Reinforcement Learning

Patricia J Riddle

Computer Science 760

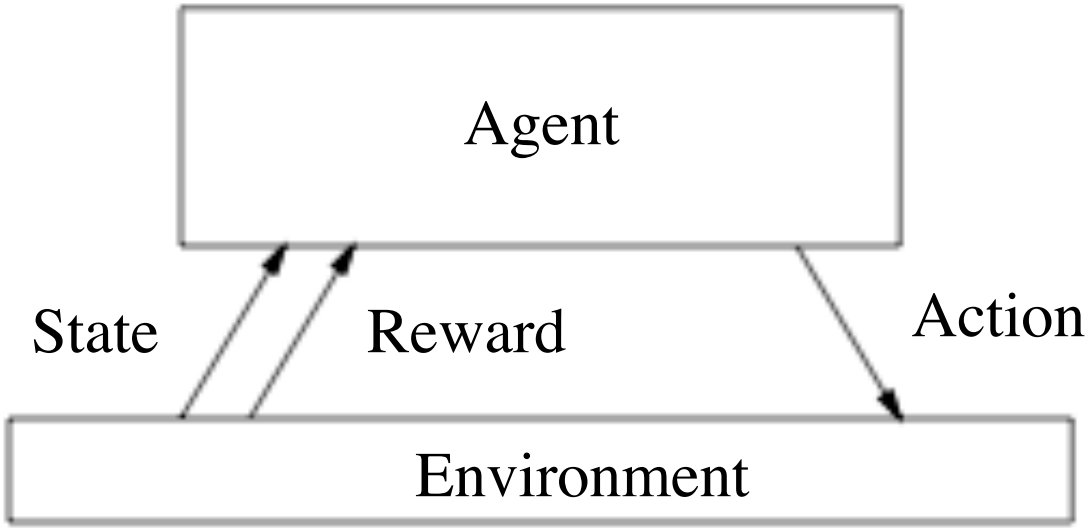
Agents

- Building a learning robot (or agent)
- Sensors observe the state of the world - camera and sonar
- A set of actions can be performed to alter the state - move forward, turn left
- Its task is to learn a control policy for choosing actions that achieve goals - docking onto a battery charger whenever its battery is low

Agent Goals

- We assume the goals of the agent can be defined by a reward function that assigns a numerical value - an immediate payoff - to each distinct action from each distinct state (a reward of +100) to state-action transitions that immediately result in a connection to the charger and 0 for all other state-action transitions

An Agent



Control Policy

- The reward function can be built into the robot or known only to an external teacher
- The task of the robot is to perform sequences of actions, observe their consequences, and learn a control policy
- The desired control policy is one that from any initial state chooses actions that maximise the reward accumulated over time by the agent

General Problem

- Learning to control sequential processes - manufacturing optimization problems where reward is goods-produced minus costs involved
- Sequential scheduling - choosing which taxis to send for passengers in a big city where reward is a function of the wait time of passengers and the total fuel costs of the taxi fleet
- Specific settings: actions are deterministic or nondeterministic, agent does or does not have prior knowledge of the effects of its actions on the environment

Reinforcement Learning Problems

- Delayed Reward
- Exploration versus Exploitation
- Partially Observable States
- Life-long Learning

Delayed Reward

- $\pi: S \rightarrow A$ that outputs an appropriate action, a, from the set A, given the current state s from the set S.

- Delayed Reward: no training example in $\langle s, \pi(s) \rangle$ form, the trainer provides only a sequence of immediate reward values as the agent executes its sequence of actions. The agent faces the problem of temporal credit assignment

Exploration versus Exploitation

- The agent influences the distribution of training examples by the action sequence it chooses. Which experimentation strategy produces most effective learning.
- The learner faces tradeoffs in choosing exploration of unknown states or exploitation of known states that it has already learned will yield high rewards

Partially Observable States

- In many practical situations sensors only provide partial information. An agent may have to consider its previous observations together with its current sensor data. The best policy may be one which chooses specifically to improve the observability of the environment.

Life-long Learning

- Agents often require that the robot learn several related tasks within the same environment. A robot might need to learn how to dock on its battery charger, how to navigate through narrow corridors, and how to pickup output from laser printers. This raises the possibility of using previously obtained experience or knowledge to reduce sample complexity when learning new tasks.

Learning Task

- Based on Markov decision processes (MDP)
- At each time step t, the agent senses a current state s_t and chooses an action a_t and performs it.
- The environment responds with a reward $r(s_t,a_t)$ and by producing the succeeding state $s_{t+1} = \delta(s_t,a_t)$

Learning Task II

- The functions r and δ are part of the environment and not necessarily known to the agent. They also only depend on the current state and action.
- We only consider finite sets S, A and deterministic functions, but these are not required.

- Learn a policy $\pi(s_t)=a_t$, with the greatest cumulative reward over time.

Reward Functions

- Discounted cumulative reward
$$V^{\pi}(s_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$
- Where r_{t+i} is generated by beginning at state s_t and repeatedly using policy π to select actions
- $0 \leq \gamma < 1$ is a constant that determines the relative value of delayed versus immediate rewards - if $\gamma = 0$ only immediate reward is considered, as γ moves closer to 1 future rewards are given more emphasis

Other Reward Functions

- Finite horizon reward

$$\sum_{i=0}^{h} r_{t+i}$$

- Average reward

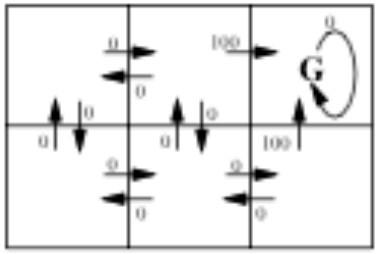
$$\lim_{h \to \infty} \frac{1}{h} \sum_{i=0}^{h} r_{t+1}$$

- We will only focus on discounted cumulative reward!

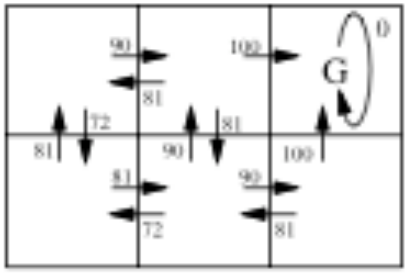
Optimal Policy

- We require the agent to learn the optimal policy,
- $\pi^* \equiv \text{argmax}_\pi V^\pi(s), (\forall s)$
- whose value function is $V^{\pi^*}(s)$ or for simplicity $V^*(s)$

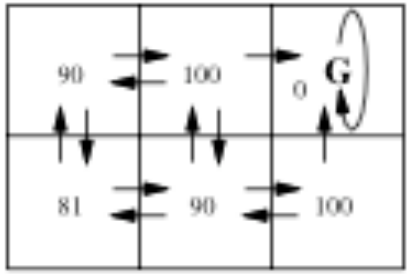
Grid-world



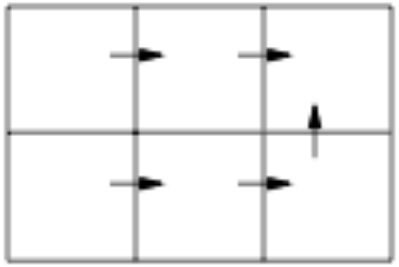
r(s,a) values
(immediate rewards)



Q(s,a) values



V*(s) values



One optimal policy

Finding Optimal Policies

- G is an absorbing state.
- If γ=0.9 an optimal policy can be defined. (any optimal policy will do)

Q Function

- Optimal action is the one that maximizes the sum r(s,a) and V^* to the immediate successor state discounted by γ
- $\pi^*(s)=\text{argmax}_a[r(s,a)+\gamma V^*(\delta(s,a))]$
- But must have perfect knowledge of reward function r and the state transition function δ!!!
- So create the Q function
- $Q(s,a)\equiv r(s,a)+\gamma V^*(\delta(s,a))$

Q Learning

- Now $\pi^*(s)=\text{argmax}_a Q(s,a)$
- Now we can select optimal actions even when we have no knowledge of r or δ
- Q value for each state-action transition equals the r value for this transition plus the V^* value for the resulting state discounted by γ

Q Learning Properties

- Still need V^* - iterative approximation or recursive definition
- $V^*(s)=\max_{a'} Q(s,a')$, so
- $Q(s,a)=r(s,a)+\gamma \max_{a'} Q(\delta(s,a),a')$
- $\hat{Q}(s,a)$, the learner's estimate of Q, is stored in a big table which is initially filled with random values or zero

Table Update

- The agent starts in some state, s, and chooses some action, a, and observes the result reward, r(s,a), and the new state, $\delta(s,a)$
- It then updates the table, $\hat{Q}(s,a) \leftarrow r+\gamma \max_{a'} \hat{Q}(s',a')$
- Doesn't need to know functions δ or r just executes the action and observes s´ and r so just sampling these functions at the current values of s and a

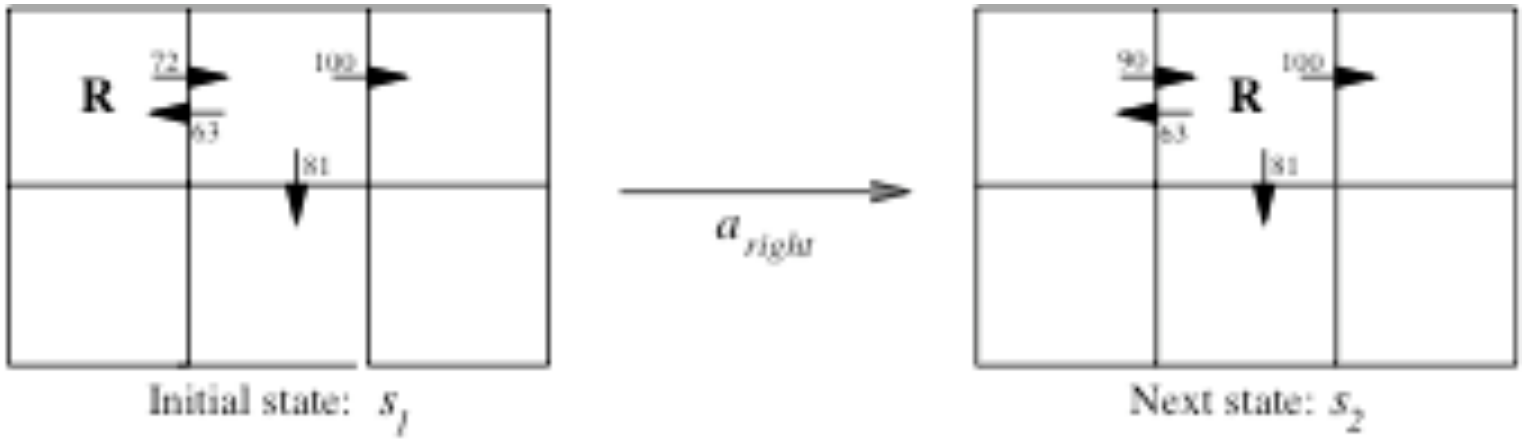
Q learning Algorithm

- For each s,a initialise the table entry Q^(s,a) to zero.
- Observe the current state s
- Do forever:
  - Select an action a and execute it
  - Receive immediate reward r
  - Observe the new state s´
  - Update the table entry for Q^(s,a) as follows:

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

  - s ← s´

Illustrative Example



$$\hat{Q}(s_1, a_{right}) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a')$$

$$\leftarrow 0 + 0.9 \max(63, 81, 100)$$

$$\leftarrow 90$$

Convergence

- $\hat{Q}$ values never decrease during training
  - $(\forall s,a,n)\hat{Q}_{n+1}(s,a) \geq \hat{Q}_{n(s,a)}$
- $\hat{Q}$ will remain in the interval between 0 and Q
  - $(\forall s,a,n)0 \leq \hat{Q}_n(s,a) \leq Q(s,a)$
- Will converge if
  1. Deterministic MDP,
  2. Immediate rewards are bounded - $|r(s,a)|<c$
  3. The agent selects actions such that it visits every state-action pair infinitely often - must execute a from s with nonzero frequency as the length of its action sequence approaches infinity

Experimentation Strategies

- Common to use probabilistic approach to selecting actions
- Actions with higher $\hat{Q}$ are assigned higher probabilities, but every action has a non-zero probability
- $P(a_i|s)$ is the probability of selecting action a_i, given the agent is in state s, where $k>0$ is the constant that determines how strongly the selection favors actions with high $\hat{Q}$ values

Probability of Selecting Action

$$P(a_i \mid s) = \frac{k^{\hat{Q}(s,a_i)}}{\sum_j k^{\hat{Q}(s,a_j)}}$$

- Sometimes k is varied with the number of iterations so the agent favors exploration during the early stages of learning, then gradually shifts toward a strategy of exploitation

Updating Sequence

- Q learning need not train on optimal action sequences to converge to the optimal policy
- After the first full episode only one entry in the table will be updated. If the agent follows the same sequence of actions the second table entry will be updates. So perform updates in reverse chronological order! Will converge in fewer iterations, although the agent has to use more memory to store the entire episode.

Retraining

- Another strategy - store past state-action transitions and immediate rewards and retrain on them periodically - This is a real win depending on relative costs (robot is very slow in comparison to replaying)
- Many more efficient techniques when the system knows the δ and r functions

Nondeterministic Rewards and Actions

- The functions $\delta(s,a)$ and $r(s,a)$ can be viewed as first producing a probability distribution over outcomes based on s and a and then drawing an outcome at random according to this distribution - nondeterministic markov decision process
- $Q(s,a) = E[r(s,a)]+\gamma\Sigma_{s'}P(s'|s,a)\max_{a'}Q(s',a')$, but is not guaranteed to converge

Decaying Weighted Average

- Decaying weighted average of the current Q^ and the revised estimate
- $\hat{Q}_n(s,a) \leftarrow (1-\alpha_n)\hat{Q}_{n-1}(s,a)+\alpha_n[r+\max_{a'} Q_{n-1}(s',a')]$, where

$$\alpha_n = \frac{1}{1 + visits_n(s,a)}$$

- Convergence long = 1.5 million games in Tesauro's backgammon program

Temporal Difference Learning

- Q learning is a special case of temporal difference learning
- Q learning can be seen as one-step lookahead, $Q^{(1)}(s_t,a_t) \equiv r_t + \gamma \max_a \hat{Q}(s_{t+1},a)$
- Two-step look ahead
- $Q^{(2)}(s_t,a_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 \max_a \hat{Q}(s_{t+2},a)$

General Formula

- N-step lookahead,
- $Q^{(n)}(s_t,a_t) \equiv r_t + \gamma r_{t+1} + \ldots + \gamma^{(n-1)} r_{t+n-1} + \gamma^n \max_a \hat{Q}(s_{t+n},a)$
- So can use a constant $0 \leq \lambda \leq 1$ to combine estimates from various lookahead distances
- $Q^{\lambda}(s_t,a_t) \equiv (1-\lambda)[Q^{(1)}(s_t,a_t) + \lambda Q^{(2)}(s_t,a_t) + \lambda^2 Q^{(3)}(s_t,a_t) + \ldots]$
- Recursive definition:
- $Q^{\lambda}(s_t,a_t) \equiv r_t + \gamma[(1-\lambda)\max_a \hat{Q}(s_t,a_t) + \lambda Q^{\lambda}(s_{t+1},a_{t+1})]$

Motivation

- The motivation is that when the agent follows an optimal policy for choosing actions if $\lambda=1$ then Q^{λ} will provide the perfect estimate of Q regardless of errors in $\hat{Q}$, if suboptimal action sequences are chosen then r_{t+i} observed far in the future can be misleading.

Generalizing from Examples

- Previous algorithms make no attempt to estimate the Q value for unseen state-action pairs, unrealistic in large or infinite spaces or when the cost of executing actions is high
- Substituted ANN for the table lookup and use each Q^(s,a) update as a training example

Multiple ANN

- A more successful alternative is to train a separate ANN for each action using state as input and Qˆ as output
- Another common alternative is to train one network with state as input and with one Qˆ output for each action
- The convergence theorems no longer hold!!

Relationship to Dynamic Programming

- Agent possesses perfect knowledge of the functions $\delta(s,a)$ and $r(s,a)$
- Focused on how to compute the optimal policy with the least computational effort, assuming the environment can be simulated
- Q learning has NO knowledge of the functions $\delta(s,a)$ and $r(s,a)$

Focus of Reinforcement Learning

- Focused on the number of real-world actions the agent must perform to converge to an acceptable policy
- In many practical domains, such as manufacturing problems, the costs in dollars and time of performing actions in the external world dominate computational costs

Summary

- Reinforcement learning - learning control strategies for autonomous agents. Training information is real-valued reward for each state-action transition. Learn action policy that maximizes total reward from any starting state.
- Reinforcement learning algorithms fit Markov decision processes where the outcome of applying an action to a state depends only on this action and state (not preceding actions or states). MDPs cover a wide range of problems - robot control, factory automation, and scheduling problems.

Summary II

- Q learning is one form of reinforcement learning where the function Q(s,a) is defined as the maximum expected, discounted, cumulative reward the agent can achieve by applying action a to state s. In Q learning no knowledge of how the actions effect the environment is required.
- Q learning is proven to converge under certain assumptions when the hypothesis Q^(s,a) is represented by a lookup table. Will converge deterministic and nondeterministic MDPs, but requires thousands of training iterations to converge in even modest problems.

Summary III

- Q learning is a member of the class of temporal difference algorithms. These algorithms learn by iteratively reducing discrepancies between estimates produced by the agent at different times.
- Reinforcement learning is closely related to dynamic programming. The key difference is that dynamic programming assumes the agent possesses knowledge of the functions $\delta(s,a)$ and $r(s,a)$ while Q learning assumes the learner lacks this knowledge.