Reinforcement Learning

Michal Jakob

Agent Technology Center

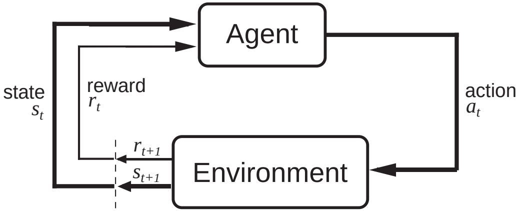
A3M33UI Spring 2010 - Lecture 11

Outline

Reinforcement Learning (RL)

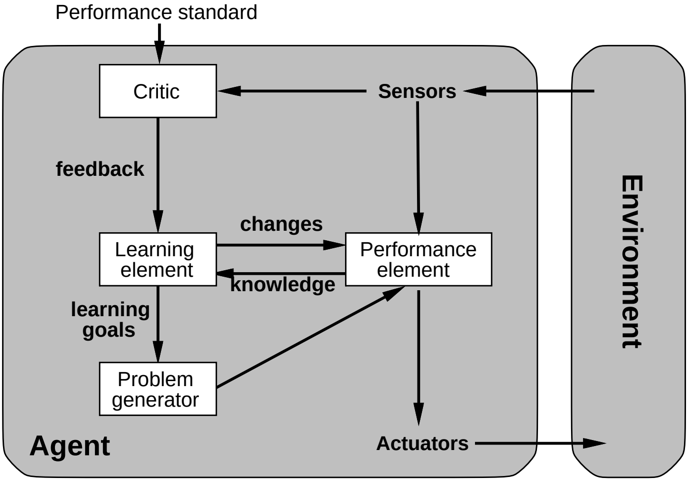
- Given an MDP model we know how to find optimal policies
  - Value Iteration or Policy Iteration
- But what if we do not have any form of model
  - Like when we were babies. . .
  - Like in many real-world applications
  - All we can do is wander around the world observing what happens, getting rewarded and punished
- $\Rightarrow$ **Reinforcement learning**

Reinforcement Learning



- Learning what to do to maximize reward
- No knowledge of the environment
  - Can only act in the world and observe states and reward
  - Try things out and see what the reward is
- Percepts received by an agent should be used not only for acting, but also for improving the agent's ability to behave optimally in the future to achieve the goal
- Extends optimal (sequential) decision making to cases where the model of the environment is unknown

Learning Agent



Examples

- Robotics: Quadruped Gait Control, Ball Acquisition (Robocup)
- Control: Helicopters
- Operations Research: Pricing, Routing, Scheduling
- Game Playing: Backgammon, Solitaire, Chess, Checkers
- Human Computer Interaction: Spoken Dialogue Systems
- Economics/Finance: Trading

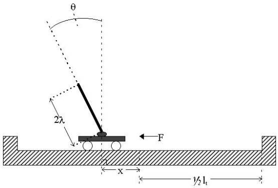


Figure: Cart-Pole balancing

RL vs Learning

- Evaluating actions vs. instructing by giving correct action
- **Evaluative feedback** – the learner is told how good an action is in terms of reward
  - Contrast with **instructive feedback** in supervised learning which gives which is the right action

RL vs MDP

- MDP
  - S is a set of states
  - A is a set of actions
  - $T(S, A, S')$ is the transition model
  - $R(S)$ is the reward function
- RL is based on MDPs but
  - Transition model is not known
  - Reward function is not known
- MDP computes an optimal policy
- RL learns an optimal policy

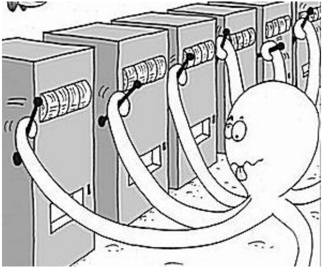
Types of RL

- Single-stage vs. sequential
  - **Single-shot**: Agent maximizes immediate feedback after a single action
  - **Sequential**: Agent maximizes aggregate feedback received over a sequence of actions
- Passive vs. active
  - **Passive**: Agent executes a fixed policy and evaluates it
  - **Active**: Agents updates policy as it learns
- Model-based vs. model-free
  - **Model-based**: Learn transition and reward model, use it to get optimal policy
  - **Model free**: Derive optimal policy without learning the model optimal policy

Outline

1. Introduction
2. **Multi-armed Bandit Problem**
3. Reinforcement Learning
   - Passive Learning
   - Model-based Active Learning
   - Model-based Active Learning
4. Conclusions

Multi-Armed Bandit Problem



- Single-stage reinforcement learning
- Choose repeatedly from n actions; each choice is called **play**
- After each play a_t, you get a **(stochastic) reward** r

$$E[r_t | a_t] = Q^*(a_t)$$

- Objective is to maximize the reward in the long term

Exploration vs. Exploitation

- To solve the multi-armed bandit problem, one must explore a variety of actions and exploit the best of them
- **Action-value estimates**: suppose by the t-th play, action a has been chosen k_a times producing rewards $r_1, r_2, \ldots, r_{k_a}$

$$Q_t(a) = \frac{r_1 + r_2 + \cdots + r_{k_a}}{k_a} \approx Q^*(a)$$

- The **greedy action** at t is

$$a_t^* = \operatorname*{argmax}_a Q_t(a)$$

  - choosing the greedy action $a_t^* \Rightarrow$ **exploitation**
  - choosing another action $a_t \neq a_t^* \Rightarrow$ **exploration**
- Must balance exploitation with exploration carefully.

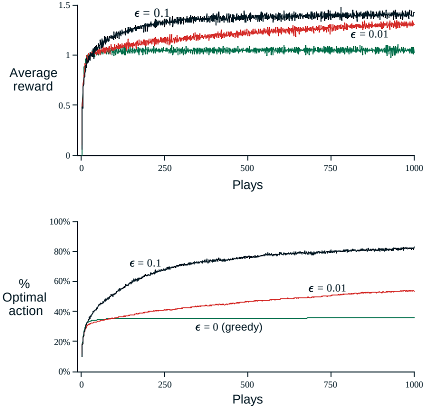
ε-greedy Action Selection

- The simplest way to balance exploration and exploitation

$$a_t = \begin{cases} a^* & \text{with probability } 1-\varepsilon \\ \text{random action } a & \text{with probability } \varepsilon \end{cases}$$

ε-greedy Action Selection Convergence

- Example for $n = 10$ and normally distributed $Q^*(a)$ and r_t



Softmax Action Selection

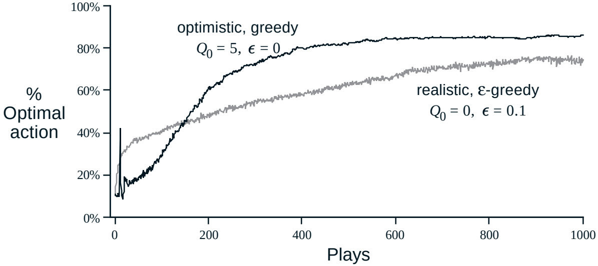
- Grade action probabilities by estimated utilities
- The most common softmax uses a Gibbs (Boltzman) distribution
- Choose action *a* on play *t* with probability

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}}$$

  - τ is the "computational temperature" (should decrease with time)

Optimistic Initial Estimates

- Previous methods depend on the initial action-value estimates $Q_0(a) \Rightarrow$ they are biased
- Instead initialize the action values optimistically
  - e.g. $Q_0(a) = 5$ for all a on the 10-armed test problem



Other Metods

- Reinforcement comparison
- Pursuit methods
- Interval estimation
- Gittins indices
- Bays optimal

Outline

Reinforcement Learning

- Sequential reinforcement learning
  - Rewards can be delayed
- **Passive learning**
  - A passive learner simply watches the world going by, and tries to learn the utility of being in various states.
  - Another perspective: a passive learner is as an agent with a fixed policy π trying to determine its benefits.
  - Serves as a component of active learning algorithms
- **Active learning**
  - Agent updates its policy as it learns
  - Agent attempts to find the optimal (or at least good) policy
  - Analogous to solving the underlying MDP

Passive Learning

- Policy π is fixed/given
  - Evaluate the policy by learning utility $U^{\pi}(s)$ of each state
- Same as policy evaluation for known transition and reward models
  - Only this time the policy is executed in the real world not simulated in agent's mind
- Several approaches
  - Direct Estimation (= LMS) - mode-free
  - Adaptive Dynamic Programming (ADP) - model-based
  - Temporal Difference Learning (TD) - model-free

Active Learning

- Agent updates its policy as it interacts with the environment
- Model-based approaches – active ADP algorithm
  - estimates the model of the environment during learning
- Model-free approaches – Q-learning
  - does not use environment model

Greedy Active ADP Agent

```
initialize U(s), T(s,a,s') and R(s) arbitrarily for all s
initialize s to the current state perceived
loop
    select a greedy optimum action a using the current R and T
    receive immediate reward r and observe the new state s'
    use the observed tuple (s,a,s',r) to update R(s') and T(s,a,s')
    (see next slide)
    calculate updated state utilities U(s) for all states
    (use any MDP algorithm)
end loop
```

Learning the Model

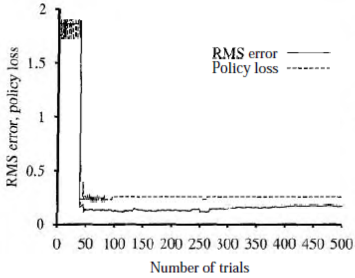
- Use simple estimation
- Learning transition model $T(s, a, s')$

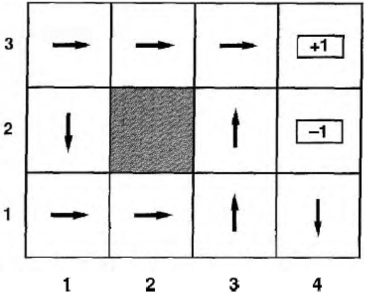
$$T(s, a, s') = \frac{N_{sas'}(s, a, s')}{N_{sa}(s, a)}$$

- Learning reward function $R(s)$ (if reward is deterministic)

$$R(s) = r(s)$$

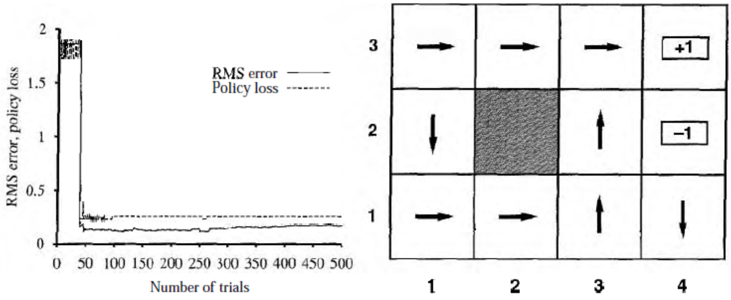
Problem - Convergence to Suboptimal Policy





- The greedy agent does not learn the true utilities of the true optimal policy!
  - Rarely converges to the optimum policy
- ⇐ Learned model is not the same as the true environment

Problem - Convergence to Suboptimal Policy



- The greedy agent does not learn the true utilities of the true optimal policy!
  - Rarely converges to the optimum policy
- $\Leftarrow$ Learned model is not the same as the true environment

Exploitation vs. Exploration

- **Exploitation**: Exploit current knowledge to maximize immediate reward.
- **Exploration**: Acquire more information to maximize long-term rewards
  - To explore requires taking actions that do not seem best according to the current model
- Managing the trade-off between exploration and exploitation is a critical issue in RL
- Basic intuition behind most approaches:
  - Explore more when knowledge is weak
  - Exploit more as we gain knowledge

Explore/Exploit Policies

- Exploration policy should be greedy in the limit of infinite exploration (GLIE)
- Agent must try each action infinite number of times
  - Rules out the chance of missing a good action
- Eventually must become greedy
- Simple GLIE
  - Choose random action $\varepsilon-$ fraction of the time
  - Use greedy policy otherwise
- Converges to the optimal policy but the convergence is very slow

Simple Exploring Active ADP Agent

initialize $U(s)$, $T(s, a, s')$ and $R(s)$ arbitrarily for all s
initialize s and r to the current state and reward observed
loop
    select action a using the explore/exploit policy on the current R and T
    receive immediate reward r and observe the new state s'
    use the tuple (s, a, s', r') to update $R(s')$ and $T(s, a, s')$
    calculate updated state utilities $U(s)$ for all states
    (use any MDP algorithm)
    $s \leftarrow s'$, $r \leftarrow r'$
end loop

Optimistic Utilities

- Smarter GLIE – give higher weights to actions not tried very often
- Modified Bellman equations with optimistic utilities $U^+(s)$

$$U^+(s) = R(s) + \gamma \max_a f\left(\sum_{s'} T(s,a,s')U^+(s'), N(s,a)\right) \quad \forall s \in S$$

  - $N(s,a)$ is the number of times a was taken in s
- The exploration function $f(u,n)$ determines how greed (preference for high values of a) is traded off against curiosity (preference for low values of n)
  - should increase with expected utility u
  - should decrease with the number of trials n
- Simple exploration function

$$f(u,n) = \begin{cases} r^+ & \text{if n} \leq N_e \\ u & \text{otherwise} \end{cases}$$

Optimistic Utilities

- Smarter GLIE – give higher weights to actions not tried very often
- Modified Bellman equations with optimistic utilities $U^+(s)$

$$U^+(s) = R(s) + \gamma \max_a f\left(\sum_{s'} T(s,a,s')U^+(s'), N(s,a)\right) \quad \forall s \in S$$

  - $N(s,a)$ is the number of times a was taken in s
- The exploration function $f(u,n)$ determines how greed (preference for high values of a) is traded off against curiosity (preference for low values of n)
  - should increase with expected utility u
  - should decrease with the number of trials n
- Simple exploration function

$$f(u,n) = \begin{cases} r^+ & \text{if n} \leq N_e \\ u & \text{otherwise} \end{cases}$$

Exploring ADP Agent with Optimistic Utilites

```
initialize U+(s), T(s, a, s') and R(s) arbitrarily for all s
initialize s and r to the current state and reward observed
loop
    choose greedy action a using the current R and T
    perform a, receive immediate reward r and observe the new state s'
    use the tuple (s, a, s', r') to update R(s') and T(s, a, s')
    calculate updated optimistic state utilities U+(s) for all states
    s ← s', r ← r'
end loop
```

- Actions towards unexplored regions are encouraged
- Fast convergence to almost optimal policy in practice

Q-Learning

- No model for learning or action selection
- Instead of learning the optimal utility function $U^*(s)$, learn the optimal action-value function $Q(a, s)$
  - utility values $U(s) = max_a Q(s, a)$
- Optimality equations for Q-values at equilibrium

$$Q^*(s, a) = R(s) + \gamma \sum_{s'} T(s, a, s') max_{a'} Q^*(s', a')$$

  - $Q(s, a)$ is the expected value of taking action a in state s and following the optimal policy thereafter
- Next action $a_{next} = \text{argmax}_a f(Q(s, a), N(s, a))$
- Converges to the optimal policy as active ADP

Temporal Difference Q-learning

- TD Q-learning does not require a model
- Iterative correction to approach the optimality equations

$$Q(s,a) \leftarrow Q(s,a) + \alpha(\underbrace{R(s) + \gamma max_{a'} Q(a',s')}_{\text{(noisy) sample of Q-value based on next state}} - Q(s,a))$$

- **Learning rate** α determines convergence to true utility
  - decrease α_s proportional to te number of state visits
  - convergence guaranteed if $\sum_{i=1}^{\infty} a_s(i) = \infty$ and $\sum_{i=1}^{\infty} a_s(i)^2 < \infty$
  - decay $\alpha_i = 1/i$ satisfies the condition

TD Q-Learning Algorithm

initialize $Q(s, a)$ arbitrarily for all a and s
initialize s and r to the current state and reward observed
loop
    select action a according to explore/exploit policy based on current $Q(s, a)$
    receive immediate reward r' and observe the new state s'
    use the tuple (s, a, s', r') to update $Q(s, a)$

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r' + \gamma max_{a'} Q(a', s') - Q(s,a))$$

    $s \leftarrow s'$, $r \leftarrow r'$
end loop

Comparison with active ADP

- Q-learning is simpler to implement since we don not need to worry about representing and learning a model
- But Q-functions can be substantially more complex than utility functions (must somehow make up for not having the model)
- Usually takes more iterations to converge
- Less efficient use of experience
- Generally does not matters for small state spaces

Outline

Conclusion

- RL is necessary for agents in unknown environments
- RL can be viewed as the ultimate level of AI
  - only minimum is given and the agent needs to learn the rest
- Single-stage decisions $\rightarrow$ multi-armed bandit problems
  - ballancing exploration with exploitation critical
- Sequential decisions $\rightarrow$ full RL
  - model-based (on-policy) – active asynchronous dynamic programming (ADP)
  - model-free – TD Q-learning
- Function approximation necessary for real/large state spaces