Computer Science • Vol. 7 • 2005

Grzegorz Głowaty*

ENHANCEMENTS OF FUZZY Q-LEARNING ALGORITHM

Fuzzy Q-Learning algorithm combines reinforcement learning techniques with fuzzy modelling. It provides a flexible solution for automatic discovery of rules for fuzzy systems in the process of reinforcement learning. In this paper we propose several enhancements to the original algorithm to make it more performant and more suitable for problems with continuous-input continuous-output space. Presented improvements involve generalization of the set of possible rule conclusions. The aim is not only to automatically discover an appropriate rule-conclusions assignment, but also to automatically define the actual conclusions set given the all possible rules conclusions. To improve algorithm performance when dealing with environments with inertness, a special rule selection policy is proposed.

Keywords: *fuzzy models, reinforcement learning, Q-Learning, automatic generation of fuzzy models*

ROZSZERZENIA ALGORYTMU FUZZY Q-LEARNING

Algorytm Fuzzy Q-Learning pozwala na automatyczny dobór reguł systemu rozmytego z użyciem technik uczenia ze wzmocnieniem. W niniejszym artykule zaproponowana została zmodyfikowana wersja oryginalnego algorytmu. Charakteryzuje się ona lepszą wydajnością działania w systemach z ciągłymi przestrzeniami wejść i wyjść. Algorytm rozszerzono o możliwość automatycznego tworzenia zbioru potencjalnych konkluzji reguł z podanego zbioru wszystkich możliwych konkluzji. Zaproponowano także nową procedurę wyboru reguł dla polepszenia prędkości działania w systemach z bezwładnością.

Słowa kluczowe: *modele rozmyte, uczenie ze wzmocnieniem, Q-Learning, automatyczne tworzenie modeli rozmytych*

1. Introduction

Automatic construction of fuzzy model is a challenging task. There are many known solutions of the problem of finding optimal fuzzy model describing a given set of data. Commonly used techniques are based on the usage of genetic algorithms [1], neural networks [4] and clustering methods [6]. The problem becomes different where no learning data is given. A variety of techniques known as reinforcement learning [11] address this kind of problems. In these kind of methods the system is provided with a critic signal r being reinforcement received after taking particular action.

*Department of Computer Science, AGH-UST, Kraków, Poland, glowaty@agh.edu.pl

Fuzzy Q-Learning algorithm proposed by Glorennec and Jouffe [2] uses reinforcement learning Q-Learning algorithm [11] to produce fuzzy model for maximization of received reinforcement signal values in a given environment. Original algorithm cannot fully depend on environment signal only. Some a priori knowledge about a model is needed. In order to run Fuzzy Q-Learning one must provide proposition of input space partitioning as well as propositions of fuzzy rules conclusions. In this paper we propose various enhancements for original Fuzzy Q-Learning algorithm to make it more performant and more suitable for problems with continuous-input continuous-output space. In the first section we present the original version of Fuzzy Q-Learning algorithm then, in the second section, we describe the problems we address in our improved version of the algorithm and our solution to these problems. In the final section we describe the results of comparison of our improved version and the original Fuzzy Q-Learning. The problem we choose as a benchmark problem is a standard Ball and Beam problem.

2. Fuzzy Q-Learning algorithm

The Fuzzy Q-Learning algorithm is the extension of Watkins' Q-Learning. The algorithm is designed to be able to learn simplified fuzzy Takagi-Sugeno model [9] in the process of reinforcement learning. The idea behind the extension of classic Q-Learning algorithm to its fuzzy version is to be able to perform reinforcement learning over a problem with continuous input/output space. As original reinforcement learning operates in the environment described by classic Markov decision process, its state space is discrete. There are other known extensions of reinforcement learning for finding solutions to problems with continuous state space [3].

A simplified Takagi-Sugeno model is a fuzzy inference system with rules with conclusions in a form of constant (it can be viewed as Mamdani [7] model with conclusions in a form of crisp singleton sets). Rule in the described inference system has the following form:

$$R_i\colon \text{IF } x_1 \text{ IS } X_{1i} \text{ AND } x_2 \text{ IS } X_{2i} \text{ AND} \ldots \text{AND } x_n \text{ IS } X_{ni} \text{ THEN } y = y_i \quad (1)$$

The evaluation of the rule depends on the chosen t-norm for representation of AND operator in the conditions part of the rule. It also depends on the choice of fuzzy implication operator representing THEN operator.

Given a set of rules containing K rules and a set of possible conclusions containing C different conclusions it is possible to build C^K different fuzzy models with rules in the form (1). Fuzzy Q-Learning algorithm's task is to find an appropriate assignment of a conclusion to each of the rules.

The evaluation of quality of each of possible conclusion to rule assignments is maintained in a special lookup vector q_i for a rule R_i such that $q_i(c_j)$ describes a quality of an assignment.

The actual Q value (by means of original Q-learning algorithm) of an inferred action a in the state x is defined as follows:

$$Q(x,a) = \frac{\sum_{i=1}^{K} \alpha_i(x) q_i(c_i^*)}{\sum_{i=1}^{K} \alpha_i(x)} \tag{2}$$

where c_i^* is a conclusion assigned to i-th rule in the evaluated time step and $\alpha_i(x)$ is an activation level of i-th rule in a state x.

Let r be a reinforcement signal received in a current learning step, x_t, x_{t+1} the current and the next (after execution of action a) state of the system, and γ the Q-Learning discount multiplier. The error of estimation for a Q function can be calculated by the following formula:

$$\Delta Q = r + \gamma Q(x_{t+1}, a^*) - Q(x_t, a) \tag{3}$$

where a^* is "the best" possible action to take in the state x_{t+1}. It is constituted of conclusions with the biggest q value for each of the rules.

A formula for updating q lookup vectors is obtained by gradient descent method and takes a form:

$$\Delta q_i(c_j) := \beta \Delta Q \frac{\alpha_i(x)}{\sum_{i=1}^{K} \alpha_i(x)} \tag{4}$$

where β is a learning rate.

Having in mind above formulas and assumptions original Fuzzy Q-Learning algorithm is described as follows:

Algorithm 1. The original Fuzzy Q-Learning algorithm:

1. Observe the current state x_t.
2. For each of the rules choose the consequence using some greedy or epsilon greedy policy with respect to current q values of conclusion for this rule.
3. Compute the global action a as a result of fuzzy inference and its corresponding Q value by using formula (2).
4. Execute action a, observe the new state x_{t+1} and reinforcement signal r.
5. Update q lookup tables using formulas (3) and (4).

3. Improvements to the Fuzzy Q-Learning algorithm

As it was said before the Fuzzy Q-Learning algorithm needs a certain amount of designer's a priori knowledge in order to discover proper fuzzy model for a given environment. This knowledge includes initial partitioning of input space into the fuzzy

sets as well as the definition of a set of possible output variables for the system to choose. At least a part of this initial knowledge can be left for the system for discovery. We address this issue in this paper presenting a method for eliminating initial output values definition and letting the system choose appropriate values.

The second issue we address in this paper is related to an adaptation of epsilon greedy action selection policy introduced in the original Fuzzy Q-Learning algorithm. Fuzzy Q-Learning algorithm is rather a straight-forward approach to adapt Q-Learning for the problems over continuous input/output spaces. Unfortunately, in such problems, especially when the system dynamics involves some kind of inertness the original adaptation of epsilon greedy policy is not enough. As an example let us consider the Ball and Beam problem (which we use as our benchmark problem). The state of the system is described by two variables – current ball position and speed. The output of the system is the desired angle of the beam. The task is to position the ball in the centre of the beam. The ball balancing should be performed in a way that prevents the ball from falling of the beam.

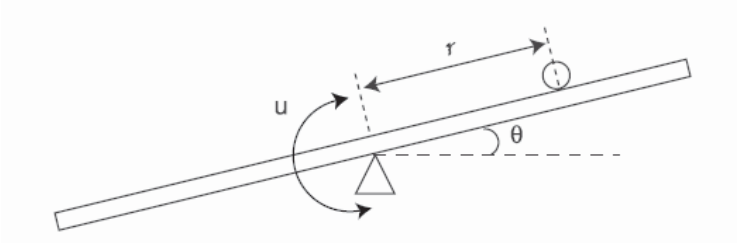

Fig. 1. Ball and Beam. The system measures current ball speed and its distance to the center of the beam r. The steering u is applied to modify the beam angle

If the angular speed of the beam is limited (which is always the case in the real system) under some conditions the desired angle of the beam cannot be realised within a short period of time (one simulation or decision step). To better illustrate the problem let us imagine the ball falling down to the left of the beam as illustrated in the Figure 1. If the random action chosen by epsilon greedy (or any other random) policy is supposed to set the angle of the beam to the opposite position it is impossible to realize the action in one simulation step. What's more, it is highly doubtable that the simple epsilon greedy policy will choose the next action to be similar to the previous one to give us some more time for positioning. Additionally because of inertness of the ball it will not move to the right immediately. It will continue moving to the left. As a result this random action will have almost no effect on the system and the used algorithm will become more like SARSA algorithm [11] than like Q-Learning. Exploration is an important part of Q-Learning and as we showed above the original Fuzzy Q-Learning algorithm may lack exploration under circumstances similar to those above. We propose a way to deal with this issue later on.

3.1. Continuous conclusion quality evaluation functions

Presented method for eliminating the need of a priori knowledge of the exact values for the rules conclusions is based on the following fact: instead of lookup tables used for storing the quality of conclusions the function approximator can be used. As the problem we deal with is a continuous one, we expect the result of the inference to be aggregated by some sort of "centre of gravity" aggregation method. Therefore we can let the conclusion values to be chosen from a continuous interval. Let

$$q_i : [c_{\min}, c_{\max}] \to \mathbb{R} \tag{5}$$

be a function describing a quality of a conclusion for a rule R_i, $c_{\min}$ and $c_{\max}$ be the boundaries of the possible conclusions interval. The task of the system will be not only to choose the appropriate conclusions from the specified set but also the definition of a set itself. In the original approach the set is given and this function is discrete with its values stored in the lookup table. In our approach the set will be defined as a set composed of n maximums of functions of type (5).

Any kind of function approximator can be used for q_i function representation as long as it is able to:

- Update function value in any given point of an interval by on-line adaptation. It is needed for calculation of formula (4) in the 5th step of the Fuzzy Q-Learning algorithm.
- Expose the maximum of an approximated function (as well as the argument for function maximum) effectively. This is needed in the calculation of formula (3) in the 5th step of the algorithm as a^* is constituted of actions with the highest quality function value.

As the first condition is held by almost every on-line function approximator, the second one is not a popular demand for a good function approximator. In our implementation we use CMAC [8, 11] as both of above conditions can be easily fulfilled with it. For easiness of computation of maximum we will use our mechanism of maximum caching for CMAC. Due to trivial construction of CMAC approximator it is easy to do so. Please notice that if neural network was chosen instead of CMAC the maximum computation wouldn't be that simple. But still it would be computable given a network structure.

One thing must be taken into the consideration when updating cached max value. In CMAC approximator, if a currently updated function value is bigger than the current cached maximum value the current value becomes a new maximum (as neighbouring values cannot become bigger). Naive approach would be to assume that this is the only case in which the maximum changes. Unfortunately it is not. The function is updated also when the current value is lower than the current maximum value. After update a check is needed to make sure that function still preserves its maximum value. If it doesn't, it means that current maximum was lowered and that there can be a new maximum elsewhere. In this case the cached value must be cleared

and calculated on demand on the next call to "get maximum" function. It can be easily proven that the average amortized time of this implementation for operations of function update and maximum retrieval is $O(1)$.

The idea behind the use of function approximator is not only to enhance abilities of performing without a priori knowledge about the conclusions. It is also supposed to speed up the learning process. As the approximated function is supposed to be continuous, the function approximator during value update changes not only the value for a given argument itself but also values from its neighbourhood.

As reinforcement learning is supposed to be used in on-line learning processes we need an efficient computation of maximum of function represented by CMAC approximator. It can be performed in constant time using maximum value caching described above.

3.2. Conclusion selection policy

Our proposition for dealing with problems of assuring proper exploration for the algorithm in the environments with inertness involved is to extend original epsilon greedy action selection policy to a different one. As we explained above, the inertness may cause insufficient exploration effect. We propose to extend the original policy used so that it preserves the chosen action as long as it is needed for action to have a desired effect on a system. Also due to the change of discrete set for possible actions it is harder to directly implement action selection policy proposed in [5].

Original policy balances between exploration and exploitation. The choice of action for a given action quality vector q is defined [5] as follows:

$$action(i) = \arg\max_{c \in C}(q_i(c) + \eta(c) + \rho_i(c)) \tag{6}$$

where $\eta(c)$ is defined by random variable with exponential distribution over all possible conclusions and $\rho_i(c)$ decreases exponentially with each use of conclusion c in rule i.

The policy prefers rarely chosen rules. Both $\rho_i(c)$ and $\eta(c)$ are scaled by appropriate factors [5] to balance between directed and undirected exploration [10]. If the action selection was based only on q function values it would be a greedy policy always selecting the best action with no exploration. Adding a random value $\eta(c)$ and selecting the action with the greatest sum introduces undirected exploration. To balance it with directed exploration in the areas which were not explored or explored rarely up to the current moment a $\rho_i(c)$ value is added. The nature of $\rho_i(c)$ function is that is bigger for conclusions selected rarely and smaller for conclusions selected often.

It is possible to implement the above policy using function approximators for both $\eta(c)$ and $\rho(c)$ functions in the continuous case. CMAC approximator initial values can be easily set to represent $\eta(c)$ with exponential probability distribution. Approximation of sum of functions represented by CMAC can be implemented by simply adding the values stored in CMAC tables. So the method mentioned before for calculation of maximum of CMAC represented function can be used to discover the action to choose using the policy above.

Whether, as it is shown above, it is possible to proceed with the original strategy we propose to exchange it with the special policy for tasks with inertness. Our rule selection algorithm is described below:

Algorithm 2. Conclusion selection policy for enhanced version of Fuzzy Q-Learning:

```
SELECT_ACTION(i) {
    IF T=0 {
        r:=rnd(0,1)
        IF r > ε THEN
                action(i):=arg max_c q_i(c)
            ELSE {
                T = rnd(0,1)l + L
                IF r ≤ bε THEN action(i):= arg max_c f_i(c)
                   ELSE action(i) = rnd(c_min, c_max)
            }
    } ELSE T:=T-1
    f_i(action(i)) := f_i((action(i)) + 1
    RETURN action(i)
}
```

T and *action* vector are local variables maintained by the policy object, so they are preserved between invocations of the method shown above. T represents inertness and is random number indicating a number of times ahead in which selected random action should be applied. Every time T is 0, a new decision is being made. With probability $1 - \epsilon$ greedy action with respect to the current policy is selected. When random action is to be selected it is selected to be executed for T times, where T varies from L to $L + l$. Random action can represent both directed and undirected exploration so parameter $b \in [0, 1]$ is introduced as the balance between directed and undirected exploration.

Exploration gets more undirected with the decrease of b parameter. For directed exploration a special set of frequency functions $f_i \colon C \to \mathbb{N}$ is introduced to maintain frequencies of appliance of actions. CMAC approximator is being used for representation of these functions.

Above algorithm is meant to be an implementation of step 2 in our modification of original Fuzzy Q-Learning algorithm (Algorithm 1).

3.3. Enhanced Fuzzy Q-Learning algorithm

As a result of above discussion we present enhanced version of Fuzzy Q-Learning algorithm.

Algorithm 3. The Enhanced Fuzzy Q-Learning algorithm:

1. `Observe the current state` x_t`.`
2. `For each of the rules choose the consequence using procedure specified by Algorithm 2.`

3. Compute the global action a as a result of fuzzy inference. Compute its corresponding Q value by using formula (2).
4. Execute action a, observe the new state x_{t+1} and reinforcement signal r.
5. Update approximation of q function using formulas (3) and 4. Please have in mind that as mentioned in sec. 3.1 selected function approximator must provide values needed by formulas (3) and 4.

Please have in mind that all references to q function in this algorithm refer to approximated value of this function by the means of function approximator used. This is one of the main differences from original approach, as there q function was stored in a lookup table indexed by pre-selected conclusion values.

4. Experiments and further work

Above improvements were applied to the Ball and Beam problem described in the previous section. For comparison original Fuzzy Q-Learning algorithm was used. The input space was partitioned with use of five fuzzy sets for each of the variables. These are BL (big left), L (left), Z (zero), R (right), BR (big right) for ball position and NB (negative big), N (negative), Z (zero), P (positive), PB (positive big) for ball speed. Ball position varies form -1 m to 1 m. Ball speed varies from -1 to 1 m/s. For the original Fuzzy Q-Learning algorithm 5 prototype conclusions were chosen. These are $C = \{-\frac{\pi}{4}, -\frac{\pi}{8}, 0, \frac{\pi}{8}, \frac{\pi}{4}\}$. For our enhanced version of the algorithm conclusions there is no need to specify exact conclusion values. These are to be chosen by the system from a given continuous interval $[-\frac{\pi}{4}, \frac{\pi}{4}]$.
Following rewards and punishments were chosen to represent the reinforcement:

- 1, when the ball is positioned on the centre of the beam, with tolerance of 2% (corresponding to beam length) and the speed of the ball is less than 0.03 m/s with ball angle set to no more than 0.03 rad;
- -1 when the ball hits the left or right side of the beam (ball doesn't fall off the beam, so the task can be continued);
- -1 when the ball stands still outside the centre of the beam for more than 3 simulation clock ticks (one clock tick is every 0.04 s), where standing still is defined as moving with a speed less than 0.03 m/s.

Beam angular speed has been limited to $\frac{\pi}{2}$ rad/s.

Calculations were performed using a simulation program written in Java. Learning parameters were set to the following values:

$$\epsilon = 0.01,\ \beta = 0.001,\ \gamma = 0.9,\ b = 0.2,\ L = 15,\ l = 15.$$

Both algorithms were implemented with use of eligibility traces [5, 11] with eligibility rate $\lambda = 0.9$ to speed-up the learning procedure. Traces were implemented by memorizing recent states visited during trial.

Every balancing trial was composed of at most 250 simulation steps (10 seconds). If there was no success after this period of time the task was interrupted. If there was reward of 1 received task was finished. At the beginning of each trial ball was positioned randomly on the beam and its speed was set randomly to any speed from $[-0.25\,\mathrm{m/s}, 0.25\,\mathrm{m/s}]$ interval.

Both algorithms performed well and with comparable results and learning time, as it is shown in Figure 2 and Figure 3. Enhanced version was able to learn proper rule base after about 150 trials. Original algorithm needed comparable time to extract the rule base.

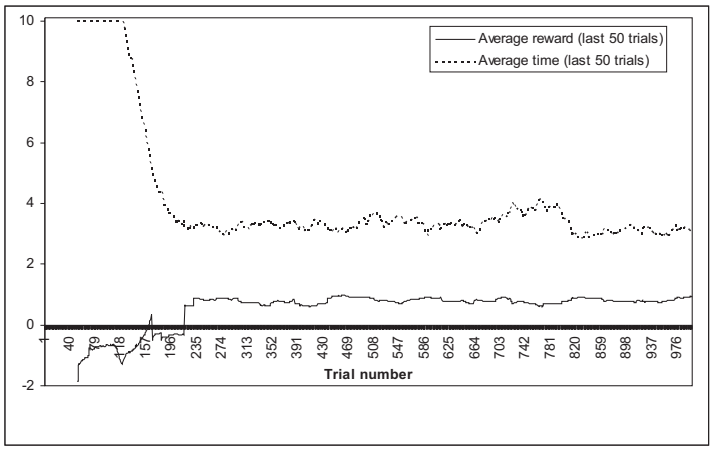

Fig. 2. Results of sample run with enhanced algorithm

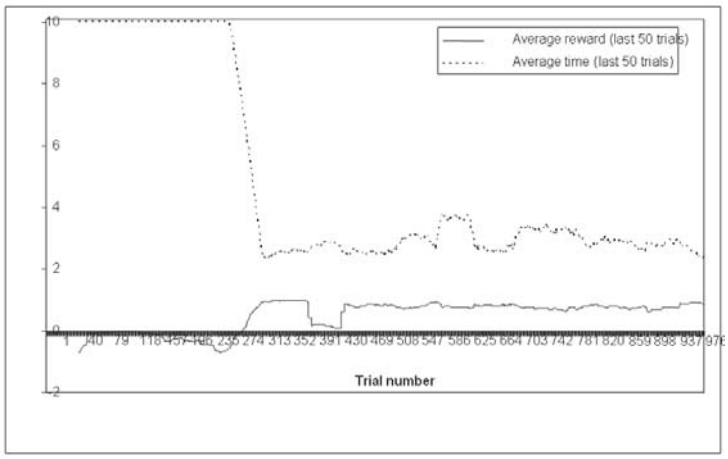

Fig. 3. Results of sample run with original algorithm

Please note, that even if the computation times were comparable, the task performed by the enhanced version was slightly different – not only appropriate conclusion rule assignment had to be done but also the definition of the conclusions themselves.

Rule bases discovered by both methods after 1000 iterations are presented in the Tables 1 and 2.

Table 1
Rule base of enhanced algorithm after 1000 trials

		Distance				
		NB	**N**	**Z**	**P**	**PB**
Speed	**NB**	−0.754	0.22	0.565	0.408	−0.126
	N	0.251	0.314	0.754	0.534	−0.251
	Z	0.408	0.754	0	−0.785	−0.723
	P	0.534	−0.471	−0.423	−0.314	−0.503
	PB	0.251	−0.754	−0.283	−0.66	−0.377

Table 2
Rule base of original algorithm after 1000 trials

		Distance				
		NB	**N**	**Z**	**P**	**PB**
Speed	**NB**	0.785	0.785	0.393	−0.785	−0.393
	N	0.393	0	−0.785	0.393	−0.393
	Z	0	0.393	0	0.393	0
	P	0.393	−0.393	0	−0.785	−0.393
	PB	−0.785	0	−0.785	−0.785	−0.393

Please note that rule bases did not change a lot in iterations 300–1000. The most significant changes to the rule bases were done in iterations 0–300.

As it was shown above enhanced version of Fuzzy Q-Learning algorithm is able to discover a set of desired conclusions for a given rule base. The learning speed of the enhanced version is not lower than the one of the original algorithm. Some further work concerning a method for automatic discovering of rules conditions may prove useful in minimizing the need for expert knowledge. Presented method can be easily enhanced to support any Takagi-Sugeno type conclusions, not only those in the simple form. Function approximators as CMAC can be used to approximate function of more than one variable, therefore we can make an assignment of linear (or any other type) function parameters to each of the rules. The learning process will remain unchanged.

References

[1] Bonarini A.: *Evolutionary Learning of Fuzzy rules: competition and cooperation.* Fuzzy Modelling: Paradigms and Practice, Kluwer Academic Press, Norwell, MA (1997), 265–284

[2] Glorennec P. Y., Jouffe L.: *Fuzzy Q-learning*. Proc. of the Sixth International Conference on Fuzzy Systems 1997

[3] Glorennec P. Y.: *Reinforcement Learning: an Overview*. ESIT 2000, Aachen, Germany 2000

[4] Jang J. R.: *ANFIS: Adaptive-Neural-Based Fuzzy Inference System*. IEEE Trans. on SMC, vol. 23 1993, 665–685

[5] Jouffe L.: *Fuzzy inference system learning by reinforcement methods*, IEEE Trans. Syst. Man. Cybernet. Part C: Appl. Rev., 28, 3, 1998

[6] Klawonn F., Kruse R.: *Constructing a fuzzy controller from data*. Fuzzy Sets and Systems, 85, 1997, 177–193

[7] Mandani E. H.: *Applications of fuzzy algorithms for simple dynamic plants*. Proc. IEE, 121, 1974, 1585–1588

[8] Sutton R. S.: *Generalization in reinforcement learning: Successful examples using sparse coarse coding*. Advances in Neural Information Processing Systems, vol. 8. MIT Press 1996

[9] Takagi H., Sugeno M.: *Fuzzy identification of systems and its applications to modeling and control*. IEEE Transactions on System, Man, and Cybernetics, 15(1) 1985, 116–132.

[10] Thrun S. B.: *Efficient exploration in reinforcement learning*, Technical Report CMU-CS-92-102, School of Computer Science, Carnegie Mellon University 1992

[11] Watkins C. J. C. H.: *Learning from Delayed Rewards*. King's College, Cambridge 1989 (PhD Thesis)