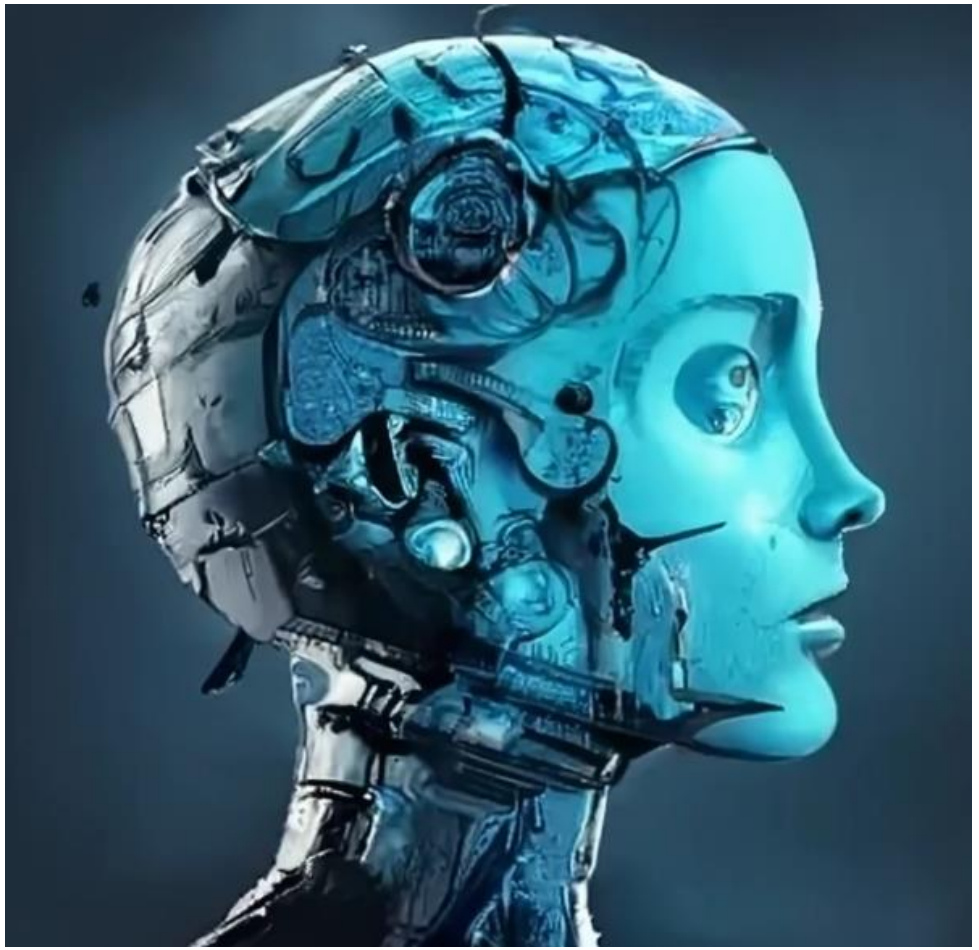
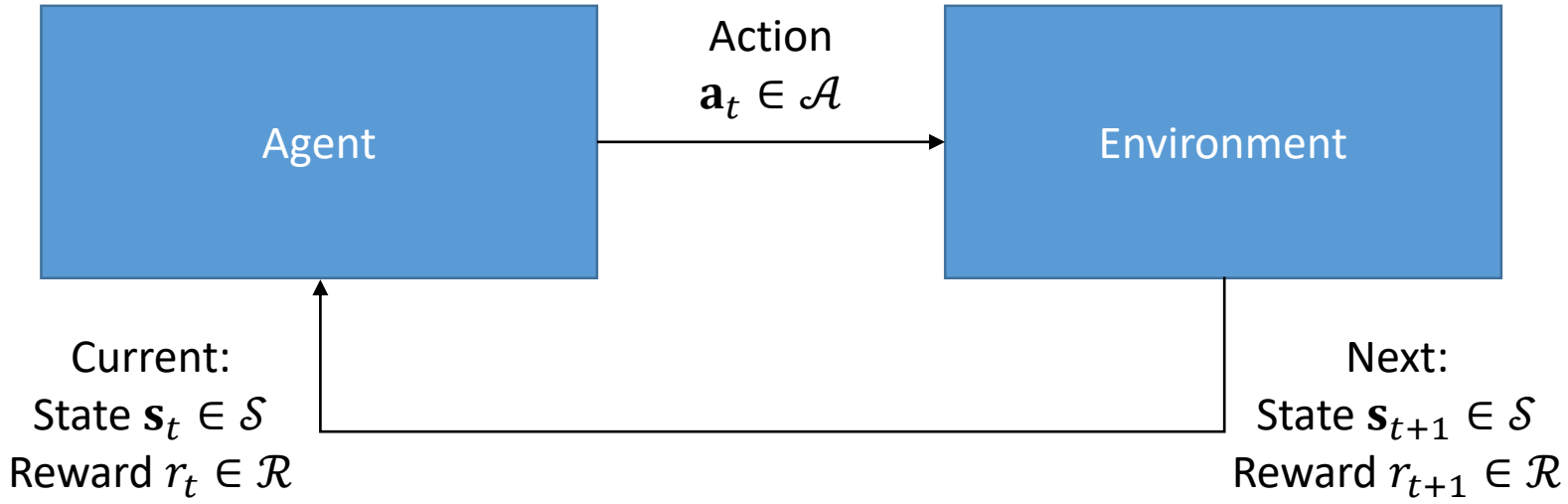
Machine Intelligence

Reinforcement Learning

Dr. Cory Merkel



The Reinforcement Learning Problem



- The goal of reinforcement learning is to learn a *policy* $\pi(\mathbf{s}, \mathbf{a})$ that is a conditional probability of taking an action $\mathbf{a}$ given some state of an environment $\mathbf{s}$
  - The policy is learned by exploring the environment and receiving positive or negative rewards based on actions
  - The agent tries to maximize its future rewards

Exploration vs. Exploitation

- A key concept in reinforcement learning is the exploration vs. exploitation dilemma
- Imagine you are lost in the wilderness and you come across an abandoned cabin near a fresh water stream
  - What should you do?
    - ❑You can stay in the cabin indefinitely and *exploit* the shelter and resources, but in this case you will never make it back home
    - ❑You can leave the cabin and *explore* your surroundings, potentially making it back to civilization, but you may make your situation worse by being exposed to the elements and not having any water to drink
  - We generally want to have a little of each, adjusting the levels based on our current situation
    - ❑Having a policy that is *stochastic* allows us to try out what are believed to be suboptimal actions in order to explore the possibility that they will ultimately lead to high rewards

Returns

- When learning a policy, we generally want to maximize the expected $return$, defined

$$R_t = f(r_{t+1}, r_{t+2}, \dots)$$

- In the simplest case, this could be just be $R_t = \sum_{i=t+1}^{T} r_i$
  - Here, there is some final timestep defined (e.g. in a game with an ending position)
    - Each time we go from $t = 0$ to $t = T$, we call it an *episode* (e.g. a single play of the game)

Returns

- More generally, we may wish to favor rewards that come sooner:

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}$$

- Here, $\gamma \in [0,1]$ is called the *discount rate*.
  - Smaller γ causes temporally distant rewards to be counted less
  - $\gamma < 1$ allows T to be infinite, for problems where there is no defined end
    - ❑Continual tasks vs. episodic tasks

Markov Property

- The Markov property is a “nice” property of the environment and the influence of our actions

$$p(\mathbf{s}_{t+1} = \mathbf{s}, r_{t+1} = r | \{\mathbf{s}_\tau, \mathbf{a}_\tau, r_\tau\} \forall \tau = 0,1 \dots, t) = p(\mathbf{s}_{t+1} = \mathbf{s}, r_{t+1} = r | \mathbf{s}_t, \mathbf{a}_t)$$

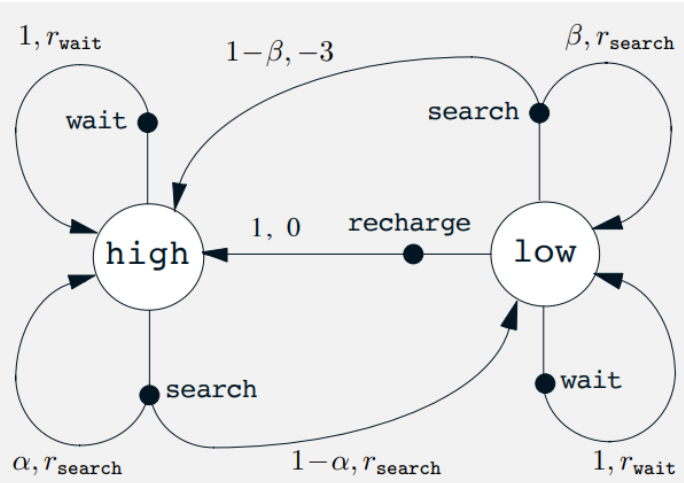
  - The probability of moving to a particular state only depends on the current state and the current action being applied
    - e.g. the current position of a game of checkers is all the matters for the future course of the game

Markov Decision Processes

- Markov Decision Processes (MDP) are RL problems that satisfy the Markov property
- Example: Recycling robot with states of high and low battery:

State Transition Probability → $\mathcal{P}^{\mathbf{a}}_{\mathbf{ss}'}$ ← $\mathcal{R}^{\mathbf{a}}_{\mathbf{ss}'}$ Expected Reward

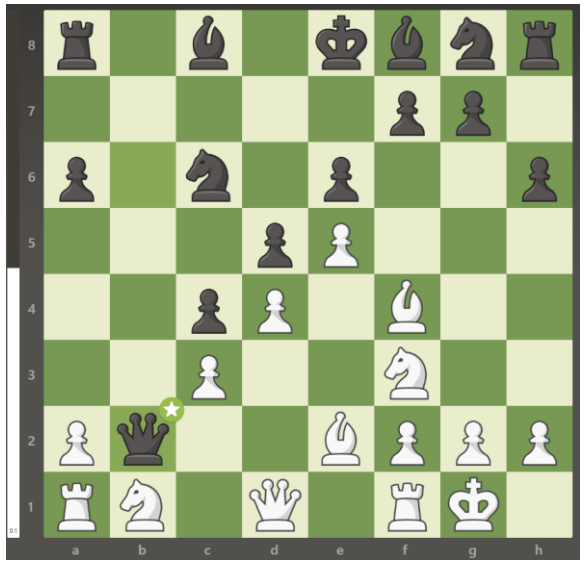
s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	$r_{\texttt{search}}$
high	search	low	$1-\alpha$	$r_{\texttt{search}}$
low	search	high	$1-\beta$	-3
low	search	low	β	$r_{\texttt{search}}$
high	wait	high	1	$r_{\texttt{wait}}$
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	$r_{\texttt{wait}}$
low	recharge	high	1	0
low	recharge	low	0	-



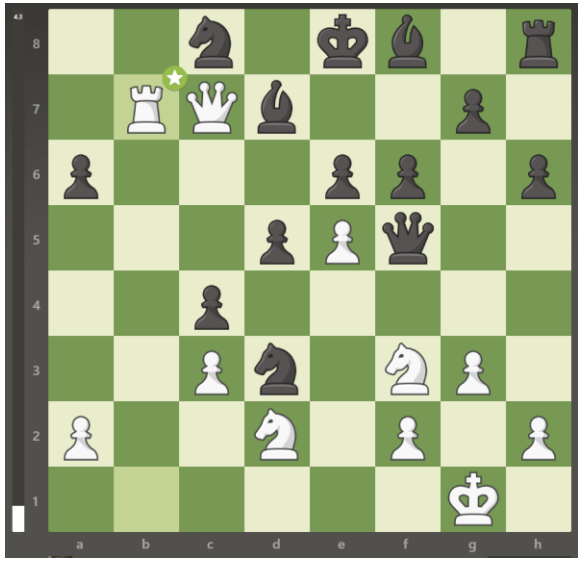
Sutton and Barto, *Reinforcement Learning*

Value Functions

- Remember, our current goal is to find a *policy* that maximizes a *return*
  - As such, it will be helpful for us to know how "good" it is to be in a particular state



Low value (for white)
Black could take rook



High(er) value (for white)
Threatening mate

Value Functions

- Formally, we can define the *state-value* function for an MDP with a given policy as

$$V^{\pi}(\mathbf{s}) \equiv \mathbb{E}_{\pi}[R_t|\mathbf{s}_t = \mathbf{s}] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \,|\mathbf{s}_t = \mathbf{s}\right]$$

- We can also define the *action-value* function for policy π:

$$Q^{\pi}(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\pi}[R_t|\mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \,|\mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}\right]$$

Bellman Equation

- Notice the following recursive relationship:

$$V^{\pi}(\mathbf{s}) \equiv \mathbb{E}_{\pi}[R_t|\mathbf{s}_t = \mathbf{s}]$$

$$= \mathbb{E}_{\pi}\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \,|\mathbf{s}_t = \mathbf{s}\right]$$

$$= \mathbb{E}_{\pi}\left[r_{t+1} + \gamma \sum_{k=0}^{T} \gamma^k r_{t+k+2} \,|\mathbf{s}_t = \mathbf{s}\right]$$

Note that we have to average over all possible combinations of the action, next state, and reward!

$$= \sum_{\mathbf{a}} \pi(\mathbf{s}, \mathbf{a}) \sum_{\mathbf{s}'} \mathcal{P}^{\mathbf{a}}_{\mathbf{ss}'} \left[\mathcal{R}^{\mathbf{a}}_{\mathbf{ss}'} + \gamma \mathbb{E}_{\pi}\left[\sum_{k=0}^{T} \gamma^k r_{t+k+2} \,|\mathbf{s}_{t+1} = \mathbf{s}'\right]\right]$$

$$V^{\pi}(\mathbf{s}) = \sum_{\mathbf{a}} \pi(\mathbf{s}, \mathbf{a}) \sum_{\mathbf{s}'} \mathcal{P}^{\mathbf{a}}_{\mathbf{ss}'} \left[\mathcal{R}^{\mathbf{a}}_{\mathbf{ss}'} + \gamma V^{\pi}(\mathbf{s}')\right]$$

This allows us to relate the value of a state to the value of its successor state.

Bellman Optimality

- The optimal policy will correspond to an optimal state-value function:

$$V^*(\mathbf{s}) = \max_{\pi} V^{\pi}(\mathbf{s}) \;\; \forall \, \mathbf{s} \in \mathcal{S}$$

and an optimal action-value function

$$Q^*(\mathbf{s}, \mathbf{a}) = \max_{\pi} Q^{\pi}(\mathbf{s}) \;\; \forall \, \mathbf{s} \in \mathcal{S}, \mathbf{a} \in \mathcal{A}$$

Bellman Optimality

- Note that the value of a state for an optimal policy π^* should correspond to the expected return for the best action in that state:

$$V^*(\mathbf{s}) = \max_{\mathbf{a} \in \mathcal{A}(\mathbf{s})} Q^{\pi^*}(\mathbf{s}, \mathbf{a})$$

$$= \max_{\mathbf{a}} \mathbb{E}_{\pi^*}[R_t | \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}]$$

$$= \max_{\mathbf{a}} \mathbb{E}_{\pi^*}\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \,|\mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}\right]$$

$$= \max_{\mathbf{a}} \mathbb{E}_{\pi^*}\left[r_{t+1} + \gamma \sum_{k=0}^{T} \gamma^k r_{t+k+2} \,|\mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}\right]$$

$$= \max_{\mathbf{a}} \mathbb{E}[r_{t+1} + \gamma V^*(\mathbf{s}_{t+1}) | \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}] = \max_{\mathbf{a}} \sum_{s'} \mathcal{P}^{\mathrm{a}}_{\mathbf{ss}'} \left[\mathcal{R}^{\mathbf{a}}_{\mathbf{ss}'} + \gamma V^*(\mathbf{s}')\right]$$

Bellman Optimality

- We can similarly define the optimality equation for the action-value, and our Bellman optimality conditions become

$$V^*(\mathbf{s}) = \max_{\mathbf{a}} \mathbb{E}[r_{t+1} + \gamma V^*(\mathbf{s}_{t+1}) | \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}] = \max_{\mathbf{a}} \sum_{\mathrm{s}'} \mathcal{P}^{\mathrm{a}}_{\mathbf{ss}'} \left[\mathcal{R}^{\mathbf{a}}_{\mathbf{ss}'} + \gamma V^*(\mathbf{s}') \right]$$

$$Q^*(\mathbf{s}, \mathbf{a}) = \mathbb{E}\left[r_{t+1} + \gamma \max_{\mathbf{a}'} Q^*(\mathbf{s}_{t+1}, \mathbf{a}') \, | \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a} \right] = \sum_{\mathrm{s}'} \mathcal{P}^{\mathrm{a}}_{\mathbf{ss}'} \left[\mathcal{R}^{\mathbf{a}}_{\mathbf{ss}'} + \gamma \max_{\mathbf{a}'} Q^*(\mathbf{s}', \boldsymbol{a}') \right]$$

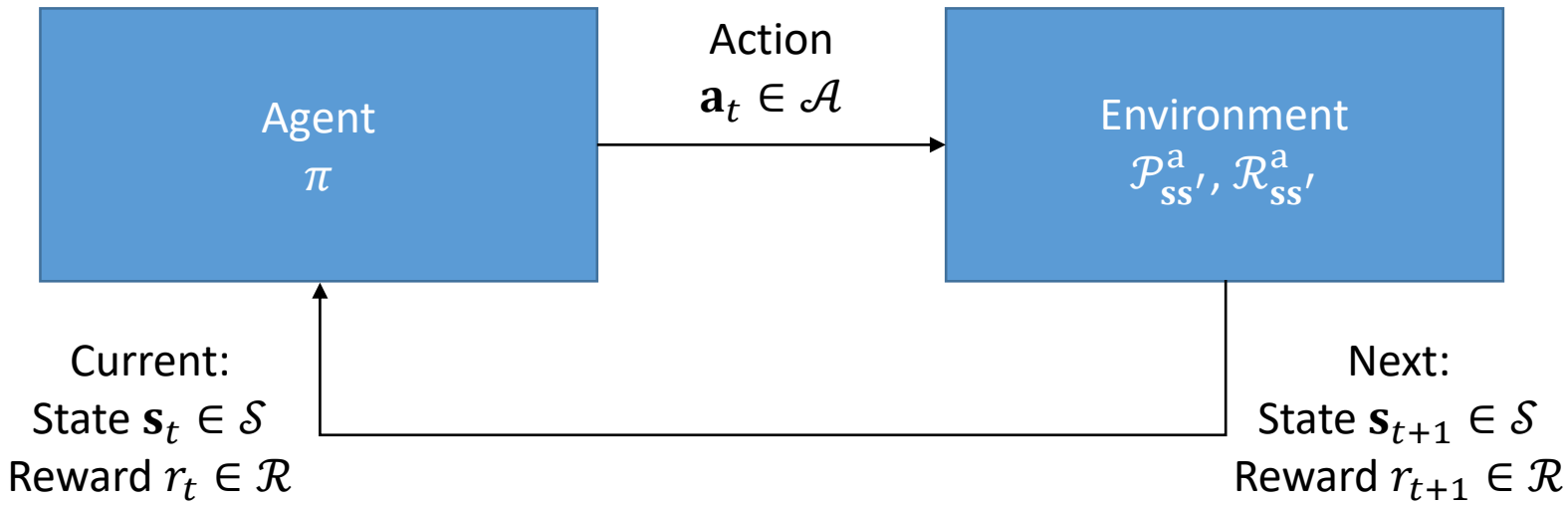
- $V^*(\mathbf{s})$ is the maximum expected return when we are in a state and take the *best* action
- $Q^*(\mathbf{s}, \mathbf{a})$ is the maximum expected return when we are in a state and take some action $\mathbf{a}$

Bellman Optimality

- Note that the Bellman optimality equations are defined for a particular state or state, action pair
- For the state-value $V^*(\mathbf{s})$ can be written for N different states (assuming the state space has N elements)
  - This give N equations in N unknowns
    - ❑Solvable if we know the environment dynamics $\mathcal{P}^{\mathbf{a}}_{\mathbf{ss}'}$ and $\mathcal{R}^{\mathbf{a}}_{\mathbf{ss}'}$
    - ❑Once $V^*(\mathbf{s})$ is known, we can choose an optimal policy by assigning actions that give us the maximum state-value
- Note that if we know $Q^*(\mathbf{s}, \mathbf{a})$, out job is even easier!
  - We just search over $\mathbf{a}$ to find the maximum without needing to know anything about the environment's dynamics

Reinforcement Learning Approaches

- We've seen the basic problem that we're trying to solve and some simplifying assumptions (e.g. Markov property), but how do we actually find π?
  - Model-based vs. model-free



Simple Approaches

- Brute force search
  - Try every possible policy and choose the best one
    - ❑Hardly ever feasible for real problems
- Stochastic search
  - Try several random policies and choose the best one
    - ❑Can be surprisingly effective for some problems

Less Simple Approaches

- Genetic algorithms - This technique mimics evolution through natural selection
  - Start with a large *population* of randomly-generated policies
  - Evaluate the *fitness* (performance) of each one
  - *Select* one or more from that population based on their fitness
  - Perform *crossover* to combine selected policies in different ways
    - e.g. Use probabilities from policy 2 when in state 1 and probabilities from policy 1 when in state 2
    - Repeat crossover until you have a new, different population
  - Randomly *mutate* some of the members of the new population
    - e.g. Modify their action probability values slightly
  - Repeat over several generations
- Simulated annealing
  - Follow a random trajectory through policy space, jumping to neighboring polices
    - How far you "jump" depends on 1.) How good the current policy is and 2.) the "temperature" of the annealing process, which decreases over time

Policy Gradients

- Recall that our goal is to find a policy that maximizes the expected return, starting from time 0 to the end of the episode:

$$\mathcal{L} = -\mathbb{E}\left[\sum_{k=0}^{T} \gamma^k r_{k+1}\right] = -\mathbb{E}[R(\tau)] = -\int_{\mathbb{T}} p(\tau) R(\tau) \mathrm{d}\tau$$

- Here, τ represents a trajectory of the agent through state space during a particular episode
  - $p(\tau)$ gives the probability of the agent taking a particular path

Policy Gradients

- REINFORCE algorithm:
  - Use policy gradients to train the parameters of a policy (gradient descent)
  - Policy can be represented as a neural network

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$

Sutton & Barto, *Reinforcement Learning*