Lecture 10 - Planning under Uncertainty (III)

Jesse Hoey
School of Computer Science
University of Waterloo

July 21, 2022

Readings: Poole & Mackworth (2nd ed.)Chapter 12.1,12.3-12.9

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.

Experiences

- We assume there is a sequence of experiences:

  state, *action*, *reward*, *state*, *action*, *reward*, ...

- What should the agent do next?
- It must decide whether to:
  - explore to gain more knowledge
  - exploit the knowledge it has already discovered

Reinforcement Learning: “Bandit” problem

Each machine has a $Pr(win)$ … but you don’t know what it is…

Which machine should you play?

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)
- Model Based RL: learn a model consisting of state transition function $P(s'|a, s)$ and reward function $R(s, a, s')$; solve this as an MDP.
- Model-Free RL 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

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

"TD formula"

- 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:

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

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'

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

$s \leftarrow s'$;

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 each action in each state enough (infinitely often).
- 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. Good actions are chosen more often than bad actions; τ defines how often good actions are chosen. For $\tau \to \infty$, all actions are equiprobable. For $\tau \to 0$, only the best is chosen.

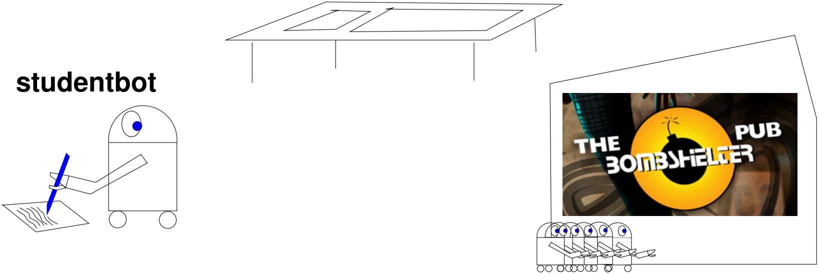
Exploration Strategies

- optimism in the face of uncertainty: initialize Q to values that encourage exploration.
- Upper Confidence Bound (UCB): Also store $N[s,a]$ (number of times that state-action pair has been tried) and use

$$\arg\max_a \left[Q(s,a) + k\sqrt{\frac{N[s]}{N[s,a]}} \right]$$

where $N[s] = \sum_a N[s,a]$

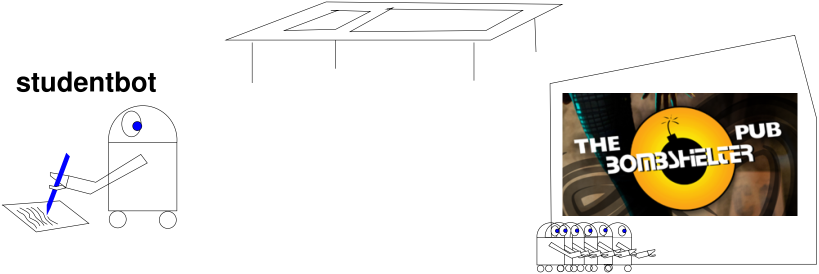
Example: studentbot



state variables:

- **tired**: studentbot is tired (no/a bit/very)
- **passtest**: studentbot passes test (no/yes)
- **knows**: studentbot's state of knowledge (nothing/a bit/a lot/everything)
- **goodtime**: studentbot has a good time (no/yes)

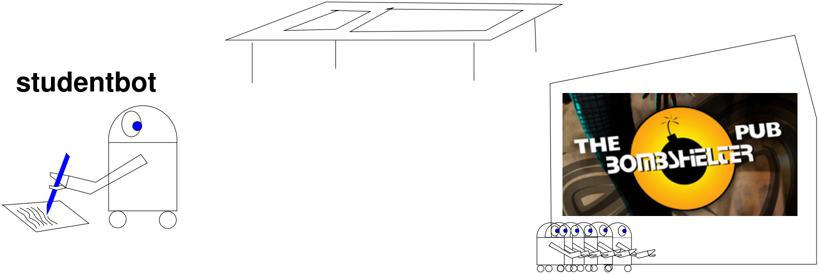
Example: studentbot



studentbot actions:

- **study**: studentbot's knowledge increases, studentbot gets tired
- **sleep**: studentbot gets less tired
- **party**: studentbot has a good time, but gets tired and loses knowledge
- **take test**: studentbot takes a test (can take test anytime)

Example: studentbot

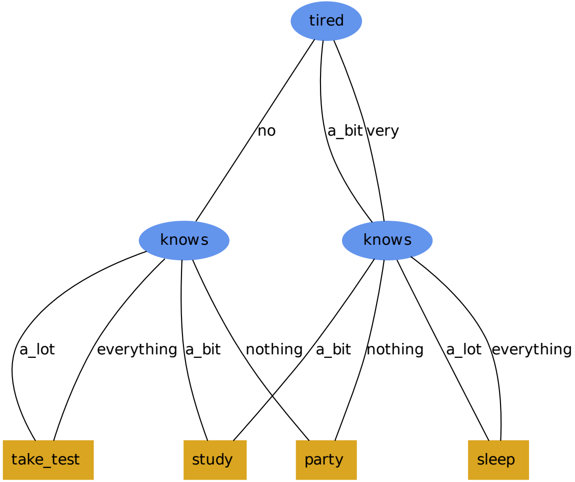


studentbot rewards:

- $+20$ if studentbot passes the test
- $+2$ if studentbot has a good time when not very tired

basic tradeoff: short term vs. long-term rewards

Studentbot Policy



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 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, R arbitrarily, T = prior counts
α is learning rate
observe current state s
repeat forever:
  select and carry out action a
  observe reward r and state s'
  $T[s, a, s'] \leftarrow T[s, a, s'] + 1$
  $R[s, a] \leftarrow \alpha \times r + (1 - \alpha) \times R[s, a]$
  repeat for a while (asynchronous VI):
    select state s_1, action a_1
    let $P = \sum_{s_2} T[s_1, a_1, s_2]$
    $$Q[s_1, a_1] \leftarrow \sum_{s_2} \frac{T[s_1, a_1, s_2]}{P} \left(R[s_1, a_1] + \gamma \max_{a_2} Q[s_2, a_2] \right)$$
  $s \leftarrow s'$

Off/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 uses the experience $\langle s, a, r, s', a' \rangle$ to update $Q[s, a]$.

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

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

$s \leftarrow s'$;

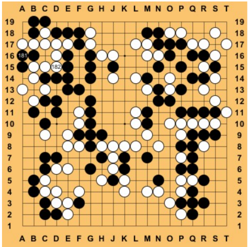
$a \leftarrow a'$;

end-repeat

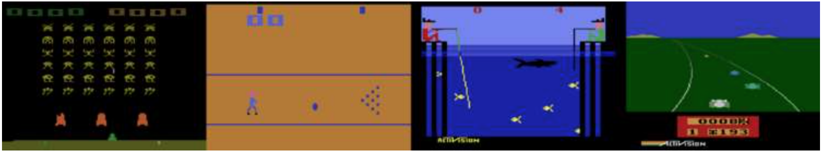
end

Large State Spaces

- Computer Go : 3^{361} *states*



- Atari Games $210 \times 160 \times 3$ dimensions (pixels)



Q-function Approximations

- Let $s = (x_1, x_2, \ldots, x_N)^T$
- Linear

$$Q_{\mathrm{w}}(s, a) \approx \sum_i w_{ai} x_i$$

- Non-linear (e.g. neural network)

$$Q_{\mathrm{w}}(s, a) \approx g(\mathrm{x}; \mathrm{w})$$

Recall: Logistic Regression

Logistic function of linear weighted inputs:

$$\hat{Y}^{\overline{w}}(e) = f(w_0 + w_1 X_1(e) + \cdots + w_n X_n(e)) = f\left(\sum_{i=0}^{n} w_i X_i(e)\right)$$

The sum of squares error is:

$$Error(E, \overline{w}) = \sum_{e \in E} \left[Y(e) - f\left(\sum_{i=0}^{n} w_i * X_i(e)\right)\right]^2$$

The partial derivative with respect to weight w_i is:

$$\frac{\partial Error(E, \overline{w})}{\partial w_i} = -2 * \delta * f'\left(\sum_i w_i * X_i(e)\right) * X_i(e)$$

where $\delta = (Y(e) - f(\sum_{i=0}^{n} w_i X_i(e)))$.
Thus, each example e updates each weight w_i by

$$w_i \quad \leftarrow \quad w_i + \eta * \delta * f'\left(\sum_i w_i * X_i(e)\right) * X_i(e)$$

Approximating the Q-function

- for experience tuple s, a, r, s' we have:
  - target Q-function: $R(s) + \gamma \max_{a'} Q_{\mathrm{w}}(s', a')$ or $R(s) + \gamma Q_{\mathrm{w}}(s', a')$
  - current Q-function: $Q_{\mathrm{w}}(s, a)$
- Squared error:

$$Err(\mathrm{w}) = \frac{1}{2}\left[Q_{\mathrm{w}}(s, a) - R(s) - \gamma \max_{a'} Q_{\mathrm{w}}(s', a')\right]^2$$

- Gradient:

$$\frac{\partial Err}{\partial \mathrm{w}} = \left[Q_{\mathrm{w}}(s, a) - R(s) - \gamma \max_{a'} Q_{\mathrm{w}}(s', a')\right] \frac{\partial Q_{\mathrm{w}(s,a)}}{\partial \mathrm{w}}$$

SARSA with linear function approximation

Given γ:discount factor; α:learning rate
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_{w})
    let $\delta = r + \gamma Q_{\mathrm{w}}(s', a') - Q_{\mathrm{w}}(s, a)$
    For $i = 0$ to n
      $w_i \leftarrow w_i + \alpha \times \delta \times \frac{\partial Q_{\mathrm{w}(s,a)}}{\partial \mathrm{w}}$
    $s \leftarrow s'$; $a \leftarrow a'$;
  end-repeat
end

Convergence

- Linear Q-learning ($Q_{\mathrm{w}}(s,a) \approx \sum_i w_{ai} x_i$) converges under same conditions as Q-learning

$$w_i \leftarrow w_i + \alpha \left[Q_{\mathrm{w}}(s,a) - R(s) - \gamma Q_{\mathrm{w}}(s',a') \right] x_i$$

- Non-linear Q-learning (e.g. neural network, $Q_{\mathrm{w}}(s,a) \approx g(\mathrm{x};\mathrm{w})$) may diverge
  - Adjusting w to increase Q at (s,a) might introduce errors at nearby state-action pairs.

Mitigating Divergence

Two tricks used in practice:

1. Experience Replay
2. Use two Q function (two networks):
   - Q network (currently being updated)
   - Target network (occasionally updated)

Experience Replay

- Idea: Store previous experiences (s, a, r, s', a') in a buffer and sample a mini-batch of previous experiences at each step to learn by Q-learning
- Breaks correlations between successive updates (more stable learning)
- Few interactions with environment needed to converge (greater data efficiency)

Target Network

- Idea: use a separate target network that is updated only periodically
- target network has weights $\overline{w}$ and computes $Q_{\overline{w}}(s, a)$
- repeat for each (s, a, r, s', a') in mini-batch:

$$w \leftarrow w + \alpha \left[Q_w(s, a) - R(s) - \gamma Q_{\overline{w}}(s', a') \right] \frac{\partial Q_w(s, a)}{\partial w}$$

- $\overline{w} \leftarrow w$

Deep Q-Network

Assign weights $\overline{w} = \langle w_0, \ldots, w_n \rangle$ at random in $[-1, 1]$

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_{w})

add (s, a, r, s', a') to experience buffer

Sample mini-batch of experiences from buffer

For each experience $(\hat{s}, \hat{a}, \hat{r}, \hat{s}', \hat{a}')$ in mini-batch:

let $\delta = \hat{r} + \gamma Q_{\overline{\mathrm{w}}}(\hat{s}', \hat{a}') - Q_{\mathrm{w}}(\hat{s}, \hat{a})$

$\mathrm{w} \leftarrow \mathrm{w} + \alpha \times \delta \times \frac{\partial Q_{\mathrm{w}}(\hat{s}, \hat{a})}{\partial \mathrm{w}}$

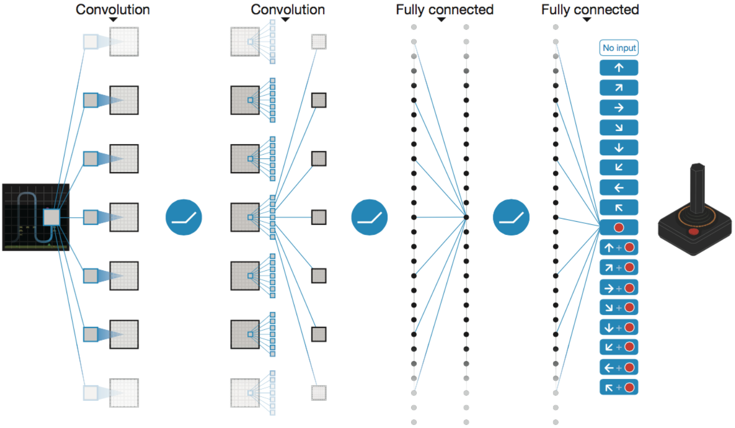
$s \leftarrow s'$; $a \leftarrow a'$;

every c steps, update target $\overline{\mathrm{w}} \leftarrow \mathrm{w}$

end-repeat

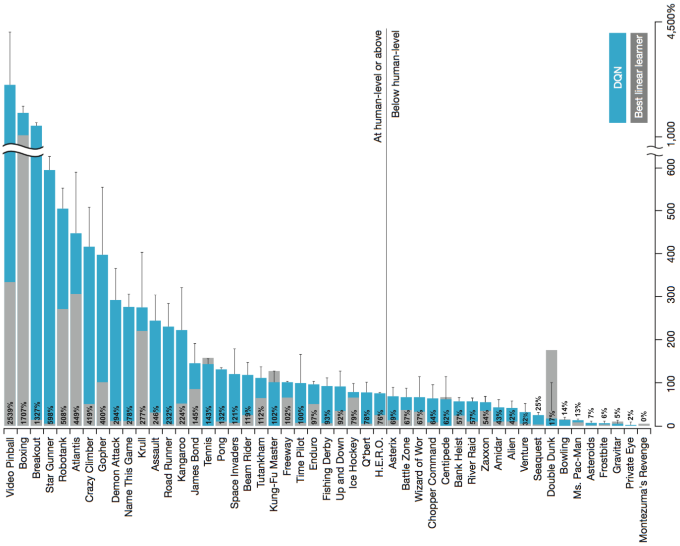
end

Deep Q-Network for Atari



from: Mnih *et. al.* Human-level control through deep reinforcement learning. *Nature* 18(7540):529–533 2015.

Deep Q-Network vs. Linear Approx.



from: Mnih *et al.*. Human-level control through deep reinforcement learning. *Nature* 18(7540):529–533 2015.

Bayesian Reinforcement Learning

- Include the parameters (transition function and observation function) in the state space
- Model-based learning though inference (belief state)
- State space is now continuous,
  belief space is a space of continuous functions
- Can mitigate complexity by modeling reachable beliefs
- optimal exploration-exploitation tradeoff.

Next:

- Recap