scheduling and selection of actions in each state is important. FNN is used as function approximator with two main properties: it is local, so it does not extrapolate too much in places where there was no experience in it. Another important feature of FNN is its linguistically interpretability which enables the designer to put his/her knowledge into the structure (although it is not as apparent as simple Fuzzy rule-based system, but it is not difficult to learn FNN to work same).

RL seems sensitive to learning parameters and choosing best parameters in some situation is not an easy task. For instance, we observed that the way we decrease α is too important and increasing it too fast may lead to sticking in a suboptimal solution while not doing so, learning is too slow and the agent learns something and forgets it again repeatedly. So it seems plausible to use some adaptive schemes to change learning parameters. One nominee is Doya metalearning theory, which states that the brain has the capability of adjusting its learning parameters dynamically using neuromodulators (most notable are dopamine, serotonin, noradrenalin, and acetylcholine) [15]. Due to its biological plausibility it is a good candidate for RL. We will consider the effect of this mechanism in our future works with some more complicated problems.

5 References

[1] W. D. Smart and L. P. Kaelbling, "Effective Reinforcement Learning for Mobile Robots," Int. Conf. Robotics and Automation, May 11-15, 2002.

[2] Ch. Balkenius, "Natural Intelligence for Autonomous Agents," Lund University Cognitive Studies, 29, 1994.

[3] L. P. Kaelbling and M. L. Littman, "Reinforcement Learning: A Survey," J. Artificial Intelligence Research, vol. 4, 1996, pp. 237-285.

[4] C. J. C. H. Watkins and P. Dayan, "Q-Learning," Machine Learning, 8, 1992, pp. 279-292.

[5] R. S. Sutton, "Learning to Predict by The Method of Temporal Differences," Machine Learning, 3(1), 1988, pp. 9-44.

[6] J. Peng and R. J. Williams, "Incremental Multi-Step Q-Learning," In Proc. 11th Int. Conf. Machine Learning, 1994, pp. 226-232, Morgan Kaufmann.

[7] B. J. Park, W. Pedrycz, and S. K. Oh, "Fuzzy Polynomial Neural Networks: Hybrid Architectures of Fuzzy Modeling," IEEE Trans. Fuzzy Syst., vol. 10, no.5, 2002.

[8] A. Bonarini, "Delayed Reinforcement, Fuzzy Q-Learning and Fuzzy Logic Controllers," In Herrera, F., Verdegay, J. L. (Eds.) *Genetic Algorithms and Soft Computing*, (Studies in Fuzziness, 8), Physica-Verlag, Berlin, D, 1996, pp. 447-466.

[9] S. Thrun and A. Schwartz, "Issues in Using Function Approximation for Reinforcement Learning," In Proceedings of the 4th Connectionist Models Summer School, 1993.

[10] J. Boyan and A. W. Moore, "Generalization in Reinforcement Learning: Safely Approximating the Value Function," in G. Tesauro, D. S. Touretzky, and T. K. Leen, eds., Advances in Neural Information Processing System 7, MIT Press, Cambridge MA, 1995.

[11] W. D. Smart and L. P. Kaelbling, "Practical Reinforcement Learning in Continuous Spaces," In Proceedings of the 7th Int. Conf. on Machine Learning (ICML 2000), 2000, pp. 903-910.

[12] R. S. Sutton, and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.

[13] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike Elements That Can Solve Difficult Learning Control Problems," IEEE Trans. Syst., Man, and Cyber., vol. 13, 1983.

[14] S. Theodoridis and K. Koutroumbas, Pattern Recognition, Academic Press, 1999.

[15] K. Doya, "Metalearning, neuromodulation, and emotion," Hatano G., Okada N., Tanabe H., Affective Minds, Elsevier Science, 2000, pp. 101-104.