ECE382N.23: Embedded System Design and Modeling

Lecture 11 – RL-Based Optimization

Sources:
A. Sutton, R. Barto, "Reinforcement Learning: An Introduction", MIT Press, 2018.
http://incompleteideas.net/book/the-book.html

Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

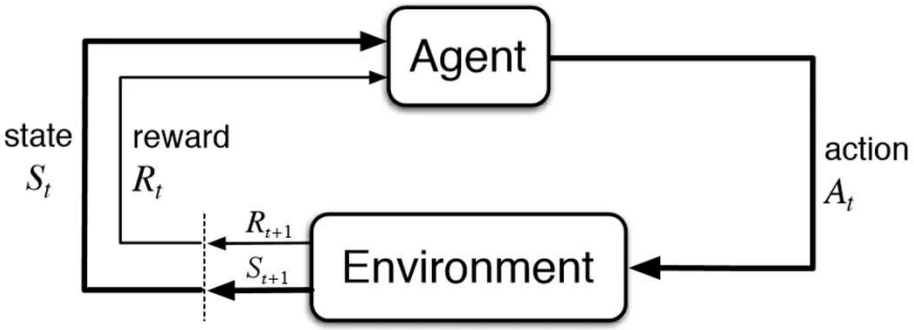
The University of Texas at Austin
Chandra Department of Electrical and Computer Engineering
Cockrell School of Engineering

Lecture 11: Outline

- **Tabular RL methods**
  - Multi-armed bandits (MAB)
  - Finite Markov decision processes (FMDP)
  - Dynamic programming
  - Monte Carlo methods
  - Temporal-difference (TD) learning
  - Q-learning
  - Monte Carlo tree search (MCTS)
- **Function approximation methods**
  - Deep reinforcement learning

Reinforcement Learning (RL)

- **Agent learns a policy**
  - Given current state S, what action A to take
  - Observe reward R and learn from past observations
  - Exploration vs. exploitation



Multi-Armed Bandits (MAB)

- **Single state**
  - Action does not affect future actions
  - Stationary or non-stationary
- **Estimated value of an action *a***

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

  - Can be computed incrementally $Q_{n+1} \doteq Q_n + \alpha\left[R_n - Q_n\right]$
    - Incremental (temporally weighted) average
- **Exploitation vs. exploration policies**
  - Greedy
  - ε-greedy
  - Upper confidence bound (UCB)

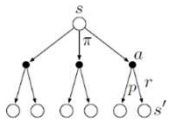
Contextual Bandits

- **Multiple n-armed bandits randomly presented**
  - Aka single n-armed bandit with different modes
  - Each mode associated with some identifying feature
- **Learn policy that distinguishes between identities**
  - Different MAB policy parameters per mode/identity
  - Associative search
- **Similar to a state in RL, but not full RL**
  - Actions do not affect mode/identity
  - Mode/identity is externally determined

Finite Markov Decision Process (FMDP)

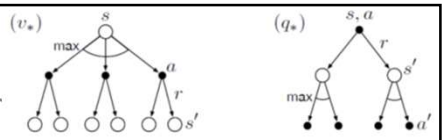
- **Actions influence subsequent states and future rewards**
  - *State* captures all history that influences future
  - FSM w/ probabilistic transitions (environment)

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$$



- **Sequence or trajectory of states, actions and rewards**
  - $S_0, A_0, R_1, S_1, A_1, R_2, \ldots$
- **Estimated value associated with actions and states**
  - Value $v_*(s)$ of each state s given optimal action selections
  - Value $q_*(s, a)$ of each action a in each state s
- **Goal**
  - Maximize expected cumulative reward (return) G_t
  - Episodic (e.g. games) vs. continuing (e.g. control) tasks
  - Discount γ for future: $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$

Dynamic Programming



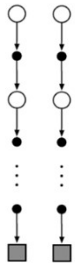
- **Recursively define optimal values (Bellman equations)**
  - Optimal action is one that maximizes value
  - Optimal value is (expected) return for optimal action

$$v_*(s) = \max_a \sum_{s',r} p(s', r\,|s, a)[r + \gamma v_*(s')] \quad \overset{\text{deterministic}}{\Rightarrow} \quad \max_a [r + \gamma v_*(s')]$$

$$q_*(s, a) = \sum_{s',r} p(s', r\,|s, a)\left[r + \gamma \max_a q_*(s', a')\right] \Rightarrow r + \gamma \max_a q_*(s', a')$$

- ➢ **Solve for optimal values and optimal policy π_***
  - Greedy policy π_*: $a(s) = \operatorname{argmax}_a [r + \gamma v_*(s')] = \operatorname{argmax}_a q_*(s, a)$
  - Can be solved iteratively
  - Policy -> value -> policy -> value -> … iteration
  - ➢ Requires environment to be known (transitions $p(s', r\,|s, a)$)
  - ➢ Computationally expensive (esp. for large state spaces)

Monte Carlo Methods



- **Sample actual or simulated trajectories**
  - Learn from actual or simulated experiences
  - Interaction with real or simulated environment (model)
  - Akin to interrelated multi-armed bandits for each state
    - Value (return) depends on whole trajectory
- **Iteratively update values and policy**
  - Sample based on exploratory policy (e.g. ε-greedy)
  - Compute state/action values as average returns
    - Can be done incrementally as in bandits
  - Update learned (target) policy based on values
  - ➢ On-policy: exploratory = target policy
  - ➢ Off-policy: exploratory ≠ target policy
    - Estimate state/action values using (weighted) importance sampling

Temporal-Difference (TD) Learning

- **Combine Monte Carlo (MC) & Dynamic Programming (DP)**
  - Learn from experience (MC): $V(S_t) \leftarrow V(S_t) + \alpha\left[G_t - V(S_t)\right]$
  - Recursive, incremental update (bootstrap) from one step:

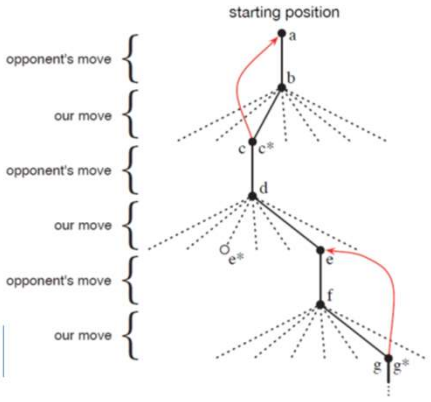
$$V(S_t) \leftarrow V(S_t) + \alpha\left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$$

- **Online learning**
  - On-policy: Sarsa

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$$

  - Off-policy: Q-learning

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$



Q-Learning

- **Q-Table**
  - Pick action derived from Q value for given state (e.g. ε-greedy)
  - Update state-action (Q) values after each step

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

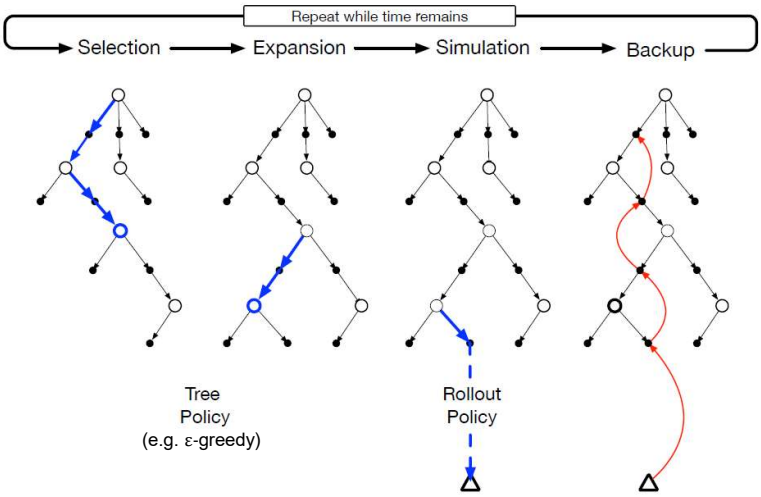
$$Q^{new}(S_t, A_t) \leftarrow (1 - \underbrace{\alpha}_{\text{learning rate}}) \cdot \underbrace{Q(S_t, A_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \Big(\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(S_{t+1}, a)}_{\text{estimate of optimal future value}} \Big)$$

$$\text{new value (temporal difference target)}$$

Source: Wikipedia

Monte Carlo Tree Search (MCTS)

- **Combine *n*-step TD w/ MC rollout from current state**
  - Tree of promising actions & states rooted at current state
  - Finally, pick best root action and repeat



Deep RL

- **Table-based approaches fail for large spaces**
  - Storage requirements
  - Very low update rates of each entry
- **Quantized state-action space**
  - Continuous -> discrete (e.g. close/far to wall)
- ➢ **Approximate table as learned function**
  - (Deep) neural network as universal function approximator
  - Deep Q-learning / RL
  - Table updates become (incremental) training
  - Or learn policy directly