Course Notes
CECS 477 : Neural Networks
Lecture of February 16, 2000
By Jim Ries, JimR@acm.org

Reinforcement Learning

- AI machine learning dates back to late 1950's (most in late 198's, early 1990's)
- Operations Research (control)
- Markov decision processes
  - Make decisions based on current state. State of world changes. New state stochastically determined by previous state and current conditions (nothing in past).
- McCallum's paper asks how add history

Markov Decision Process

- Discrete time, stochastic system with discrete state space

t=0,1,2,…
s=(1,2,…}
given state I, choose action $u \in U(I)$
i➔ U(I)
i➔ cost, our task is to minimize total cost
or
i➔ reward, our task is to maximize total reward
given i,u ➔ j (new state) $P_{ij}(u)$
where $P_{ij}(u)$ is the probability of going from state i to j given action u
action policy:
μ_i i➔u
$J^{\mu}(i)$ - estimate of plan μ starting form state i
infinite horizon: $J^{\mu}(i) = E_{\mu}[$ for t=0 to t=∞ $\sum \gamma^t c_t \mid s_0=i]$

c_t = cost in time step t
s_0 = start state
γ^t = discount factor (if reward is in future, it matters less)
$0 < \gamma \leq 1$ (typically, γ would be decreasing with time)

finite horizon is same, but iterates to M (limit) only, and thus discount factor can be 1 without problem.

Sometimes, we want average cost, e.g.,

E_{μ} [for t=0 to t=∞ $\sum \gamma^t c_t \mid s_0=i] / M$
(or the lim M➔ ∞ of above in infinite case)

We use dynamic programming to find optimal policy. Reinforcement learning is a variant of dynamic programming.

value iteration

policy iteration

offline - traditional dynamic programming
online - reinforcement learning

<u>Bellman equation</u>

- idea: improve policy and estimates of value of policy through many iterations (successive approximations) and at the same time improve the policy.

$J_{k+1}(i)$ = min where $u \in U(i)$ $[C_\gamma(u) + \gamma * j \in s \sum P_{ij}(u) J_k(i)] = Q(i,u)$
above equation without "k's" is Bellman equation

greedy policy

"one-step backup" - uses next state to update estimate of current state.

<u>Policy Iteration</u>

- attempts to decouple learning of policy from learning of the value function

$J(i) = c_\gamma(u) + \gamma * j \in s \sum P_{ij}(u) J(j)$
where u is action chose $u = \mu(i)$
update policy, go back, …

<u>algorithm:</u>

init current policy
while current policy ≠ next policy
learn value function of policy
update policy (greedy actions based on current learned value function)
endwhile

Process will converge to optimal policy

- In value iteration, we use greedy policy at every step. Since value function is changing, so is the policy. Once we learn value function, the policy will stop changing.
- In policy iteration,…

Claim 1: Only policies that are greedy with regard to their own value function are optimal policies.
$Q^{J^*}(i,\mu^*(i))$ = min $u \in U(i)$ $Q^{J^*}(i,u)$
μ^* is optimal policy
J^* is value function of μ^*

Claim 2: A necessary and sufficient condition for value function to be value function of the optimal policy is:
$J(i)$ = min $u \in U(i)$ $[C_i(u) + \gamma * j \in s \sum P_{ij}(u) J(j)]$ (Bellman Equation)

Thus, a solution to Bellman's equation leads to an optimal policy.

Synchronous update: $J_{k+1}(i) = \min Q(i,u)$ (update all states in parallel)
Asynchronous update: update each state iteratively.
Both converge.

<u>Real-time Dynamic Programming (an instance of on-line)</u>
- learn value function while interacting with the world
- update only states we "visit" (asynchronous updating)
- intermediate results will be used in control
- must update every state infinitely often to converge (can use random exploration scheme to assure all states are explored). Can also estimate confidence for each region, and intentionally visit states not often visited.

Random exploration

$\text{Prob}(i,a) = e^{Q(i,a)/t}/\sum$ for all a $e^{Q(i,a)/t}$

t = "temperature"; t starts very high and settles ("cools") down after a while. So, we start random and make it more deterministic over time.

- this is essentially "simulated annealing"

<u>Dynamic Programming with incomplete information</u>
- e.g., we don't have probability of transition
- Baysian or Non-baysian estimate of transition probabilities

Non-baysian - indirect estimate. Direct estimate is Q-Learning (model-free RL)

Bartol, et. al. 1990

<u>indirect estimate</u>
$P_{ij}(u) = \eta_{ij}^{u} / \sum l \in s\ \eta_{il}^{u}$
where η_{ij}^{u} is # of observed transitions from i to j after action u
(maximum likelihood estimate)

<u>Q-Learning</u>
Off-line Q-learning
Adaptive Q-learning
Real-time Q-learning (also just called "Q-learning"; original Q-learning)

<u>off-line Q-learning</u>
$Q_{k+1}(i,u) = (1- \alpha_k(i,u))Q_k(i,u) + \alpha_k(i,u)[c_i(u) + \gamma * \min$ over u' $Q(succ(i,u), u')]$
where α is the learning rate; rewrite as:
$\Delta Q_{k+1}(i,u) = \alpha_k(i,u)[c_i(u) + \gamma * \min$ over u' $Q(succ(i,u), u') - Q_k(i,u)]$

So, since we update many times, and only learn a little (α) each time, we can try many successors and learn most probable ones.

In real-time, we have no model of world so we just get successor from the real world (prevailing conditions).

α should decrease over time. If each pair updated infinitely often and α decreases over time, algorithm will converge with probability 1.

Q-Learning	**Value Iteration**
more space	only need value of each state
update value many times	one-step updates
each step fast (good for real-time)	each step slow (due to $\sum$)

Real-time and offline can be mixed. "Contemplate" after actions learned.

Reference:
Barto, A.G., Bradtke, S.J., and Singh, S.P. (1995) Learning to act using real-time dynamic programming, Artificial Intelligence 72: 81–138.
(see http://www.umass.edu/neuro/faculty/barto.html for more on Andrew Barto)

Storing Value functions (Q or J)
Exact Method
Lookup table representation, i.e.,

State	Action	Q-value
…	…	…

Approximate methods:
- linear, weighted sums
- back propagation net
- decision trees
- radial basis function
- statistical clustering to minimize space usage
- Baysian Network

Paper presented by Jonathon Marjamaa
"Self-Improving Reactive Agents Based On Reinforcement Learning, Planning, and Teaching"
By Long-Ji Lin, Carnegie Mellon University 1992

http://riesj.hmi.missouri.edu/CECS477/RL/

Self-Improving Reactive Agents Based On Reinforcement Learning, Planning and Teaching

By Long-Ji Lin, Carnegie Mellon University 1992

Presented By Jonathon Marjamaa
February 16, 2000

Parrtially observable decision Markvo Process (can't see all of the states)

Overview

- Introduction
- Reinforcement learning frameworks
  - AHC-learning: Framework AHCON
  - Q-Learning: Framework QCON
  - Experience Replay: Frameworks AHCON-R and QCON-R
  - Using Action Models: Frameworks AHCON-M and QCON-M
  - Teaching: Frameworks AHCON-T and QCON-T
- A dynamic environment
- The Learning agents
- Experimental results
- Discussion
- Limitations
- Conclusion

Introduction

- Goals:
  - Apply connectionist reinforcement learning to non-trivial learning problems.
  - Study method for speeding up reinforcement learning.
- Tests:
  - AHC (adaptive heuristic critic)
  - Q-Learning
  - AHC and Q-learning with experience replay, action models, and teaching.
- These will be tested in a non-deterministic dynamic environment.

Reinforcement Learning Frameworks

- 3 stages of a reinforcement learner:

  1 - Learning agent receives sensory input from the environment

  2 - The agent selects and performs an action

  3 - The agent receives a scalar signal from the environment

  The signal can be +(reward), -(punishment), or 0.

- The learners goal is to create a optimal action selection policy.
- Performance is measured by utility:

$$V_t=\sum_{k=0}^{infinity} \gamma^k r_{t+k} \quad (1)$$

V_t Utility from time t

γ discount factor ($0 <= \gamma <= 1$)

r_{t+1} reinforcement from r_t to r_{t+1}

Reinforcement learning frameworks

- A framework will attempt to learn a evaluation function, *eval(y)*, to predict the utility.

$$\text{util}(x, a) = r + \gamma * \text{eval}(y) \qquad (2)$$

util(x, a)	expected utility of action ‘a’ on world state x.
r	immediate reinforcement value
eval(y)	utility of the next state

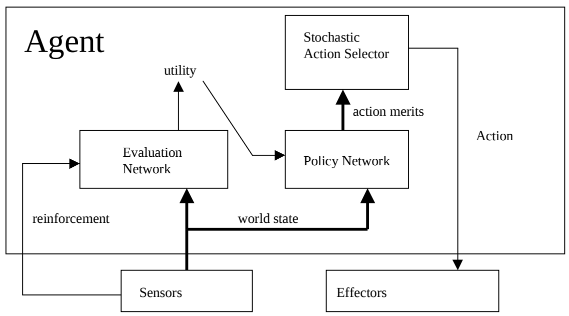
R=-c, if c is a cost and r is a reward

AHC-learning: Framework AHCON

- 3 components: evaluation network, policy network, stochastic action selector
- Decomposes reinforcement learning into 2 subtasks:

1. Construct a model of eval(x) using the evaluation network.

2. Assign higher merits to actions that result in higher utilities (as measured by the evaluation network) in the Policy Network.



AHC-Learning: Framework AHCON

1. xϕcurrent state; eϕeval(x);

2. aϕselect(policy(x),T);

3. Perform action a; (y,r)ϕnew state and reinforcement;

4. e'ϕ r + γ eval(y);

5. Adjust evaluation network by backpropogating TD error (e' - e) through it with input x;

6. Adjust policy network by backpropogating error Δ through it with input x, where

   Δ_i= e'-e   if i = a, and 0 otherwise

7. Go to 1.

---

select(p, T) is based on the follow probability function

$$\text{Prob}(a_i) = e\text{^}(m_i/T)/\sum_k e\text{^}(m_k/T) \qquad (4)$$

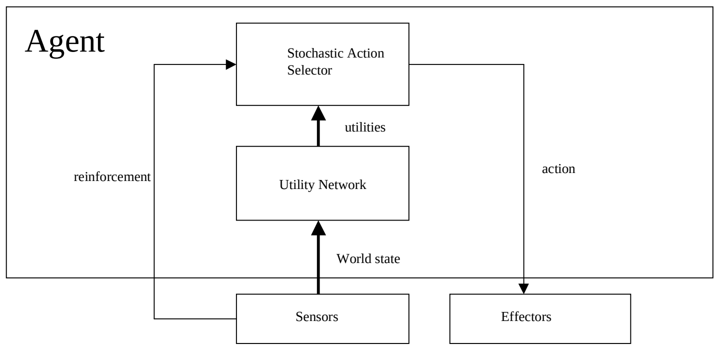
where m_i is the merit of action a_i, and the temperature T adjusts the randomness

TD -> Temporal Difference

Q-Learning: Framework QCON

- QCON learns a utility network that models util(x, a)
- Given a utility net., a state, the agent chooses the action with the maximum util(x, a).

$$\text{util}(x,a) = r + \gamma\,\text{Max}\{\ \text{util}(\ y, k\ )\ |\ k, \text{ an element of actions }\} \tag{5}$$



Q-Learning: Framework QCON

1. xϕcurrent state; for each action i, $U_i \phi$util(x,i);

2. aϕselect(U,T);

3. Perform action a; (y,r)ϕnew state and reinforcement;

4. u'ϕr + γ * max{ util(y,k) | k is an element of actions };

5. Adjust utility network by backpropogating error ΔU through it with input x, where ΔU_i=u'-U_i if i = a, otherwise 0;

6. Go to 1;

Experience Replay

- Learns faster by replaying experiences (x, a, y, r)
- In AHCON-R one only replays policy actions so that a non-policy action does not ruin the utility of a good state.
- In QCON-R one only replays policy actions so that bad actions do not make a network underestimate the value of a good state.
- Policy actions are those above a set threshold.
- Only recent experiences are replayed, so the their significance is not overplayed.

- better if done backward since more rewards are given at the end.

- If no NN implementation, there should be no interfereance.

Action Models

- Action models attempt to build a function from (x,a) to (y,r).
- Determines how ‘a’ acts upon ‘x’.

- Only represents most likely outcome, rather than a distribution.

Framework AHCON-M

- Uses the relaxation planning algorithm
- Produces a series of look-aheads using the action model.
- Since all actions are examined, relative merits of actions can more directly be assigned than in standard AHCON.

1. $x \phi$ current state; $e \phi eval(x)$;
2. Select promising actions S according to policy(x);
3. If there is only one action in S, go to 8;
4. For a, an element of S, do
    - 4a. Simulate action a; $(y,r) \phi$ predicted new state and reinforcement
    - 4b. $E_a \phi r + \gamma * eval(y)$;
5. $\mu \phi \Sigma_a Prob(a) * E_a$; $max \phi Max\{E_a \mid a$ is an element of S$\}$
6. Adjust Eval. Net. by backpropogating error (max-e) through it with input x;
7. Adjust policy net. by backpropogating error Δ through it with input x,
    where $\Delta = E_a - \mu$ if a is an element of S, and 0 otherwise
8. Exit.

- used to speed learning

Framework QCON-M

- Used in the same way as with AHCON-M.

1. $x \phi$current state; for each action i, $U_i \phi util(x,i)$;
2. Select promising action S, according to U;
3. If there is only one action in S, go to 6;
4. For every ‘a’, an element of S, do

    4a. Simulate action a; $(y,r)\phi$predicted new state and reinforcement;

    4b. $U_a’ \phi r + \gamma$ * Max{ util(y,k) | k is an element of actions };

5. Adjust util. net. by backpropogating error ΔU through it with input x, where

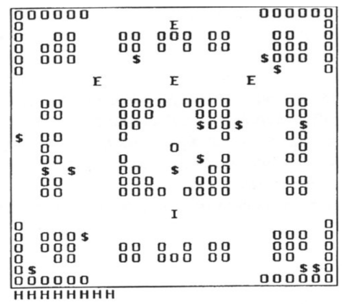
    $\Delta U_a = U_a’ - U_a$ if ‘a’ is an element of S, 0 otherwise.

6. Exit.

Teaching: Frameworks AHCON-T and QCON-T

- Builds upon the Action Replay frameworks.
- An external teacher provides the learner with a lesson (a set of actions.)
- The agent can play taught lessons just like experienced ones.
- Agents can learn from both positive and negative examples.

The test environment



I = agent

E = Enemy, Enemies move randomly, and towards the Agent.

O = Obstacle

$ = Food (+ 15 Health)

H = Health

Each move costs 1 health.

When an agent dies, they are brought to a new map, learning nets preserved.

- goal is to get all food without dying

The Learning Agents

The Reinforcement Signal

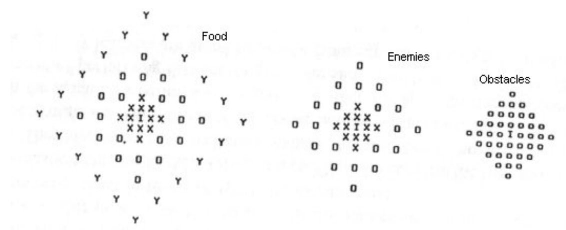
- -1.0 if the agent dies
- 0.4 if the agent gets food
- 0.0 otherwise

Action Representation

- Global: Actions are North, South, East and West
- Local: Actions are Forward, Backward, Left and Right

- should try for food even without reward since it dies without food

Input Representation



Each network has 145 input units belonging to the following five groups:

1. Enemy Map
2. Food Map
3. Obstacle Map
4. Energy Map
5. History Information (previous action choice, and if it resulted in an obstacle collision.)

Output Representation

Global:

1 policy net. finds the merit of moving North.

Other directions are determined by rotating state maps.

1 utility net. finds the utility of moving North.

Local:

No symmetry is used.

AHC uses 4 policy networks, Q-Learning uses 4 utility networks.

All output are truncated to be between -1 and 1.

Action Models

AHCON-M and QCON-M used two 2-layer networks

- Reinforcement Network: predicts the immediate reinforcement signal.
- Enemy Network: predicts enemy movement.

Enemy networks only took the enemy, obstacle maps as input.

Reinforcement networks took all 145 inputs.

Active Exploration

The learner uses the Stochastic action selector and sets the temperature to be higher when it gets stuck in order to balance between learning and gaining rewards.

Prevention of over-training

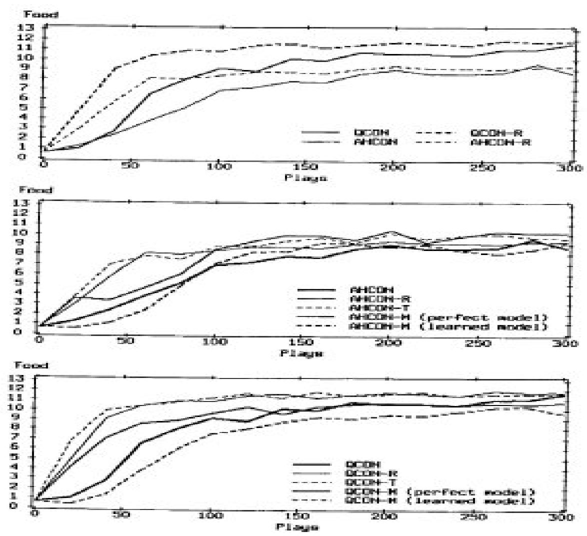
After each play, only n of the last 100 learned lessons are played back. Lessons are chosen randomly, with the most recent lessons most likely to be chosen.

n is a decreasing number between 12 and 4

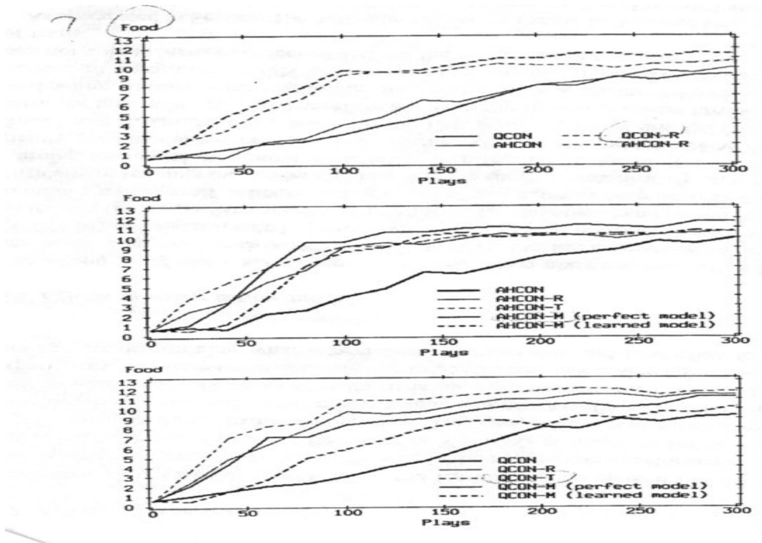
After each play, the agent chooses taught lessons to play. Lessons have a decreasing probability of being chosen between 0.5 and 0.1.

- “being chosen” means probability of being chosen for replay

Experimental Results (Global Representation)



Experimental Results (Local Representation)



QCON-T results

Got all food	Got Killed	Ran out of Energy
39.9%	31.9%	28.2%

#	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
%	0.1	0.3	0.8	1.8	2.2	2.9	4.0	4.1	3.8	3.7	3.4	4.1	5.4	8.2	15.2	39.9

Amount of food found

- note that he uses neural network representation and there is thus no guarantee of convergence.

- Is it possible to always get all food (probably not)?

Discussion

AHCON vs. QCON

Effects of experience replay

Effects of using action models

Effects of teaching

Experience replay vs. using action models

Why not perfect performance?

1. Insufficient input information
2. The problem is too complex for the network.

- Speeds learning, but doesn’t effect result (should have counted number of updates to compare).

- Humans could see the entire board?

Limitations

Representation dependent: An optimal input representation must be found first.

Discrete time and discrete actions: It would be difficult to apply this to continuous time applications.

Unwise use of sensing: Some input should be filtered.

History insensitive: Agents are reactive, and do not make decisions based of past information.

Perceptual Aliasing: Sometimes different states might appear the same to an agent.

No Hierarchical control: TD work less accurately over longer series of action. A way of creating sub-tasks would be ideal.

E.g., robot opens box, puts something in, closes box, and now thinks this is a start state (erroneously).

Conclusions

1. QCON was generally better at learning than AHCON.

2. Action models were not very good in this dynamic, non-deterministic world.

3. Experience replay was more effective than action models in this case.

4. Experience replay increase the learning rate.

5. Teaching effectively reduces the learning time by reducing the necessary trial-and-error, and helping avoid local maxima.