Passive learning

• Here we assume that the agent executes a fixed policy $\pi$
• The goal is to evaluate how good $\pi$ is, based on some sequence of trials performed by the agent – i.e., compute $U_\pi(s)$

Learning $U_\pi(s)$ does not lead to an optimal policy, why?
• the models are incomplete/inaccurate
• the agent has only tried limited actions, we cannot gain a good overall understanding of $T$
• This is why we need active learning
Goal of active learning

- Let’s first assume that we still have access to some sequence of trials performed by the agent
  - The agent is not following any specific policy
  - We can assume for now that the sequences should include a thorough exploration of the space
  - We will talk about how to get such sequences later

- The goal is to learn an optimal policy from such sequences
Active Reinforcement Learning Agents

We will describe two types of Active Reinforcement Learning agents:

- Active ADP agent
- Q-learner (based on TD algorithm)
Active ADP Agent
(Model-based)

• Using the data from its trials, the agent learns a transition model \( T \) and a reward function \( R \)
• With \( T(s,a,s') \) and \( R(s) \), it has an estimate of the underlying MDP
• It can compute the optimal policy by solving the Bellman equations using value iteration or policy iteration

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

• If we can learn an accurate MDP model of the environment, we will converge to optimal policy
Q-learning

So far, we have focused on the utilities for states

- \( U(s) \) = utility of state \( s \) = expected maximum future rewards

An alternative is to store Q-values, which are defined as:

- \( Q(a,s) \) = utility of taking action \( a \) at state \( s \)
  = expected maximum future reward if action \( a \) at state \( s \)

- Relationship between \( U(s) \) and \( Q(a,s) \)?

\[
U(s) = \max_a Q(a,s)
\]
Q-learning can be model free

• Note that after computing $U(s)$, to obtain the optimal policy, we need to compute:

$$\pi(s) = \max_a \sum_{s', a} T(s, a, s') U(s')$$

  – This requires $T$, the model of world
  – So even if we use TD learning (model free), we still need the model to get the optimal policy

• However, if you successfully estimate $Q(a, s)$ for all $a$ and $s$, we can compute the optimal policy without using the model:

$$\pi(s) = \max_a Q(a, s)$$
Q-learning

At equilibrium when the Q-values are correct, we can write the constraint equation:

\[ Q(a, s) = R(s) + \gamma \sum_{s'} T(s, a, s') U(s') \]

- **Reward at state** \( s \)
- **Expected value for action-state pair** \((a, s)\)
- **Expected value averaged over all possible states** \( s' \) that can be reached from \( s \) after executing action \( a \)
Q-learning

At equilibrium when the Q-values are correct, we can write the constraint equation:

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

- **Reward at state s**
- **Best expected value for action-state pair (a, s)**
- **Best value averaged over all possible states s’ that can be reached from s after executing action a**
- **Best value at the next state = max over all actions in state s’**
Q-learning Without a Model

- We can use a temporal differencing approach which is model-free
- After moving from state $s$ to state $s'$ using action $a$:

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

- New estimate of $Q(a, s)$
- Learning rate $0 < \alpha < 1$
- Old estimate of $Q(a, s)$
- Difference between old estimate $Q(a, s)$ and the new noisy sample after taking action $a$
Q-learning: Estimating the Policy

Q-Update: After moving from state $s$ to state $s'$ using action $a$:

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

Note that $T(s,a,s')$ does not appear anywhere!

Further, once we converge, the optimal policy can be computed without $T$.

This is a completely model-free learning algorithm.
Q-learning Convergence

- Guaranteed to converge to the true Q values given enough exploration
- Very general procedure (because it’s model free)
- Converges slower than ADP agent (because it is completely model free and it doesn’t enforce consistency among values through the model)
• So far, we have assumed that all training sequences are given and they fully explore the state space and action space
• But how do we generate all the training trials?
  – We can have the agents random explore first, to collect training trials
  – Once we accumulate enough trials, we perform the learning (either ADP, or Q-learning)
  – We then choose the optimal policy
• How much exploration do we need to do?
• What if the agent is expected to learn and perform reasonably constantly, not just at the end
A greedy agent

• At any point, the agent has a current set of training trials, and we’ve got a policy that is “optimal” based on our current understanding of the world

• A greedy agent can execute the optimal policy for the learned model at each time step
A greedy Q-learning agent

**function** Q-learning-agent(\textit{percept}) **returns** an action

**inputs:** \textit{percept}, a percept indicating the current state \( s' \) and reward signal \( r' \)

**static:** \( Q \), a table of action values index by state and action

\( N_{sa} \) a table of frequencies for state-action pairs, initially zero

\( s, a, r \) the previous state and action, initially null

\textbf{if} \( s \) is not null, \textbf{then do}

increment \( N_{sa}[s,a] \)

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

\textbf{if} TERMINAL?\[s'\] \textbf{then do} \( s, a \leftarrow \text{null} \)

\textbf{else} \( s, a, r \leftarrow s', \arg \max_{a} Q(s',a), r' \)

**return** \( a \)

Always choose the action that is deemed the best based on current Q table
The agent finds the lower route to get to the goal state but never finds the optimal upper route. The agent is stubborn and doesn’t change so it doesn’t learn the true utilities or the true optimal policy.
What happened?

• How can choosing an optimal action lead to suboptimal results?
• What we have learned (T/R, or Q) may not truly reflect the true environment
• In fact, the set of trials observed by the agent was often insufficient

How can we address this issue?
   We need good training experience ...
Exploitation vs Exploration

• Actions are always taken for one of the two following purposes:
  – **Exploitation**: Execute the current optimal policy to get high payoff
  – **Exploration**: Try new sequences of (possibly random) actions to improve the agent’s knowledge of the environment even though current model doesn’t believe they have high payoff

• Pure exploitation: gets stuck in a rut
• Pure exploration: not much use if you don’t put that knowledge into practice
Optimal Exploration Strategy?

• What is the optimal exploration strategy?
  – Greedy?
  – Random?
  – Mixed? (Sometimes use greedy sometimes use random)

• It turns out that the optimal exploration strategy has been studied in-depth in the N-armed bandit problem
N-armed Bandits

• We have N slot machines, each can yield $1 with some probability (different for each machine)

• What order should we try the machines?
  – Stay with the machine with the highest observed probability so far?
  – Random?
  – Something else?

• Bottom line:
  – It’s not obvious
  – In fact, an exact solution is usually intractable
GLIE

• Fortunately it is possible to come up with a **reasonable** exploration method that eventually leads to optimal behavior by the agent

• Any such exploration method needs to be **Greedy** in the **Limit of Infinite Exploration (GLIE)**

• Properties:
  – Must try each action in each state an unbounded number of times so that it doesn’t miss any optimal actions
  – Must eventually become greedy
Examples of GLIE schemes

• ε-greedy:
  – Choose optimal action with probability \((1-ε)\)
  – Choose a random action with probability \(ε/(\text{number of actions}-1)\)

• Active ε-greedy agent
  1. Start from the original sequence of trials
  2. Compute the optimal policy under the current understanding of the world
  3. Take action use the ε-greedy exploitation-exploration strategy
  4. Update learning, go to 2
Another approach

• Favor actions the agent has not tried very often, avoid actions believed to be of low utility (based on past experience)

• We can achieve this using an *exploration function*
An exploratory Q-learning agent

**function** Q-learning-agent(*percept*) **returns** an action

**inputs:** *percept*, a percept indicating the current state $s'$ and reward signal $r'$

**static:** $Q$, a table of action values index by state and action

$N_{sa}$ a table of frequencies for state-action pairs, initially zero

$s, a, r$ the previous state and action, initially null

if $s$ is not null, then do

increment $N_{sa}[s,a]$

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

if TERMINAL?[s'] then $s, a \leftarrow$ null

else $s, a, r \leftarrow s', \arg \max_{a'} f(Q(a',s'), N_{sa}[s',a']), r'$

return a

Exploration function:

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

Exploration function $f(u,n)$:

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

- Trades off greedy (preference for high utilities $q$) against curiosity (preference for low values of $n$ – the number of times a state-action pair has been tried)
- $R^+$ is an optimistic estimate of the best possible reward obtainable in any state with any action
- If $a$ hasn’t been tried enough in $s$, you assume it will somehow lead to gold – optimistic
- $N_e$ is a limit on the number of tries for a state-action pair
Model-based/Model-free

- Two broad categories of reinforcement learning algorithms:
  1. Model-based eg. ADP
  2. Model-free eg. TD, Q-learning
- Which is better?
  - Model-based approach is a knowledge-based approach (ie. model represents known aspects of the environment)
  - Book claims that as environment becomes more complex, a knowledge-based approach is better
What You Should Know

• Exploration vs exploitation
• GLIE schemes
• Difference between model-free and model-based methods
• Q-learning