Reinforcement Learning

What should an agent do given:

- **Prior knowledge** possible states of the world
  possible actions
- **Observations** current state of world
  immediate reward / punishment
- **Goal** act to maximize accumulated reward

Like decision-theoretic planning, except model of dynamics and model of reward not given.

Reinforcement Learning Examples

- Game - reward winning, punish losing
- Dog - reward obedience, punish destructive behavior
- Robot - reward task completion, punish dangerous behavior

Experiences

- We assume there is a sequence of experiences:

  $$state, action, reward, state, action, reward, \ldots.$$

- At any time it must decide whether to
  - explore to gain more knowledge
  - exploit the knowledge it has already discovered

Why is reinforcement learning hard?

- What actions are responsible for the reward may have occurred a long time before the reward was received.
- The long-term effect of an action of the robot depends on what it will do in the future.
- The explore-exploit dilemma: at each time should the robot be greedy or inquisitive?

Reinforcement learning: main approaches

- search through a space of policies (controllers)
- learn a model consisting of state transition function $P(s'|a,s)$ and reward function $R(s,a,s')$; solve this an an MDP.
- learn $Q^*(s,a)$, use this to guide action.

Temporal Differences

- Suppose we have a sequence of values:

  $$v_1, v_2, v_3, \ldots$$

  And want a running estimate of the average of the first k values:

  $$A_k = \frac{v_1 + \cdots + v_k}{k}$$

Temporal Differences (cont)

- When a new value v_k arrives:

$$\begin{aligned} A_k &= \frac{v_1 + \cdots + v_{k-1} + v_k}{k} \\ kA_k &= v_1 + \cdots + v_{k-1} + v_k \\ &= (k-1)A_{k-1} + v_k \\ A_k &= \frac{k-1}{k} A_{k-1} + \frac{1}{k} v_k \end{aligned}$$

Let $\alpha = \frac{1}{k}$, then

$$\begin{aligned} A_k &= (1-\alpha)A_{k-1} + \alpha v_k \\ &= A_{k-1} + \alpha(v_k - A_{k-1}) \end{aligned}$$

- Often we use this update with α fixed.

Q-learning

- Idea: store $Q[State, Action]$; update this as in asynchronous value iteration, but using experience (empirical probabilities and rewards).
- Suppose the agent has an experience $\langle s, a, r, s' \rangle$
- This provides one piece of data to update $Q[s, a]$.
- The experience $\langle s, a, r, s' \rangle$ provides the data point:

which can be used in the TD formula giving:

Q-learning

- Idea: store $Q[State, Action]$; update this as in asynchronous value iteration, but using experience (empirical probabilities and rewards).
- Suppose the agent has an experience $\langle s, a, r, s' \rangle$
- This provides one piece of data to update $Q[s, a]$.
- The experience $\langle s, a, r, s' \rangle$ provides the data point:

$$r + \gamma \max_{a'} Q[s', a']$$

which can be used in the TD formula giving:

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

Q-learning

begin

initialize $Q[S, A]$ arbitrarily

observe current state s

repeat forever:

select and carry out an action a

observe reward r and state s'

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

$s \leftarrow s'$;

end-repeat

end

Properties of Q-learning

- Q-learning converges to the optimal policy, no matter what the agent does, as long as it tries the each action in each state enough.
- But what should the agent do?
  - exploit: when in state s,
  - explore:

Properties of Q-learning

- Q-learning converges to the optimal policy, no matter what the agent does, as long as it tries the each action in each state enough.
- But what should the agent do?
  - exploit: when in state s, select the action that maximizes $Q[s, a]$
  - explore: select another action

Exploration Strategies

- The ϵ-greedy strategy: choose a random action with probability ϵ and choose a best action with probability $1-\epsilon$.
- Softmax action selection: in state s, choose action a with probability

$$\frac{e^{Q[s,a]/\tau}}{\sum_a e^{Q[s,a]/\tau}}$$

  where $\tau > 0$ is the *temperature*.
- "optimism in the face of uncertainty": initialize the Q function to values that encourage exploration.

Problems with Q-learning

- It only does one-step backup. You can use the same data to provide information to more states (even without using a model).
- It only does one backup between each experience.
    - In many domains, you can do lots of computation between experiences (e.g., if the robot has to move to get experiences).
    - You can make better use of the data by building a model, and using MDP methods to determine optimal policy.

On-policy Learning

- Q-learning does off-policy learning: it learns the value of the optimal policy, no matter what it does.
- This could be bad if the exploration policy is dangerous.
- On-policy learning learns the value of the policy being followed.

  e.g., act greedily 80% of the time and act randomly 20% of the time
- If the agent is actually going to explore, it may be better to optimize the actual policy it is going to do.

SARSA

begin

initialize $Q[S, A]$ arbitrarily

observe current state s

select action a using a policy based on Q

repeat forever:

carry out an action a

observe reward r and state s'

select action a' using a policy based on Q

$Q[s, a] \leftarrow Q[s, a] + \alpha \left(r + \gamma Q[s', a'] - Q[s, a]\right)$

$s \leftarrow s'$;

$a \leftarrow a'$;

end-repeat

end

Multi-step backups

Suppose you are considering updating $Q[s_t, a_r]$ based on "future" experiences:

$$s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, r_{t+2}, s_{t+2}, a_{t+2}, r_{t+3}, s_{t+3}, a_{t+3}, \ldots$$

- How can you use more than one-step lookahead?
- Is an off-policy or on-policy method better?
- How can we update $Q[s_t, a_t]$ by looking "backwards" at time $t + 1$, then at $t + 2$, then at $t + 3$, etc.?

Multi-step lookaheads

lookahead	Weight	Return
1 step	$1-\lambda$	$r_{t+1}+\gamma V(s_{t+1})$
2 step	$(1-\lambda)\lambda$	$r_{t+1}+\gamma r_{t+2}+\gamma^2 V(s_{t+2})$
3 step	$(1-\lambda)\lambda^2$	$r_{t+1}+\gamma r_{t+2}+\gamma^2 r_{t+3}+\gamma^3 V(s_{t+3})$
4 step	$(1-\lambda)\lambda^3$	$r_{t+1}+\gamma r_{t+2}+\gamma^2 r_{t+3}+\gamma^3 r_{t+4}+\gamma^4 V(s_t$
...	...	...
n step	$(1-\lambda)\lambda^{n-1}$	$r_{t+1}+\gamma r_{t+2}+\gamma^2 r_{t+3}+\cdots+\gamma^n V(s_{t+n})$
...	...	...
total	1	

Function Approximation

- Usually we don't want to reason in terms of states, but in terms of features.
- In the state-based methods, information about one state cannot be used by similar states.
- If there are too many parameters to learn, it takes too long.
- Idea: Express the value function as a function of the features. Most typical is a linear function of the features.

Gradient descent

To find a (local) minimum of a real-valued function $f(x)$:

- assign an arbitrary value to x
- repeat

$$x \leftarrow x - \eta \frac{df}{dx}$$

where η is the step size

Gradient descent

To find a (local) minimum of a real-valued function $f(x)$:

- assign an arbitrary value to x
- repeat

$$x \leftarrow x - \eta \frac{df}{dx}$$

where η is the step size

To find a local minimum of real-valued function $f(x_1, \ldots, x_n)$:

- assign arbitrary values to $x_1, \ldots, x_n$
- repeat:

  for each x_i

$$x_i \leftarrow x_i - \eta \frac{\partial f}{\partial x_i}$$

Linear Regression

- A linear function of variables $X_1, \ldots, X_n$ is of the form

$$f^{\overline{w}}(X_1, \ldots, X_n) = w_0 + w_1 \times X_1 + \cdots + w_n \times X_n$$

  where $\overline{w} = \langle w_0, w_1, \ldots, w_n \rangle$ is a tuple of weights. (Let $X_0 = 1$).

- Given a set E of examples, where example e has input value $X_i = e_i$ for each i and an observed value, o_e let

$$Error_E(\overline{w}) = \sum_{e \in E} (f^{\overline{w}}(e_1, \ldots, e_n) - o_e)^2$$

- Minimizing the error using gradient descent, each example should update w_i using:

SARSA with linear function approximation

- One step backup provides the examples that can be used in a linear regression.
- Suppose $F_1, \ldots, F_n$ are the features of the state and the action.
- So $Q_{\overline{w}}(s, a) = w_0 + w_1 F_1(s, a) + \cdots + w_n F_n(s, a)$
- An experience $\langle s, a, r, s', a' \rangle$ where s, a has feature values $F_1 = e_1, \ldots, F_n = e_n$, provides the "example":

  input: $Q_{\overline{w}}(s, a)$

  output: $r + \gamma Q_{\overline{w}}(s', a')$

SARSA with linear function approximation

Given γ:discount factor; η:step size
Assign weights $\overline{w} = \langle w_0, \ldots, w_n \rangle$ arbitrarily
begin
  observe current state s
  select action a
  repeat forever:
    carry out action a
    observe reward r and state s'
    select action a' (using a policy based on $Q_{\overline{w}}$)
    let $\delta = r + \gamma Q_{\overline{w}}(s', a') - Q_{\overline{w}}(s, a)$
    For $i = 0$ to n
      $w_i \leftarrow w_i + \eta \delta F_i(s, a)$
    $s \leftarrow s'$; $a \leftarrow a'$;
  end-repeat
end

Model-based Reinforcement Learning

- Model-based reinforcement learning uses the experiences in a more effective manner.
- It is used when collecting experiences is expensive (e.g., in a robot or an online game), and you can do lots of computation between each experience.
- Idea: learn the MDP and interleave acting and planning.
- After each experience, update probabilities and the reward, then do some steps of asynchronous value iteration.

Model-based learner

Data Structures: $Q[S, A]$, $T[S, A, S]$, $R[S, A]$
Assign Q arbitrarily, T = prior counts, $R = 0$
observe current state s
repeat forever:
    select and carry out action a
    observe reward r and state s'
    $T[s, a, s'] \leftarrow T[s, a, s'] + 1$
    $R[s, a] \leftarrow R[s, a] + \alpha(r - R[s, a])$
    repeat for a while
        Select state s_1, action a_1
        let $P = \sum_{s_2} T[s_1, a_1, s_2]$

$$Q[s_1, a_1] \leftarrow \sum_{s_2} \frac{T[s_1, a_1, s_2]}{P} \left(R[s_1, a_1] + \gamma \max_{a_2} Q[s_2, a_2] \right)$$

    end repeat
end-repeat