Reinforcement Learning
Lecture 13

Gillian Hayes

19th February 2007

School of informatics

Using Q-Learning: Asterix and Obelix

- Asterix and Obelix robots
- Q-learning
- Statistical Clustering

A Q-Learning Example

- Obelix – Mahadevan and Connell (IBM)
- Asterix – John Hoar (DAI MSc 1996)
- A behaviour-based robot learning to push a box
- Learns behaviours by trial and error: Q-learning
- Rewards/punishments for doing right/wrong thing
- Simulator and real robot
- Reimplemented (here) on Asterix

How to frame a problem so that we can use reinforcement learning with it

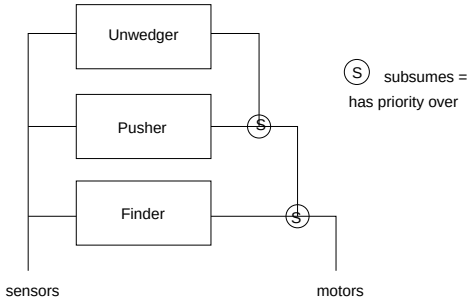
Obelix Robot

- Actions: move forward, turn left, turn hard left, turn right, turn hard right
- Sensors: sonar (35ft, 20degrees, time to echo), infrareds (4 inches), motorcurrent (increases if robot stuck)
- Can measure distance to object to nearest 1/4 inch, but use range bins: near/far
- Blind spot: object permanence a problem – if the robot turns through a small angle, small object disappear until the next sensor picks them up
- State space $2^{18} = 262144$ states: is this too many?
- Find box, push it, don't get stuck.
- How to organise the RL problem? In one monolithic block? In submodules that learn parts of the task?

See diagrams of Obelix, its environment, its sonars, its sensor quantisation

Architecture for Obelix: Subsumption Architecture

- Behaviour-based modules: unwedger, pusher, finder



- Each module gets own reward
- 3 copies of Q-learning algorithm

- Priorities: Unwedger > Pusher > Finder (= default behaviour)
- Modules have applicability predicates
- So combine BB modules with RL, rather than using a monolithic controller

Applicability Predicates

Under what sensor conditions is a module applicable?

e.g.

```
PUSHER_APPLICABLE
if BUMP sensors on
    return TRUE
else if BUMP recently activated    (perceptual
    return TRUE                    aliasing)
else return FALSE
```

Current state + recent activation allows perceptual aliasing to be dealt with (a bump = false state often occurs after pushing the box; box bounces away from bump sensor)

- If > 1 applicable, priority ordering decides which gets a chance to learn

- Default behaviour = box_finder (so no applicability predicate needed)
- Perceptual aliasing – same state needs different actions depending on context, e.g. BUMP

See diagram of control flow

Reward Functions $R^a_{ss'}$

One per module

e.g. BOX_FINDER_REWARD(old state, action, new state)

```
if action = FORWARD and
    FRONT_NEAR_SONAR(new state)     facing
                                    object
    return 3
else if ¬ FRONT_NEAR_SONAR(new state)
    return -1
else return 0
```


Procedure:

What modules are applicable?

Arbitration ⇒ module chosen

Reward calculated

Module learns

- Robot gets plan sketch for box-pushing. It fill in details.

Learning:$Q(s, a)$

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

Always update using *maximum* Q value available from next state: then $Q \Rightarrow Q^*$, optimal action-value function

Problem

OBELIX has 18 sensory bits and 262144 states.
Can't experience all in reasonable time

Solution

Generalise:

1. 'Virtual' sensors combine readings from real ones ⇒ 9 bits
2. Don't just update current state, update all 'similar' states.

Take weighted Hamming distance between state vectors – weights give relative importance of bits, e.g. bump/stuck = 5, near = 2, far = 1.

Structural credit assignment

Task

So, given

- states
- actions
- behaviour-based task modules and priority ordering
- applicability and reward functions

learn

- Transfer function from state ⇒ action which maximises cumulative expected reinforcement

1. Initialise $Q(s, a)$ for all x, a to 0, $\alpha = 0.5$, $\gamma = 0.9$

2. Do forever:

   - Observe current state s_t
   - 90% of time choose an action that has maximum $Q(s_t, a_t)$, 10% choose some other action. Action is a_t.
   - Carry out action a_t in world. New state is s_{t+1}
   - Immediate reward for carrying out a in s_t is r_{t+1}
   - Update Q: $\Delta Q = \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$ for state s_t and all states within weighted Hamming distance of 2 from s_t

Alternative Generalisation Method – Statistical Clustering

Weighted Hamming distance OK for noisefree simulations, but use statistical clustering for noisy real world environment

For each action

A set of clusters of states with similar Qs

Cluster: $< p_{bit1}, p_{bit2}, \ldots p_{bitn}, Q_c >$

p_{bit1} is $p(bit1 \mid s \in c)$, i.e. probability that bit 1 = 1 given that the state s is in the cluster c – you can get this from counting over all the states that are already in the cluster and from sensor statistics. Q_c is the Q value of the cluster

To be in the cluster, a state's bits must match those of the cluster closely and the state must have a Q value very close to that of the cluster and differing from it by no more than δ

Updating Clusters and Action Selection

New cluster: if difference between current state and existing clusters is too big, start a new cluster

Merge two clusters: if close enough merge and update new cluster statistics

Action selection: match state against all clusters, all actions. Calculate its Q-value for each action in turn:

$$Q(s, a) = \frac{\sum_{\text{clustersforthataction}} Q_{\text{cluster}} \times \text{weight}}{\sum \text{weight}}$$

where weight = match probability for the state being in that cluster.

Pick action with the biggest $Q(s, a)$

Scales better than Hamming

Experiments

Hamming compared with:

- random agent: chooses actions randomly
- hand-coded: give it the reward for each action that the learning agent would have got
- statistical clustering

on

- simulator
- robots

Results

- learning curves
- learning agents often as good as hand-coded

Issues

Asterix: a good internal learning curve doesn't always correspond with good behaviour in world

- push walls, not boxes
- find boxes, then move away
- zoom around world as quickly as possible

Difficulty: designing a good reward function. Learner will ALWAYS take advantage....

So always evaluate on real task, even if learning curves increase properly

RL for Real World Problems

Slow

Noisy sensors – can states be reidentified?

Generalisation – how to do it? 40% of processor time spent matching states to clusters in Asterix

Actions – reliability and repeatability

Monolithic vs. modular architecture

Specifying reward function difficult

But may be easier than writing controller

Most of effort goes into setting up the problem so that the RL algorithm can be applied.

BUT: if we get the right level of abstraction it can be done

References

John Hoar: Reinforcement Learning Applied to a Real Robot Task. DAI MSc dissertation, University of Edinburgh (1996).

S.Mahadevan and J.Connell: Automatic Programming of Behavior-Based Robots Using Reinforcement Learning. Artificial Intelligence 55, 311–365 (1992).

Jeremy Wyatt, John Hoar, Gillian Hayes: Design, Analysis and Comparison of Robot Learners. Robotics and Autonomous Systems 24(1–2), 17–32 (1998).