Reinforcement Learning

COMP3431 Robot Software Architectures
A Simple Learning Robot
Reinforcement Learning

• “Stumpy” receives a \textit{reward} after each action
  • Did it move forward or not?
  • After each move, updates its \textit{policy}
  • Continues trying to maximise its reward
Pole Balancing

- Pole balancing can be learned the same way except that reward is only received at the end after falling or hitting the end of the track.
Boxes

- State space is discretised
- Each “box” represents a subset of state space
- When system lands in a box, execute action specified
  - left push
  - right push
MENACE
Simulation

\[ x_{t+1} = x_t + \tau \dot{x}_t \]
\[ \dot{x}_{t+1} = \dot{x}_t + \tau \ddot{x}_t \]
\[ \theta_{t+1} = \theta_t + \tau \dot{\theta}_t \]
\[ \dot{\theta}_{t+1} = \dot{\theta}_t + \tau \ddot{\theta}_t \]

\[ \dot{x}_t = \frac{F_t + m_p l \left[ \dot{\theta}_t^2 \sin \theta_t - \dot{\theta}_t \cos \theta_t \right]}{m_c + m_p} \]

\[ \ddot{\theta}_t = \frac{g \sin \theta_t + \cos \theta_t}{l} \left[ \frac{-F_t - m_p l \dot{\theta}_t^2 \sin \theta_t}{m_c + m_p} \right] \]

\[ l \left[ \frac{4}{3} - \frac{m_p \cos^2 \theta_t}{m_c + m_p} \right] \]
Parameters

\( m_c = 1.0 \text{ kg} \quad \text{mass of cart} \)

\( m_p = 1.0 \text{ kg} \quad \text{mass of pole} \)

\( l = 0.5 \text{ m} \quad \text{distance of centre of mass of pole from the pivot} \)

\( g = 9.8 \text{ ms}^{-2} \quad \text{acceleration due to gravity} \)

\( F_t = \pm 10 \text{ N} \quad \text{force applied to cart} \)

\( t = 0.02 \text{ s} \quad \text{time interval of simulation} \)
The BOXES Algorithm

- Each box contains statistics on performance of controller, which are updated after each failure

- How many times each action has been performed \((usage)\)

- The sum of lengths of time the system has survived after taking a particular action \((LifeTime)\)

- Each sum is weighted by a number less than one which places a discount on earlier experience.
Update Rule

if an action has not been tested

choose that action

else if \( \frac{LeftLife}{LeftUsage^k} > \frac{RightLife}{RightUsage^k} \)

choose left

else

choose right

\[ k \] is a bias to force exploration

e.g. \( k = 1.4 \)
Performance

• BOXES is *much* faster than genetic algorithm
  • Only 75 trials, on average, to reach 10,000 time steps
  • But only works for *episodic* problems
    • i.e. has a specific termination
  • Doesn’t work for continuous problems like Stumpy
State Transition Graph
States and Actions

- Each node is a *state*
- *Actions* cause transitions from one state to another
- A *policy* is the set of transition rules
  - i.e. which action to apply in a given state
- Agent receives a *reward* after each action
- Actions may be non-deterministic
  - Same action may not always produce same state
Markov Decision Process (MDP)

- Assume that current state has all the information needed to decide which action to take
Grid World Example
Expected Reward

• Try to maximise expected future reward:

\[ V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \]
\[ = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \]

• \( V \) is the value of state \( S \) under policy \( \pi \)
• \( \gamma \) is a discount factor \((0..1)\)
Q Function

• How to choose an action in a state?

\[ Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a)) \]

• The Q value for an action, a, in a state, s, is the immediate reward for the action plus the discounted value of following the optimal policy after that.

• \( V^* \) is value obtained by following the optimal policy.

• \( \delta(s,a) \) is the succeeding state, assuming the optimal policy.
Q Learning

initialise $Q(s,a)=0$ for all $s$ and $a$
observe current state $s$
repeat
    select an action $a$ and execute it
    observe immediate reward $r$ and next state $s'$
    $Q(s,a) \leftarrow r + \max_{a'} Q(s',a')$
    $s \leftarrow s'$
Exploration vs Exploitation

• How do you choose an action?

  • Random

  • Pick the current “best” action

• Combination:

  • most of the time pick the best action

  • occasionally throw in random action
Background

- Reinforcement learning is based in earlier work in optimisation: dynamic programming

- Text book: Sutton & Barto
Reinforcement Learning

Variants

• There are *many* variations on reinforcement learning to improve search.

• RL was one of the components of alphaGo, which recently beat a Go master

• Used to learn helicopter aerobatics