ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

08
Chapiter

Reinforcement Learning

2I1AE1: Artificial Intelligence

Régis Clouard, ENSICAEN - GREYC

"If people do not believe that mathematics is simple,
it is only because they do not realize how complicated life is."
John Von Neumann, 1947

In this chapter

- Learning by reinforcement
  - In which we examine how an agent can learn from success and failure, from reward and punishment.
- Plan
  - Learning agent
  - Reinforcement Learning
    - Q-Learning algorithm
    - Dilemma Exploration / Exploitation
    - Utility Function Approximation

1. Learning Agent

- An agent **learns** if its performance at a task improves with experience.
- Why programming agent that learns ?
  - An agent can be in an unknown or complex environment (it has to discover it).
  - Even if the environment is known beforehand, it can change overtime in unpredictable ways.
  - Sometime, we have no idea how to program the performance function for an agent.

Types of Machine Learning

Supervised Learning

Task Driven

Learning from labeled data "training examples".

- Classification: qualitative
- Regression: quantitative

Unsupervised Learning

Data Driven

Learning from unlabeled data looking for patterns and structure.

- Clustering
- Anomaly detection

Reinforcement Learning

Reward Driven

Learning by interaction with the environment and outcomes.

- Decision Making

2. Reinforcement Learning

- What distinguishes reinforcement learning from other automatic learning paradigms:
  - There is no supervisor who indicates the right solution, but rather a reward signal.
  - The reward can be delayed; it is not necessarily instantaneous.
  - The actions of the agent influence the future data the agent will receive.

Reinforcement Learning Agent


```
function INTELLIGENT-AGENT(percept, goal) returns an action
   static: state, the agent's memory of the world state
   state ← UPDATE-STATE-FROM-PERCEPTS(state, percept)
   IF previous_action != None
        LEARN_FROM_TRANSITION(previous_state, previous_action,
                              state, reward)
   action ← CHOOSE-BEST-ACTION(state)
   state ← UPDATE-POLICY-STATE-FROM-ACTION(state, action)
   previous_action ← action
   previous_state ← state
   return action
```

General Features

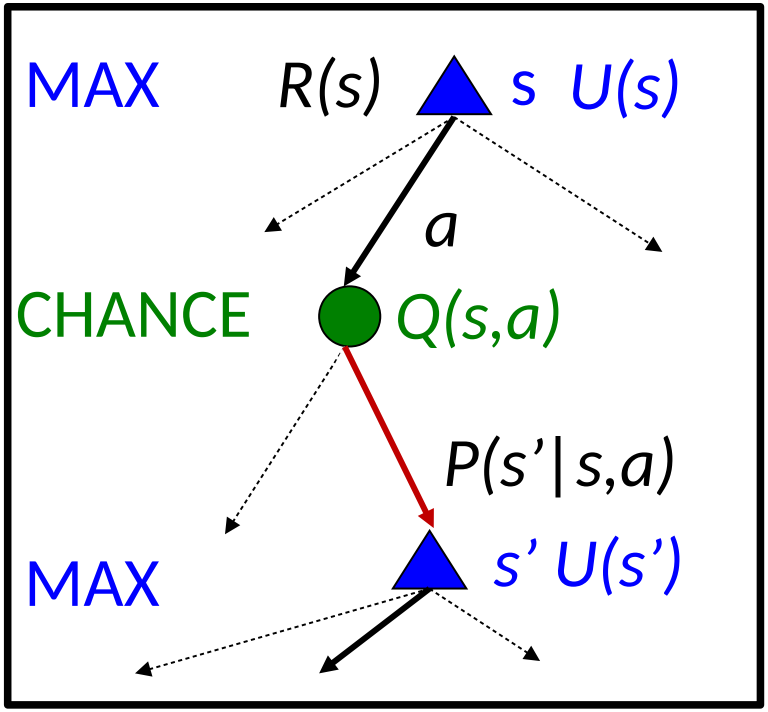
- We still have a Markov Decision Process (MDP).
- Let
  - S : a set of finite **states** (including an initial state s_0 and terminal states).
  - A(s) : a set of possible **actions** from state s.
  - P(s’|s, a) : a **transition** model, where a ϵ A(s).
  - R(s): the **reward** function (how good is to be in state *s*).
  - The environment is perfectly observable.
- We still are looking for the optimal policy π* that maximizes the expected sum of the rewards.
  - $U^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s, \pi(s))\, U^{\pi}(s')$ (where γ is the decay discount factor).
  - $\pi^*(s) = \mathrm{argmax}_{\pi} \sum_{s' \in S} P(s'|s, a)\, U^{\pi}(s')$

Learning

- New consideration: solve MDP, ie find an optimal policy π, when
  - P(s'|s, a) is unknown
  - R(s) is unknown
- Trial and get information from percepts after choosing an action
  - Percepts provide information on the current state and the immediate reward.

3. Q-learning algorithm

- **Q-Learning** learns the action-value function Q(s, a).
  - Q(s, a) is the expected sum of the rewards from *s* and the execution of *a* until the end of the optimal policy.
    - $Q(s,a) = R(s) + \gamma \sum_{s' \in S} P(s'|s, a)\, U(s')$
  - The link between Q (s, a) and *U*(s) is that *U*(s) = $\max_a$ Q(*s*, *a*).
    - $Q(s,a) = R(s) + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) \max_{a'} Q(s',a')$
- The policy of the agent
  - $\pi(s) = \arg\max_a Q(s, a)$.
  - Advantage: for the choice of the action, no need to learn *P*(*s*'|*s*, *a*) or R(s) (hidden in Q(s,a))



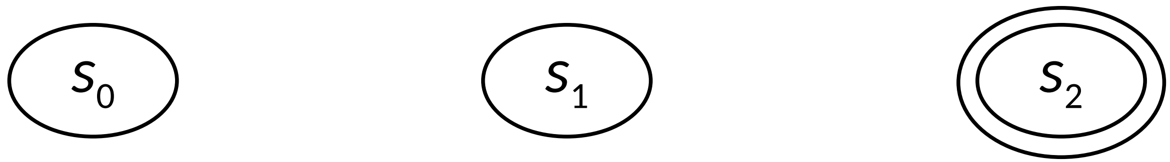
Learning with Q-learning

- According to the definition of $Q(s_t, a_t)$, we have:
  - $Q(s_t, a_t) = [R_{t+1} + \gamma\ R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid s_t, a_t]$
  - R_{t+1} is the reward the agent gains after taking action at time step t.
- We translate this equation by updating based on a learning rate.
  - When a transition $s \rightarrow s'$ occurs from state s to state s', we apply the following update to Q(s, a):
    - $Q(s, a) \leftarrow Q(s, a) + \underbrace{\alpha}_{\text{learning rate}} (\ correction\ )$
    - $correction = difference = \underbrace{R(s) + \gamma \max_{a'} Q(s', a')}_{\text{Current reward + Maximum expected future reward}} \underbrace{- Q(s, a)}_{\text{Previous value}}$
- Theorem: if each action is executed an infinite number of times in each state, the values of Q lead to the optimal policy.

Learning Rate α

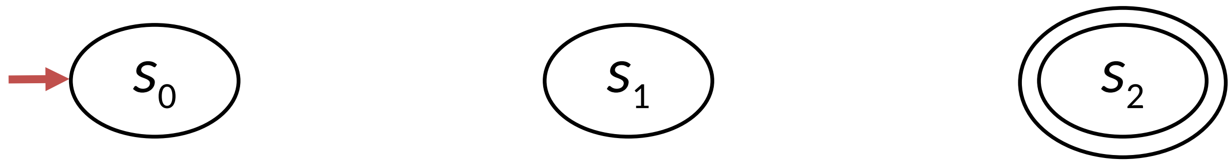
- The learning rate α determines to what extent newly acquired information overrides old information.
  - $Q(s, a) \leftarrow Q(s, a) + \boldsymbol{\alpha}\ (\ R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a)\ )$
  - $Q(s, a) \leftarrow \boldsymbol{(1 - \alpha)}\ Q(s, a) + \boldsymbol{\alpha}\ (R(s) + \gamma \max_{a'} Q(s', a'))$
  - A factor $\boldsymbol{\alpha} = 0$ makes the agent learn nothing (exclusively exploiting prior knowledge).
  - A factor $\boldsymbol{\alpha} = 1$ makes the agent consider only the most recent information (memoryless: ignoring prior knowledge to explore possibilities).
  - In practice, often a constant learning rate is used, such as $\alpha = 0.1$.

Learning with Q-learning. Example



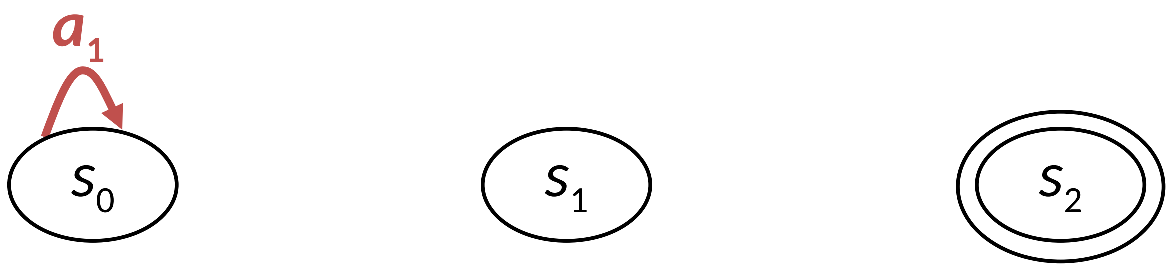
- MDP
  - $S=\{s_0, s_1, s_2\}$, $A=\{a_0, a_1, a_2\}$
- Initialization:
  - $Q(s_0, a_0) = 0$ $Q(s_0, a_1) = 0$ $Q(s_0, a_2) = 0$
    $Q(s_1, a_0) = 0$ $Q(s_1, a_1) = 0$ $Q(s_1, a_2) = 0$
    $Q(s_2, *) = 1$ known terminal node.
- Given:
  - $\alpha = 0.5$, $\gamma = 0.5$
- Note: initial values can be obtain from experiment (eg, helicopter)

Learning with Q-learning. Example



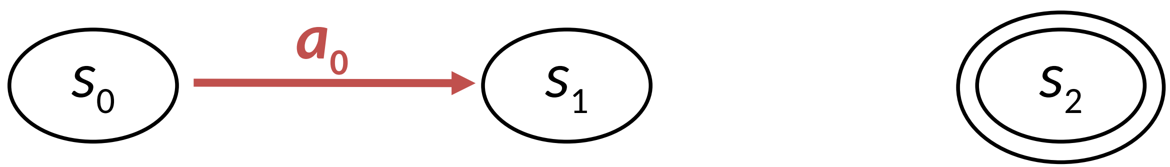
- Observations: $(s_0)_{-0.1}$
  - Nothing to do (we need a triplet (s, a, s'))
  - Chosen action $\pi(s_0) = \arg\max\{ Q(s_0, a_0),\ Q(s_0, a_1), Q(s_0, a_2) \}$
    $= \arg\max\{ 0, 0, 0 \}$
    $= a_1$ (arbitrary, could be a_0 or a_2)

Learning with Q-learning. Example



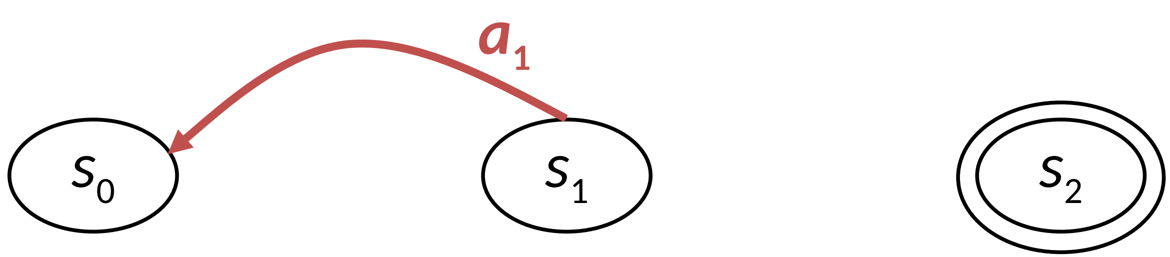
- Observations: $(s_0)_{-0.1} \xrightarrow{a_1} (s_0)_{-0.1}$
  - Recall: $Q(s, a) \leftarrow Q(s,a) + \alpha\, (\, R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)\,)$ for transition $s \rightarrow s'$
  - $\mathbf{Q(s_0,a_1)} \leftarrow Q(s_0, a_1) + \alpha\, (\, R(s_0) + \gamma \max\{\, Q(s_0, a_0), Q(s_0, a_1), Q(s_0, a_2)\} - Q(s_0, a_1)\,)$
    $= 0 + 0.5\, (\, -0.1 + 0.5 \max\{\, 0, 0, 0\} - 0)$
    $= -0.05$
  - Chosen action $\pi(s_0) = \arg\max\{\, Q(s_0, a_0), Q(s_0, a_1), Q(s_0, a_2)\, \}$
    $= \arg\max\{\, 0, -0{,}05, 0\, \}$
    $= a_0$ (arbitrary, could be a_2)
    (change policy! Previously $\pi(s_0) = a_1$)

Learning with Q-learning. Example



- Observations: $(s_0)_{-0.1} \xrightarrow{a_1} (s_0)_{-0.1} \xrightarrow{a_0} (s_1)_{-0.1}$
  - Recall: $Q(s, a) \leftarrow Q(s, a) + \alpha\ (\ R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a)\ )$
  - $\mathbf{Q(s_0, a_0)} \leftarrow Q(s_0, a_0) + \alpha\ (\ R(s_0) + \gamma \max\{\ Q(s_1, a_0), Q(s_1, a_1), Q(s_1, a_2)\} - Q(s_0, a_0)\ )$
    $= 0 + 0.5\ (\ -0.1 + 0.5 \max\{\ 0, 0, 0\} - 0)$
    $= -0.05$
  - Chosen action $\pi(s_1)$ $= \arg\max\{\ Q(s_1, a_0), Q(s_1, a_1), Q(s_1, a_2)\ \}$
    $= \arg\max\{\ 0, 0, 0\ \}$
    $= a_1$ (arbitrary, could be a_0 or a_2)

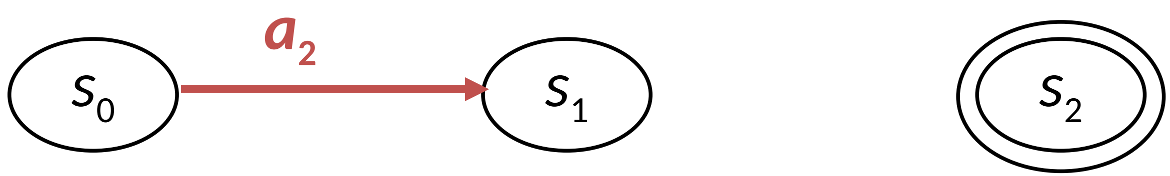
Learning with Q-learning. Example



- Observations: $(s_0)_{-0.1} \xrightarrow{a_1} (s_0)_{-0.1} \xrightarrow{a_0} (s_1)_{-0.1} \xrightarrow{a_1} (s_0)_{-0.1}$

  - Recall: $Q(s, a) \leftarrow Q(s, a) + \alpha\ (\ R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a)\ )$
  - $\mathbf{Q(s_1, a_1)} \leftarrow Q(s_1, a_1) + \alpha\ (\ R(s_1) + \gamma \max\{\ Q(s_0, a_0), Q(s_0, a_1), Q(s_0, a_2)\} - Q(s_1, a_1)\ )$
    $= 0 + 0.5\ (\ -0.1 + 0.5 \max\{\ -0.05, -0.05, 0\} + 0)$
    $= -0.0625$
  - Chosen action $\pi(s_0)$ $= \arg\max\{\ Q(s_0, a_0), Q(s_0, a_1), Q(s_0, a_2)\ \}$
    $= \arg\max\{\ -0.05, -0.05, 0\ \}$
    $= a_2$ **(change policy! Previously $\pi(s_0) = a_0$)**

Learning with Q-learning. Example

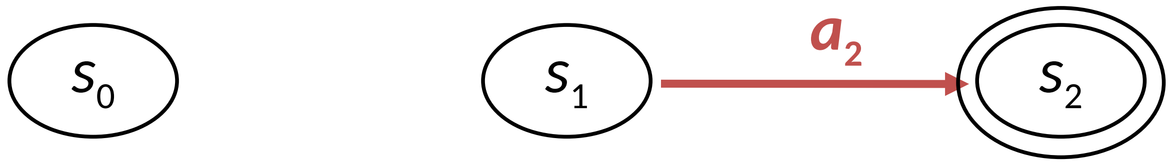


- Observations: $(s_0)_{-0.1} \xrightarrow{a_1} (s_0)_{-0.1} \xrightarrow{a_0} (s_1)_{-0.1} \xrightarrow{a_1} (s_0)_{-0.1} \xrightarrow{a_2} (s_1)_{-0.1}$

  - $\boldsymbol{Q(s_0, a_2)} \leftarrow Q(s_0, a_2) + \alpha\ (\ R(s_0) + \gamma \max\{\ Q(s_1, a_0), Q(s_1, a_1), Q(s_1, a_2)\} - Q(s_0, a_2)\ )$
    $= 0 + 0.5\ (\ -0.1 + 0.5 \max\{\ -0.0625, 0, 0\ \} + 0)$
    $= -0.065625$

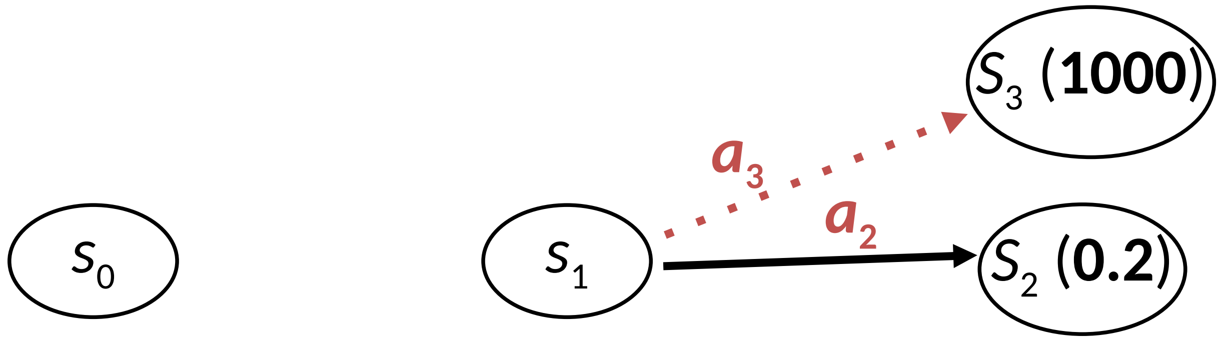
  - Chosen action $\pi(s_1)$ $= \arg\max\{\ Q(s_1, a_0), Q(s_1, a_1), Q(s_1, a_2)\ \}$
    $= \arg\max\{\ 0, -0.0625, 0\ \}$
    $= a_2$ **(change policy! Previously $\pi(s_1) = a_1$)**

Learning with Q-learning. Example



- Observations: $(s_0)_{-0.1} \xrightarrow{a_1} (s_0)_{-0.1} \xrightarrow{a_0} (s_1)_{-0.1} \xrightarrow{a_1} (s_0)_{-0.1} \xrightarrow{a_2} (s_1)_{-0.1} \xrightarrow{a_2} (s_2)_1$
  - **Terminal state:** $Q(s_2, *) = 1$
  - Recall: $Q(s, a) \leftarrow Q(s, a) + \alpha\ (\ R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a)\ )$
  - $Q(s_1, a_2) \leftarrow Q(s_1, a_2) + \alpha\ (\ R(s_1) + \gamma \max\{\ Q(s_2, *)\ \} - Q(s_1, a_2)\ )$
    $= 0 + 0.5\ (\ -0.1 + 0.5 \max\{\ 1\ \} + 0)$
    $= 0.2$
- Let's go for a new try…

4. Dilemma Exploration vs. Exploitation



- The current approach is greedy: always uses the best action (*exploitation*)
- Problem:
  - Suppose we could do action a_3 in state s_1, that leads to state s_3 such as $R(s_3) = 1000$.
  - Since $Q(s_1, a_3) = 0$ at initialization, and current $Q(s_1, a_2) = 0.2$, after a try $Q(s_1, a_2) > Q(s_1, a_3)$, a greedy approach will never explore s_3!
- Solution : use random actions (exploration)

Dilemma Exploration vs. Exploitation

- Analysis
  - Too much exploitation leads to an agent that relies on suboptimal plans.
  - Too much exploration leads to an agent that wastes his time to learn.
- Finding the optimal balance between exploration and exploitation is an open problem in general.
  - Optimal exploration / exploitation strategies exist only in very simple cases.
  - Only practical heuristics are available.

Solution 1: ε-Greedy

- The simplest strategy
  - Consider parameter $\varepsilon \in [0,1]$
  - Every time step, flip a coin → p
  - Exploration: with small probability ($p \leq \varepsilon$), act randomly
  - Exploitation: with large probability ($p > \varepsilon$), act on current policy
- Problems with random actions?
  - You do eventually explore the space, but keep acting randomly once learning is done.
  - One solution: lower ε over time.

Solution 2: Exploration Function

- A more adaptive strategy: the exploration function *f(u, n)*
  - This function increases artificially the future rewards of unexplored actions to favor exploration.
  - Q(s, a) ← Q(s, a) + α(R(s) + γ max$_{a'}$ **f(** Q[a', s'], **N(s', a'))** – Q[a, s])
    where N(s,a) is the number of times action a has been chosen in state *s*

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

    - R^{+} : an artificial optimistic estimate of the best possible reward obtainable in any state (*problem dependent*)
    - N_e a fix parameter. Guaranty that action a will be chosen in *s* at least N_e times during learning.
    - The exploration function should decrease for u and increase for n.

Q-learning Algorithm


```
function Q-LEARNING-AGENT(percept) returns an action
input: percept, indicating the current state s′ and reward signal r′
persistent: Q, a table of action values index by state and action
            Nsa, a table of frequencies for state-action pairs
            s,a,r the previous state, action, reward, initially null

  IF s is not null THEN
     increment Nsa[s,a]
     Q[s,a] ← Q[s,a] + α(f(r + γ max_a′ Q[s′,a′], Nsa[s′,a′]) – Q[s,a])
  s, a, r ← s′, arg max_a′ f(Q[s′, a′], Nsa[s′,a′]), r′
  return a
end
```

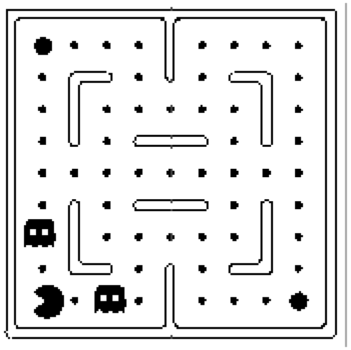
f(u,n) is the exploration function.

5. Limits of this modeling

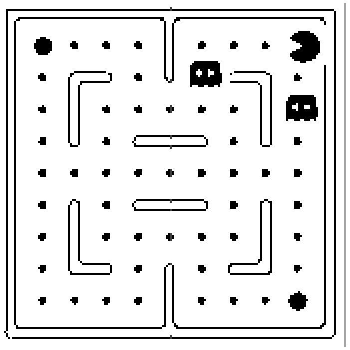
- Problem 1: Limited for real applications
  - **Q(state, action)** values are stored in tables of size state*action.
    - Too large to hold in memory.
    - Too large to learn all the Q values. Time to convergence increases rapidly as the space gets larger.

Limits of this modeling

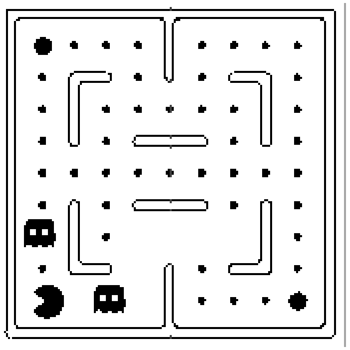
- Problem 2: Generalization
- Example: Pacman
  - Suppose we learn during tries that this states is bad (low *U(s) value*).



  - With the described Q-Learning approach, we cannot deduce anything about this state.



  - Or even for this one (*one dot less*).

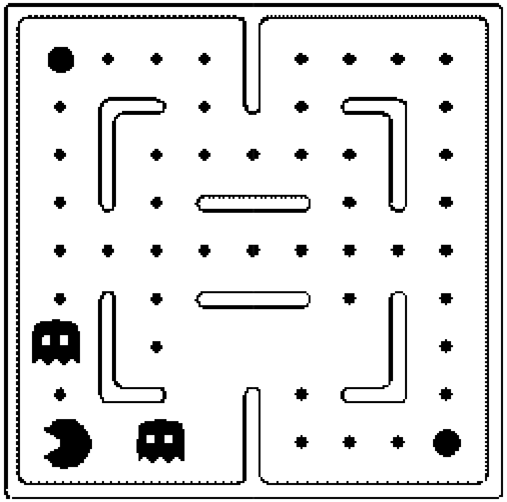


Solution: Approximation of Utility Function

- Describe a state using a vector of features (cf. adversarial search).
  - $f(s)= \Sigma_i \, w_i \, . \, f_i(s, a)$
  - Features f_i are floating point functions that capture the salient properties of states.
- Advantages
  - Represent utility functions for a very large number of states.
  - Generalize to similar states.

Example of State Abstraction: Pacman

- Examples of features:
  - Distance to the nearest ghost
  - Distance to the nearest food.
  - Number of ghosts.
  - 1 / (distance to closest dot)
  - Is Pacman in a tunnel? (0/1)
  - Is Pacman close to a ghost? (0/1)
  - etc.



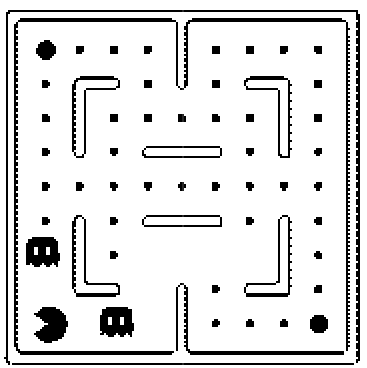
Abstraction using Approximated Linear Functions

- Q values becomes:
  - $Q_f(s, a) = \Sigma_i\, w_i \,.\, f_i(s, a)$
- Q-Learning learning algorithm is used to learn the weight vector W, similar to updating Q-values when a transition *s* → *s'* by action a occurs:
  - $\forall i,\ w_i \leftarrow w_i + \alpha[\text{difference}] \,.\, f_i(s, a)$
  - $\text{difference} = [\ R(s, a) + \gamma \max_{a'} Q(s', a'))\ ] - Q(s, a)$
- Intuitive interpretation:
  - Adjust weights of active features
  - e.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features.

Example: Q-Pacman

- Suppose 2 features: $Q(s,a) = 4.0\, f_{DOT}(s, a) - 1.0\, f_{GST}(s, a)$ and $\alpha = 0.004$
  - f_{DOT} = 1 / (distance to closest dot)
  - f_{GST} = distance to the nearest ghost

s

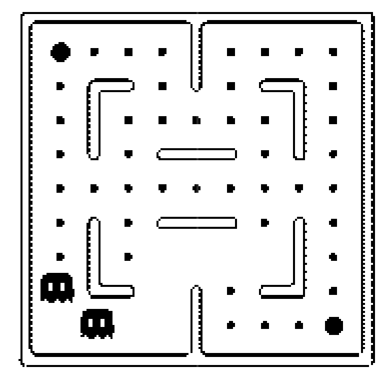


$f_{DOT}(s, \text{NORTH}) = 0.5$

$f_{GST}(s, \text{NORTH}) = 1.0$

a = NORTH
r = -500

s'



$Q(s, \text{NORTH}) = 4.0*0.5 - 1.0*1.0 = +1$

$Q(s', *) = 0$

  - $r + \gamma \max_{a'} Q(s', a') = -500 + 0$
  - difference = $[\, R(s, a) + \gamma \max_{a'} Q(s', a')) \,] - Q(s, a)$

$= -500 - 1$

$w_{DOT} \leftarrow 4.0 + \alpha[-501]\,.\,0.5 = 3$

$w_{GST} \leftarrow -1.0 + \alpha[-501]\,.\,1.0 = -3$

- Update: $Q(s, a) = 3.0\, F_{DOT}(s, a) - 3.0\, F_{GST}(s, a)$

Conclusion

- Reinforcement learning is used to learn sequential decision making.
- This is a very active area of research:
  - There are more and more applications in robotics, self-driving, and other areas.
- Reinforcement learning is more difficult when the reward is far (eg. at the very end of a game).
  - Sometimes appropriate positive intermediate reinforcements are added.
    - Drawback: need expertise.

[Boston Dynamics.sh]