Reinforcement Learning

CSci 5512: Artificial Intelligence II

Outline

- Reinforcement Learning
- Passive Reinforcement Learning
- Active Reinforcement Learning
- Generalizations
- Policy Search

Reinforcement Learning (RL)

- Learning what to do to maximize reward
  - Learner is not given training
  - Only feedback is in terms of reward
  - Try things out and see what the reward is
- Different from Supervised Learning
  - Teacher gives training examples

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

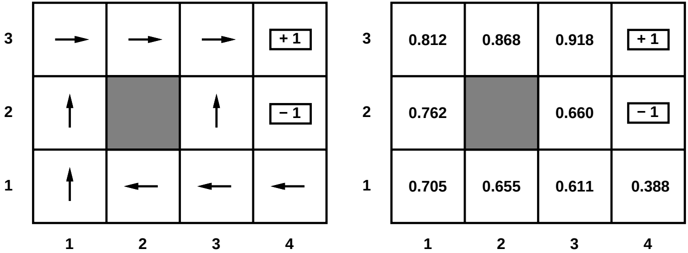
MDP Vs RL

- Markov decision process
  - Set of states S, set of actions A
  - Transition probabilities to next states $T(s, a, a')$
  - Reward functions $R(s)$
- RL is based on MDPs, but
  - Transition model is not known
  - Reward model is not known
- MDP *computes* an optimal policy
- RL *learns* an optimal policy

Types of RL

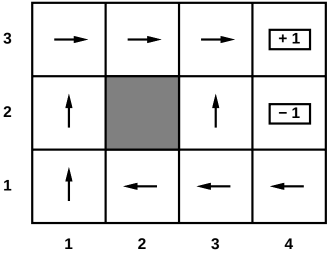
- Passive Vs Active
  - Passive: Agent executes a fixed policy and evaluates it
  - Active: Agents updates policy as it learns
- Model based Vs Model free
  - Model-based: Learn transition and reward model, use it to get optimal policy
  - Model free: Derive optimal policy without learning the model

Passive Learning



- Evaluate how good a policy π is
- Learn the utility $U^{\pi}(s)$ of each state
- Same as policy evaluation for known transition & reward models

Passive Learning (Contd.)



Agent executes a sequence of trials:

$(1,1) \to (1,2) \to (1,3) \to (1,2) \to (1,3) \to (2,3) \to (3,3) \to (4,3)_{+1}$
$(1,1) \to (1,2) \to (1,3) \to (2,3) \to (3,3) \to (3,2) \to (3,3) \to (4,3)_{+1}$

$(1,1) \to (2,1) \to (3,1) \to (3,2) \to (4,2)_{-1}$

Goal is to learn the expected utility $U^\pi(s)$

$$U^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s\right]$$

Direct Utility Estimation

- Reduction to inductive learning
  - Compute the empirical value of each state
  - Each trial gives a sample value
  - Estimate the utility based on the sample values
- Example: First trial gives
  - State (1,1): A sample of reward 0.72
  - State (1,2): Two samples of reward 0.76 and 0.84
  - State (1,3): Two samples of reward 0.80 and 0.88
- Estimate can be a running average of sample values
- Example: $U(1,1) = 0.72, U(1,2) = 0.80, U(1,3) = 0.84, \ldots$

Direct Utility Estimation (Contd.)

- Ignores a very important source of information
- The utility of states satisfy the Bellman equations

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U^{\pi}(s')$$

- Search is in a hypothesis space for U much larger than needed
- Convergence is very slow

Adaptive Dynamic Programming (ADP)

- Make use of Bellman equations to get $U^{\pi}(s)$

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U^{\pi}(s')$$

- Need to estimate $T(s, \pi(s), s')$ and $R(s)$ from trials
- Plug-in learnt transition and reward in the Bellman equations
- Solving for U^{π}: System of n linear equations

ADP (Contd.)

- Estimates of T and R keep changing
- Make use of modified policy iteration idea
  - Run few rounds of value iteration
  - Initialize value iteration from previous utilities
  - Converges fast since T and R changes are small
- ADP is a standard baseline to test ‘smarter’ ideas
- ADP is inefficient if state space is large
  - Has to solve a linear system in the size of the state space
  - Backgammon: 10^{50} linear equations in 10^{50} unknowns

Temporal Difference (TD) Learning

- Best of both worlds
  - Only update states that are directly affected
  - Approximately satisfy the Bellman equations
- Example:

$(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \rightarrow (4,3)_{+1}$
$(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \rightarrow (3,2) \rightarrow (3,3) \rightarrow (4,3)_{+1}$

$(1,1) \rightarrow (2,1) \rightarrow (3,1) \rightarrow (3,2) \rightarrow (4,2)_{-1}$

  - After the first trial, $U(1,3) = 0.84, U(2,3) = 0.92$
  - Consider the transition $(1,3) \rightarrow (2,3)$ in the second trial
  - If deterministic, then $U(1,3) = -0.04 + U(2,3)$
  - How to account for probabilistic transitions (without a model)
- TD chooses a middle ground

$$U^{\pi}(s) \leftarrow (1-\alpha)U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s'))$$

- Temporal difference (TD) equation, α is the learning rate

TD Learning (Contd.)

- The TD equation

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s))$$

- TD applies a correction to approach the Bellman equations
  - The update for s' will occur $T(s, \pi(s), s')$ fraction of the time
  - The correction happens proportional to the probabilities
  - Over trials, the correction is same as the expectation
- Learning rate α determines convergence to true utility
  - Decrease α_s proportional to the number of state visits
  - Convergence is guaranteed if

$$\sum_{m=1}^{\infty} \alpha_s(m) = \infty \qquad \sum_{m=1}^{\infty} \alpha_s^2(m) < \infty$$

  - Decay $\alpha_s(m) = \frac{1}{m}$ satisfies the condition
- TD is model free

TD Vs ADP

- TD is mode free as opposed to ADP which is model based
- TD updates observed successor rather than all successors
- The difference disappears with large number of trials
- TD is slower in convergence, but much simpler computation per observation

Active Learning

- Agent updates policy as it learns
- Goal is to learn the optimal policy
- Learning using the passive ADP agent
  - Estimate the model $R(s)$, $T(s, a, s')$ from observations
  - The optimal utility and action satisfies

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s')$$

  - Solve using value iteration or policy iteration
- Agent has "optimal" action
- Simply execute the "optimal" action

Exploitation Vs Exploration

- The passive approach gives a greedy agent
- Exactly executes the recipe for solving MDPs
- Rarely converges to the optimal utility and policy
  - The learned model is different from the true environment
- Trade-off
  - Exploitation: Maximize rewards using current estimates
  - Agent stops learning and starts executing policy
  - Exploration: Maximize long term rewards
  - Agent keeps learning by trying out new things

Exploitation Vs Exploration (Contd.)

- Pure Exploitation
  - Mostly gets stuck in bad policies
- Pure Exploration
  - Gets better models by learning
  - Small rewards due to exploration
- The multi-armed bandit setting
  - A slot machine has one lever, a one-armed bandit
  - n-armed bandit has n levers
- Which arm to pull?
  - Exploit: The one with the best pay-off so far
  - Explore: The one that has not been tried

Exploration

- Greedy in the limit of infinite exploration (GLIE)
    - Reasonable schemes for trade off
- Revisiting the greedy ADP approach
    - Agent must try each action infinitely often
    - Rules out chance of missing a good action
    - Eventually must become greedy to get rewards
- Simple GLIE
    - Choose random action $1/t$ fraction of the time
    - Use greedy policy otherwise
- Converges to the optimal policy
- Convergence is very slow

Exploration Function

- A smarter GLIE
  - Give higher weights to actions not tried very often
  - Give lower weights to low utility actions
- Alter Bellman equations using optimistic utilities $U^+(s)$

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

- The exploration function $f(u,n)$
  - Should increase with expected utility u
  - Should decrease with number of tries n
- A simple exploration function

$$f(u,n) = \begin{cases} R^+, & \text{if } n < N \\ u, & \text{otherwise} \end{cases}$$

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

Q-Learning

- Exploration function gives a active ADP agent
- A corresponding TD agent can be constructed
  - Surprisingly, the TD update can remain the same
  - Converges to the optimal policy as active ADP
  - Slower than ADP in practice
- Q-learning learns an action-value function $Q(a, s)$
  - Utility values $U(s) = \max_a Q(a, s)$
- A model-free TD method
  - No model for learning or action selection

Q-Learning (Contd.)

- Constraint equations for Q-values at equilibrium

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

- Can be updated using a model for $T(s,a,s')$
- The TD Q-learning does not require a model

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

- Calculated whenever a in s leads to s'
- The next action $a_{next} = \text{argmax}_{a'} f(Q(a',s'), N(s',a'))$
- Q-learning is slower than ADP
- Trade-off: Model-free vs knowledge-based methods

Function Approximation

- Practical problems have large state spaces
  - Example: Backgammon has $\approx 10^{50}$ states
- Hard to maintain models and statistics
  - ADP maintains $T(s, a, s'), N(a, s)$
  - TD maintains $U(s), Q(a, s)$
- Approximate with parametric utility functions

$$\hat{U}_\theta(s) = \theta_1 f_1(s) + \cdots + \theta_n f_n(s)$$

  - f_i are the basis functions
  - Significant compression in storage
- Learning the approximation leads to generalization
  - $\hat{U}_\theta(s)$ is a good approximation on visited states
  - Provides a predicted utility for unvisited states

Learning Approximations

- Trade-off: Choice of hypothesis space
  - Large hypothesis space will give better approximations
  - More training data is needed
- Direct Utility Estimation
  - Approximation is same as online supervised learning
  - Update estimates after each trial

Online Least Squares

- In grid world, for state (x, y)

$$\hat{U}_\theta(x, y) = \theta_0 + \theta_1 x + \theta_2 y$$

- The error on trial j from state s onwards is

$$E_j(s) = \frac{1}{2}(\hat{U}_\theta(s) - u_j(s))^2$$

- Gradient update to decrease error

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j(s)}{\partial \theta_i} = \theta_i + \alpha (u_j(s) - \hat{U}_\theta(s)) \frac{\partial \hat{U}_\theta}{\partial \theta_i}$$

- Widrow-Hoff rule for online least-squares

Learning Approximations (Contd.)

- Updating θ_i changes utilities for all states
- Generalizes if hypothesis space is appropriate
  - Linear functions may be appropriate for simple problems
  - Nonlinear functions using linear weights over derived features
  - Similar to kernel trick used in supervised learning
- Approximation can be applied to TD learning
  - TD-learning updates

$$\theta_i \leftarrow \theta_i + \alpha[R(s) + \gamma \hat{U}_\theta(s') - \hat{U}_\theta(s)] \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$$

  - Q-learning updates

$$\theta_i \leftarrow \theta_i + \alpha[R(s) + \gamma \max_{a'} \hat{Q}_\theta(a', s') - \hat{Q}_\theta(a, s)] \frac{\partial \hat{Q}_\theta(a, s)}{\partial \theta_i}$$

- Learning in partially observable environments
  - Variants of Expectation Maximization can be used
  - Assumes some knowledge about the structure of the problem

Policy Search

- Represent policy as a parameterized function
- Update the parameters till policy improves
- Simple representation $\pi_\theta(s) = \max_a \ \hat{Q}_\theta(a, s)$
- Adjust θ to improve policy
  - π is a non-linear function of θ
  - Gradient based updates may not be possible
- Stochastic policy representation

$$\pi_\theta(s, a) = \frac{\exp(\hat{Q}_\theta(s, a))}{\sum_{a'} \exp(\hat{Q}_\theta(s, a))}$$

  - Gives probabilities of actions
  - Close to deterministic if a is far better than others

Policy Gradient

- Updating the policy
  - Policy value $\rho(\theta)$ is the accumulated reward using π_θ
  - Update can be made using the policy gradient $\nabla_\theta \rho(\theta)$
  - Such updates converge to a local optimum in policy space
- Gradient computation in stochastic environments is hard
  - May compare $\rho(\theta)$ and $\rho(\theta + \Delta\theta)$
  - Difference will vary from trial to trial
- Simple solution: Run lots of trials and average
  - May be very time-consuming and expensive

Stochastic Policy Gradient

- Obtain an unbiased estimate of $\nabla\rho(\theta)$ by sampling
- In the "immediate" reward setting

$$\nabla_\theta\rho(\theta) = \nabla_\theta \sum_a \pi_\theta(s,a)R(a) = \sum_a (\nabla_\theta\pi_\theta(s,a))R(a)$$

- Summation approximated using N samples from $\pi_\theta(s,a)$

$$\nabla_\theta\rho(\theta) = \sum_a \pi_\theta(s,a)\frac{(\nabla_\theta\pi_\theta(s,a))R(a)}{\pi_\theta(s,a)} \approx \frac{1}{N}\sum_{j=1}^{N}\frac{(\nabla_\theta\pi_\theta(s,a_j))R(a_j)}{\pi_\theta(s,a_j)}$$

- The sequential reward case is similar

$$\nabla_\theta\rho(\theta) \approx \frac{1}{N}\sum_{j=1}^{N}\frac{(\nabla_\theta\pi_\theta(s,a_j))R_j(s)}{\pi_\theta(s,a_j)}$$

- Much faster than using multiple trials
- In practice, slower than necessary

Summary

- RL is necessary for agents in unknown environments
- Passive Learning: Evaluate a given policy
  - Direct utility estimate by supervised learning
  - ADP learns a model and solves linear system
  - TD only updates estimates to match successor state
- Active Learning: Learn an optimal policy
  - ADP using proper exploration function
  - Q-learning using model-free TD approach
- Function approximation necessary for real/large state spaces
- Policy search
  - Update policy following gradient
  - Sample-based estimate for stochastic environments