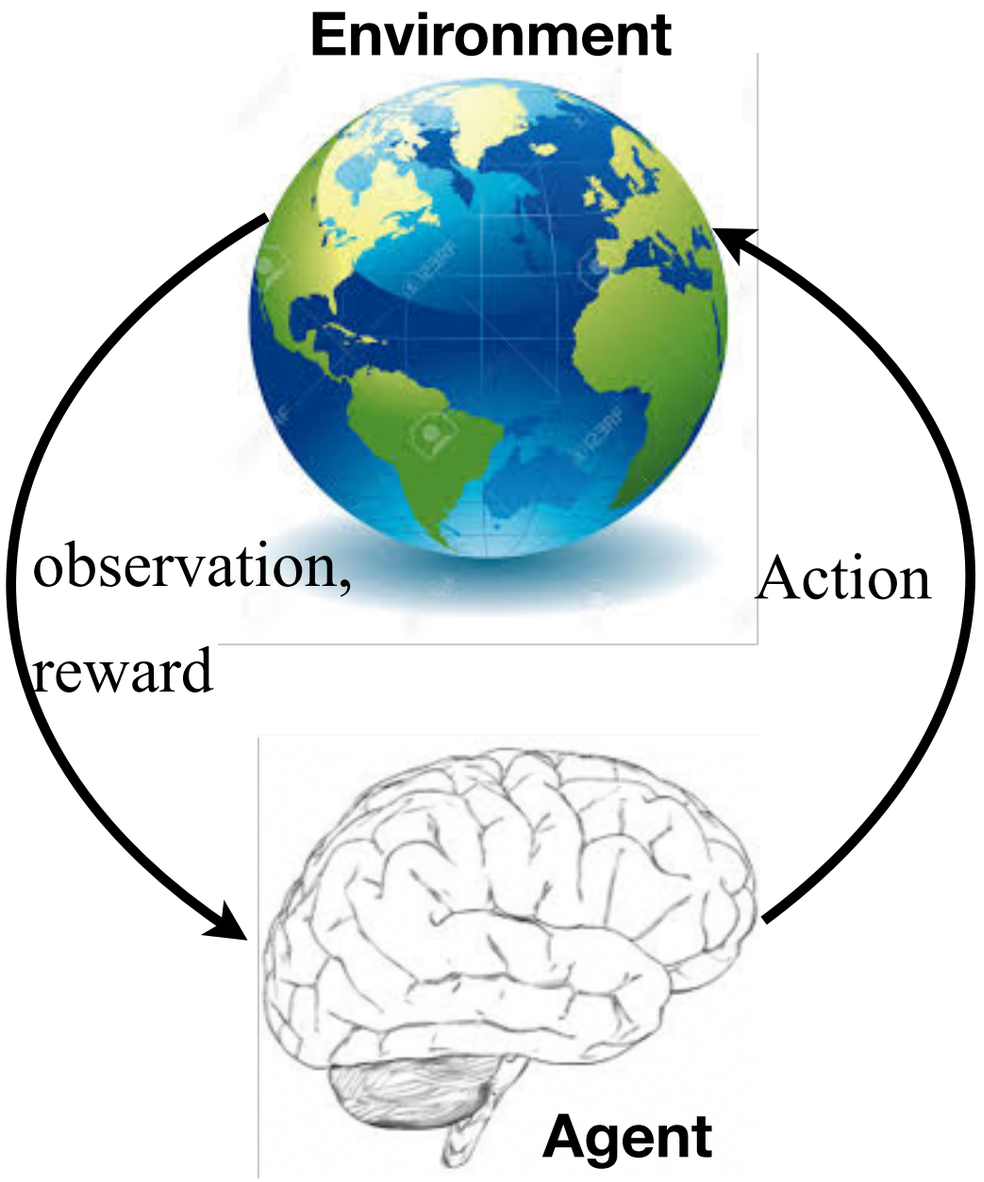
Reinforcement Learning

Khimya Khetarpal
School of Computer Science, McGill University
Mila, Montreal

IVADO/MILA Summer School, 2019

Reinforcement Learning

- Learning by trial-and-error, in real time
- Improves with experience
- Inspired by psychology



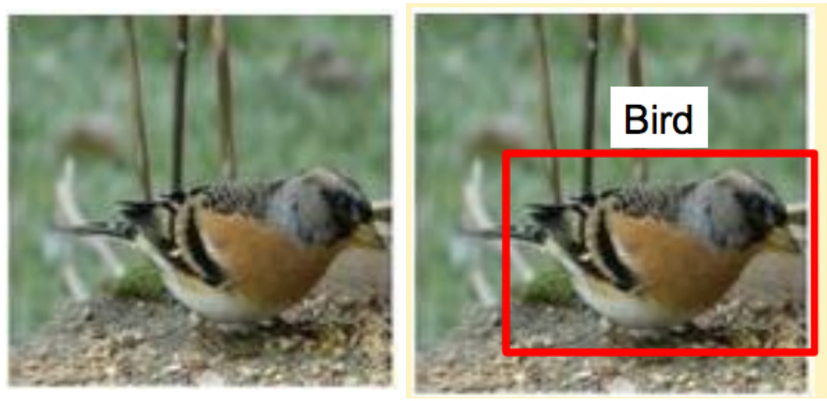
Progress In Recent Years

AlphaGo Zero

Starting from scratch

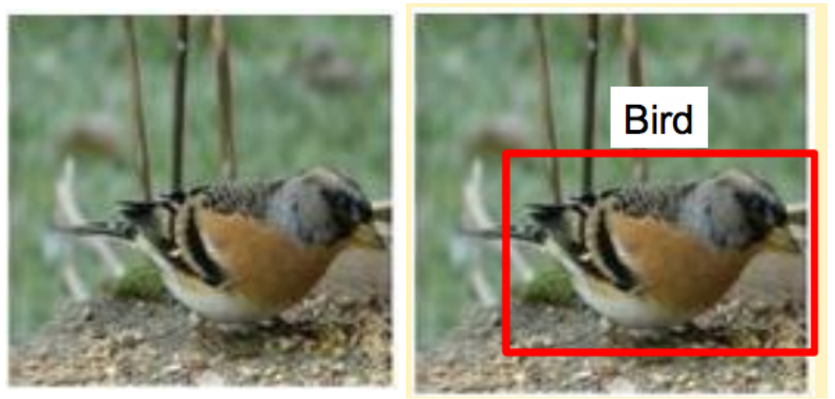
RL vs other types of learning

Supervised Learning: the training data comprises examples of the input vectors along with their corresponding target vectors. Examples: classification, regression

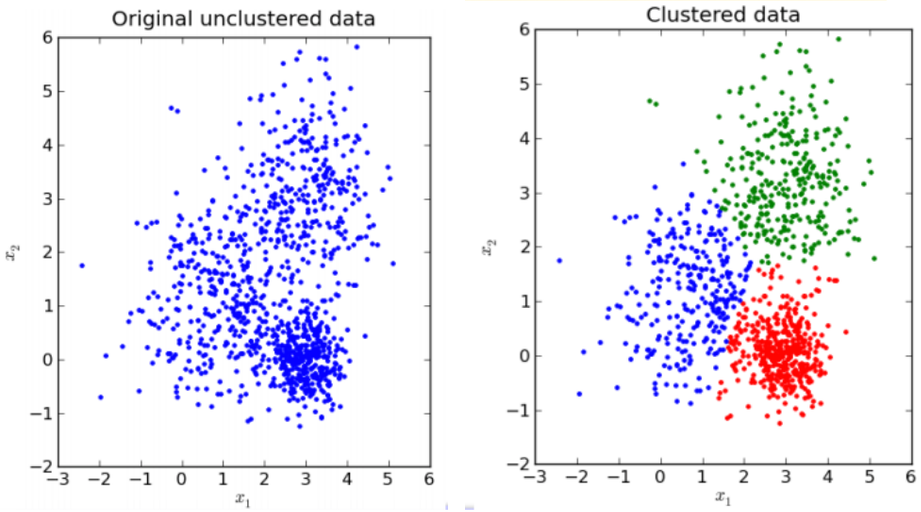


RL vs other types of learning

Supervised Learning: the training data comprises examples of the input vectors along with their corresponding target vectors. Examples: classification, regression

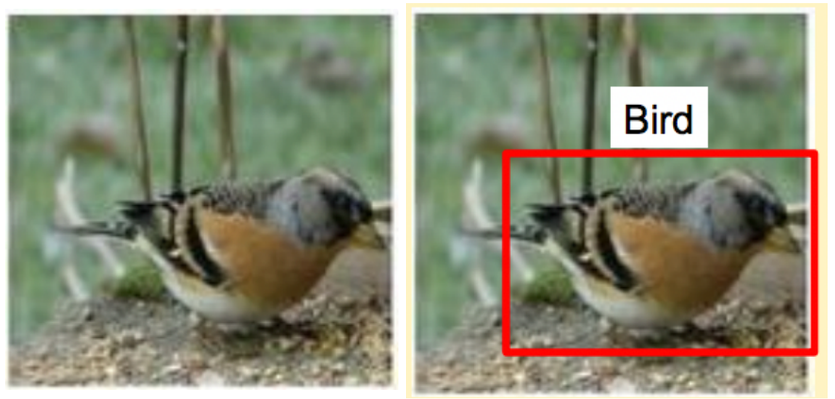


Unsupervised Learning: the training data consists of a set of input vectors x without any corresponding target values.

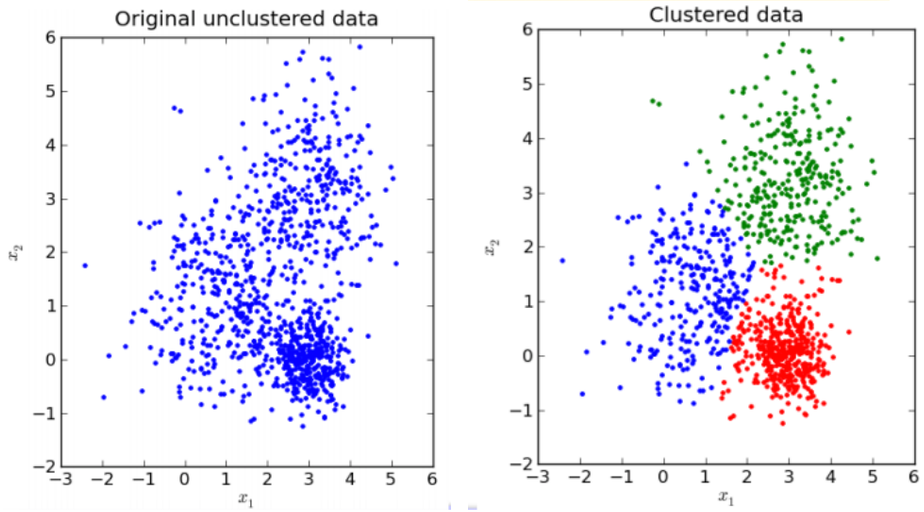


RL vs other types of learning

Supervised Learning: the training data comprises examples of the input vectors along with their corresponding target vectors. Examples: classification, regression



Unsupervised Learning: the training data consists of a set of input vectors x without any corresponding target values.



Reinforcement Learning: the problem of finding suitable actions to take in a given situation in order to maximize reward(s).

Characteristics of Reinforcement Learning

Supervised Learning: the training data comprises examples of the input vectors along with their corresponding target vectors. Examples: classification, regression

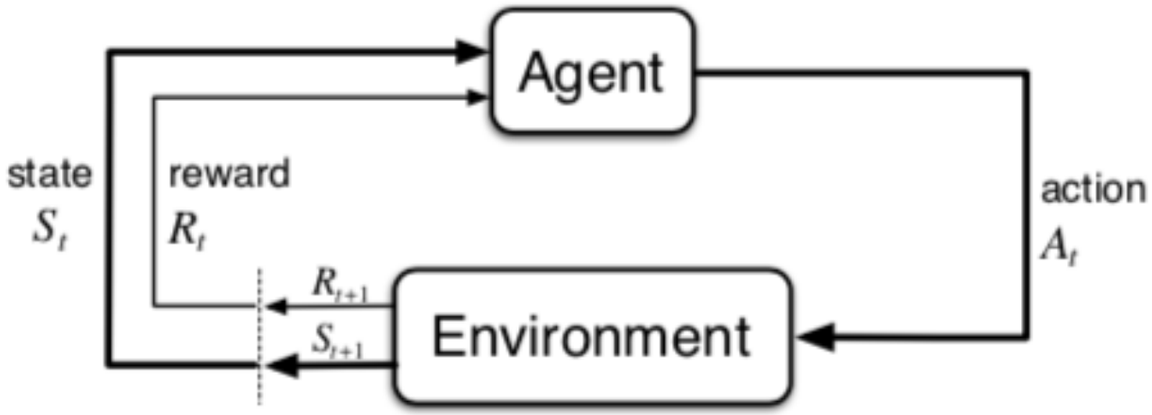
Unsupervised Learning: the training data consists of a set of input vectors x without any corresponding target values.

Reinforcement Learning: the problem of finding suitable actions to take in a given situation in order to maximize reward(s).

What makes RL different from other ML paradigms?

- There is no supervisor, only a reward signal
- Feedback is delayed, not instantaneous
- Time really matters: data is sequential (non i.i.d)
- Agent's actions affect the subsequent data it receives

Reinforcement Learning



Agent-Environment Interaction
(Fig. from Sutton & Barto)

At each time step, the ***agent***:

- Observes state $S_t \in S$
- Executes action $A_t \in A$
- Receives reward R_t

At each time step, the ***environment***:

- Receives action A_{t+1}
- Emits new state S_{t+1}
- Emits scalar reward R_{t+1}

Markov Property

The future is independent of the past given the present.

$$P(S_{t+1} \mid S_t, A_t) = P(S_{t+1} \mid S_1, A_1, S_2, A_2 \ldots . S_t, A_t)$$

- The state captures all relevant information from the history

- The state is a sufficient statistic of the future

- ***Markovian assumption*:** current state provides sufficient information to describe the distribution of immediate reward and next state

Markov Decision Processes

- **Markov decision processes (MDP)** formally describes an environment for reinforcement learning

- A finite discrete-time MDP is a tuple $\langle S, A, R, P, \gamma \rangle$

Markov Decision Processes

- **Markov decision processes (MDP)** formally describes an environment for reinforcement learning
- A finite discrete-time MDP is a tuple $\langle S, A, R, P, \gamma \rangle$
- One-step ***model*** of the environment:
  - One-step ***state-transition probabilities***

$$p(s' \mid s, a) \doteq P^a_{ss'} = Pr(S_{t+1} = s' \mid S_t = s, A_t = a) = \sum_{r \in R} p(s', r \mid s, a)$$

  - One-step ***expected rewards***

$$r(s, a) = R^a_s = E[R_{t+1} \mid S_t = s, A_t = a] = \sum_{r \in R} r \sum_{s' \in S} p(s', r \mid s, a)$$

Agent's Learning Task

- The agent's behavior is determined by its ***policy***, a mapping from states to probabilities of taking each of the admissible primitive actions

$$\pi : S \times A \rightarrow [0,1]$$

- The agent executes actions in an environment, observe results, and learn policy (strategy, way of behaving)

- MDP policies depend on the current state (not the history)

- i.e. Policies are ***stationary*** (time-independent) $A_t \sim \pi(\,.\,|\,S_t), \forall t > 0$

- Note that the target function is $\pi : S \rightarrow A$, but we have ***no training examples*** of form $\langle s, a \rangle$

- Training examples are of form $\langle \langle s, a \rangle, r, s', \ldots \rangle$

Agent's objective

- The goal of the agent is to maximize the expected value of the accumulated discounted reward from time-step t.

- Maximize the expected ***return***

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- The ***discount*** $\gamma \in [0,1]$ is the present value of the future rewards

- The value of receiving reward R after k+1 time steps is $\gamma^k R$

- This values immediate reward over delayed reward

  - γ close to 0 leads to "myopic"evaluation
  - γ close to 1 leads to "far-sighted" evaluation

Agent's objective

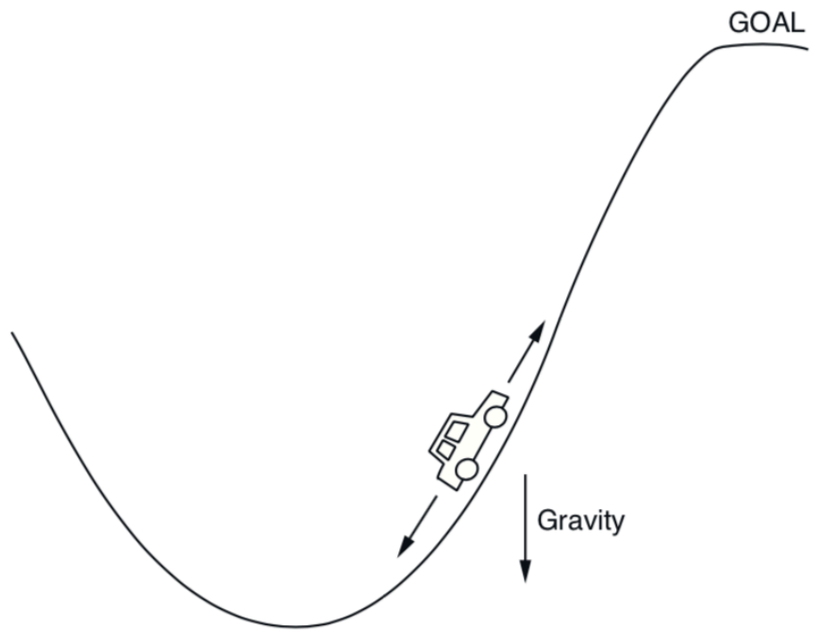
- **Episodic:** consider return over finite horizon

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \ldots R_T$$

- **Continuing Task:** consider return over infinite horizon

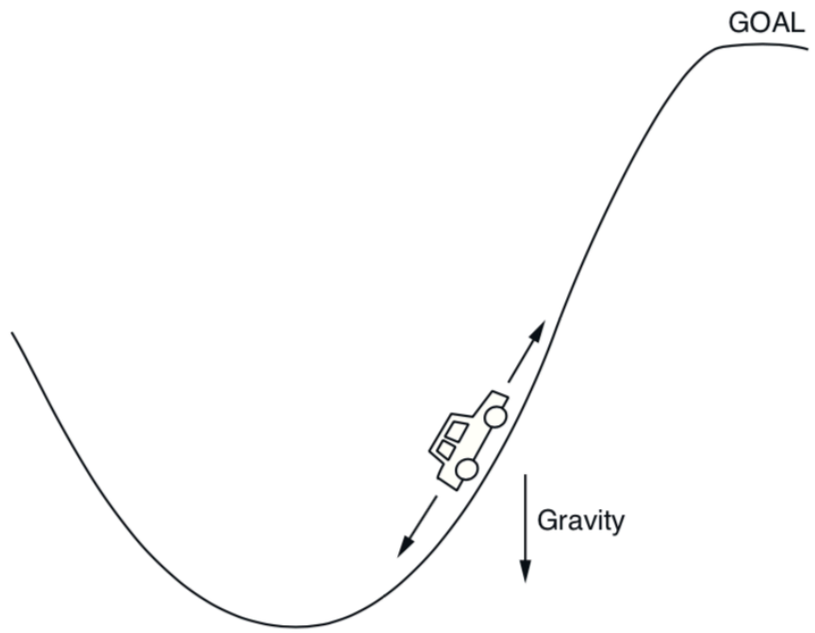
$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Example: Mountain Car



- States: position and velocity
- Actions: accelerate forward, accelerate backward
- Aim: agent to move the car to get to the top of the hill as quickly as possible
- What are the rewards and the returns?

Example: Mountain Car



- States: position and velocity
- Actions: accelerate forward, accelerate backward
- Possible Reward formulations:
  - Reward = -1 for every time step, until the car reaches the top
  - Reward = 1 at the top, 0 otherwise , $\gamma < 1$

Value Function

- The ***value of being in a state*** is the expected return starting from state s, and then following policy π .

$$v_\pi(s) = E_\pi[G_t \mid S_t = s]$$

Value Function

- The ***value of being in a state*** is the expected return starting from state s, and then following policy π .

$$v_\pi(s) = E_\pi[G_t \,|\, S_t = s]$$

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \,|\, S_t = s]$$

Value Function

- The ***value of being in a state*** is the expected return starting from state s, and then following policy π .

$$v_\pi(s) = E_\pi[G_t \,|\, S_t = s]$$

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \,|\, S_t = s]$$

$$= E_\pi[R_{t+1} + \gamma G_{t+1} \,|\, S_t = s]$$

Value Function

- The ***value of being in a state*** is the expected return starting from state s, and then following policy π .

$$v_\pi(s) = E_\pi[G_t \,|\, S_t = s]$$

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \,|\, S_t = s]$$

$$= E_\pi[R_{t+1} + \gamma G_{t+1} \,|\, S_t = s]$$

$$= E_\pi[R_{t+1} + \gamma E_\pi[G_{t+1} \,|\, S_{t+1} = s']]$$

Value Function

- The ***value of being in a state*** is the expected return starting from state s, and then following policy π .

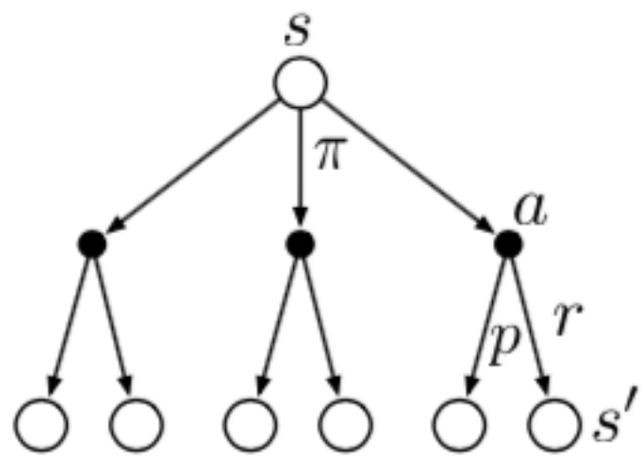
$$v_\pi(s) = E_\pi[G_t \mid S_t = s]$$

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s]$$

$$= E_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

$$= E_\pi[R_{t+1} + \gamma E_\pi[G_{t+1} \mid S_{t+1} = s']]$$

$$= \sum_{a \in A} \pi(a \mid s) \left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right]$$



Value Function

- The ***value of being in a state*** is the expected return starting from state s, and then following policy π .

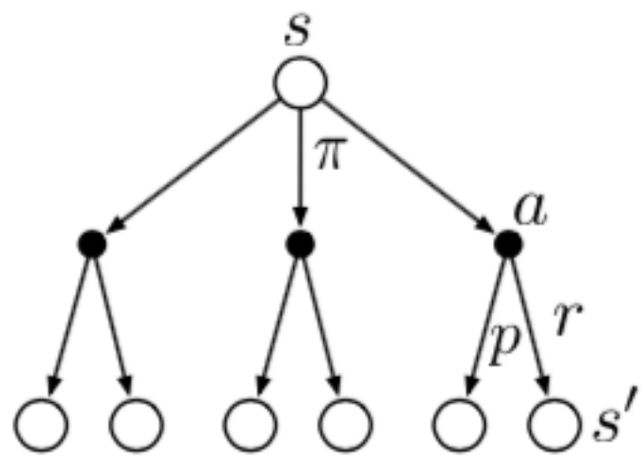
$$v_\pi(s) = E_\pi[G_t \mid S_t = s]$$

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s]$$

$$= E_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

$$= E_\pi[R_{t+1} + \gamma E_\pi[G_{t+1} \mid S_{t+1} = s']]$$

$$= \sum_{a \in A} \pi(a \mid s) \left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right]$$



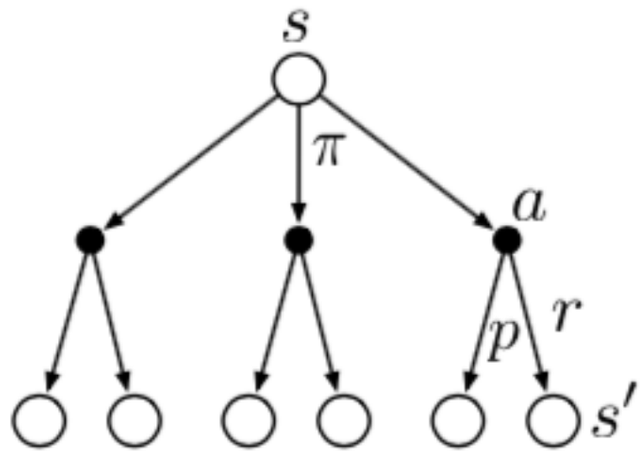
- Values can be written in terms of successor values: ***Bellman equations***

Value Function

- The ***value of being in a state*** is the expected return starting from state s, and then following policy π .

$$v_\pi(s) = E_\pi[G_t \mid S_t = s]$$

$$= \sum_{a \in A} \pi(a \mid s) \left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right]$$



- If the state-space is finite, the collection of values of all states, $v_\pi(s)$ can be represented as a vector of size equal to the number of states

- The function mapping states to expected returns is called ***state-value-function.***

Action-Value Function

- The ***value of taking an action a in a state*** s under policy π

$$q_\pi(s,a) = E_\pi[G_t \,|\, S_t = s, A_t = a]$$

Action-Value Function

- The ***value of taking an action a in a state*** s under policy π

$$q_\pi(s,a) = E_\pi[G_t \,|\, S_t = s, A_t = a]$$

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a' \,|\, s') q_\pi(s', a')$$

Optimal Policies and Value Functions

- Value functions define a partial order over policies:

$$\pi_1 \geq \pi_2 \textbf{ iff } v_{\pi_1}(s) \geq v_{\pi_2}(s), \forall s \in S$$

- If a policy is better than another policy if and only if, it generates at least the same amount of return at all states

- The optimal state-value function $v*(s)$ is the maximum value function over all policies

$$\mathbf{v}^*(\mathbf{s}) = \max_{\pi} \mathbf{v}_{\pi}(\mathbf{s})$$

- The optimal action-value function $q^*(s, a)$ is the maximum action-value function over all policies

$$\mathbf{q}^*(\mathbf{s}, \mathbf{a}) = \max_{\pi} \mathbf{q}_{\pi}(\mathbf{s}, \mathbf{a})$$

Optimal Policies and Value Functions

For any MDP:

- There exists an **optimal policy** π^* that is better than or equal to all other policies, $\pi^* \geq \pi, \forall \pi$

- All optimal policies achieve the ***optimal value function***, $v_{\pi^*}(s) = v^*(s)$

- All optimal policies achieve the ***optimal action-value function***, $q_{\pi^*}(s, a) = q^*(s, a)$

Dynamic Programming

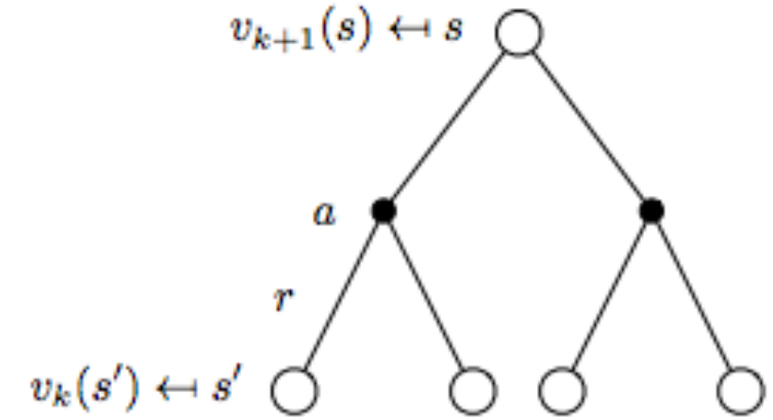
- **Key Idea:** Turn Bellman equations into update rules

- For instance, we can use DP for
  - **Iterative Policy Evaluation**
  - **Policy Iteration**
  - **Value Iteration**

Iterative Policy Evaluation

- **Problem:** Evaluate a given policy π

- **Solution**: iterative application of Bellman expectation backup
  - $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$
  - Using synchronous backups,
    - At each iteration $k+1$
    - For all states $s \in S$
    - Update $v_{k+1}(s)$ from $v_k(s')$
    - where s' is a successor state of s



$$v_{k+1} = \sum_{a \in A} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

Convergence of Iterative Policy Evaluation

Consider the absolute error in our estimate

$$| v_{k+1}(s) - v_\pi(s) |$$

Convergence of Iterative Policy Evaluation

Consider the absolute error in our estimate

$$| v_{k+1}(s) - v_\pi(s) |$$

$$= \left| \sum_{a \in A} \pi(a \mid s) \left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right] - \sum_{a \in A} \pi(a \mid s) \left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right] \right|$$

As long as $\gamma < 1$ **the error contracts**, and eventually goes to 0

Convergence of Iterative Policy Evaluation

Consider the absolute error in our estimate

$$| v_{k+1}(s) - v_\pi(s) |$$

$$= \left| \sum_{a \in A} \pi(a \mid s) \left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right] - \sum_{a \in A} \pi(a \mid s) \left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right] \right|$$

$$= \gamma \left| \sum_{a \in A} \pi(a \mid s) \sum_{s' \in S} P_{ss'}^a \left(v_k(s') - v_\pi(s') \right) \right|$$

Convergence of Iterative Policy Evaluation

Consider the absolute error in our estimate

$$|\, v_{k+1}(s) - v_\pi(s) \,|$$

$$= \left| \sum_{a \in A} \pi(a \,|\, s) \left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right] - \sum_{a \in A} \pi(a \,|\, s) \left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right] \right|$$

$$= \gamma \left| \sum_{a \in A} \pi(a \,|\, s) \sum_{s' \in S} P_{ss'}^a \Big(v_k(s') - v_\pi(s') \Big) \right|$$

$$\leq \gamma \sum_{a \in A} \pi(a \,|\, s) \sum_{s' \in S} P_{ss'}^a \left| v_k(s') - v_\pi(s') \right|$$

Convergence of Iterative Policy Evaluation

Consider the absolute error in our estimate

$$|\, v_{k+1}(s) - v_\pi(s) \,|$$

$$= \left| \sum_{a \in A} \pi(a \,|\, s) \left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right] - \sum_{a \in A} \pi(a \,|\, s) \left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right] \right|$$

$$= \gamma \left| \sum_{a \in A} \pi(a \,|\, s) \sum_{s' \in S} P_{ss'}^a \Big(v_k(s') - v_\pi(s') \Big) \right|$$

$$\leq \gamma \sum_{a \in A} \pi(a \,|\, s) \sum_{s' \in S} P_{ss'}^a \left| v_k(s') - v_\pi(s') \right|$$

As long as $\gamma < 1$ **the error contracts**, and eventually goes to 0

How to Improve a Policy

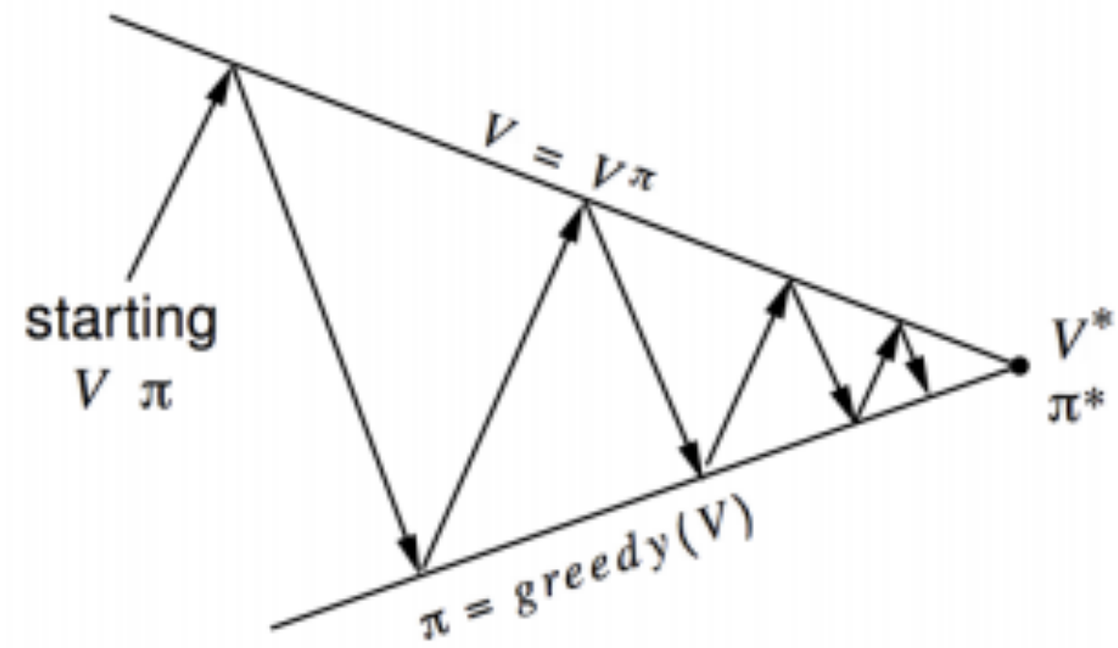
- Given a policy π
  - **Evaluate** the policy π

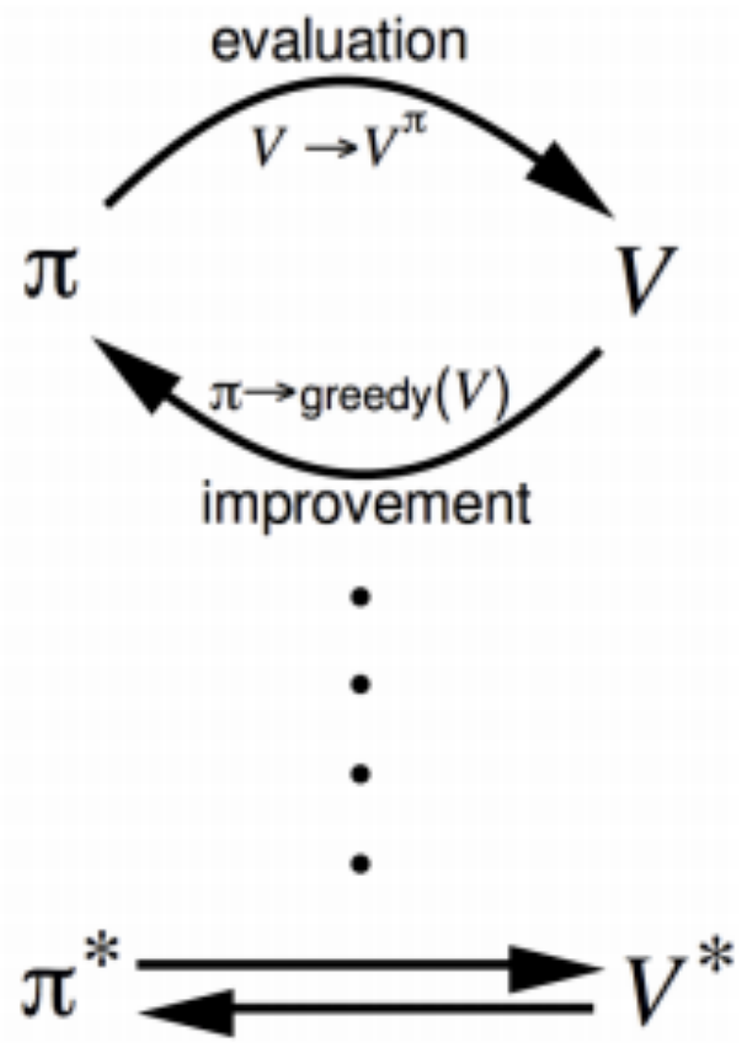
$$v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s]$$

  - **Improve the** the policy π by acting greedily with respect to v_π

$$\pi' = \texttt{greedy}(\texttt{v}_\pi)$$

Policy Iteration

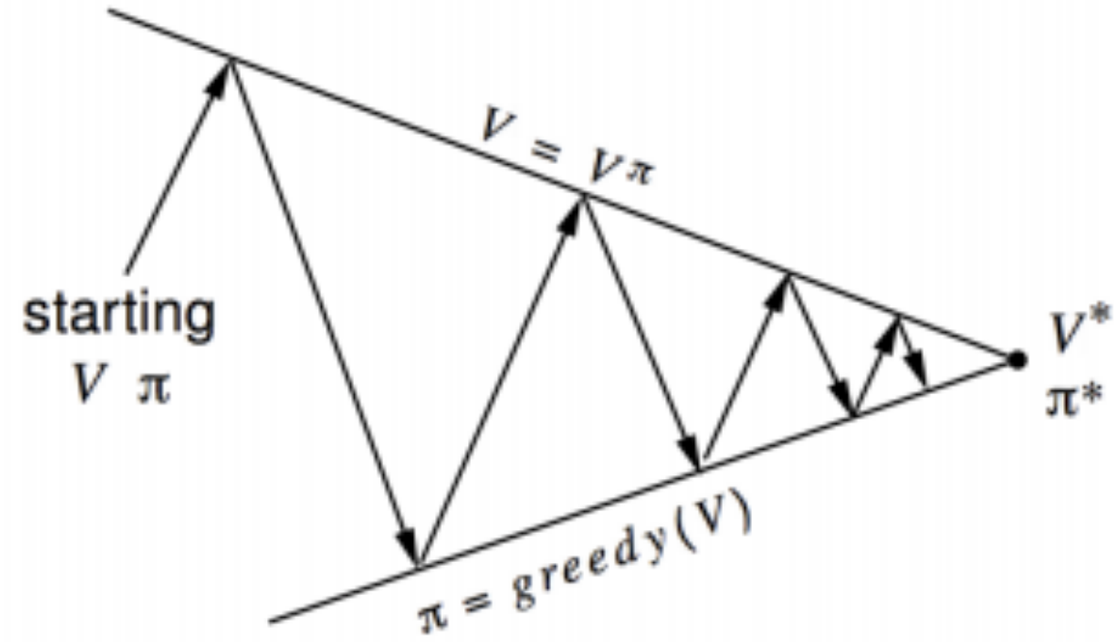


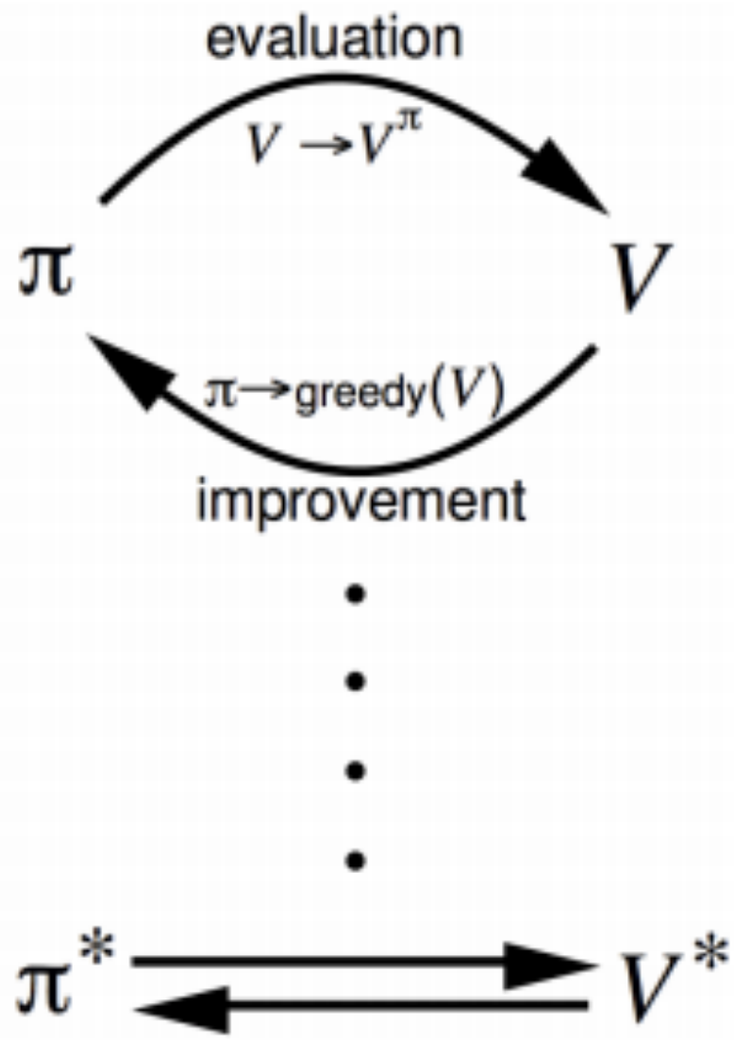


- ***Policy evaluation:*** Estimate v_π,
  Iterative policy evaluation

- ***Policy improvement:*** Generate $\pi' \geq \pi$
  Greedy policy improvement

Generalized Policy Iteration





- ***Policy evaluation:*** Estimate v_π,
  Any policy evaluation

- ***Policy improvement:*** Generate $\pi' \geq \pi$
  Any policy improvement

Model-based reinforcement learning

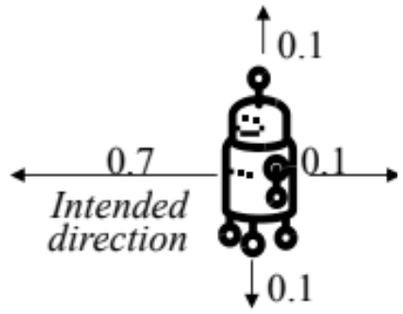
- Usually, the model of the environment ($R_s^a, P_{ss'}^a$) is ***unknown***
- Instead, the learner observes transitions and rewards in the environment
- ***Model-based learning algorithms*** use this data to build an ***approximate*** model $\hat{R}_s^a, \hat{P}_{ss'}^a$
- Note that this is just a supervised machine learning problem
- In the simplest case, $\hat{R}$ can be estimated as the mean reward for every state-action pair, and $\hat{P}$ can be estimated using counts
- Using this ***approximate*** model we built, we can then compute the value function as before.

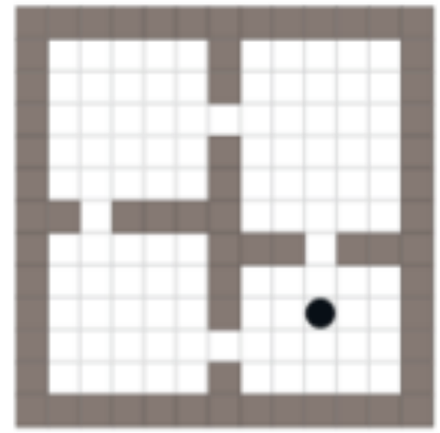
Challenges in model based approach

- Large or **infinite state** and **action space** would require many data points
- If the model is not estimated correctly, **errors** are amplified during control

Curse of Dimensionality

- **Tabular:** Can store in memory a list of the states and their value.





- The number of states go ***exponentially*** with the number of state variables ,
  E.g. in Go, there are 10^{170} states

- The ***action set*** may also be very large or continuous, E.g. in Go, branching factor is ~100

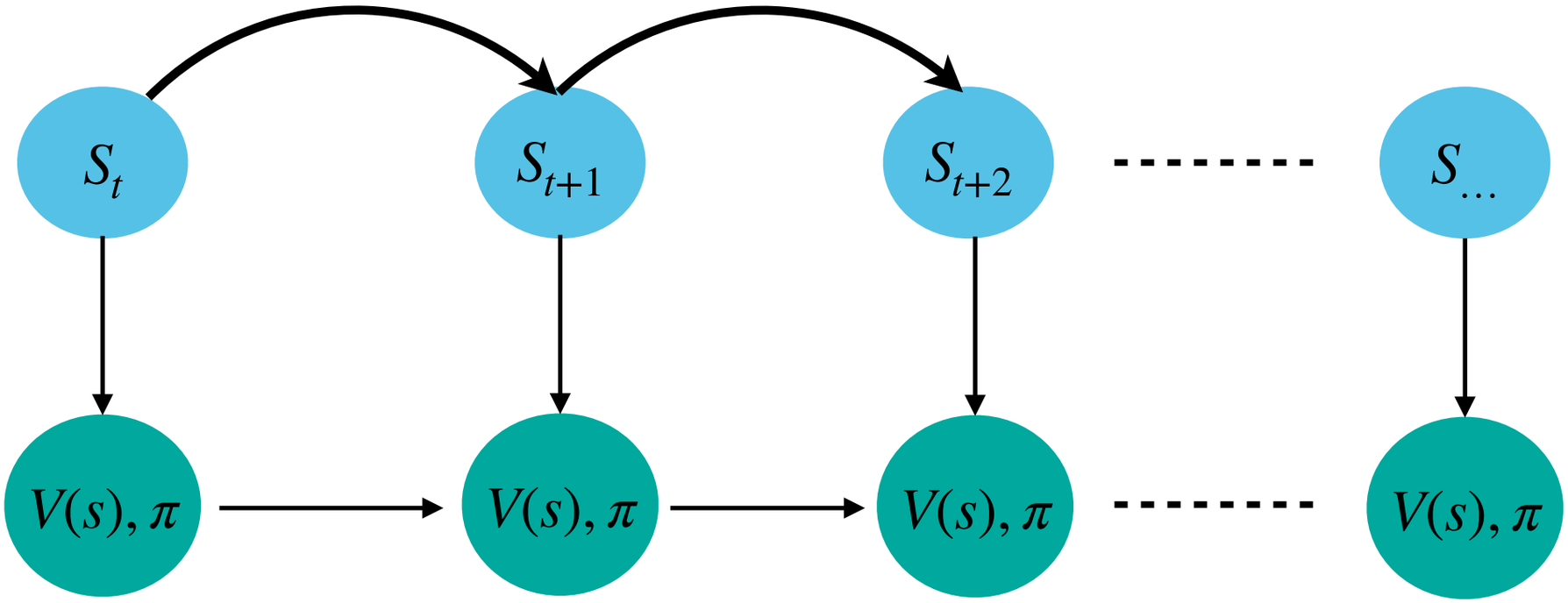
Key Challenges in RL

To solve large problems, we need to:

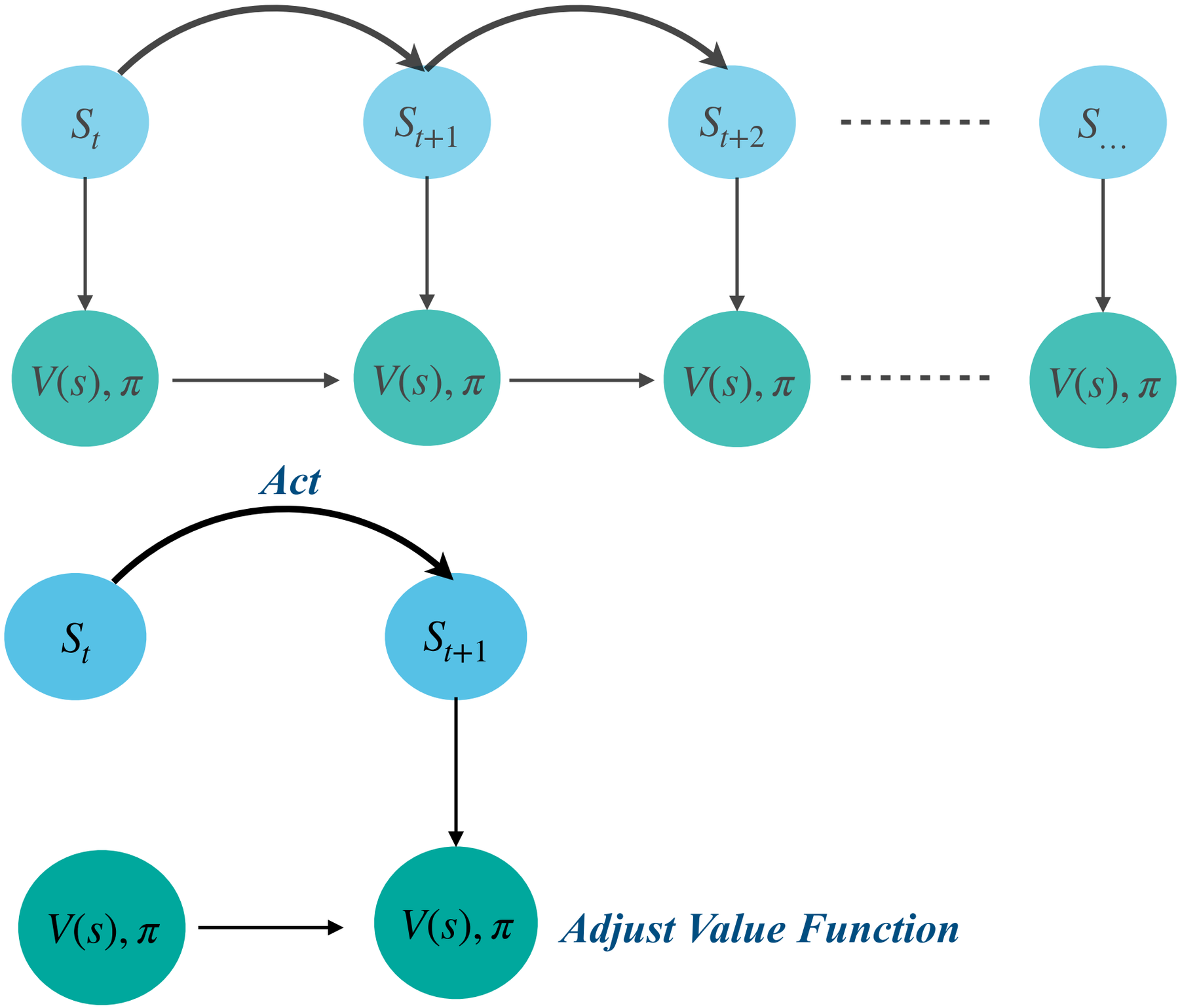
- ***Approximate the iterations*** (using sampling, cf. asynchronous dynamic programming, temporal-difference learning)

- ***Generalize*** the value function to unseen states using **function approximation**

Learning *online* using experience



Learning *online* using experience



Monte-Carlo Learning

- The idea of MC methods is to **learn directly from the episodes of experiences** i.e. learn from complete episodes which involves no bootsrapping.
- Use empirical return from complete episodes
- MC methods are model-free i.e. no knowledge of MDP transitions or rewards is give
- It is well suited for episodic tasks only wherein the value function = mean return of the episodes.
- Experience is divided over episodes, and all episodes eventually terminate.

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t))$$

Learning rate **Target**

Temporal-Difference (TD) Learning

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t))$$

- One can re-write the update based on the ***change in the prediction from one moment to the next***, called the ***temporal difference***

$$\delta_t = R_{t+1} + \gamma V(s_{t+1}) - V(s_t) \qquad \textbf{TD error}$$

Temporal-Difference (TD) Learning

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t))$$

- One can re-write the update based on the ***change in the prediction from one moment to the next***, called the ***temporal difference***

$$\delta_t = R_{t+1} + \gamma V(s_{t+1}) - V(s_t) \qquad \textbf{TD error}$$

$$V(s_t) \leftarrow V(s_t) + \alpha(\underline{R_{t+1} + \gamma V(s_{t+1})} - V(s_t))$$

Learning rate

TD Target

Q Learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[R_{t+1} + \max_{a'} \gamma Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

Learning rate

Target

In large state spaces: need approximation

$$\hat{Q}^{\pi}(s,a) = \sum_{i}^{d} \theta_i \phi_i(s,a)$$

Feature vector

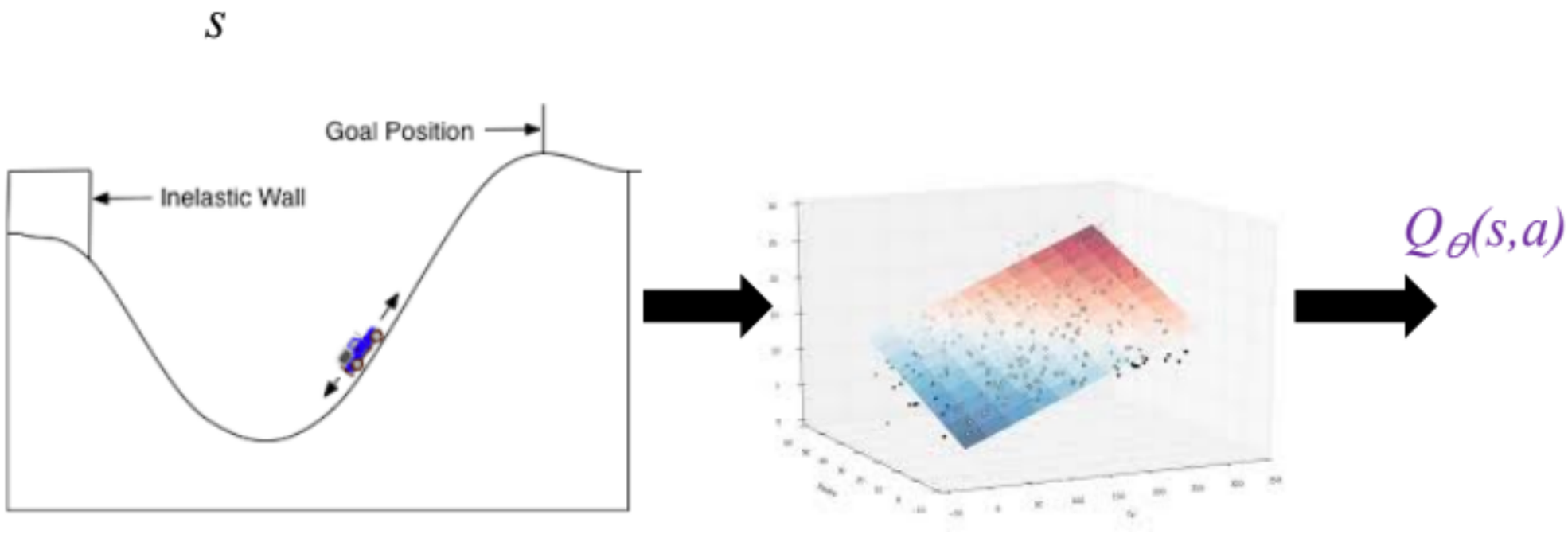
In large state spaces: need approximation

$$\hat{Q}^{\pi}(s,a) = \sum_{i}^{d} \theta_i \phi_i(s,a)$$

Feature vector

***Challenge*: Finding Good Features**

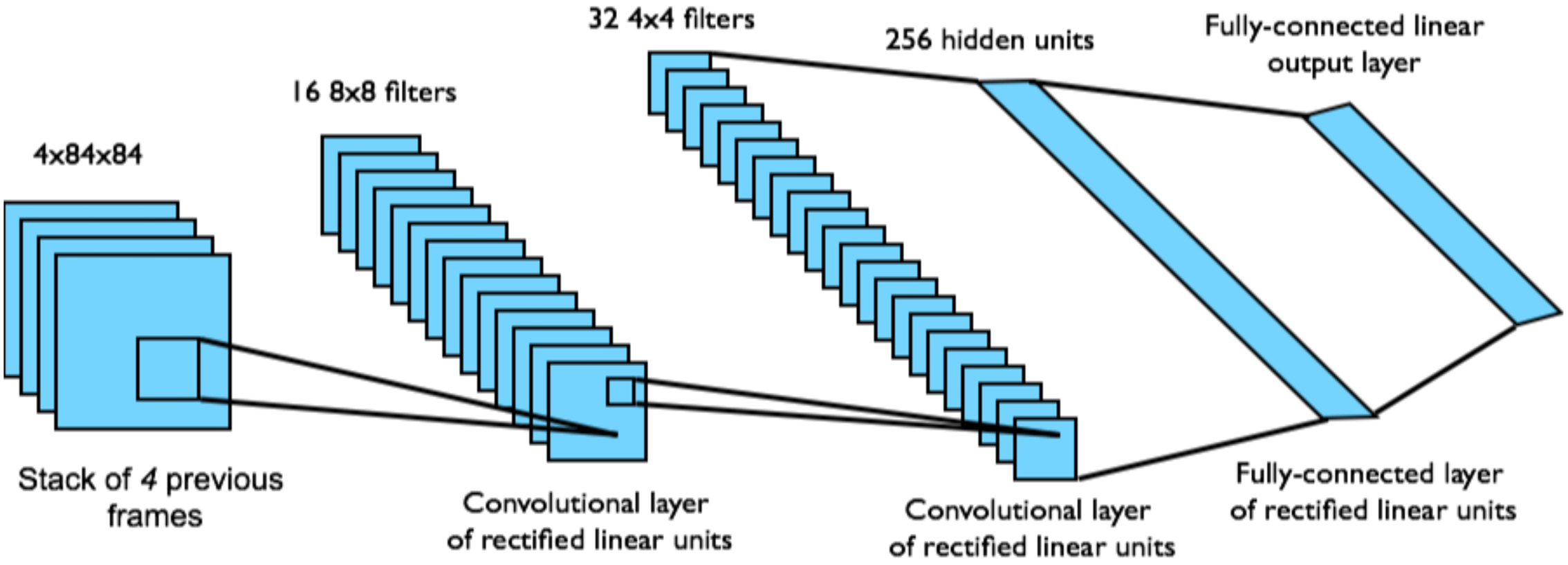
Learning representations for RL



Original state **Linear function**

(Fig. from & Joelle Pineau's slides)

Deep Reinforcement Learning



(Mnih et. al)

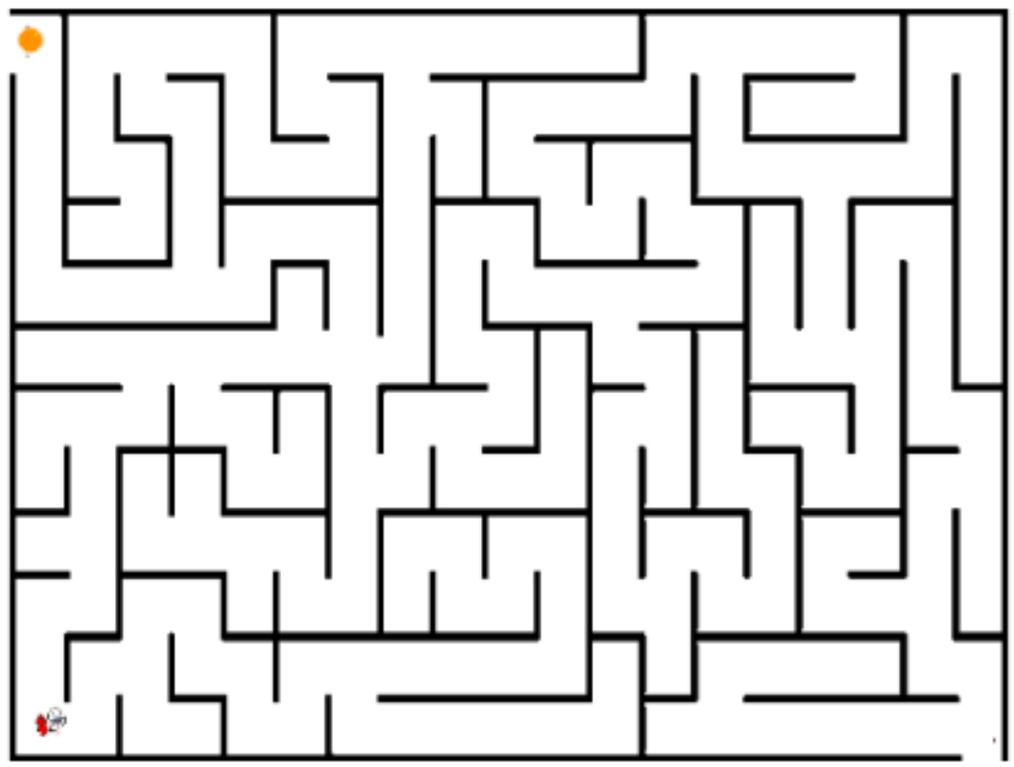
Key Challenges in RL: Exploration/Exploitation

- Suppose you form estimates

$$\hat{Q}(s,a)$$

- The ***greedy*** action at time t is:

$$a_t^* = \texttt{argmax}\hat{\texttt{Q}}(\texttt{s},\texttt{a})$$



Key Challenges in RL: Exploration/Exploitation

- Suppose you form estimates

$$\hat{Q}(s,a)$$

- The ***greedy*** action at time t is:

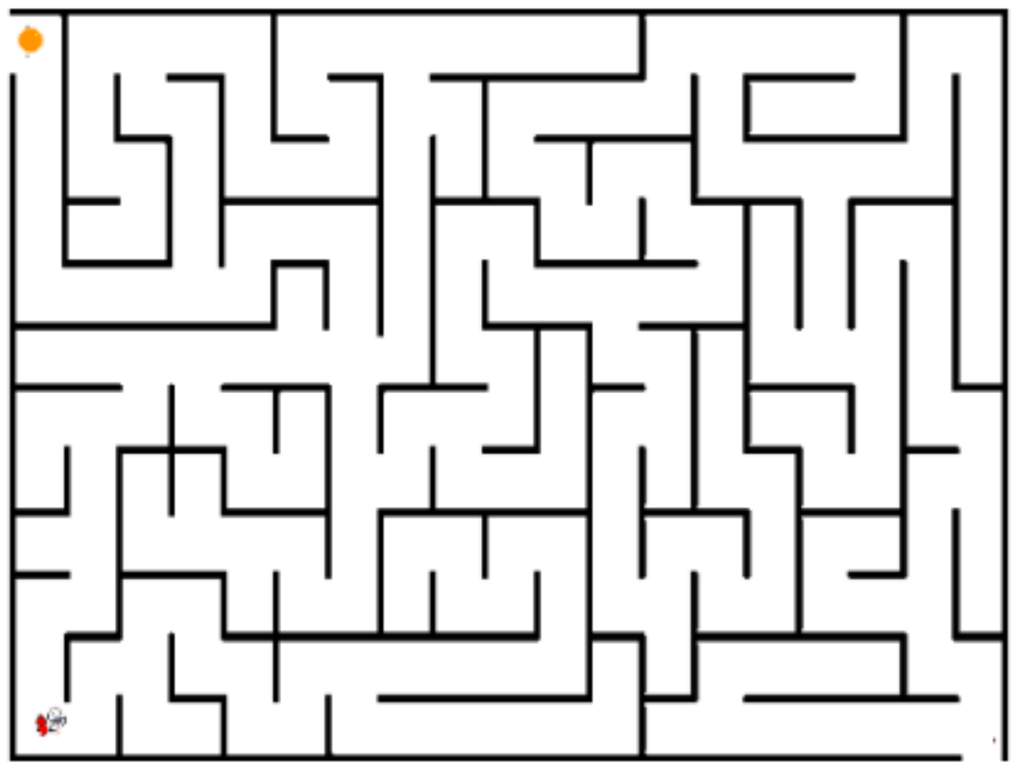
$$a_t^* = \texttt{argmax}\hat{\texttt{Q}}(\texttt{s},\texttt{a})$$

- ***Exploitation*** is acting greedily with respect to the best you know

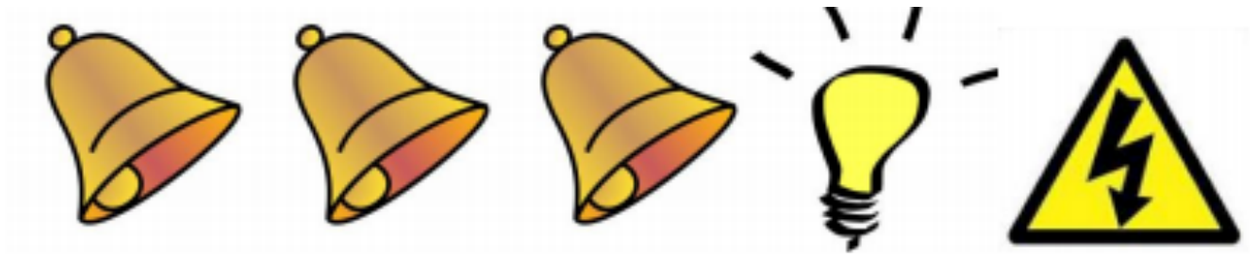
- ***Exploration*** is not the greedy action

- You can't exploit all the time, you can't explore all the time

- You can never stop exploring,
  but you should always reduce exploring

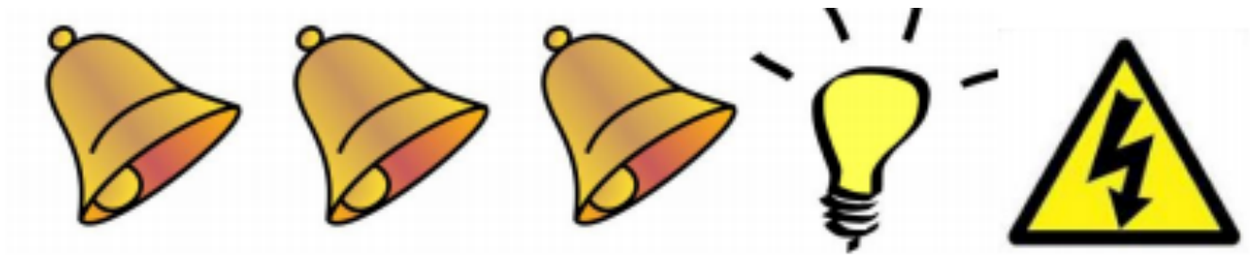


Key Challenges in RL: Credit Assignment



(Fig. from & David Silver's slides)

Key Challenges in RL: Credit Assignment



- **How to credit for an outcome over a sequence of steps that led to the outcome?**
- Deep learning: sequence of activations leading to loss
- Reinforcement learning: sequence of actions leading to reward

Questions

DQN Notebook

https://colab.research.google.com/drive/1GKatNQWF2lCVPsS_Sk29835Th1v1xzQd