COMP90054 – Week 10 tutorial

Last updated: 8 May 2023

Q-function approximation

Idea

- Q-table has two main limitations
    - Cannot provide estimates of $Q(s, a)$ that are not previously encountered
    - Q-table has size $|S| \times |A|$ where $|S|$ can be very large
- Solution: approximate Q-function with
    - a linear function
    - a deep neural network

Linear function approximation

Overview

- Approximate Q-function using a weighted linear sum of features
- Given a state s and an action a, approximate $Q(s, a)$ with n features:

$$Q(s,a) = w_1^a \cdot f_1(s,a) + w_2^a \cdot f_2(s,a) + \cdots + w_n^a \cdot f_n(s,a)$$

Process

- For the states, consider what are the features that determine its representation
- During learning, perform updates based on the weights of features instead of states
- Estimate $Q(s, a)$ by summing the features and their weights

Component

- Feature vector $\boldsymbol{f}$
    - a vector of $n \cdot |A|$ functions for n features and $|A|$ actions.
    - Each function extracts a state-action feature value from a state-action pair (s, a)
- Weight vector $\boldsymbol{w}$
    - a vector of $n \cdot |A|$ weights for each feature-action pair
    - w_i^a defines the weight of a feature i and action a
    - Weight updates: $w_i^a \leftarrow w_i^a + \alpha \cdot \delta \cdot f_i(s,a)$, where δ depends on algorithm used
        - e.g. In Q-learning $\delta = r + \gamma \max_{a' \in A(s')} Q(s', a') - Q(s, a)$

Monte-Carlo Tree Search (MCTS)

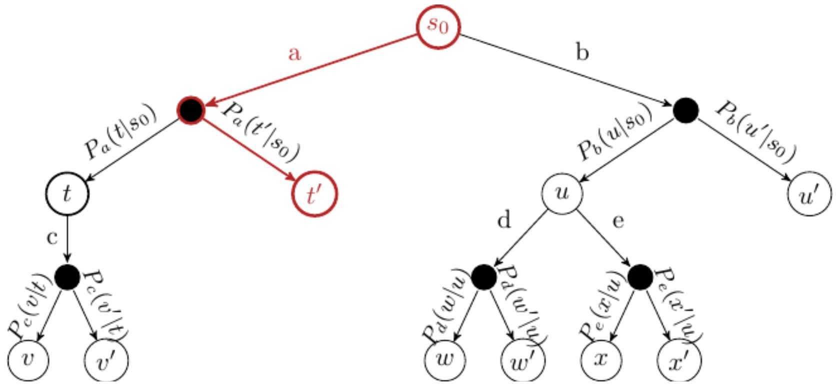
Idea

- MCTS is an *online* algorithm that builds a search tree through simulation to evaluate and select the most promising moves in a game or decision-making process
    - Online: plan immediately before executing an action. Once an action (or a sequence of actions) is executed, start planning again from the new state
    - Offline: solve the problem offline for all possible states, and then use the solution (a policy) online to act (e.g. Value iteration)

Step

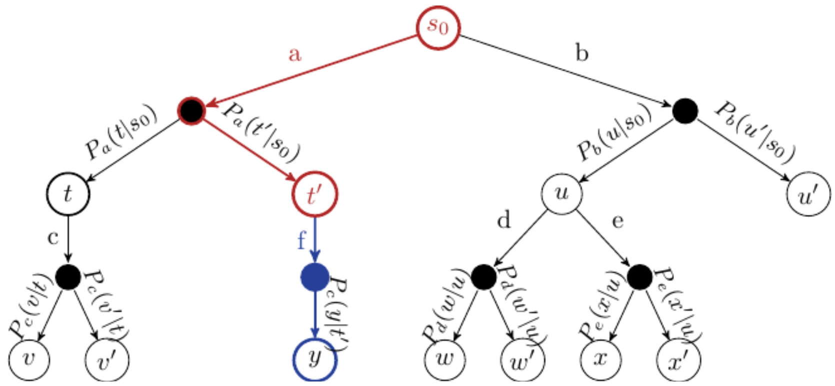
- In MCTS, states that have been evaluated are stored in a search tree
- Each node in the tree stores
    - a set of children nodes
    - pointers to parent node and parent action
    - the number of times it has been visited
- The set of evaluated states is incrementally built be iterating over four steps:
    - *Select*: select a single node in the tree that has at least one child not yet explored
    - *Expand*: Expand this node by applying one available action from the node
    - *Simulate*: From one of the outcomes of the expanded, perform a complete random simulation of the MDP to a terminating state
    - *Backpropagate*: The reward from the simulation is backpropagated from the selected node to its ancestors recursively
- Repeat these steps until some stopping condition has been reached (e.g. computational time limits)
- Select the action that maximises expected return with $\underset{a \in A(s)}{\operatorname{argmax}} Q(s, a)$, apply the action and observe new state s', then start over the MCTS with s' as root
    - If applicable, can keep the sub-tree from s' simulated in the previous search

Selection

- Select a node s to explore using some multi-armed bandit algorithm
- Upper Confidence Tree (UCT) algorithm: MCTS with UCB1 strategy to select next node to explore
- UCT selection strategy: $\underset{a \in A(s)}{\operatorname{argmax}}[Q(s,a) + 2C_p\sqrt{\frac{2\ln N(s)}{N(s,a)}}]$
  - $N(s)$: the number of times a state node s has been visited
  - $N(s,a)$: the number of times a has been selected from s
  - $C_p > 0$: exploration constant
- 

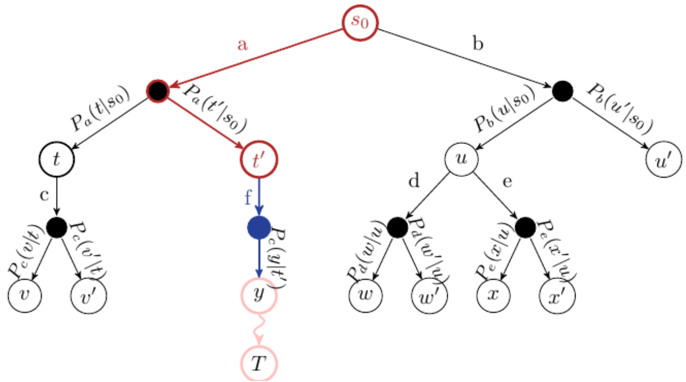
Expansion

- Select an action a to apply in state s, either randomly or using a heuristic
- Get the outcome s' of applying a in s
- Expand a new state node from s'



Simulation

- Perform a randomised simulation of the MDP until we reach a terminating state
  - Note the difference between “selection” in simulation and the selection step
- G is the cumulative discounted reward received from the simulation starting at s' until the simulation terminates



Backpropagation

- The reward from the simulation is backpropagated from the selected node to its ancestors recursively
  - For each node, reward = immediate reward for reaching the state + discounted future rewards from child state
  - Q-value for each state-action pair is the average reward from multiple iterations
- Algorithm

Procedure – $\text{Backpropagation}(s : S; a : A)$

Input: state-action pair (s, a)

Output: none

do

$N(s,a) \leftarrow N(s,a) + 1$

$G \leftarrow r + \gamma G$

$Q(s,a) \leftarrow Q(s,a) + \frac{1}{N(s,a)}[G - Q(s,a)]$

$s \leftarrow$ parent of s

$a \leftarrow$ parent action of s

while $s \neq s_0$