CS 380: Artificial Intelligence

Lecture 13: Reinforcement Learning

Machine Learning

- Computational methods for computers to exhibit specific forms of learning. For example:
  - Learning from Examples:
    - Supervised learning
    - Unsupervised learning
  - Reinforcement Learning
  - Learning from Observation (demonstration/imitation)

Examples

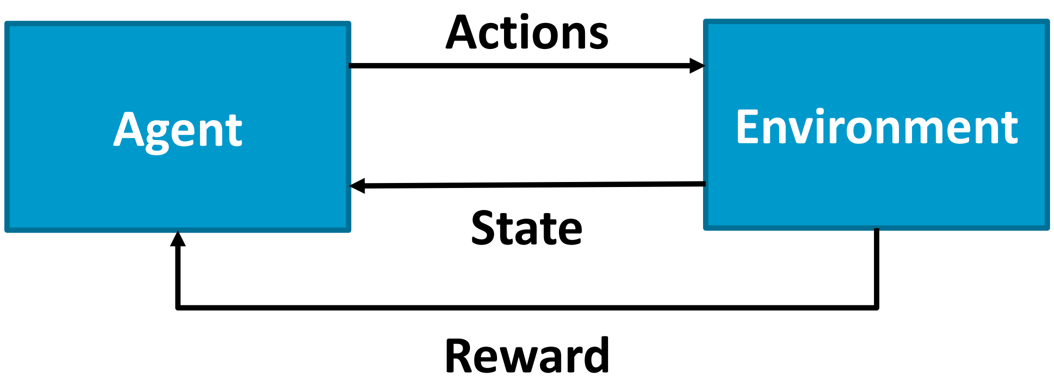
- **Reinforcement Learning**: learning to walk

Examples

- **Reinforcement Learning**:
  - https://www.youtube.com/watch?v=hx_bgoTF7bs
  - https://www.youtube.com/watch?v=e27TUmMkOA0
  - https://www.youtube.com/watch?v=0JL04JJjocc

Reinforcement Learning

- How can an agent learn to take actions in an environment to maximize some notion of reward



- Assumption: Environment is **unknown** and maybe **stochastic**

Basic Concepts

- State (S):
  - The configuration of the environment, as perceived by the agent
- Actions (A):
  - The set of different actions the agent can perform
  - We assume for now that it's discrete (but this doesn't need to be true for other RL algorithms)
- Reward (R):
  - Each time the agent performs an action, it receives a reward
  - Real-valued

Policies and Plans

- Plan:
  - Sequence of actions generated to achieve a certain goal from a given starting state
- Policy:
  - A mapping of states to actions
  - i.e.: a function that defines which action to perform in every possible state
- For RL, given an initial state, does the agent need to learn a plan or a policy? (environment is stochastic)

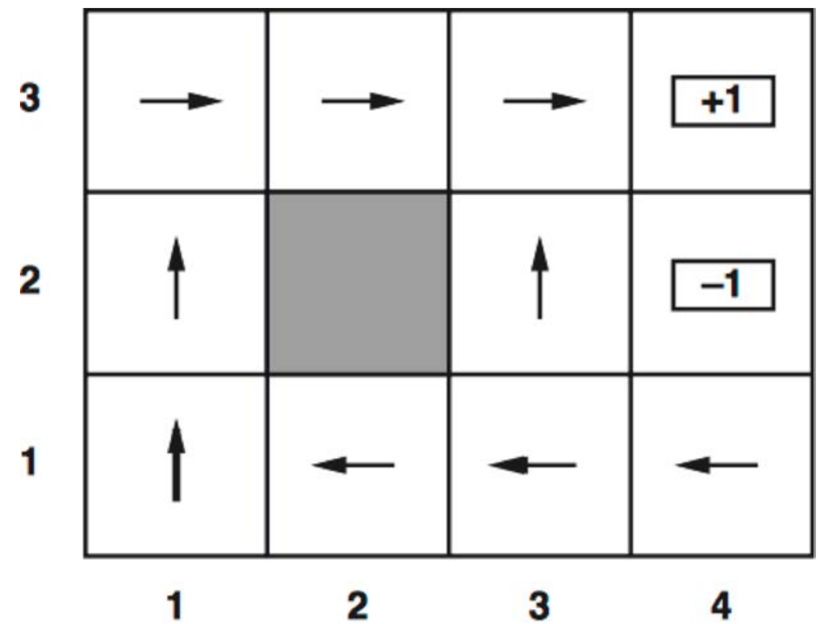
Policies and Plans

- Plan:
  - Sequence of actions generated to achieve a certain goal from a given starting state
- Policy:
  - A mapping of states to actions
  - i.e.: a function that defines which action to perform in every possible state
- For RL, given an initial state, does the agent need to learn a plan or a policy? (environment is stochastic)
  - A **Policy**, since plans assume deterministic execution

Policies

- RL algorithms learn **Policies**
- How do we represent a policy?
  - Example: as a table (if it's a deterministic policy)

State	Action
s_0	right
s_1	right
s_2	up
…	…
s_n	left

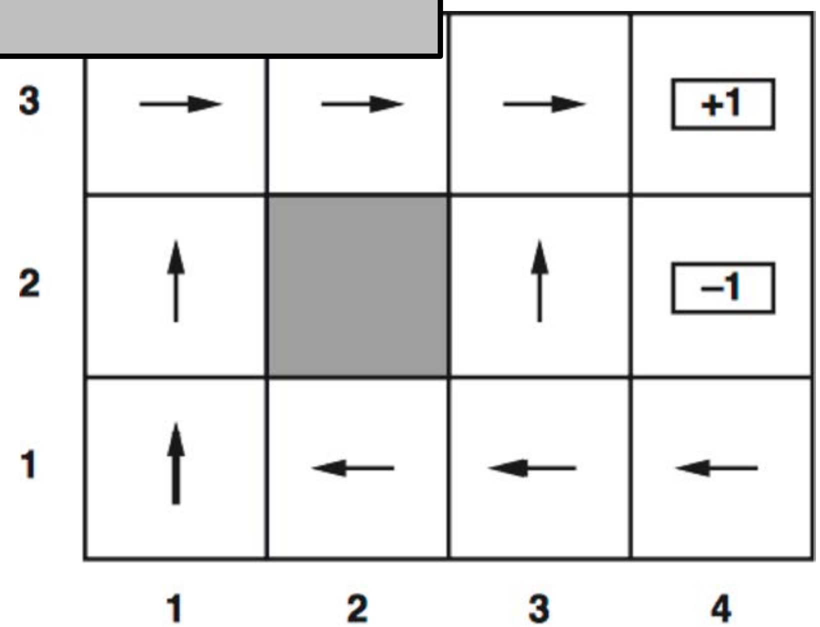


Policies

- RL algorithms learn **Policies**
- How do we re
  - Example: as   policy)

A **stochastic** policy would specify the probability of each action in each state.

State	Action
s_0	right
s_1	right
s_2	up
…	…
s_n	left



Value Function

- Given a policy *P*, the **Value** of a state *S* using policy *P* is the expected reward we would get if we execute *P* starting from *S*:

$$V^P(S) = \mathbf{E}\left[\sum_{t=0\ldots\infty} R(S_t, P(S_t))|S_0 = S\right]$$

  – This is the expected sum of all rewards in the future ($t = 0 \ldots \infty$) given that the start state is S ($S_0 = S$)

Value Function

- However, calculating infinite time is hard ☺
- Also, we might not care as much what happens really long into the future
- So we assume a **discount factor** γ (between 0 and 1) that discounts future rewards:

$$V^P(S) = \mathbf{E}\left[\sum_{t=0\ldots\infty} \gamma^t R(S_t, P(S_t))|S_0 = S\right]$$

  – (Note how γ^t shrinks to zero over time)

State-Action Value Function (Q)

- Given a policy *P*, the **Q value** of a state *S* and an action *A* is the expected reward we would get if we execute *A* and then follow policy *P* starting from *S*:

$$Q^P(S, A) = \mathbf{E}\left[\sum_{t=0\ldots\infty} \gamma^t R(S_t, P(S_t)) | S_0 = S, A_0 = A\right]$$

Q table

- A **Q table** is a matrix with one row per state, and one column per action with the Q value of each state, action pair

State	right	…	up
s_0	0.4	…	0.1
s_1	0.5	…	0.1
s_2	0.3	…	0.05
…	…	…	…
s_n	0.1	…	0.8

Q table

- A **Q table** is a mat te, and
  one column per a f each
  state, action pair

A Q table defines a deterministic policy as taking the action with the maximum Q value in each state.

State	right	…	up
s_0	0.4	…	0.1
s_1	0.5	…	0.1
s_2	0.3	…	0.05
…	…	…	…
s_n	0.1	…	0.8

State	Action
s_0	right
s_1	right
s_2	right
…	…
s_n	up

Q Learning

- Q Learning is a fundamental but powerful reinforcement learning algorithm
  - Goal: to learns the Q table for a given domain
  - Starts with an initial (e.g., all zeroes) Q table
  - Updates the Q table iteratively over time using **Bellman Equations**

Bellman Equations

- Imagine that:
  - We have a current estimate of the Q table
  - An agent is in state *S*
  - It performs action *A* (which takes it to state *S'*)
  - And observes reward *R*
- How do we update the Q table with this new piece of information?

$$Q^{new}(S,A) = (1-\alpha)Q(S,A) + \alpha\left[R + \gamma max_{A'} Q(S',A')\right]$$

Bellman Equations

- Imagine that:
  - We have a current estimate of the Q table
  - An agent is in state *S*
  - It performs action *A* (which takes it to state *S'*)
  - Ar … *R*
- How … e Q table with t…
  of in…

Previous Q value estimate

New Q value estimate

$$Q^{new}(S,A) = (1-\alpha)\boxed{Q(S,A)} + \alpha\,\boxed{[R + \gamma max_{A'} Q(S',A')]}$$

Learning rate

Discount factor

α is a value from 0 to 1
High α → fast learning
Low α → slow learning

Q Learning

1. Initialize Q table to some uniform value (e.g., 0)
2. S = initial state
3. A = choose action based on Q table and current state S
4. Execute action A:
   - S' = new state after executing A
   - R = observed reward
5. Update Q table:

$$Q^{new}(S,A) = (1-\alpha)Q(S,A) + \alpha\left[R + \gamma max_{A'} Q(S',A')\right]$$

6. Go to Step 3

Q Learning

1. Initialize Q table to some uniform value (e.g., 0)
2. S = initial state
3. A = choose action based on Q table and current state S
4. Execute action A:
   – S’ = new state after executing A
   – R = observed reward
5. Update Q table:

$$Q^{new}(S,A) = (1-\alpha)Q(S,A) + \alpha\left[R + \gamma max_{A'} Q(S',A')\right]$$

6. Go to Step 3

How do we choose an action?

Exploration vs Exploitation

- During learning, the agent is in a given state S, and has to choose an action using the Q table:

State	right	left	forward
s_0	0.4	0.9	0.1
s_1	**0.5**	0.3	0.1
s_2	0.3	0.1	0.05
…	…	…	…
s_n	0.1	0.3	0.8

Current state

Action that maximizes Q value

Exploration vs. Exploitation

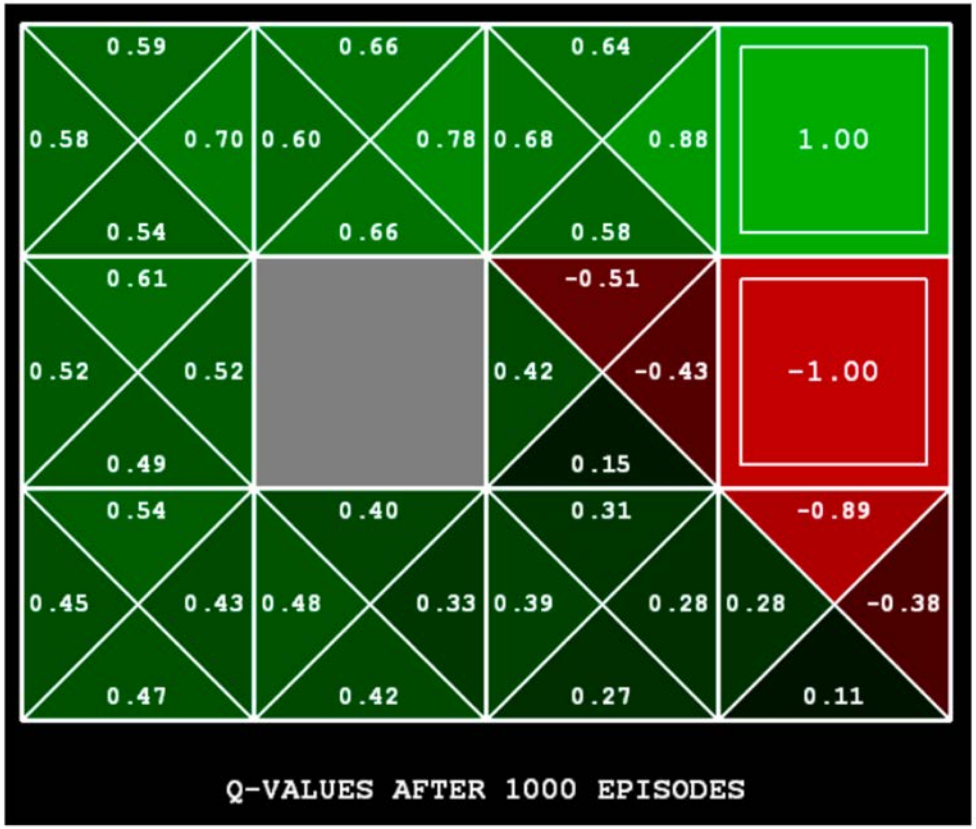
- During learning, instead of choosing an action that maximizes Q value, we can use a policy to balance exploration and exploitation
  - Remember MCTS? (Monte Carlo tree search)
  - Same idea here!
- For example: ϵ-greedy
  - $\epsilon = 0.1$ (or some small value between 0 and 1)
  - With probability ϵ, choose an action at random
  - With probability $(1 - \epsilon)$, choose action with maximum Q
- Why?

Exploration vs. Exploitation

- During learning, instead of choosing an action that maximizes Q value, we can use a policy to balance exploration and exploitation
  - Remember MCTS? (Monte Carlo tree search)
  - Same idea here!
- For example: ϵ-greedy
  - $\epsilon = 0.1$ (or some small value between 0 and 1)
  - With probability ϵ, choose an action at random
  - With probability $(1 - \epsilon)$, choose action with maximum Q
- Why?
  - The action currently believed to be the best might just happen to be by coincidence. So, we need to keep exploring just in case other actions turn out to be better.

Q Learning

- Example output of Q Learning (Q table):



(from Hal Daumé's CS421 slides)

Problems with Q Learning

- Biggest problem: Lack of generalization
  - If two states are very similar, Q learning does not exploit this — it learns the Q values for each state independently
  - There are techniques to address this:
    - Function approximation
    - Feature-based state representation
    - Deep Q-learning: uses a neural network to represent the Q table (implicit generalization)

Examples (Again)

- **Reinforcement Learning**:
  - https://www.youtube.com/watch?v=hx_bgoTF7bs
  - https://www.youtube.com/watch?v=e27TUmMkOA0
  - https://www.youtube.com/watch?v=0JL04JJjocc

Machine Learning

- Computational methods for computers to exhibit specific forms of learning. For example:
  - Learning from Examples:
    - Supervised learning
    - Unsupervised learning
  - Reinforcement Learning
  - Learning from Observation (demonstration/imitation)