FUZZ-IEEE 2007

IEEE International Conference on
Fuzzy Systems

Imperial College, London, UK
23-26 July, 2007
:: Intelligence is Fuzzy ::

Tutorial 3: Fuzzy Reinforcement Learning
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:
  \[ \pi_{t+1}(a) = \frac{e^{\theta(s,a)T}}{\sum_{a'} e^{\theta(s,a')T}} \]
  where \( \pi_{t+1}(a) \) is the probability of selecting action \( a \) 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, \( T = \infty \) 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:
  \[ r = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{s=0}^{\infty} \gamma^s r_{t+s} \]
  where \( \gamma \) 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 \( \gamma \) 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 \( \pi \) is defined:
  \[ Q^\pi(s,a) = E_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s, a_0 = a \right] \]
  where \( Q^\pi \) is the action-value function for policy \( \pi \).

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

\[ V^* = E_s \sum_{t=0}^{\infty} r_{t+1} \]
\[ = E_s \left( r_s + \sum_{t=0}^{\infty} r_{t+1} \right) \]
\[ = \sum_s p(s) \sum_a p_a(s) \left[ r_s + \gamma E_{s'} \left( V^* \right) \right] \]
\[ = \sum_s p(s) \sum_a p_a(s) \left[ r_s + \gamma V^*(s') \right] \]

where \( p_a(s) \) represents the probability of reaching state \( s' \) while taking action \( a \) in state \( s \) and \( r_s \) is its associated return.

- Requires a complete environment model.
Finite State DP

Markov Property

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

\[ P_r\{s_{t+1} = s', r_{t+1} = r|s_t, a_t\} \]

Dynamic Programming

State-Value function for Policy \( \pi \)

\[ V^\pi(s) = E_x[R_t|s_t = s] = E_x\{\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_t = s\} \]

\[ V^\pi(s) = \max_a E_r[\gamma V^\pi(s_{t+1}) | s_t = s, a_t = a] \]

or

\[ Q^\pi(s,a) = E_r[\gamma \max_a Q^\pi(s_{t+1}, a') | s_t = s, a_t = a] \]

\[ = \sum_a P^\pi_{sa} [R^a_t + \gamma \max_a Q^\pi(s', a') ] \]

Dynamic Programming

\[ V^\pi(s) = E_x\{V_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s\} \]

\[ = \sum_a \pi(s,a) \sum_s P^\pi_{sa} [R^a_t + V^\pi(s')] \]

Policy Evaluation

Input \( \pi \), the policy to be evaluated

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

Repeat

\[ \Delta \leftarrow 0 \]

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

\[ V(s) \leftarrow \sum_v \pi(s,a) \sum_{s'} P^\pi_{sa} [R^a_t + V^\pi(s')] \]

\[ \Delta \leftarrow \max(\Delta, |V(s)|) \]

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

output \( V = V^\pi \)

\[ Q^\pi(s,a) = E_r[\gamma \max_a Q^\pi(s_{t+1}) | s_t = s, a_t = a] \]

\[ = \sum_a P^\pi_{sa} [R^a_t + \gamma V^\pi(s')] \]

Policy Improvement

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

Repeat

\[ \Delta \leftarrow 0 \]

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

\[ V(s) \leftarrow \max_a \sum_r P^a_r [R^r + V^\pi(s')] \]

\[ \Delta \leftarrow \max(\Delta, |V(s)|) \]

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

output a deterministic policy \( \pi \) such that

\[ \pi(s) = \arg \max_a \sum_r P^a_r [R^r + V^\pi(s')] \]
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

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

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
  $$\Delta V(s) = \alpha \left[ r + V(s') - V(s) \right]$$
  where $\alpha$ 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 $V(s)$.
  $$\Delta Q(s, a) = \alpha \left[ r + Q^\pi(s', a) - Q(s, a) \right]$$
- Sarsa: quintuple $(s, a, r, s', a')$ for transition form one state-action pair to the next.
- Sarsa is an on-policy control algorithm which continually estimate $Q$ for the behavior policy $\pi$.

Reinforcement Learning Basics
Monte Carlo method
- Estimate value functions $V(s)$ by maintaining an average for all the actual returns that have followed the state since the policy $\pi$.
- 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 $Q(s, a)$.
- Problem: not practical in large problems with many states and actions

Tabular TD(0) for estimating $V^\pi$
- Initialize $V(s)$ arbitrarily, $\pi$ the policy to be evaluated
- Repeat (for each episode):
  - Initialize $s$
  - Repeat (for each step of episode):
    - $a \leftarrow$ action given by $\pi$ for $s$
    - $V(s) = V(s) + \alpha \left[ r + V(s') - V(s) \right]$
    - Take action $a$, observe reward $r$, and next state $s'$
  - $S \leftarrow S'$
  - until $s$ is terminal
### TD(λ)
- \( \Delta V(s) = \alpha((r + \gamma V'(s',s)) - V(s))\epsilon(s) \)
- \( \epsilon(s) \) is the state's eligibility
- \( \epsilon(s) = \lambda^k \) where \( k \) is the number of steps since \( s \) was visited

### Q-Learning
- Introduced by Watkins for Reinforcement Learning.
- Q-learning maintains an estimate \( Q(x,a) \) 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
<table>
<thead>
<tr>
<th></th>
<th>Elapsed Time</th>
<th>Predicted Time to Go</th>
<th>Predicted Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leaving office,</td>
<td>0</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Reach car, raining</td>
<td>5</td>
<td>35</td>
<td>40</td>
</tr>
<tr>
<td>Exiting highway</td>
<td>20</td>
<td>15</td>
<td>35</td>
</tr>
<tr>
<td>Secondary road,</td>
<td>30</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>behind truck</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Entering home street</td>
<td>40</td>
<td>3</td>
<td>43</td>
</tr>
<tr>
<td>Arrive home</td>
<td>43</td>
<td>0</td>
<td>43</td>
</tr>
</tbody>
</table>

### 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
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 two-layer feedforward net with sigmoid everywhere except in the output layer.

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 \( \frac{\partial v}{\partial F} \) can be computed by the instantaneous difference ratio
  \[ \frac{\partial v}{\partial F} = \frac{v(t) - v(t-1)}{F(t) - F(t-1)} \]

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 (Cont.)

- The output unit of the evaluation network:
  \[ v[t+1] = \sum_{i} h_i(x, t) \hat{x}_i + \sum_{j} c_j(x, t) \]
  where \( v \) is the prediction of reinforcement.
- Evaluation of the recommended action:
  \[ \hat{r}[t+1] = \begin{cases} 0 & \text{start;} \\ v[t+1] - v[t, t] & \text{failure;} \\ v[t+1] + \gamma (v[t+1] - v[t, t]) & \text{else} \end{cases} \]
  where \( 0 < \gamma \leq 1 \) is the discount rate.

Rule strength calculation using softmin operator

- Using the softmin, the strength of Rule 1 is:
  \[ w_i = \frac{\mu_k(x_i) e^{-\alpha x_i} + \mu_j(x_i) e^{-\beta x_i}}{e^{-\alpha x_i} + e^{-\beta x_i}} \]
  Similarly we can find \( w_i \) for Rule 2.
- The control output of rule 1:
  \[ z_1 = \mu_i(x_i), \]
  and for Rule 2:
  \[ z_2 = \mu_i^{-1}(w_i), \]
- 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 $\tilde{f}$.
- The output of the units in the hidden layer is:
  $$y_{[t,t+1]} = g\left(\sum a_t[t]x_t[t+1]\right)$$
  where
  $$g(s) = \frac{1}{1 + e^{-s}}$$
  and $t$ and $t+1$ are successive time steps.

Rule strength calculation using softmin operator

- Using the softmin, the strength of Rule 1 is:
  $$w_1 = \mu_a(x_1)e^\frac{-\mu_a(y_1)}{\mu_a(y_2)} + \mu_b(x_2)e^\frac{-\mu_b(y_2)}{\mu_b(y_1)}$$
  Similarly, we can find $w_2$ for Rule 2.
- The control output of rule 1:
  $$z_1 = \mu_a(y_1)$$
  and for Rule 2:
  $$z_2 = \mu_b(y_2)$$
- Using a weighted averaging approach, $z_1$ and $z_2$ are combined to produce the combined result $\hat{z}$.

GARIC Applied to Cart-Pole Balancing

The GARIC-Q Architecture (Cont..)

- The FQ values are updated according to:
  $$\Delta FQ = \beta(\nu + \tilde{y} - V(x))$$
  where $\nu$ is the value of state $x$ and action $a_k$ selected through a Boltzman process.
- $\tilde{y}$ is the value of the best state-agent pair defined by:
  $$V(y) = \max FQ(y,a_k)$$
  where $k = 1$ to $K$, $i$ is the agent number and $a_k$ is its recommended final action.

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 $K$.”
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 \( x \).

\[
FQ(x, a) = E[(r + \gamma V(y)) \cdot \mu_c(x, a)]
\]

- Update Rule:

\[
\Delta FQ(x, a) \leftarrow \beta [(r + \gamma V(y)) \cdot \mu_c(x, a) - FQ(x, a)]
\]

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 \( \hat{f}(t-1) \) 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

- \( TD(\lambda) \): 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:

\[
\Delta Q_\lambda(s_t, a_t) = \alpha [r_{t+1} + \gamma \max_a Q_\lambda(s_{t+1}, a) - Q_\lambda(s_t, a_t)]
\]
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 state 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 goals 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 (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 $x$.
- $FQ(x,a) = E[(r + \hat{r}(y)) \land \mu_c(x,a)]$

- Update Rule:
  \begin{align*}
  \Delta FQ(x,a) & \leftarrow \beta[(r + \hat{r}(y)) \land \mu_c(x,a) - FQ(x,a)] \\
  FQ(x,a) & \leftarrow FQ(x,a) + \beta[(r + \hat{r}(y)) \land \mu_c(x,a) - FQ(x,a)]
  \end{align*}

The FQ-Learning Algorithm

- Initialize FQ values
- Until FQ values converge do {
  1. $x \leftarrow$ 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 $FQ(x,a) \leftarrow FQ(x,a) + \beta[(r + \hat{r}(y)) \land \mu_c(x,a) - FQ(x,a)]$}
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 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 $p^*$

The GARIC-Q Architecture (Cont.)

- An approach similar to Giorenc’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
  $$ R = \sum_{k=1}^{K} R_k $$
- $R_j$ refers to rule number $i$ of agent $j$. Associated with each rule $R_j$ is a $f_q$, which represents the $f_q$ of rule $R_j$.

The Architecture of GARIC-Q

- Action Evaluation Network
- GARIC for agent 1
- GARIC for agent 2
- GARIC for agent 3
- GARIC for agent 4
- Plant
- External Reinforcement

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 $f_q$ values of all the rules used by the agent

The GARIC-Q Architecture (Cont.)

- The FQ value for an agent $k$ is calculated from:
  $$ FQ_k = \frac{\sum_{i=1}^{R_k} f_q_i \cdot \alpha_i}{\sum_{i=1}^{R_k} \alpha_i} $$
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:
  \[ V(y) = \max_a FQ(y, a) \]
  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:
  \[ r(t) = \begin{cases} +1 & \text{if within the success region} \\ 0 & \text{Viable zone} \\ -1 & \text{Failure} \end{cases} \]
- Within each agent or rule base \( k \), the reward or punishment is distributed based on the activity of rule \( i \):
  \[ \rho_i = \frac{\alpha_i}{\sum_i \alpha_i} \]
  where \( \alpha_i \) is the strength of rule \( i \).

The GARIC-Q Architecture (Cont..)

- The \( f_q \) values are updated for the selected agent \( j \) using:
  \[ \Delta f_{q_i} = \gamma + \rho \cdot \Delta FQ \]
- 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

<table>
<thead>
<tr>
<th>( \theta )</th>
<th>( \theta )</th>
<th>( x )</th>
<th>( i )</th>
<th>( F )</th>
</tr>
</thead>
<tbody>
<tr>
<td>PO1</td>
<td>PO2</td>
<td>null</td>
<td>null</td>
<td>PL</td>
</tr>
<tr>
<td>PO1</td>
<td>ZE2</td>
<td>null</td>
<td>null</td>
<td>PL</td>
</tr>
<tr>
<td>PO1</td>
<td>NE2</td>
<td>null</td>
<td>null</td>
<td>ZE</td>
</tr>
<tr>
<td>ZE1</td>
<td>PO2</td>
<td>null</td>
<td>null</td>
<td>ZE</td>
</tr>
<tr>
<td>ZE1</td>
<td>ZE2</td>
<td>null</td>
<td>null</td>
<td>ZE</td>
</tr>
<tr>
<td>ZE1</td>
<td>NE2</td>
<td>null</td>
<td>null</td>
<td>ZE</td>
</tr>
<tr>
<td>NE1</td>
<td>PO2</td>
<td>null</td>
<td>null</td>
<td>NS</td>
</tr>
<tr>
<td>NE1</td>
<td>ZE2</td>
<td>null</td>
<td>null</td>
<td>NL</td>
</tr>
<tr>
<td>NE1</td>
<td>NE2</td>
<td>null</td>
<td>null</td>
<td>NS</td>
</tr>
<tr>
<td>VS1</td>
<td>VS2</td>
<td>PO3</td>
<td>PO4</td>
<td>PS</td>
</tr>
<tr>
<td>VS1</td>
<td>VS2</td>
<td>PO3</td>
<td>PS4</td>
<td>PV</td>
</tr>
<tr>
<td>VS1</td>
<td>VS2</td>
<td>NE3</td>
<td>NE4</td>
<td>NS</td>
</tr>
<tr>
<td>VS1</td>
<td>VS2</td>
<td>NE3</td>
<td>NS4</td>
<td>NVS</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>( \theta )</th>
<th>( \theta )</th>
<th>( x )</th>
<th>( i )</th>
<th>( F )</th>
</tr>
</thead>
<tbody>
<tr>
<td>PO1</td>
<td>PO2</td>
<td>null</td>
<td>null</td>
<td>PL</td>
</tr>
<tr>
<td>PO1</td>
<td>ZE2</td>
<td>null</td>
<td>null</td>
<td>PL</td>
</tr>
<tr>
<td>PO1</td>
<td>NE2</td>
<td>null</td>
<td>null</td>
<td>ZE</td>
</tr>
<tr>
<td>ZE1</td>
<td>PO2</td>
<td>null</td>
<td>null</td>
<td>ZE</td>
</tr>
<tr>
<td>ZE1</td>
<td>ZE2</td>
<td>null</td>
<td>null</td>
<td>ZE</td>
</tr>
<tr>
<td>ZE1</td>
<td>NE2</td>
<td>null</td>
<td>null</td>
<td>ZE</td>
</tr>
<tr>
<td>NE1</td>
<td>PO2</td>
<td>null</td>
<td>null</td>
<td>NS</td>
</tr>
<tr>
<td>NE1</td>
<td>ZE2</td>
<td>null</td>
<td>null</td>
<td>NL</td>
</tr>
<tr>
<td>NE1</td>
<td>NE2</td>
<td>null</td>
<td>null</td>
<td>NS</td>
</tr>
<tr>
<td>VS1</td>
<td>VS2</td>
<td>PO3</td>
<td>PO4</td>
<td>PS</td>
</tr>
<tr>
<td>VS1</td>
<td>VS2</td>
<td>PO3</td>
<td>PS4</td>
<td>PV</td>
</tr>
<tr>
<td>VS1</td>
<td>VS2</td>
<td>NE3</td>
<td>NE4</td>
<td>NS</td>
</tr>
<tr>
<td>VS1</td>
<td>VS2</td>
<td>NE3</td>
<td>NS4</td>
<td>NVS</td>
</tr>
</tbody>
</table>

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.

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

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

USING FUZZY REINFORCEMENT LEARNING FOR POWER CONTROL IN WIRELESS TRANSMITTERS

David Vengerov
Hamid Berenji
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 $I_t$: $\text{Prob}(\text{success} | p_t, I_t) = 1 - e^{-\frac{I_t}{p_t}}$
- 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
Determine optimal constant power $p^*$
Initialize $p_1, \ldots, p_6$ to $p^*$
Let ACFRL tune $p_1, \ldots, p_6$

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

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

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

- 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: \[ \max_{a_t \in \mathcal{A}} \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) \right] \]

Subject to the constraint on the evolution of sequence of states:
\[ s_{t+1} = f(s_t, a_t). \]

Q-value: \[ Q(s, a) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) \right| s_0 = s, a_0 = a], \]
equivalent 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: \[ Q(s, a) = \mathbb{E} \left[ r + \gamma \max_a Q(s', a) \right] \]

Idea of Q-learning: compute a noisy sample of Bellman’s error:
\[ \delta_t = \mathbb{E} \left[ r + \gamma \max_a Q(s', a) \right] - Q(s, a) \]
\[ = r + \gamma \max_a Q(s', a) - Q(s, a) \]

Stochastic update in small discrete state-action spaces:
\[ Q(s, a) \leftarrow Q(s, a) + \alpha \delta_t \]
In large or continuous state-action spaces:
\[ \theta_t \leftarrow \theta_t + \alpha \delta_t \nabla_\theta Q(s, a, \theta) \]
Computational Theory of Perceptions

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

\[
Q(s, a, \theta) = \sum_{i=1}^{M} \theta^i w_i(s, a)
\]

Perception-based Q-Learning

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

\[
V_{\theta} Q(s_i, a_i, \theta) \text{ becomes } (w_1(s_i, a_i) , ..., w_M(s_i, a_i))^T
\]

Continuous update equation \( \theta' \leftarrow \theta + \alpha \delta V_{\theta} Q(s, a, \theta) \)

for perception-based rules becomes component-wise

\[
\theta^i \leftarrow \theta^i + \alpha \delta w_i(s, a, i = 1, ..., M)
\]

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

\[
\theta' \leftarrow \theta' + \alpha \delta \sum_{t=0}^{\gamma} (\gamma \kappa)^t w(s, a_t)
\]

AV Reward Functions

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

\[
r_j = \sum_{t=0}^{\gamma} \left( \frac{V_t}{1 + d_{t+1}^2} \frac{1}{\sum_{s=1}^{M} 1 + d_{s+1}^2} \right)
\]

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 \( j \):
  \( s[1] = \sum_{i=1}^{M} \frac{V_t}{1 + d_{t+1}^2} \)
  Sum of potentials of all other UAVs near target \( j \):
  \( s[2] = \sum_{t=0}^{\gamma} \frac{1}{1 + d_{t+1}^2} \)

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

- \( y[1] = \text{"target potential"} \)
- \( y[2] = \text{"AV potential"} \)

Potential Surface of AVs

Darker locations have higher target potentials

Rules for sensors alignment

- If \( (s_1, \text{is SMALL}) \) and \( (s_2, \text{is SMALL}) \) then \( \theta^1 \)
- If \( (s_1, \text{is SMALL}) \) and \( (s_2, \text{is LARGE}) \) then \( \theta^2 \)
- If \( (s_1, \text{is LARGE}) \) and \( (s_2, \text{is SMALL}) \) then \( \theta^3 \)
- If \( (s_1, \text{is LARGE}) \) and \( (s_2, \text{is LARGE}) \) then \( \theta^4 \)
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 $\lambda$:

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda = 0$</td>
<td>1.1</td>
<td>2.55</td>
</tr>
<tr>
<td>$\lambda = 0.5$</td>
<td>1.1</td>
<td>2.52</td>
</tr>
<tr>
<td>$\lambda = 0.9$</td>
<td>1.1</td>
<td>2.25</td>
</tr>
</tbody>
</table>

Decrease in performance for higher $\lambda$ => 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

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
- Agents learn the value of signals in the context of their local environments

Q-learning
- In discrete state and action spaces:
  \( Q(s,a) \leftarrow Q(s,a) + \alpha \gamma Q(s_{i+1},a) - Q(s,a) \). \\
- \( \alpha \gamma \) 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,
  \( \sum_{i=0}^{\infty} \alpha_{i} = \infty, \sum_{i=0}^{\infty} \alpha_{i} < \infty. \)

Q-learning with state generalization
\[
\theta_i \leftarrow \theta_i - \frac{1}{2} \alpha \nabla_{\theta} Q(s_{i+1},a_{i},\theta_{i}) - Q(s_i, a_i, \theta_i) \]
\[
\theta_i \leftarrow \theta_i + \alpha \nabla_{\theta} Q(s_{i+1},a_{i},\theta_{i}) \nabla_{\theta} Q(s_i, a_i, \theta_i) \]
- \( Q(s,a,\theta) \) approximates \( Q(s,a) \)
- \( \theta \) is the set of all parameters arranged in a single vector.

Q-learning
- \( Q(s,a) \) is the expected reward in state \( s \) after taking action \( a \) and following the optimal policy thereafter:
  \[
  Q(s,a) = E \{ R_{i+1} | s_i = s, a_i = a \} \\
  = E \{ \sum_{i=0}^{\infty} \gamma^i r_{i+1} | s_i = s, a_i = a \} 
  \]
- \( r_i \): the reward received after taking action \( a \) in state \( s \).
- \( \gamma \): is the discounting factor.

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

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 $j$.
  \[ P_j = \frac{V_i}{1 + d_{ij}^2} \]
- 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 $Q_i$)

- Final value of moving to location $L_i$:
  \[ Q' = \sum_k \mu_k Q_i \]
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


References (Cont...)