FUZZ-IEEE 2007

IEEE International Conference on Fuzzy Systems

Tutorial 3: Fuzzy Reinforcement Learning

IEEE · IEEE Computational Intelligence Society · BT

Fuzzy Reinforcement Learning

A Tutorial
Presented by
Dr. Hamid Berenji
FUZZ-IEEE07, London

Outline

1. Reinforcement Learning
2. Dynamic Programming
3. Monte Carlo Methods
4. Temporal Difference
5. Function Approximation
6. Fuzzy Reinforcement Learning
7. GARIC Architecture
8. FQ Learning
9. Co evolutionary Learning
10. Conclusion

Learning Methods

- Supervised Learning, Reinforcement Learning, Unsupervised Learning
- In supervised learning, a teacher provides the desired control objective at each time step
- In reinforcement learning, the teacher's response is not as direct, immediate, and informative as in supervised learning
- The presence of a supervisor to provide the correct response is not assumed in unsupervised learning

Reinforcement Learning

- What is it?
  - Learning by interaction with the environment
  - Is learning what to do
  - How to map situations to actions

Reinforcement Learning basics

- Has its roots in animal learning
- Draws upon many insights from the fields of control theory, operations research, neural networks, and artificial intelligence

Reinforcement Learning basics

- A policy is a decision making function which specifies what action to take in each situation
- A policy may be stochastic
- A reward function maps the state to a reward and the goal of the agent is to maximize this reward over the long run

Reinforcement Learning basics

- A value function determines the expected reward in the long run
- The value of a state is the sum of the rewards that it collects over long run or expects to accumulate in the future starting from that state
- A state may receive a low immediate reward but be of high value because it is often followed by states which receive high rewards

Reinforcement Learning Basics Boltzmann distribution

- Boltzmann distribution:

$$\pi_{t+1}(a_i) = \frac{e^{Q_t(a_i)/T}}{\sum_{i=1}^{m} e^{Q_t(a_i)/T}}$$

where $\pi_{t+1}(a_i)$ is the probability of selecting action a_i in the next time step,

- T is called a temperature parameter where the high values of T will make actions more equi-probable and low values will lead to a more selective policy,
- m is the number of actions available to the agent at time $t+1$

Reinforcement Learning Basics Reward functions

- Maximize the expected return.
- For processes which always end in a final time step such as in games, this reward will be

$$r_t = r_{t+1} + r_{t+2} + \cdots + r_T$$

where T is the final time step.

- For infinite-horizon problems, T =∞ and hence the expected reward can become infinite.

Reinforcement Learning Basics Reward functions (Cont..)

- This problem is solved in reinforcement learning by calculating a discounted reward

$$r_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{j=0}^{\infty} \gamma^j r_{t+j+1},$$

where γ is the discount rate and $0 \leq \gamma \leq 1$

- If $\gamma = 0$, then the agent is concerned with only maximizing the immediate rewards (r_{t+1}).
- As γ gets closer to 1, then the agent considers future rewards more strongly.

Reinforcement Learning Basics Action values

- The action value of taking action a in state s using policy π is defined

$$Q^{\pi}(s,a) = E_{\pi}\left\{ \sum_{j=0}^{\infty} \gamma^j r_{t+j+1} \middle| s_t = s, a_t = a \right\}$$

where Q^{π} is the action-value function for policy π .

Reinforcement Learning basics greedy methods

- Keep an estimate of values for different actions and always select the action with the highest action value
- A greedy method only exploits its environment and does not explore
- RL methods work best when one keeps a delicate balance between exploration and exploitation

Elements of Reinforcement Learning

- Policy: Way of behaving at a given time
- Reward function: defines the goal
- Value function: what is good in the long run.
- Model of environment: mimics the behavior of the environment.

Reward function vs. Value function

- We seek actions that bring about states of highest value, not highest reward
- Because these actions obtain the greatest amount of reward for us over the long run.

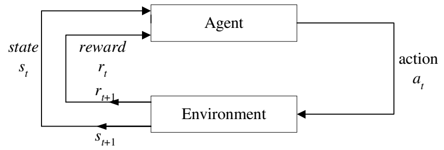
Exploration vs. Exploitation

- Exploitation: always select actions that result in highest state values
- Exploration: once in awhile, select non-max actions to allow exploring for higher values
- Softmax action selection

Solving full Reinforcement Learning

- Dynamic Programming
- Monte Carlo Method
- Temporal Difference Learning
- A Unified View

Dynamic Programming



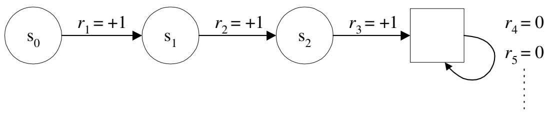
Agent-environment interaction

Reinforcement Learning Basics Bellman equations

- $$\begin{aligned} V^\pi &= E_\pi\left\{\sum_{j=0}^{\infty}\gamma^j r_{t+j+1}\right\} \\ &= E_\pi\left\{r_{t+1} + \gamma\sum_{j=0}^{\infty}\gamma^j r_{t+j+2}\right\} \\ &= \sum_a \pi(s,a)\sum_{s'} P_{ss'}^a\left[R_{ss'}^a + \gamma E_\pi\left\{\gamma^j r_{t+j+2}\right\}\right] \\ &= \sum_a \pi(s,a)\sum_{s'} P_{ss'}^a\left[R_{ss'}^a + \gamma V^\pi(s')\right], \end{aligned}$$

  where $P_{ss'}^a$ represents the probability of reaching state s' while taking action *a* in state *s* and $R_{ss'}^a$ is its associated return.
- Requires a complete environment model.

Finite State DP



Markov Property

$$P_r\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \cdots, r_1, s_0, a_0\}$$

$$P_r\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\}$$

Dynamic Programming

$$V^\pi(s) = E_\pi\{R_t \mid s_t = s\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

State-Value function for Policy π

$$V^*(s) = \max_a E\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\}$$

or

$$Q^*(s,a) = E\{r_{t+1} + \gamma \max Q^*(s_{t+1}, a') \mid s_t = s, a_t = a\}$$
$$= \sum_{s'} P_{ss'}^a \left[R_{ss'}^a + \gamma \max Q^*(s', a')\right]$$

Dynamic Programming

$$V^\pi(s) = E_\pi\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \mid s_t = s\}$$
$$= E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s\}$$
$$= \sum_a \pi(s,a) \sum_{s'} P_{ss'}^a \left[R_{ss'}^a + \gamma V^\pi(s')\right]$$

$$V_{k+1}(s) = E_\pi\{r_{t+1} + \gamma V_k(s_{t+1}) \mid s_t = s\}$$
$$= \sum_a \pi(s,a) \sum_{s'} P_{ss'}^a \left[R_{ss'}^a + \gamma V_k(s')\right]$$

Policy Evaluation

Input π, the policy to be evaluated

Initialize V(s) = 0, for all $s \in S^+$

Repeat

$\Delta \leftarrow 0$

for each $s \in S$: $v \in V(s)$

$V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} P_{ss'}^a \left[R_{ss'}^a + \gamma V(s')\right]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

output $V \approx V^*$

$$Q^\pi(s,a) = E_\pi\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\}$$
$$= \sum_{s'} P_{ss'}^a R_{ss'}^a \left[R_{ss'}^a + \gamma V^*(s')\right]$$

Policy Improvement

Initialize V arbitrarily, e.g., V(s) = 0, for all $s \in S^+$

Repeat

$\Delta \leftarrow 0$

for each $s \in S$:

$v \in V(s)$

$V(s) \leftarrow \max_a \sum_{s'} P_{ss'}^a \left[R_{ss'}^a + \gamma V(s')\right]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

output a deterministic policy π such that

$$\pi(s) = \arg\max_a \sum_{s'} P_{ss'}^a \left[R_{ss'}^a + \gamma V(s')\right]$$

Monte Carlo Methods

- Any estimation method whose operation involves a significant random component.
- Based on averaging complete returns
- Ideas carry over from DP, they both compute the same value functions

Reinforcement Learning Basics Monte Carlo method

- Estimate value functions $V^{\pi}(s)$ by maintaining an average for all the actual returns that have followed the state since the policy π.
- Similarly, maintain an average for all the occasions that action *a* has been tried when visiting state *s* and it will converge to the true action value $Q^{\pi}(s, a)$.
- Problem: not practical in large problems with many states and actions

First-Visit MC method for estimating V^{π}

Initialize
$\pi \leftarrow$ Policy to be evaluated
$V \leftarrow$ an arbitrary state-value function
Returns(s) $\leftarrow$ an empty list, for all $s \in S$
Repeat forever:
(a) Generate an episode using π
(b) For each state *s* appearing in the episode
    $R \leftarrow$ return following the first occurrence of s
    Append R to Returns(s)
    $V(s) \leftarrow$ Average (Returns(s))

Temporal Difference (TD) Learning

- Learns from experience without a need for a model.
- Similar to dynamic programming, TD methods update their estimates based on other learned estimates.
- Unlike Monte Carlo methods, TD methods do not have to wait until the end of a trial to update their estimates.
- TD methods learn by the following update
  $$\Delta V_t(s_t) = \alpha[r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)]$$
  where α is a step size parameter

Temporal Difference Learning and Sarsa

- In order to apply TD methods in control, one has to learn an action-value function $Q^{\pi}(s,a)$ instead of a state-value function $V^{\pi}(s)$.
- $\Delta Q_t(s_t, a_t) = \alpha[r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)]$
- Sarsa: quintuple (s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})for transition form one state-action pair to the next.
- Sarsa is an on-policy control algorithm which continually estimate Q^{π} for the behavior policy π.

Tabular TD(0) for estimating V*

Initialize $V(s)$ arbitrarily, π the policy to be evaluated
Repeat (for each episode):
Initialize s
Repeat (for each step of episode):
$a \leftarrow$ action given by π for s
$$V^*(s) = V(s) + \alpha[r + \gamma V(s') - V(s)]$$
Take action a; observe reward r, and next state s'
$s \leftarrow s'$
until s is terminal

TD(λ)

- $\Delta V(s) = a((r_t + \gamma V(s_{t+1})) - V(s_t))e(s)$
- $e(s)$ is the state's eligibility
- $e(s) = \lambda^k$ where k is the number of steps since s was visited

Q-Learning

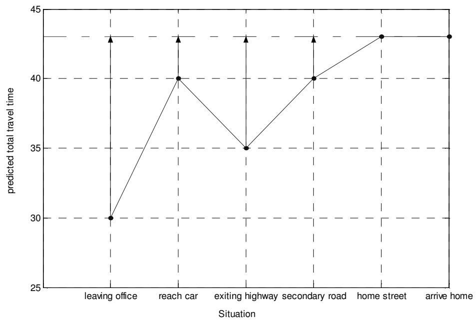
- Introduced by Watkins for Reinforcement Learning.
- Q-learning maintains an estimates $Q(x,a)$ of the values of taking action a in state x and continuing with the optimal policy after a new state is reached.
- The values of a state can be defined as the value of the state's best state-action pair:

$$V(x) = Max_a Q(x,a)$$

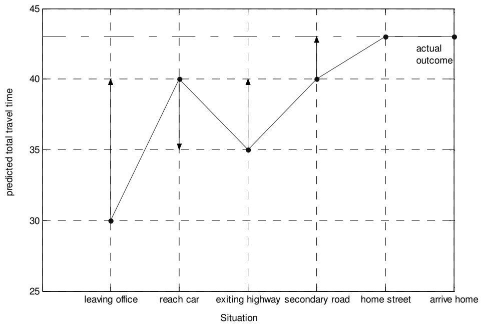
Driving home example

	Elapsed Time	Predicated Time to Go	Predicated Total Time
Leaving office, Friday at 6	0	30	30
Reach car, raining	5	35	40
Exiting highway	20	15	35
Secondary road, behind truck	30	10	40
Entering home street	40	3	43
Arrive home	43	0	43

Driving home example (Monte Carlo Changes)



Driving home example (TD Methods Changes)

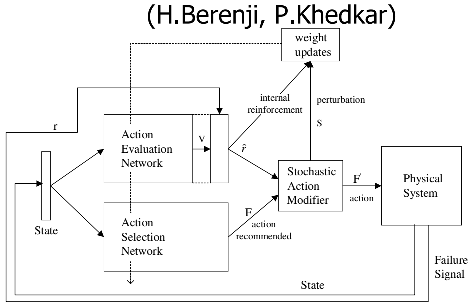


Generalization and Function Approximation

- Gradient Descent Methods
- Radial Basis Functions
- Coarse Coding and Tile Coding
- Linear Functions

Generalized Approximate Reasoning-based Intelligent Control (GARIC)

(H.Berenji, P.Khedkar)



Action Selection Network

- **Layer 1:** the input layer, consisting of the real-valued input variables.
- **Layer 2:** nodes represent possible values of linguistic variables in layer 1.
- **Layer 3:** conjunction of all the antecedent conditions in a rule using *softmin* operation.
- **Layer 4:** a node corresponds to a consequent label with an output
- **Layer 5:** nodes as output action variables where the inputs come form Layer 3 and Layer 4.

The Action Evaluation Network

- The AEN plays the roles of an adaptive critic element and constantly predicts reinforcements associated with different input states.
- The only information received by the AEN is the state of the physical system in terms of its state variables and whether or not a failure has occurred.
- The AEN is a standard tow-layer feedforward net with sigmoids everywhere except in the output layer.

The Action Evaluation Network (Cont..)

- The output unit of the evaluation network:

$$v[t,t+1]=\sum_{i=1}^{n} b_i[t]x_i[t+1]+\sum_{i=1}^{h} c_i[t]y_i[t+1]$$

where v is the prediction of reinforcement.

- Evaluation of the recommended action:

$$\hat{r}[t+1]=\begin{cases}0 & \text{start;}\\ r[t+1]-v[t,t] & \text{failure;}\\ t[t+1]+\gamma v[t,t+1]-v[t,t] & \text{else}\end{cases}$$

where 0≤ γ≤1 is the discount rate.

Learning in ASN

- We use the following learning rule

$$\Delta p=\eta\frac{\partial v}{\partial p}=\eta\frac{\partial v}{\partial F}\frac{\partial F}{\partial p}$$

- We assume that $\partial v/\partial F$ can be computed by the instantaneous difference ratio

$$\frac{\partial v}{\partial p}\approx\frac{dv}{dF}\approx\frac{v(t)-v(t-1)}{F(t)-F(t-1)}$$

Rule strength calculation using softmin operator

- Using the softmin, the strength of Rule 1 is:

$$w_1=\frac{\mu_{A_1}(x_0)e^{-k\mu_{A_1}(x_0)}+\mu_{B_1}(y_0)e^{-k\mu_{B_1}(y_0)}}{e^{-k\mu_{A_1}(x_0)}+e^{-k\mu_{B_1}(y_0)}}$$

- Similarly we can find w_2 for Rule 2.
- The control output of rule 1:

$$z_1=\mu_{C_1}^{-1}(w_1),$$

and for Rule 2:

$$z_2=\mu_{C_2}^{-1}(w_2),$$

- Using a weighted averaging approach, z_1 and z_2 are combined to produce the combined result z^*.

The Action Evaluation Network (Cont..)

- The input is the state of the plant, and the output is an evaluation of the state (a score), denoted by v.
- The v-value is suitably discounted and combined with the external failure signal to produce internal reinforcement $\hat{r}$.
- The output of the units in the hidden layer is:

$$y_i[t,t+1] = g(\sum_{j=1}^{n} a_{ij}[t]x_j[t+1])$$

where

$$g(s) = \frac{1}{1+e^{-s}}$$

and t and t+1 are successive time steps.

The Action Evaluation Network (Cont..)

- The output unit of the evaluation network:

$$v[t,t+1] = \sum_{i=1}^{n} b_i[t]x_i[t+1] + \sum_{i=1}^{h} c_i[t]y_i[t+1]$$

where v is the prediction of reinforcement.

- Evaluation of the recommended action:

$$\hat{r}[t+1] = \begin{cases} 0 & \text{start;} \\ r[t+1] - v[t,t] & \text{failure;} \\ r[t+1] + \gamma v[t,t+1] - v[t,t] & \text{else} \end{cases}$$

Rule strength calculation using softmin operator

- Using the softmin, the strength of Rule 1 is:

$$w_1 = \frac{\mu_{A_1}(x_0)e^{-k\mu_{A_1}(x_0)} + \mu_{B_1}(y_0)e^{-k\mu_{B_1}(y_0)}}{e^{-k\mu_{A_1}(x_0)} + e^{-k\mu_{B_1}(y_0)}}$$

- Similarly we can find w_2 for Rule 2.
- The control output of rule 1:

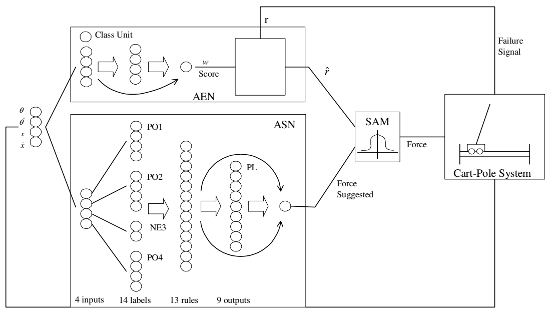
$$z_1 = \mu_{C_1}^{-1}(w_1),$$

and for Rule 2:

$$z_2 = \mu_{C_2}^{-1}(w_2),$$

- Using a weighted averaging approach, z_1 and z_2 are combined to produce the combined result z^*.

GARIC Applied to Cart-Pole Balancing



Fuzzy Q-Learning

- Introduced by Berenji in 1993 for Fuzzy Reinforcement Learning
- Fuzzy Q-Learning extends Watkin's Q-learning method for decision process in which the goals and/or the constraints, but not necessarily the system under control, are fuzzy in nature.
- An example of a fuzzy constraint is: "the weight of object *A* must not be substantially heavier than *w*" where *w* is a specified weight. Similarly, an example of a fuzzy goal is: "the robot must be in the vicinity of door *k*".

The GARIC-Q Architecture (Cont..)

- The FQ values are updated according to:

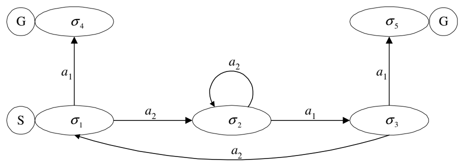
$$\Delta FQ = \beta(r + \gamma V(y) - V(x))$$

- $V(x)$ is the value of state x and action a_k selected through a Boltzman process.
- $V(y)$ is the value of the best state-agent pair defined by:

$$V(y) = Max_a FQ(y, a_k)$$

where k = 1 to K, is the agent number and a_k is its recommended final action.

Reinforcement Learning Basics Markov Decision Process (MDP)



- An example of a MDP with 5 states with two goals (i.e., terminal states) where two actions a_1 and a_2 are available at each non-terminal state.

FQ-Learning (Cont..)

- FQ is the *confluence* of the immediate reinforcements plus the discounted value of the next state and the constraints on performing action *a* in state *x*.

$$FQ(x,a) = E\{(r + \gamma V(y)) \wedge \mu_C(x,a)\}$$

- Update Rule:

$$\Delta FQ(x,a) \leftarrow \beta[(r + \gamma V(y)) \wedge \mu_C(x,a) - FQ(x,a)]$$

The GARIC-Q Architecture (Cont..)

- At each time step, using Fuzzy Q-Learning, GARIC-Q selects a winner among the GARIC agents and switches the control to that agent for that time step.
- The agent takes over and:
  - Calculates what action to apply using the current set of rules, within the selected agent, and their fuzzy labels.
  - Using SAM and $\hat{r}(t-1)$ calculates a new action F'

The GARIC-Q Architecture

- The GARIC-Q method presents an algorithm to model a society of rule bases (i.e., agents)
- Each agent operates internally with the methodology of GARIC and at the top level, using a modified Fuzzy Q-learning to select the best agent at each particular time step.

TD Method

- Real-time dynamic programming (Barto et al 1995)
- RTDP combines value function idea with simulation idea
- *TD*(1): Supervised training
- *TD*(0): Train for one-step
- *TD*(λ): Mixture

Q-Learning

- The development of Q-learning by Watkins is one of the most significant breakthroughs in reinforcement learning.
- Q-learning is an off-policy TD control algorithm and uses the following update rule:

$$\Delta Q_t(s_t, a_t) = \alpha\left[r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a_t) - Q_t(s_t, a_t)\right]$$

The GARIC Architecture

- The Action Selection Network maps a state vector into a recommended action F, using fuzzy inference.
- The Actor Evaluation Network maps a state vector and a failure signal into a scalar score which indicates sate goodness. This is also used to produce internal reinforcement $\hat{r}$.
- The Stochastic Action Modifier uses both *F* and $\hat{r}$ to produce an action *F'* which is applied to the plant.

Fuzzy Dynamic Programming

- Developed by Bellman and Zadeh, 1970
- Goals and Constraints can be fuzzy
- Provides a symmetrical view over gorals and constraints
- Decision:
  Confluence of goals and constraints

Fuzzy Q-Learning

- Introduced by Berenji in 1993 for Fuzzy Reinforcement Learning
- Fuzzy Q-Learning extends Watkin's Q-learning method for decision process in which the goals and/or the constraints, but not necessarily the system under control, are fuzzy in nature.
- An example of a fuzzy constraint is: "the weight of object *A* must not be substantially heavier than *w*" where *w* is a specified weight. Similarly, an example of a fuzzy goal is: "the robot must be in the vicinity of door *k*".

Fuzzy Q-Learning

- FQ-learning maintains an estimate *FQ*(*x*,*a*) of the value of taking action *a* in state *x* and continuing with the optimal policy after a new state is reached.
- The value of a state can be defined as the value of the state's best state-action pair:

$$V(x) = Max_a FQ(x,a)$$

FQ-Learning (Cont..)

- FQ is the *confluence* of the immediate reinforcements plus the discounted value of the next state and the constraints on performing action *a* in state *x*.

$$FQ(x,a) = E\{(r + \gamma V(y)) \wedge \mu_C(x,a)\}$$

- Update Rule:

$$\Delta FQ(x,a) \leftarrow \beta[(r + \gamma V(y)) \wedge \mu_C(x,a) - FQ(x,a)]$$

The FQ-Learning Algorithm

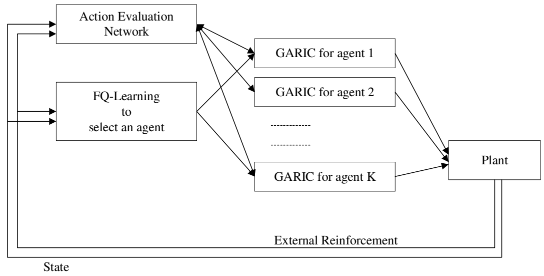
- Initialize FQ values
- Until FQ values converge do {
  1. x ← current state
  2. Select the action with the highest FQ. If multiple exist, select randomly among them.
  3. Apply action, observe the new state (y) and reward (r)
  4. Update

$$FQ(x,a) \leftarrow FQ(x,a) + \beta[(r + \gamma V(y)) \wedge \mu_C(x,a) - FQ(x,a)]$$

The GARIC-Q Architecture

- The GARIC-Q method presents an algorithm to model a society of rule bases (i.e., agents)
- Each agent operates internally with the methodology of GARIC and at the top level, using a modified Fuzzy Q-learning to select the best agent at each particular time step.

The Architecture of GARIC-Q



The GARIC-Q Architecture (Cont..)

- At each time step, using Fuzzy Q-Learning, GARIC-Q selects a winner among the GARIC agents and switches the control to that agent for that time step.
- The agent takes over and:
  - Calculates what action to apply using the current set of rules, within the selected agent, and their fuzzy labels.
  - Using SAM and $\hat{r}(t-1)$ calculates a new action F'

The GARIC-Q Architecture (Cont..)

- Calculates the internal reinforcement $\hat{r}(t)$
- Updates the weights of AEN
- Updates the parameters of the fuzzy labels in ASN
- Updates the *fq* values of all the rules used by the agent

The GARIC-Q Architecture (Cont..)

- An approach similar to Glorennec's method for selecting a rule base among the competing rule bases.
- Assuming that there are *K* agents and each agent *k* has R_k rules, then the total number of rules considered by the system is

  $R = \sum_{k=1}^{K} R_k$.
- R_{ij} refers to rule number *i* of agent *j*. Associated with each rule R_{ij} is a fq_{ij} which represnts the *fq* of rule R_{ij}.

The GARIC-Q Architecture (Cont..)

- The FQ value for an agent *k* is calculated from:

$$FQ_k = \frac{\sum_{i=1}^{R_k} fq_i * \alpha_i}{\sum_{i=1}^{R_k} \alpha_i}$$

The GARIC-Q Architecture (Cont..)

- The FQ values are updated according to:

$$\Delta FQ = \beta(r + \gamma V(y) - V(x))$$

- *V*(*x*) is the value of state *x* and action a_k selected through a Boltzman process.
- *V*(*y*) is the value of the best state-agent pair defined by:

$$V(y) = Max_a FQ(y, a_k)$$

  where *k* = 1 to *K*, is the agent number and a_k is its recommended final action.

The GARIC-Q Architecture (Cont..)

- The reinforcement r(t) can take:

$$r(t) = \begin{cases} +1 & \textit{if within the success region} \\ 0 & \textit{Viable zone} \\ -1 & \textit{Failure} \end{cases}$$

- Within each agent or rule base *k*, the reward or punishment is distributed based on the activity of rule *i*.

$$\rho_i = \frac{\alpha_i}{\sum_{i=1}^{R_k} \alpha_i}$$

  where α_i is the strength of rule *i*.

The GARIC-Q Architecture (Cont..)

- The *fq* values are updated for the selected agent *j* using:

$$\Delta fq_i = \lambda * \rho_i * \Delta FQ$$

- Upon each success or failure the state of the system is returned to an initial state (can be a random state) in the viable zone and learning restarts.
- Agents compete until the whole process converges to a unique agent or a combination of different agents have been able to control the process for an extended time.

Experiments

θ	$\dot{\theta}$	x	$\dot{x}$	F
PO1	PO2	null	null	PL
PO1	ZE2	null	null	PM
ZE1	NE2	null	null	ZE
ZE1	PO2	null	null	PS
ZE1	ZE2	null	null	ZE
NE1	NE2	null	null	NS
NE1	PO2	null	null	ZE
NE1	ZE2	null	null	NM
NE1	NE2	null	null	NL
VS1	VS2	PO3	PO4	PS
VS1	VS2	PO3	PS4	PVS
VS1	VS2	NE3	NE4	NS
VS1	VS2	NE3	NS4	NVS

The 13 rules used by each agent with 7 labels for force

θ	$\dot{\theta}$	x	$\dot{x}$	F
PO1	PO2	null	null	PL
PO1	ZE2	null	null	PL
PO1	NE2	null	null	ZE
ZE1	PO2	null	null	PS
ZE1	ZE2	null	null	ZE
ZE1	NE2	null	null	NS
NE1	PO2	null	null	ZE
NE1	ZE2	null	null	NL
NE1	NE2	null	null	NL
VS1	VS2	PO3	PO4	PS
VS1	VS2	PO3	PS4	PVS
VS1	VS2	NE3	NE4	NS
VS1	VS2	NE3	NS4	NVS

Conclusion

- GARIC-Q improves the speed of GARIC
- More importantly, GARIC-Q provided the facility to design and test different types of agents.
- These agents may have different number of rules, use different learning strategies on the local level, and have different architectures.

Conclusion (Cont..)

- GARIC-Q provided the first step toward a true intelligent system where at the lower level, agents can explore the environment and learn from their experience, while at the top level, a super agent can monitor the performance and learn how to select the best agent for each step of the process.

MULTI-GARIC-Q

- MULTI-GARIC-Q extends the GARIC-Q.
- The evaluator or AEN to learn not only based on the trials of the winning agent but also learn based on all the hypothetical experiences gained by the non-winning agents.
- The AEN in this model acts like a classroom teacher that learns by observing what each individual student is doing but only listens to the best student who has won the competition at that cycle.

IIS CORP Intelligent Inference Systems Corp.

NASA

USING FUZZY REINFORCEMENT LEARNING FOR POWER CONTROL IN WIRELESS TRANSMITTERS

David Vengerov
Hamid Berenji

State Generalization

- In large state spaces, most states will be visited only once
- Need to generalize learning experience across similar states
- Function approximation for generalizing state values

Limitations of Q-learning With State Generalization

- Q-learning can diverge even for linear approximation architectures
- Requires solving a nonlinear programming problem at each time step when action space is continuous

Actor-Critic Algorithms

- Actor-critic (AC) algorithms can be used in continuous action spaces because actor can be parameterized
- Tsitsiklis and Konda (1999) presented a practical convergent AC algorithm
- Actor is a parameterized function that has to satisfy certain conditions

Actor-Critic Fuzzy Reinforcement Learning (ACFRL) algorithm

- Actor is represented by a fuzzy rulebase
- Convergence proven in Fuzz-IEEE 2000

Power Control for Wireless Transmitters

- Transmitter -- finite-buffer FIFO queue
- The transmission probability is a function increasing with power p_t and decreasing with channel interference i_t: $\text{Prob}(\text{success} \mid p_t, i_t) = 1 - e^{-\frac{p_t}{i_t}}$
- The transmission cost at time t is a function of transmitter's backlog b_t and the power used p_t: $C_t = \alpha\, p_t + b_t$.
- When a packet arrives to a full buffer, an overflow cost L is incurred.

Power Control for wireless transmitters

- Agent observes current interference i_t and backlog b_t and chooses a power level p_t
- Objective: minimize the average cost per time step.

Tradeoff to be learned

- Higher power incurs a higher immediate cost but also increases the probability of a successful transmission thereby reducing the future backlog.

Agent Structure

- An agent is a fuzzy rulebase, which specifies transmission power as a function of backlog(b) and interference(i):
- If (b is SMALL) and (i is SMALL) then (power is p1)
- If (b is SMALL) and (i is MEDIUM) then (power is p2)
- If (b is SMALL) and (i is LARGE) then (power is p3)
- If (b is LARGE) and (i is SMALL) then (power is p4)
- If (b is LARGE) and (i is MEDIUM) then (power is p5)
- If (b is LARGE) and (i is LARGE) then (power is p6)

Motivation for the rulebase structure

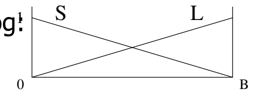
Bambos and Kandukuri (INFOCOM 2000) analytically derived a special-case power control policy:

Hump-shaped interference response resulting in a backoff behavior

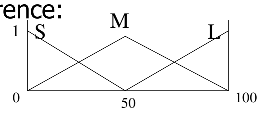
The size of the hump grows with backlog

Fuzzy Labels

- Fuzzy input labels:
  - backlog:



  - interference:



Simulation Procedure

Determine optimal constant power p*

Initialize p1,…,p6 to p*

Let ACFRL tune p1,…p6

Problem Parameters

Problem setup of Bambos and Kandukuri:

Poisson arrivals, uniform i.i.d. interference, finite buffer

Simulated arrival rates 0.1 through 0.6, corresponding to low and high stress levels on the transmitter

Results

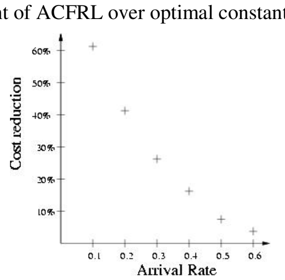
ACFRL learns a hump-shaped interference response

The size of the hump grows with backlog

Corresponds to a special-case analytical study by Bambos and Kandukuri

Results

Cost improvement of ACFRL over optimal constant power policy:



For high arrival rates there is less freedom of buffering the arriving packets and waiting for better future channel conditions

Conclusions

- Demonstrated how ACFRL can be applied to a challenging delayed reward problem
- ACFRL converges to a policy that significantly improves upon optimal constant policy
- ACFRL learns the same function of the input variables as predicted by analytical investigations for a special case

Co-evolutionary Perception-based Reinforcement Learning for Sensor Allocation in Autonomous Vehicles

Hamid Berenji, David Vengerov, Jayesh Ametha
IIS Corp

Fuzz-IEEE, St. Louis
May 26, 2003

Distributed Sensor Allocation in Teams of Automated Vehicles

- "Curse of dimensionality" problem
- At the team level, treat each AV as a composite sensor
- Distribute AVs to different regions of search space
- An AV Must be aware of other nearby AVs (e.g., not to track the same targets)

Perception-based Reinforcement Learning (PRL)

- Uses Perception-based Rules for Generalizing decision strategy across similar states
- Uses Reinforcement Learning for adapting these rules to the uncertain, dynamic environment

Co-evolutionary PRL for Sensor Allocation in AVs

- AVs must learn two complementary policies:
  - How to allocate their individual sensors
  - How to distribute themselves as a team in space to match the density and importance of targets
- Learn policies separately but with a common reward function => co-evolution toward the common objective

Reinforcement Learning (RL)

Objective: $\max_{a_t,\, t=0,1,\ldots} E\left[\sum_{t=0}^{\infty}\gamma^t r_t(s_t,a_t)\right]$

Subject to the constraint on the evolution of sequence of states:

$$s_{t+1} = f(s_t, a_t).$$

Q-value: $Q(s,a) = E\{\sum_{t=0}^{\infty}\gamma^t r_t(s_t,a_t) \mid s_0 = s, a_0 = a\}$,

expected long-term benefit of taking action *a* in state *s* and following the optimal policy thereafter.

Then, the optimal action in state *s* is $a^*(s) = \arg\max_a Q(s,a)$

Example of RL: Q-learning

Q-value satisfies Bellman's equation: $Q(s_t,a) = E\{r_t + \gamma \max_a Q(s_{t+1},a)\}$

Idea of Q-learning: compute a noisy sample of Bellman's error:

$$\delta_t = E\{r_t + \gamma \max_a Q(s_{t+1},a)\} - Q(s_t,a)$$
$$= r_t + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)$$

Stochastic update in small discrete state-action spaces: $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha_t\delta_t$

In large or continuous state-action spaces: $\theta_t \leftarrow \theta_t + \alpha_t\delta_t\nabla_\theta Q(s_t,a_t,\theta_t)$

Computational Theory of Perceptions

- Based on Computing with Words
- Granulation based on *perceptions* plays a critical role
- Combining rules with different θ^i, recommendation of a Q-value
- Weighted by $w^i(s,a)$, normalized applicability of each rule

$$Q(s,a,\boldsymbol{\theta}) = \sum_{i=1}^{M} \theta^i w^i(s,a)$$

Perception-based Q-Learning

Given $Q(s,a,\boldsymbol{\theta}) = \sum_{i=1}^{M} \theta^i w^i(s,a)$,

$\nabla_{\boldsymbol{\theta}} Q(s_t, a_t, \boldsymbol{\theta}_t)$ becomes $(w^1(s_t,a_t),\ldots,w^M(s_t,a_t))^T$

Continuous update equation $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_t + \alpha_t \delta_t \nabla_{\boldsymbol{\theta}} Q(s_t, a_t, \boldsymbol{\theta}_t)$

for perception-based rules becomes component-wise

$$\theta^i \leftarrow \theta^i + \alpha_t \delta_t w^i(s_t, a_t), i = 1,\ldots,M$$

TD(λ) updates rules according to how much they have contributed to decision-making in the past, discounting by $\gamma\lambda$:

$$\theta^i \leftarrow \theta^i + \alpha_t \delta_t \sum_{\tau=0}^{t} (\gamma\lambda)^{t-\tau} w^i(s_\tau, a_\tau)$$

AV Reward Functions

Reward received by AV k for tracking all targets within its sensor range after aligning itself with target j:

$$r_{kj} = \sum_{n=1}^{N} \left(\frac{V_n}{1+d_{kn}^2} \right) \left(\frac{\frac{1}{1+d_{kn}^2}}{\sum_{m=1}^{M} \frac{1}{1+d_{mn}^2}} \right)$$

State variables

Evaluating a target for individual sensor allocation:

Sum of potentials for all targets that an AV expects to track after aligning itself with target j:

$$x[1] = \sum_{n=1}^{N} \left(\frac{V_n}{1+d_{kn}^2} \right)$$

Sum of potentials of all other UAVs near target j:

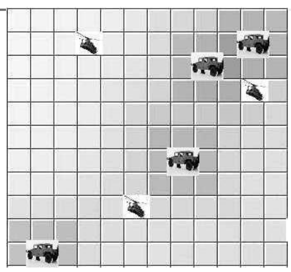
$$x[2] = \sum_{m=1}^{P} \left(\frac{1}{1+d_{jm}^2} \right)$$

Choosing direction of motion for allocating AVs in a search space:

y[1]="target potential"

y[2]="AV potential"

Potential Surface of AVs



Darker locations have higher target potentials

Rules for sensors alignment

- If (s_1 is SMALL) and (s_2 is SMALL) then θ^1
- If (s_1 is SMALL) and (s_2 is LARGE) then θ^2
- If (s_1 is LARGE) and (s_2 is SMALL) then θ^3
- If (s_1 is LARGE) and (s_2 is LARGE) then θ^4

Experiments

- 3 AVs to track 6 targets
- Use Player-Stage to simulate
- 2D square-shaped environment of length 2
- Size of AV and targets is .05 and .025

Sensors on each AV

- Sony EVID30 pan-tilt-zoom camera set to a range of 60 degrees
- SICK LMS-200 laser rangefinder for measuring distance
- GPS device for exact location position

Experimental Results

Measuring average team performance for different values of the TD parameter λ:

	$\lambda = 0$	$\lambda = 0.5$	$\lambda = 0.9$
Before Learning	1.1	1.1	1.1
After Learning	2.55	2.52	2.25

Decrease in performance for higher λ => decreased importance of past actions due to co-evolution with the second policy

Conclusions

- Co-evolutionary Perception based Reinforcement Learning algorithm performs well and it is feasible for AVs
- Joint optimization of individual sensor allocation policy and the team motion policy
- The methodology can be used in other domains such as robotic swarms

IIS CORP — Intelligent Inference Systems Corp. NASA

Adaptive Coordination Among Fuzzy Reinforcement Learning Agents

David Vengerov
Hamid Berenji
Alexander Vengerov

Task distribution in multi-agent systems

- Traditional task distribution in multi-agent systems:
  - Centralized allocation
  - Allocation by auction (directly or through brokers)
  - Allocation by acquaintances
- Works well in static, known environments

Emergent allocation methods

- Interested in dynamic, a priori unknown environments
- Emergent allocation methods: signal-based rather than message-based.
- Agents learn the value of signals in the context of their local environments

Q-learning

- $Q(s,a)$ is the expected reward in state s after taking action a and following the optimal policy thereafter:

$$Q(s,a) = E\{R_t \mid s_t = s, a_t = a\}$$
$$= E\{\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a\}$$

- r_t : the reward received after taking action a in state s_t,
- γ: is the discounting factor.

Q-learning

- In discrete state and action spaces:

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha_t (r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a)),$$

- α_t: is the learning rate at time t.
- Converges to optimal Q-values (Watkins, 1989) if each action is tried in each state infinitely many times,

$$\sum_{t=0}^{\infty} \alpha_t = \infty, \quad \sum_{t=0}^{\infty} \alpha_t < \infty.$$

State Generalization

- In large state spaces, most states will be visited only once
- Need to generalize learning experience across similar states
- Function approximation for generalizing state values

Q-learning with state generalization

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_t - \frac{1}{2} \alpha_t \nabla_{\boldsymbol{\theta}_t} [r_{t+1} + \gamma \max_a Q(s_{t+1}, a, \boldsymbol{\theta}_t) - Q(s_t, a_t, \boldsymbol{\theta}_t)]^2.$$

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_t + \alpha_t \nabla_{\boldsymbol{\theta}_t} Q(s_t, a_t, \boldsymbol{\theta}_t)[r_{t+1} + \gamma \max_a Q(s_{t+1}, a, \boldsymbol{\theta}_t) - Q(s_t, a_t, \boldsymbol{\theta}_t)].$$

- $Q(s,a,\boldsymbol{\theta})$ approximates $Q(s,a)$
- $\boldsymbol{\theta}$ is the set of all parameters arranged in a single vector.

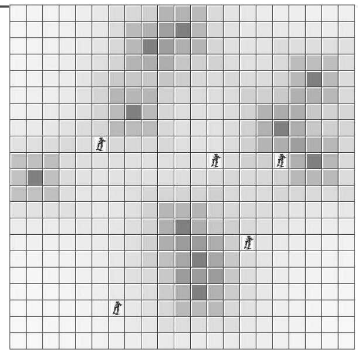
Distributed Dynamic Web Caching

- Servers distributed throughout the Internet
- Replicate content for faster access
- Main focus so far: directing requests to the "best" server
- Important issue: dynamically moving relevant content to servers located in "hot spots"

Agent-based View

- Agents represent content blocks
- Need to allocate themselves in proportion to the demand in each area
- Natural tradeoff for an agent:
  - moving to the highest demand area
  - ensuring adequate coverage of the whole area by the team

Tileworld Simulation



Tileworld Description

- Demand sources appear and disappear randomly
- Location-based similarity of interests
- Potential field model: demand source *i* contributes demand potential to location *j*:

$$P_{ij} = \frac{V_j}{1 + d_{ij}^2}$$

- Total potential at each location: $P_j = \sum_i P_{ij}$

Tileworld Description

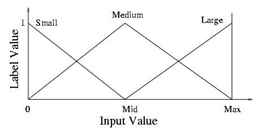
- Agent at location *i* extracts reward from source *j* equal to P_{ij}
- The value of each demand source decreases at each time step by the total reward extracted by all agents from this source
- Agent's goal: maximize average reward per time step

Agent Coordination

- Information about the team is presented to each agent in the form of "agent potential"
- Just like demand potential with agents being the sources

Decision Making

- Agents evaluate 8 adjacent locations
- Sample rule k: IF (demand potential at L_i is LARGE) and (agent potential at L_i is SMALL) then (Q-value of moving to L_i is $Q^i{}_k$)



- Final value of moving to location L_i:

$$Q^i = \sum_k \mu_k^i Q_k^i$$

Experimental Setup

- 20-by-20 tileworld with 10 demand sources and 5 agents
- Agents are trained using fuzzy Q-learning for 1000 time steps and then tested for 100 time steps
- Sensory radius: 5 units of distance or unlimited

Results

- Agents learn rules that prefer higher demand potential and smaller agent potential
- Coordinating agents perform 50-100% better than random agents
- Independent agents perform *worse* than random agents because they crowd together

Conclusions

- Fuzzy rulebased agents can learn successfully in continuous state spaces
- A new method for adaptive coordination among fuzzy reinforcement learning agents
- Agents learn an efficient group behavior in a dynamic resource allocation problem

References

- David Vengerov, Nicholas Bombos, Hamid Berenji, Reinforcement Learning Approach to Power Control in Wireless Transmitters, IEEE Transactions on Systems, Man, and Cybernetics, August 2005.
- Richard Sutton, Andrew Barto, Reinforcement Learning, An Introduction, MIT Press, 1988.
- Hamid Berenji, Pratap Khedkar, Generalized Approximate Reasoning based Intelligent Control, IEEE Transactons on Neural Networks, August 1992.
- Hamid Berenji, David Vengerov, On Convergence of Fuzzy Reinforcement Learning, IEEE Fuzzy systems, 2001.

References (Cont…)

- Hamid Berenji, David Vengerov, A Convergent actor critic based fuzzy reinforcement learning with application to power management of wireless transmitters, IEEE Transactionof Fuzzy Aystems, vpl. 11, no.4, 478-485, August 2003.
- Hamid Berenji, David Vengerov, Cooperation and Coordination between Fuzzy Reinforcement Learning Agents in Continuous State Partially Observable Markov Decision Processes,IEEE Conference on Fuzzy Systems, 2002.