1 The Problem

The general reinforcement learning (RL) problem: an agent acts in an unknown environment and receives observations and rewards in cycles. The agent’s task is to act so as to receive as much reward as possible.

2 Feature RL

Feature Reinforcement Learning aims to automatically reduce a complex real-world problem to a useful representation (MDP) i.e. create a map $\phi$ from an agent’s history to an MDP state. $\phi$ is then a function that produces a relevant summary of the history, $\phi(h) = s_t$.

In order to select the best $\phi$, we need a cost function on $\phi$ and a way to search over the space containing $\phi$. The original cost function used is given below.

$$Cost(\phi[h]) = CL(s_{t+n}|s_{t+n}) + CL(s_{t+n}|s_{t+n-1}) + CL(\phi)$$

- A global stochastic search (e.g. simulated annealing) is used to find the $\phi$ with minimal cost.
- Traditional RL methods can then be used to find the optimal policy given the minimal $\phi$.

3 Model-free Cost

- The above cost function is model-based and cannot be directly used in large observation spaces.
- A viable alternative is to use a model-free cost that can then be function approximated.

For each $\phi$ we define a Q-table based on the state space given by $\phi$ to be of the form $Q(\phi(h), a)$. We use the squared pathwise Q-learning error to find a suitable map $\phi : H \rightarrow S$ by selecting $\phi$ to minimise the following cost,

$$Cost_{QL}(\phi) = \min \frac{1}{Q} \sum_{t=1}^{n} (r_{t+1} + \gamma \max_a Q(\phi(h_{t+1}), a) - Q(\phi(h_t), a_t))^2 + Reg(\phi)$$

This can easily be extended to the linear function approximation setting by representing $Q$ as linear function over some features. The stochastic search is then over these features.

4 Algorithm

An intuitive explanation of the algorithm is as follows.

- We are given some environment which produces an observation and reward when given an action.
- Act randomly for a predefined amount of time.
- Loop the following.
  - Run simulated annealing on the existing history using the above Cost function to get $\phi$.

- Map the history sequence to a state sequence.
- Use Q-learning on the state sequence to find a suitable policy.
- Act according to this policy for some time while adding the resulting observations and rewards to the history.

5 Features

- For the tabular approach, we used suffix trees which allow us to uniquely map history sequences to state sequences.
- For the linear function approximation we defined a set of features known as event selectors (and a modified version called bit selectors).
- An event selector is a set of features $\xi_j$. Each feature $\xi_j$ consists of a position $m$ and an observation $o$. Feature $\xi_j$ is on if the $(m-n)$th position in the history has observation $o$.
- A bit selector is similar but picks out bits of the history instead of observations.

6 Experiments

- The algorithm was tested on three domains. Tiger, Cheesemaze and Partially Observable Pacman (Pocman).
- On Tiger and Cheesemaze we compared against the standard $\phi$ MDP algorithm using suffix trees and event selectors.
- On Pocman, we compared against MC-AIXI using bit selectors.
- In the smaller environments, our algorithm performed competitively achieving optimal performance but being more feature-efficient in the function approximation case.
- On Pocman, we perform better than MC-AIXI and use 25 times less space and are twice as fast.

7 Conclusion

- The algorithm presented above can be viewed as an extension of Q-learning to the history-based setting.
- It allows us to scale the feature RL framework to large environments.