

A Survey of Reinforcement Learning

Literature

Kaelbling, Littman, and Moore

Sutton and Barto

Russell and Norvig

Presenter

Prashant J. Doshi

CS594: Optimal Decision Making

Overview

Part I

- Reinforcement Learning model
- Exploitation vs Exploration
- Learning Optimal Policies using Model-based Methods
- Learning Optimal Policies using Model-free Methods
- Computing Optimal Policies by Learning Models

Part II

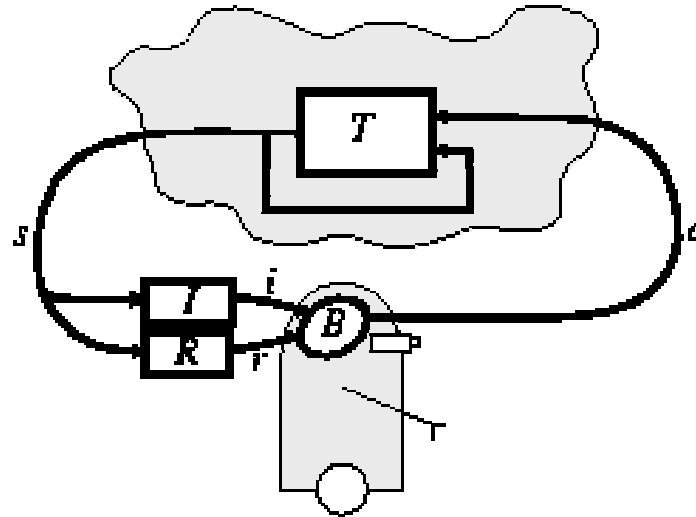
- Generalizations
- Partially Observable Environments
- Reinforcement Learning Applications

Part I: Roadmap

- Reinforcement Learning(RL) Model
 - Definition
 - Key Concepts
 - Models of Optimal Behaviour
 - Learning Performance Metrics
- Exploitation vs Exploration
 - Tradeoff involved
 - Formally Justified Exploration Techniques
 - Ad-hoc Exploration Techniques
- Delayed Reward
 - Learning Policies: Given a Model
 - Learning Policies: Model-free methods
 - Learning Policies: Model-based methods

RL Model

- Problem faced by an agent that must learn behaviour through trial -and-error interactions with a dynamic environment.
 - Class of problems opposed to a set of techniques
- Formal Definition of a RL model
 - Discrete set of environment states, S ;
 - Discrete set of environment actions, A ; and
 - Set of scalar reinforcement signals; $\{0, 1\}$



Key Concepts

- Policy: Mapping from perceived states to actions to be taken when in those states - stimulus-response rules
- Reward: Short term intrinsic desirability of the state. RL agent's sole objective is to maximize the total reward it receives in the long run
- Value: Total amount of reward an agent can expect to accumulate over the future starting from that state. Long term desirability of the state
- Backup: Updating the value of a state using values of future states
- Sweep: A sweep consists of applying a backup operation to each state

Other AI Methods vs RL

Supervised Learning	Reinforcement Learning
Presentation of input/output pairs	Agent is told immediate reward and the next state agent is not told which action is best in the long term
Learning occurs offline	Online performance is important system is evaluated while the agent is learning
No exploration of the environment	Explicit exploration of the environment is required

Search	Reinforcement Learning
Entire state space need not be enumerated	Requires entire state space to be enumerated and stored in memory

Models of Optimal Behaviour

● Finite Horizon Model

$$E\left(\sum_{t=0}^h r_t\right)$$

- Non-stationary policy π_t : *h step optimal action, (h-1) step optimal action ... 1 step optimal action*

● Infinite Horizon Discounted Model

$$E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right)$$

- $H_T = \gamma H_{T-1}$ where $H_T = E\left(\sum_{t=0}^T \gamma^t r_t\right)$
- Stationary policy π

● Average Reward Model

$$\lim_{h \rightarrow \infty} E\left(\frac{1}{h} \sum_{t=0}^h r_t\right)$$

- Gain optimal policy
- Bias optimal model

Measuring Learning Performance

- Eventual Convergence to optimal
 - Provable guarantee of asymptotic convergence to optimal behavior
 - eg. Value functions in an MDP
- Speed of convergence to optimality
 - Speed of convergence to near-optimality
 - Level of performance after a given time
- Regret
 - Difference between the expected total reward gained by following a learning algorithm and expected total reward one could gain by playing for the maximum expected reward from the start.

Exploitation vs Exploration

- A Simple RL problem: k -armed bandit problem
 - Agent is permitted h pulls $\rightarrow h$ step finite horizon
 - Immediate boolean payoff 0|1 with a probability p_i
- Exploit or explore
 - Typically premature sub-optimal decisions may affect optimal strategy
- Solutions
 - Formally Justified Techniques
 - Ad-Hoc Techniques

Formally Justified Techniques

- Dynamic Programming Approach
 - *Belief State*: $\{n_1, w_1, n_2, w_2, \dots, n_k, w_k\}$
 - Each p_i has an independent prior uniform distribution (Beta)
 - $V^*(n_1, w_1, n_2, w_2, \dots, n_k, w_k)$: expected future payoff when we act optimally
 - If $\sum_i n_i = h$, then $V^*(n_1, w_1, n_2, w_2, \dots, n_k, w_k) = 0 \dots$ (Basis)
 - $V^*(n_1, w_1, n_2, w_2, \dots, n_k, w_k) = \max_i E(\text{Future payoff of performing action } i \text{ then acting optimally for the remaining pulls})$
 $= \max_i E(\rho_i V^*(n_1, w_1, \dots, n_i + 1, w_i + 1, \dots, n_k, w_k)$
 $+ (1 - \rho_i) V^*(n_1, w_1, \dots, n_i + 1, w_i, \dots, n_k, w_k))$
 - ρ_i is the posterior payoff probability of action i paying off given n_i, w_i and our prior probability distribution \rightarrow Bayesian updating

Formally Justified Techniques (Contd)

● Gittins Allocation Indices

- Utilizes the Discounted Expected Reward Model
- Gives a table of *allocation index* values for different discount factors: $I(n_i, w_i)$
- Index value: Expected payoff of action i + value of information in selecting i
- At each step choosing action with the largest index guarantees optimal balance between exploration and exploitation
- Indexes are computed through an iterated dynamic programming approach
- Simple table lookup makes it computationally efficient

Ad-Hoc Techniques

● Greedy Strategy

- Select the action with the highest estimated payoff

true optimal action may get starved

- *Optimism in the face of uncertainty*

Assume optimistic prior beliefs; Actions are not easily eliminated from consideration

● Randomized Strategy

- p : Random action; $1-p$: Greedy action

p controls the amount of exploration

- *Boltzmann exploration*: $P(a) = \frac{e^{ER(a)/T}}{\sum_{a' \in A} e^{ER(a')/T}}$

T controls the amount of exploration

Ad-Hoc Techniques (Contd)

● Interval-based Estimation

- Tabulate for each a_i : n_i, w_i
- Compute an upper bound with $100 \cdot (1 - \alpha)\%$ confidence interval on p_i
- Select an action with the largest upper bound
- Small α encourage greater exploration

Delayed Reward

- Agent may take a long sequence of actions receiving insignificant rewards and then finally arrive at a state with a high reward
- Markov Decision Processes(MDP)
 - A set of states S
A set of actions A ,
A reward function $R : S \times A \rightarrow \mathbb{R}$
A state transition function $T : S \times A \times S \rightarrow \Pi(S)$
 - Model is markov if the state transitions are independent of any previous states or actions

Learning Policies: Given a Model

● Results

- For the infinite horizon discounted model, there exists an optimal deterministic stationary policy
- *Optimal value* of state: $V^* = \max_{\pi} E(\sum_{t=0}^{\infty} \gamma^t r_t)$
- Any policy that is greedy w.r.t. to V^* is an *Optimal policy*

● Methods to learn optimal policy

- Value Iteration
- Policy Iteration

Learning Policies: Model-free Methods

- RL is primarily concerned with the task of learning policies when the model is not known in advance. (T and R are not known)
- Agent interacts with the environment directly to obtain information
- *Temporal credit assignment problem*: When we get a large reward after a series of actions, how do we figure out which action had the most impact?
- *Temporal difference methods*:
NewEstimate = OldEstimate + LearningRate[Target - OldEstimate]

Learning Policies: Model-free Methods

● Monte-Carlo Method

- Accumulate experience over an entire episode
- Occurrence of state s in an episode is called a *visit* to s
- Std Deviation of the error is $\frac{1}{\sqrt{n}}$

● Algorithm:

$\pi \leftarrow$ policy to be evaluated $V \leftarrow$ an arbitrary state-value function

$Returns(s) \leftarrow$ an empty list, for all $s \in S$

Repeat Forever:

 Generate an episode using π

 For each state s appearing in the episode

$R \leftarrow$ return following the first occurrence of s

 Append R to $Returns(s)$

$V(s) \leftarrow average>Returns(s)$

Learning Policies: Model-free Methods

- Adaptive Heuristic Critic and TD(0)

- Adaptive version of the policy iteration method

- AHC: Using r and π it computes the expected discounted value function for each state

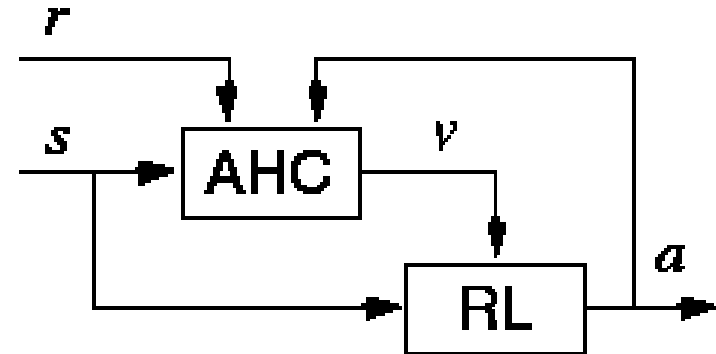
- RL: Computes π' by maximizing over v
Value Function

- *Experience tuple*: $\langle s, a, r, s' \rangle$

- Sutton's $TD(0)$ algorithm:

$$V(s) := V(s) + \alpha(r + \gamma V(s') - V(s)) \quad \text{Sample backup}$$

- $TD(\lambda)$



Learning Policies: Model-free Methods

- Q-Learning (Off-Policy TD(0))

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

- *Q-Learning Rule (1-step)*

- *Experience tuple: $\langle s, a, r, s' \rangle$*

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

- Q values are guaranteed to converge provided each action-value pair is tried out an infinite number of times

- *Exploration insensitive*

- Exploration strategy will not affect the guarantee but may affect the speed of convergence of the Q values

Learning Policies: Model-free Methods

Q-Learning Algorithm

Initialize $Q(s, a)$ arbitrarily

Repeat(for each episode):

Initialize s

Repeat(for each step of episode):

Choose a from s using policy derived from Q (eg. greedy)

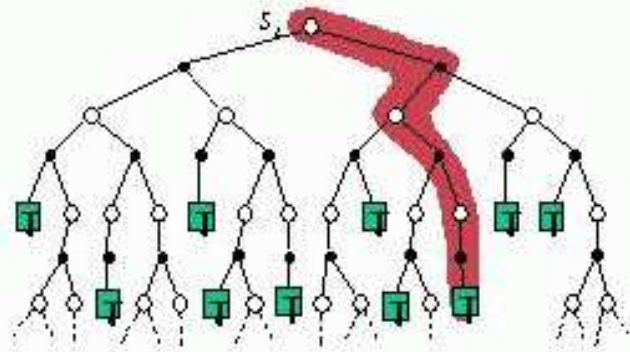
Take action a , observe r, s'

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

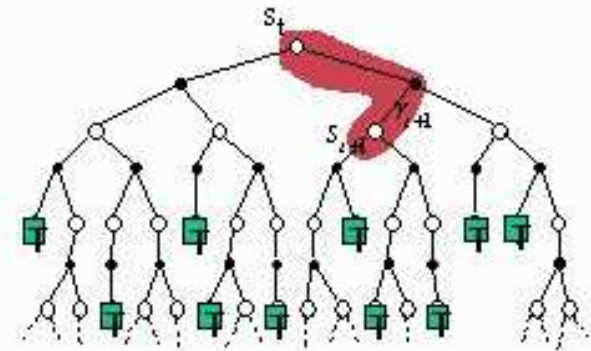
$$s \leftarrow s'$$

Until s is terminal

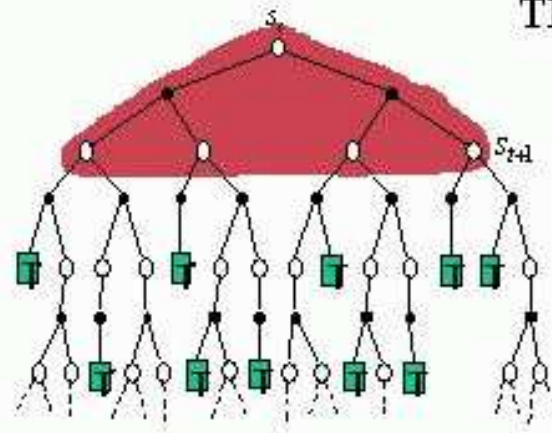
Backup Comparisons



Monte-Carlo Backup



TD Sample Backup



DP Full Backup

Learning Policies: Model-based Methods

- Disadvantage of Model-free methods
 - Require a great deal of experience; they make inefficient use of the gathered data
 - Use the experience to learn the models?
- Certainty Equivalent Methods
 - Algorithm:
 - Use the experience to statistically learn T and R
 - Use Value Iteration or Policy Iteration to learn the policy
 - Limitations:
 - Arbitrary division between the learning phase and acting phase
 - Non-stationary environments

Learning Policies: Model-based Methods

- Sutton's Dyna

- Interleaves model learning with acting
- Computationally efficient than the certainty equivalence method

- Algorithm:

- Update the model \hat{T} and \hat{R}
- Update the action-value function at s using \hat{T} and \hat{R}
$$Q(s, a) := \hat{R}(s, a) + \gamma \sum_{s' \in S} \hat{T}(s, a, s') \max_{a'} Q(s', a')$$
- Perform k additional updates at random
$$Q(s_k, a_k) := \hat{R}(s_k, a_k) + \gamma \sum_{s' \in S} \hat{T}(s_k, a_k, s') \max_{a'} Q(s', a')$$
- Choose an action a' to perform in state s' that may be greedy on Q

Learning Policies: Model-based Methods

- Dyna is relatively undirected
- Prioritized Sweeping/Queue-Dyna
 - Updates are prioritised rather than random; we update V (not Q)
 - Algorithm:
 - Select a high priority state from a queue
 - Remember the current value of the state $V_{old} = V(s)$
 - Update $V(s) := \max_a (\hat{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \hat{T}(s, a, s') V(s'))$
 - Reset the state's priority back to 0
 - Compute $\Delta = |V_{old} - V(s)|$
 - Set the priorities of the *predecessors* of s to $\Delta * \hat{T}(s, a, s')$
 - If $\Delta > threshold$, then insert the *predecessors* into a queue
 - Priority of backing up and updating depends on size of change and the current transition probabilities

Learning Policies: Model-based Methods

Performance Comparison

Problem domain: A 3277 state grid world formulated as a shortest path learning problem, which yields the same result as if a reward of 1 is given at the goal, and a reward of 0 elsewhere and a discount factor is used .

Algorithm	Steps	Backups
Q-Learning	531,000	531,000
Dyna	62,000	3,055,000
Prioritized sweeping	28,000	1,010,000

End of Part I: Recap

- RL provides us with an intuitive mechanism for learning policies
- 3 models of optimal behaviour and some measures of learning performance
- k -armed bandit problem:
 - Formally justified techniques
 - Ad-hoc techniques
- Learning Policies for Delayed Reward
 - Given a model: Value and Policy Iteration
 - Model-free: TD(0) and Q-Learning
 - Model-based: Dyna and Prioritized Sweeping
 - Q-Learning requires most steps before convergence; prioritized learning requires least steps and backups before convergence

Part II: Roadmap

- Generalization
 - Generalization over input
 - Generalization over actions
 - Hierarchical Methods
- Partially Observable Environments
- Reinforcement Learning Applications
 - Game Playing
- Discussion

Generalization

- Trivial problems
 - Enumeration of state and action spaces
 - Store state values as tables
 - Experience is handled inefficiently
- Non-trivial problems
 - Large state and action spaces(possibly continuous)
 - Possible to aggregate over state and action spaces
- Generalization:
 - How can experience with a limited subset of the state space be usefully generalized to produce a good approximation over a much larger subset?
- Generalization technique: Function approximation
 - Gradient Descent methods such as Backpropagation NN

Generalization Over Input

- Approximate V^π : Value prediction with function approximation
- Method of Approximation
 - $V \approx V_t$ - parameterized functional form with parameter vector θ_t
 $\theta_t = (\theta_t(1), \theta_t(1), \dots, \theta_t(n))^T$
eg. $V_t(s) = \theta_t^T \phi_s = \sum_{i=1}^n \theta_t(i) \phi_s(i)$ (Linear)
where $\phi_t = (\phi_s(1), \phi_s(1), \dots, \phi_s(n))^T$ is the column of features that characterize a state s
 - Task: Learn θ_t using supervised learning on a training set of $s \rightarrow v$ where $v = r + \gamma V_t(s')$ in case of TD(0)
 - Supervised learning methods seek to minimize mean squared error over some distribution of the states
$$MSE(\theta_t) = \sum_{s \in S} P(s) (V_\pi(s) - V_t(s))^2$$

Generalization Over Input

- Gradient Descent: Adjust the parameter after each example by a small amount in the direction that would reduce the error on that example
- Method
 - $\theta_{t+1} = \theta_t - \frac{1}{2}\alpha \nabla_{\theta_t} (V^\pi(s_t) - V_t(s_t))^2$
 $\theta_{t+1} = \theta_t + \alpha(V^\pi(s_t) - V_t(s_t)) \nabla_{\theta_t} V_t(s_t)$
where $\nabla_{\theta_t} V_t(s_t) = \phi_s$ (linear model)
 - But we don't know $V^\pi(s_t)$; however from the t th training example,
 $s_t \rightarrow v_t$
 $\theta_{t+1} = \theta_t + \alpha(v_t - V_t(s_t)) \nabla_{\theta_t} V_t(s_t)$
- Sample Function Approximations
 - Function computed by NN utilizing the backpropagation algorithm with θ_t as the connection weights

Generalization Over Actions

- Approximate Q^π : Action-value prediction using function approximation
- Method of Approximation
 - $Q_t \approx Q^\pi$ - parameterized functional form with parameter vector θ_t
 - Training examples are of the form $s_t, a_t \rightarrow v_t$
where $v_t = r + \gamma Q(s', a')$
 - Gradient Descent:
$$\theta_{t+1} = \theta_t + \alpha(v_t - Q_t(s_t, a_t)) \nabla_{\theta_t} Q_t(s_t, a_t)$$
- We can then combine action-value prediction with techniques for policy improvement and action selection

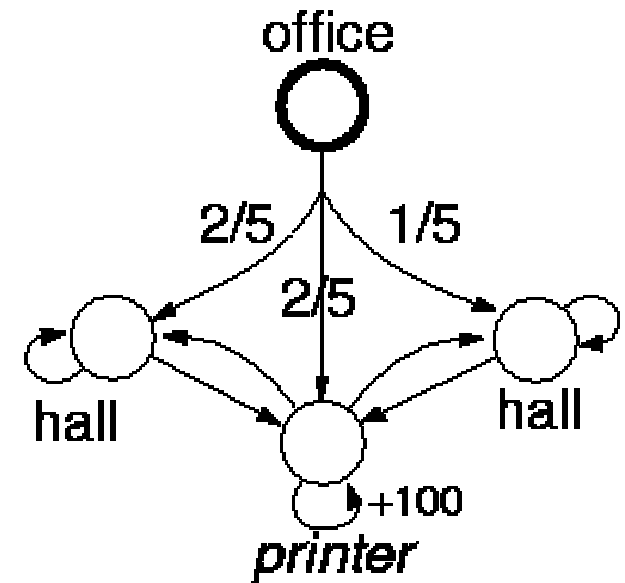
Hierarchical Methods

- Large state spaces → Hierarchy of learning problems
- Hierarchical learners as Gated behaviors
 - Collection of *behaviors* and a *gating* function that selects a particular behavior based on the state of the environment
- Feudal Q-Learner
 - High-level *Master* and low-level *Slave* modules
 - Master:
 - Receives reinforcement from the environment and sends commands to the slave
 - Learns a mapping from states to commands
 - Slave:
 - Receives reinforcement from the master for taking actions that satisfy the commands
 - Learns a mapping from state, commands to actions

Partially Observable Environments

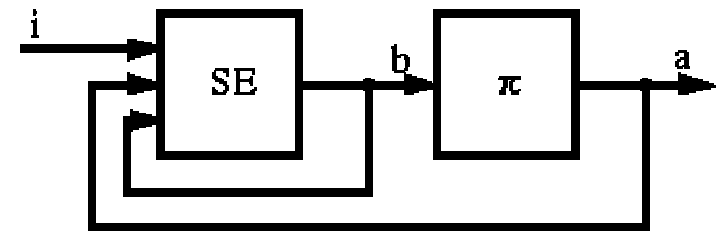
● Naive Approach

- Ignore partial observability and treat observations as if they were states of the environment
- TD(0) and Q-Learning can be applied but they lead to suboptimal behaviour



● POMDP Approach

- State Estimator: Computes a new expected belief state b
- Policy π : Compute from a piece-wise linear and convex function over the belief space



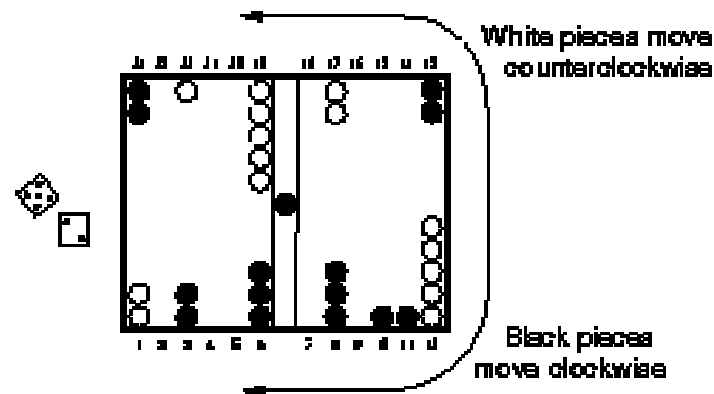
Reinforcement Learning Applications

- Game Playing: Tesauro's TD-Gammon 0.0

- Combination of TD(λ) and a non-linear function(V_t) approximation(10^{20} states).

- $V_t \rightarrow$ Estimate of the probability of victory for the current player

modeled as a multi-layer neural network trained using backpropagation algorithm(uses gradient-descent method)



- Training examples: Obtained through constant self-play and a greedy exploration strategy

- Achieved tournament-level performance

- TD-Gammon 1.0 utilized inductive biases

Discussion

- For complex problems, tabulation of values may not be enough. Inductive biases will give leverage to the learning process
 - Shaping
 - Imitation
 - Problem decomposition
 - Reflexes
- Future work: Methods for approximating, decomposing and incorporating bias into problems