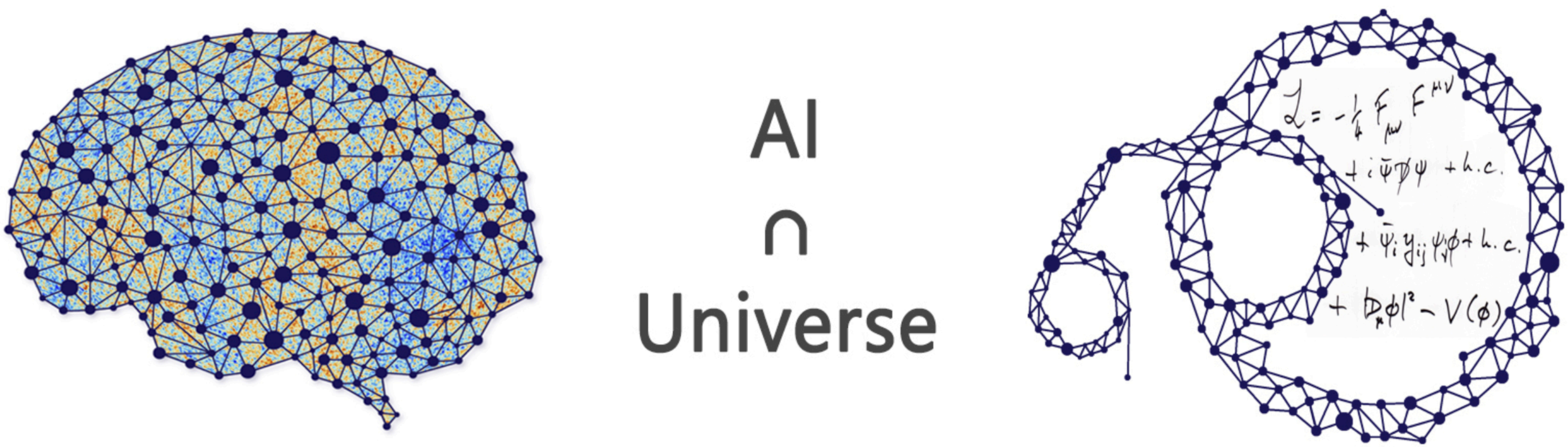
PHY 835: Machine Learning in Physics

Lecture 21: Reinforcement Learning Part 1

April 9, 2024



Gary Shiu

Introduction

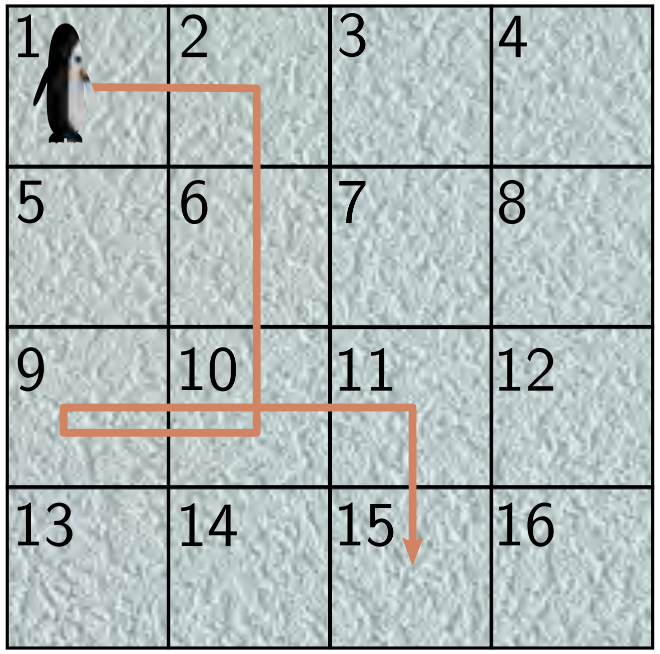
- Reinforcement leaning is a sequential decision making framework in which agents learned to perform actions in an environment with the goal of maximizing rewards.
- RL controls the **actions** of an **agent** in an **environment** to maximize the **reward**.
- RL applications: Go/Chess/Atari, robotics, financial trading, string theory, optimal experimental design, ….
- RL is often used when problem involves searching a large configuration space.
- Other related approaches: genetic algorithms (GAs) which mimic natural selection. RL and GA work complementarily like nurture and nature.
- References: Sutton and Barto: http://incompleteideas.net/book/the-book-2nd.html, Simon Prince, Understanding Deep Learning: https://udlbook.github.io/udlbook/, Fabian Ruehle, Data science applications to string theory, https://inspirehep.net/literature/1779782
- https://github.com/Farama-Foundation/Gymnasium (formerly https://github.com/openai/gym)

Challenges of RL

- Illustrate the challenges with chess game. A reward of +1, −1, or 0 is given at the end of the game if the agent wins, loses, or draws and 0 at every other time step. The challenges:
  - The reward is sparse; we must play an entire game to receive feedback.
  - **Temporal credit assignment problem**: The reward is temporally offset from the action that caused it; a decisive advantage might be gained thirty moves before victory. We must associate the reward with this critical action. (other examples?)
  - **The environment is stochastic**; the opponent doesn't always make the same move in the same situation, so it's hard to know if an action was truly good or just lucky.
  - **Exploration-exploitation trade-off**: The agent must balance exploring the environment (e.g., trying new opening moves) with exploiting what it already knows .

Markov Processes

- In RL, we learn a **policy** that maximizes the expected return in a Markov decision process.
- The word Markov implies that the probability of being in a state depends only on the previous state and not on the states before.
- The changes between states are captured by the transition probabilities $Pr(s_{t+1} \mid s_t)$ of moving to the next state s_{t+1} given the current state s_t, where t indexes the time step.
- A Markov process is an evolving system that produces a sequence $s_1, s_2, s_3, \ldots$ of states.

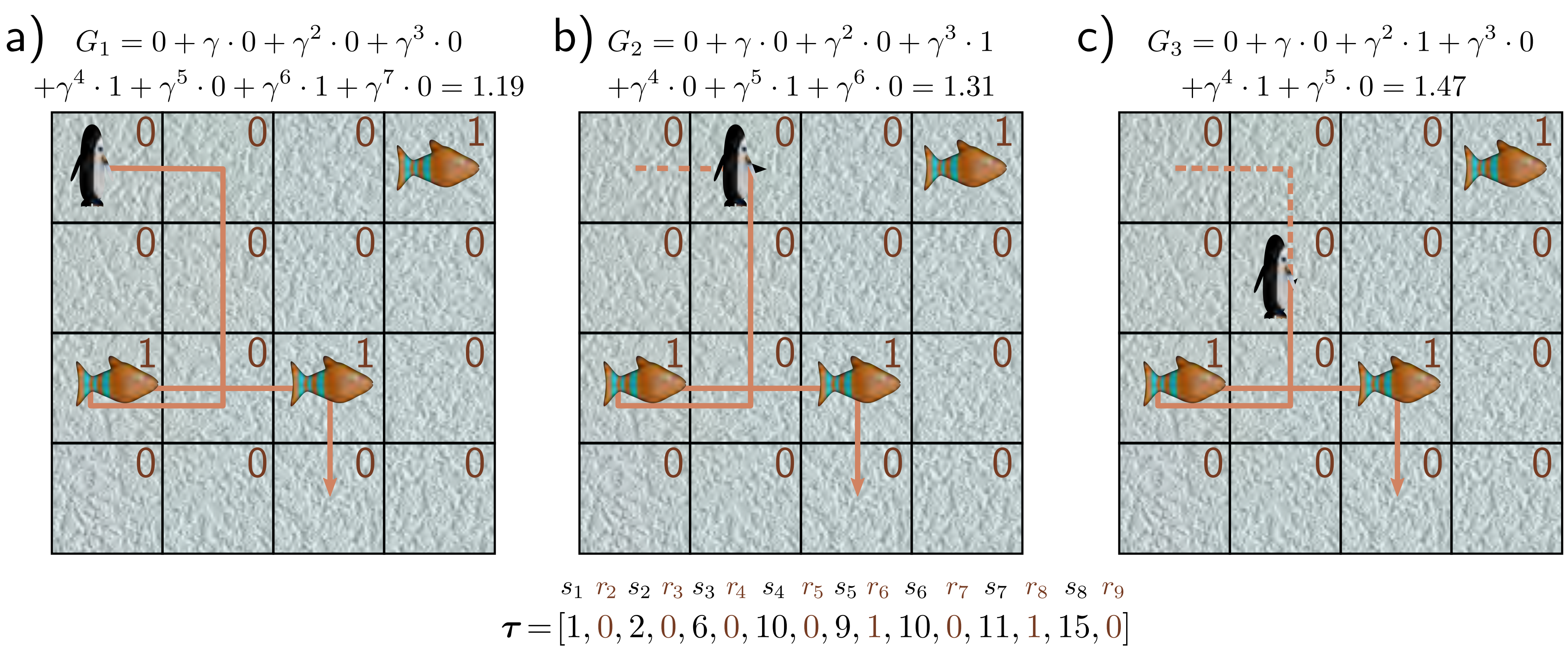


$$\begin{array}{l} \quad\; s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8 \\ \tau = [1, 2, 6, 10, 9, 10, 11, 15] \end{array}$$

Markov Reward Processes

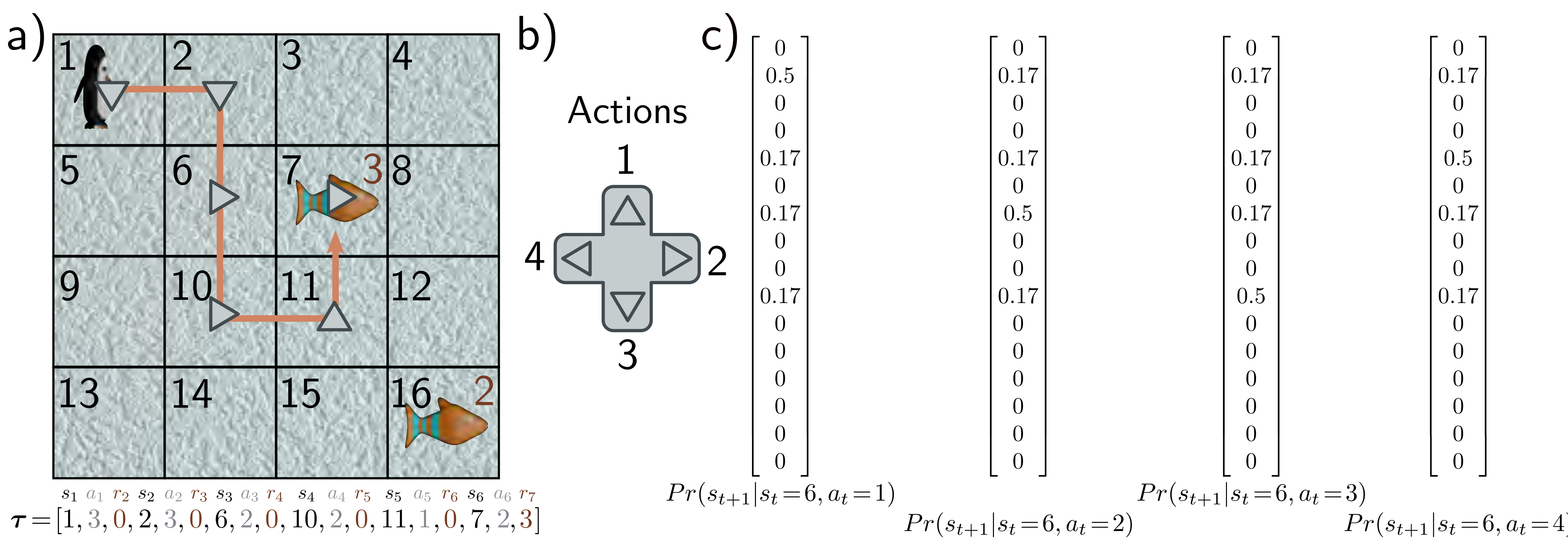
- A Markov reward process also includes a distribution $Pr(r_{t+1} \mid s_t)$ over the possible rewards r_{t+1} received at the next step, given s_t.
- Introduce a discount factor $\gamma \in (0,1]$ to compute the **return** G_t:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}.$$



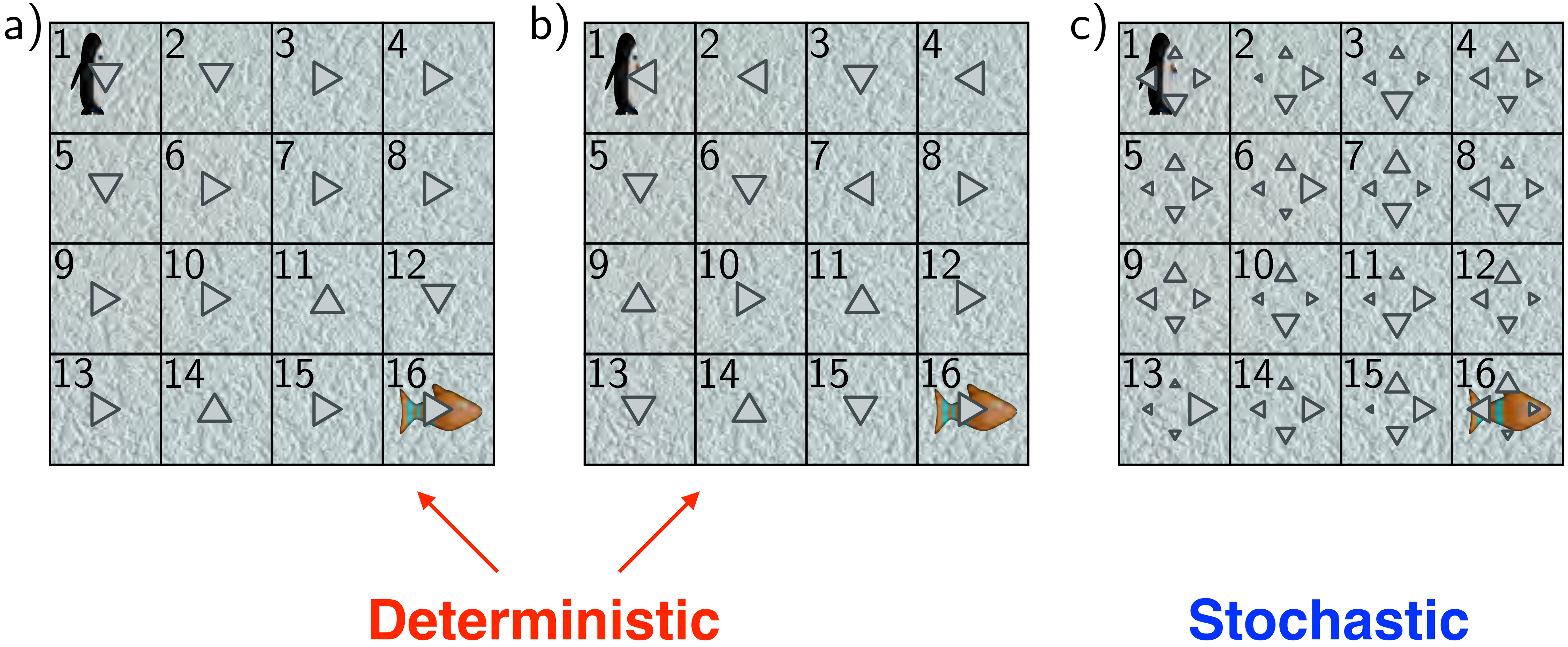
Markov Decision Processes

- A Markov decision process (MDP) adds a set of possible action a_t at each step which changes the transition probabilities $Pr(s_{t+1} \mid s_t, a_t)$.
- The rewards can also depend on the action: $Pr(r_{t+1} \mid s_t, a_t)$.
- MDP produces a sequence $s_1, a_1, r_2, s_2, a_2, r_3, s_3, a_3, \ldots$ of states, actions & rewards. The entity that performs the actions is the **agent**.



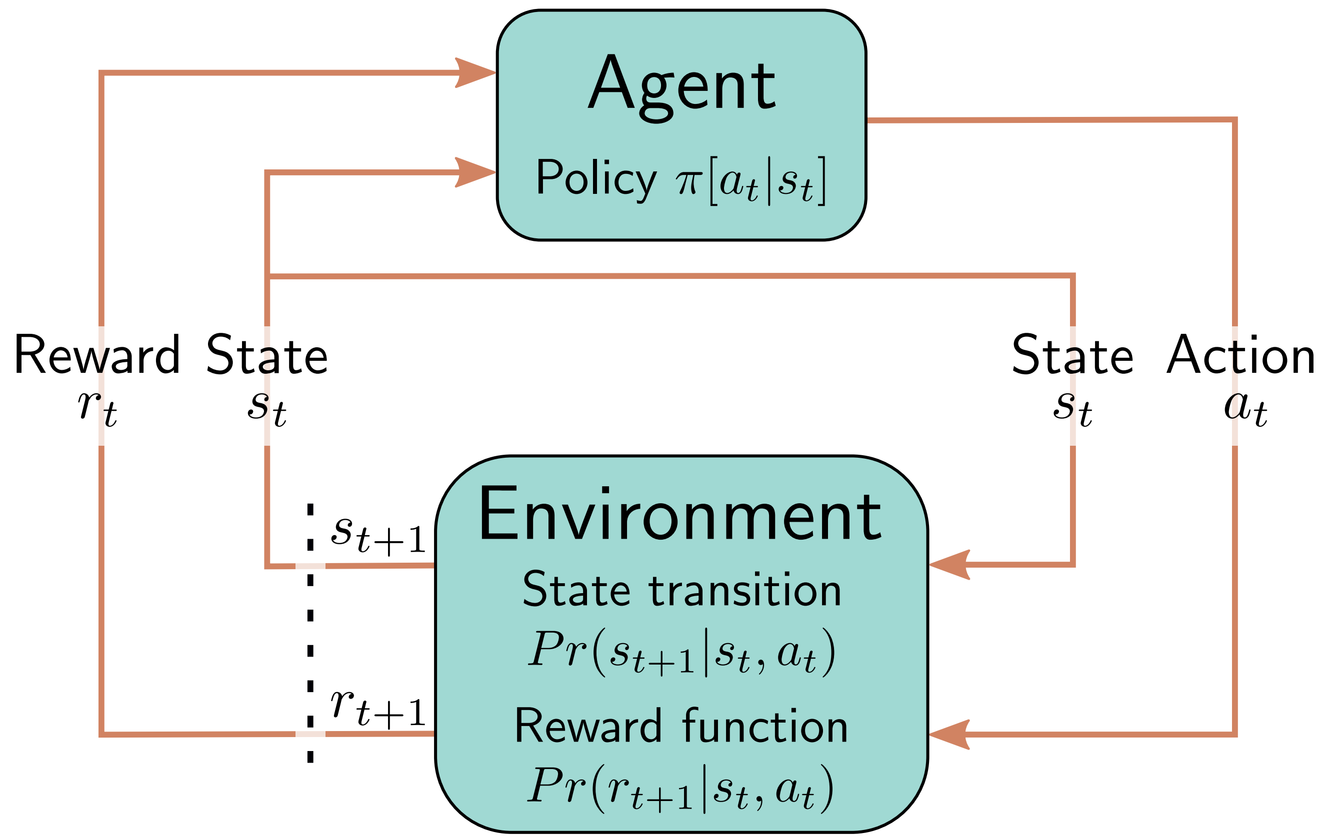
Policy

- The rules that determine the agent's action are known as the **policy**.
- The policy can be **deterministic** (one action for a given state) or **stochastic** (a probability distribution over each possible action):



Reinforcement Learning Loop

- The environment and the agent form a loop:



- The agent receives the state and reward from the last time step. Based on the policy, the agent chooses the next action.
- The environment then assigns the next state according to $Pr(s_{t+1} \,|\, s_t, a_t)$ and the reward according to $Pr(r_{t+1} \,|\, s_t, a_t)$.

Expected return: state and action values

- The return G_t depends on the state s_t and the policy $\pi[a \mid s]$
- Characterize how "good" a state is under a given policy π by considering the expected return $v[s_t \mid \pi]$. **State-value** function (long-term return on average from sequences that starts from s_t):

$$v[s_t|\pi] = \mathbb{E}\Big[G_t|s_t, \pi\Big].$$

- **Action value** or state-action value function $q[s_t, a_t \mid \pi]$ is the expected return from executing action a_t in state s_t:

$$q[s_t, a_t|\pi] = \mathbb{E}\Big[G_t|s_t, a_t, \pi\Big].$$

- Through this quantity, RL algorithms connect future rewards to current actions (i.e., resolve the temporal credit assignment problem).

Optimal Policy

- We want a policy that maximizes the expected return.
- For MDPs, there ∃ a deterministic, stationary (depends only on the current state, not the time step) policy that maximizes the value of every state.
- If we know this optimal policy, then we get the optimal state-value function:

$$v^*[s_t] = \max_{\pi}\bigg[\mathbb{E}\Big[G_t|s_t,\pi\Big]\bigg].$$

- Similarly, the optimal state-action value function:

$$q^*[s_t,a_t] = \max_{\pi}\left[\mathbb{E}\Big[G_t|s_t,a_t,\pi\Big]\right].$$

- Turning this around, if we knew the optimal action-values, we can derive the optimal policy. RL algorithms estimate the action and policy alternately.

$$\pi[a_t|s_t] \leftarrow \underset{a_t}{\operatorname{argmax}}\Big[q^*[s_t,a_t]\Big].$$

Tabular RL

- RL algorithms that do not rely on function approximation.
- **Model-based methods** use the MDP structure explicitly and find the best policy from the transition matrix $Pr(s_{t+1} \mid s_t, a_t)$ and reward $r[s, a]$.
- If the transition matrix & reward are known (often not), a straightforward optimization problem is dynamic programming.
- It not, they must first be observed from observed MDP trajectories.
- **Model-free methods** fall into two classes:
    - Value estimation - estimate the optimal state-action value and then assign the policy according to the action with the greatest value.
    - Policy estimation - estimate the optimal policy using gradient descent w/o the intermediate steps of estimating the model or values.

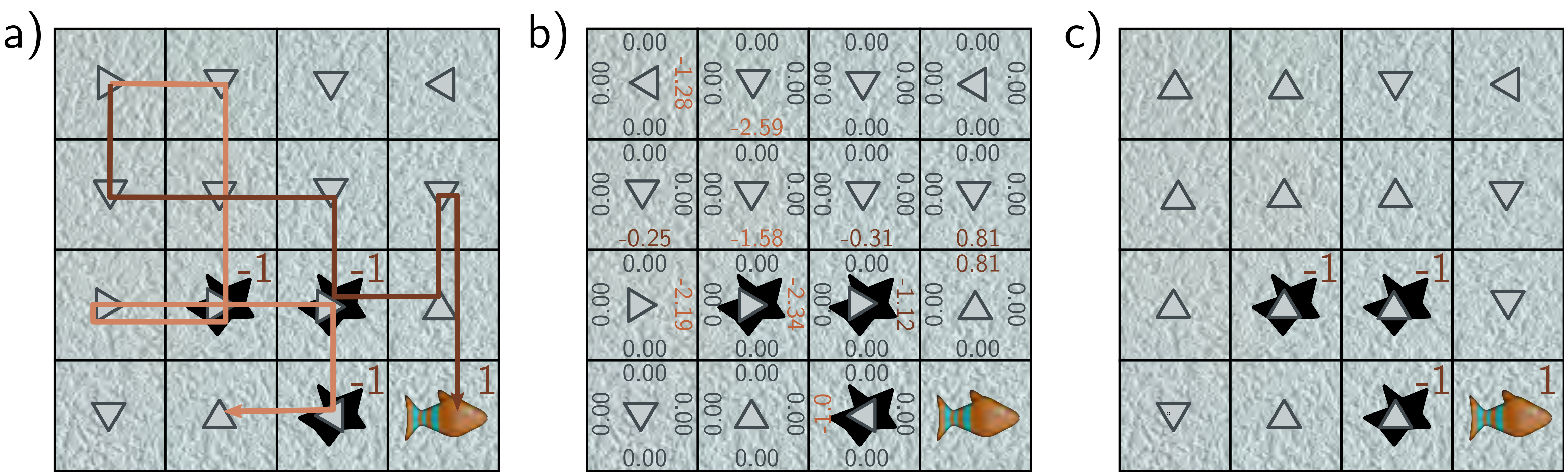
Tabular RL

- **Monte Carlo** methods simulate many trajectories through the MDP for a given policy to gather information to improve this policy.
- **Temporal difference** methods update the policy while the agent traverses the MDP.
- We will later contrast tabular RL algorithms with the use of deep learning in RL that does not require storing the large transition matrix.

Monte Carlo Methods

- Alternate between computing the action values (based on repeatedly sampling trajectories) & updating the policy (based on action values).
- The action value is estimated as the average of the empirical returns.
- The policy is updated by choosing the action with the maximum value at each state:

$$\pi[a|s] \leftarrow \underset{a}{\operatorname{argmax}}\Big[q[s,a]\Big]$$



On/off-policy methods

- **On-policy** method: the current best policy is used to guide the agent through the environment.
- It is not possible to estimate the value of actions that have not been used, & there is nothing to encourage the algorithm to explore them.
- **Exploring starts**: episodes with all possible state-action pairs are initiated, so every combination is observed at least once. (impossible for large configuration space).
- ϵ-greedy policy: random action is taken with ϵ probability and optimal action with $1 - \epsilon$ probability (exploitation/exploration trade-off).
- **Off-policy** method: the optimal policy π (the target policy) is learned based on episodes generated by a different behavior policy π'.
- We want π' to explore the environment (stochastic) and the learned policy π to be efficient.

Temporal difference methods

- Update the values/policy while the agent traverses the states of MDP.
- **SARSA** (State-Action-Reward-State-Action) is an on-policy algorithm with update:

$$q[s_t, a_t] \leftarrow q[s_t, a_t] + \alpha\Big(r[s_t, a_t] + \gamma \cdot q[s_{t+1}, a_{t+1}] - q[s_t, a_t]\Big),$$

where $\alpha \in \mathbb{R}^+$ is the learning rate. The bracketed term is TD error.

- **Q-learning** is an off-policy algorithm with update:

$$q[s_t, a_t] \leftarrow q[s_t, a_t] + \alpha\Big(r[s_t, a_t] + \gamma \cdot \max_a\big[q[s_{t+1}, a]\big] - q[s_t, a_t]\Big),$$

where the choice of action is derived from a different policy π'.

- In both cases, the policy is updated by maximizing the action values:

$$\pi[a|s] \leftarrow \operatorname*{argmax}_a\Big[q[s, a]\Big]$$