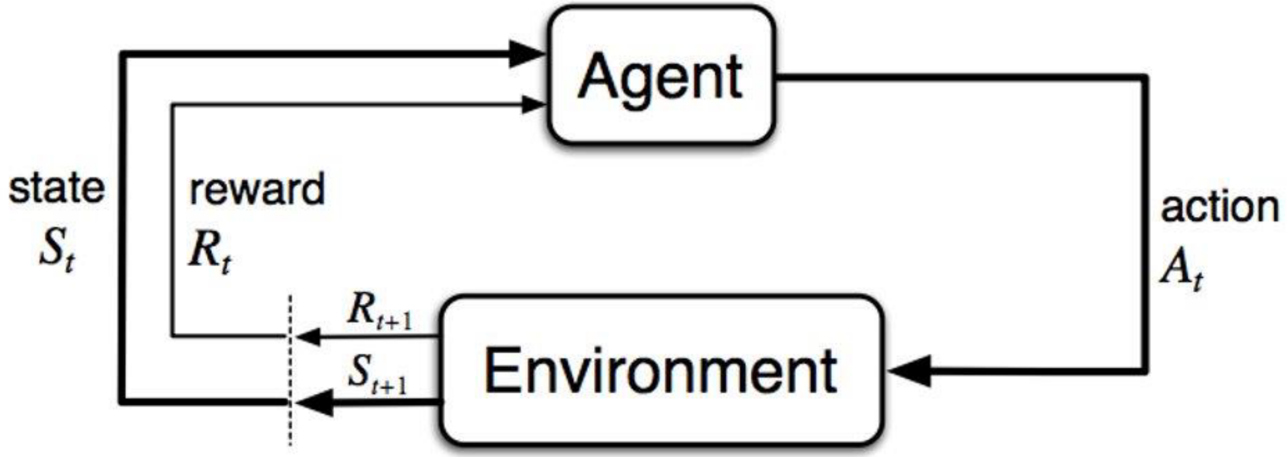
Policy Gradient Reinforcement Learning

Presented by Andrew Li

CSC2515 Tutorial – Fall 2021

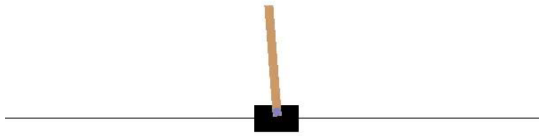
Past slides contributors: Tianshi Cao, Mohammad Firouzi
Adapted from slides by Sergey Levine and David Silver

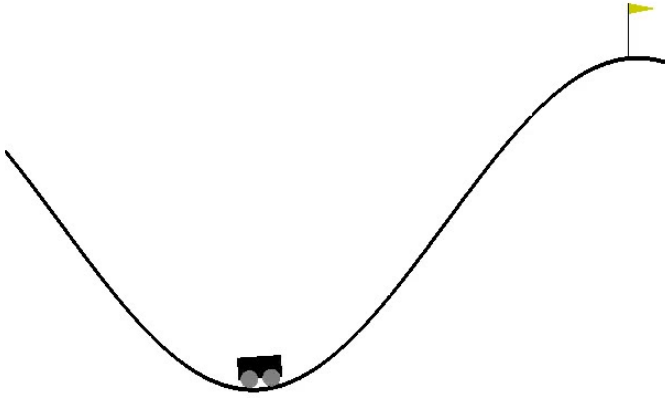
Reinforcement Learning

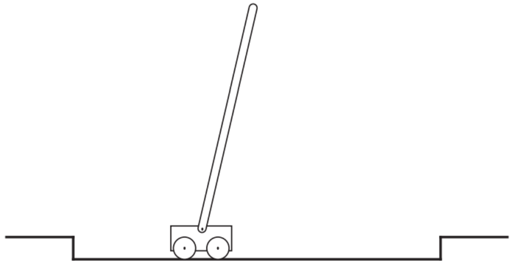


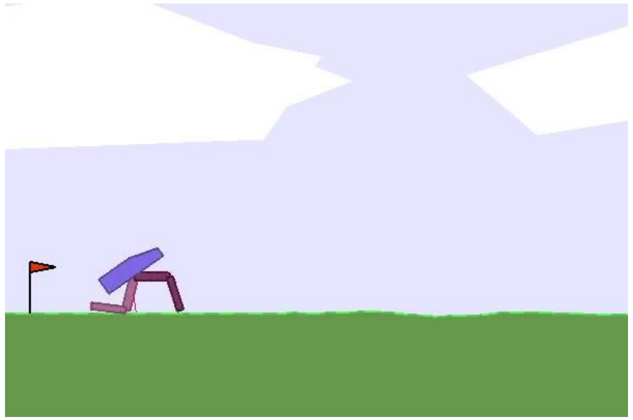
Examples

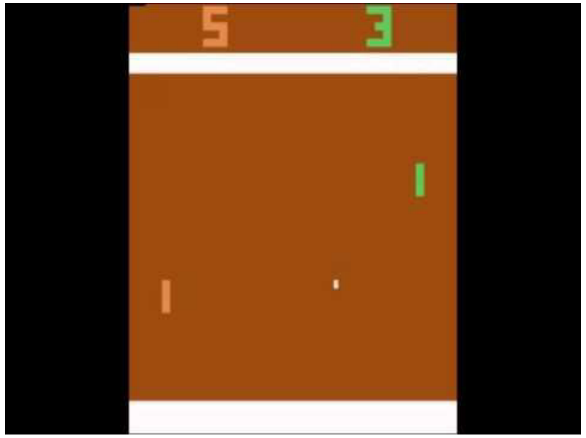
- MountainCar
- CartPole
- BipedalWalker
- Atari games









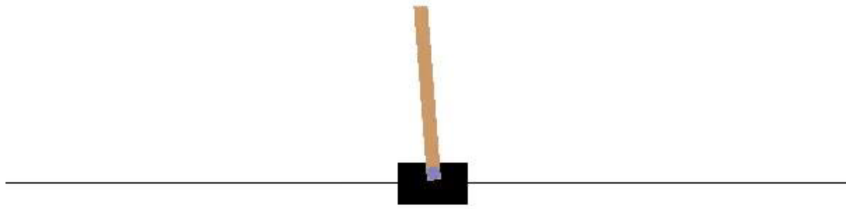


Reinforcement Learning Task

- Environment is a **Markov Decision Process.**
  - S: State Space
  - A: Action Space
  - $F(S' \mid S, A)$: State-action transition distribution
  - $\rho_0(S)$: Initial state distribution
  - $R(S)$: Reward function (can also depend on action)
  - γ : Discount factor $(0 \leq \gamma \leq 1)$

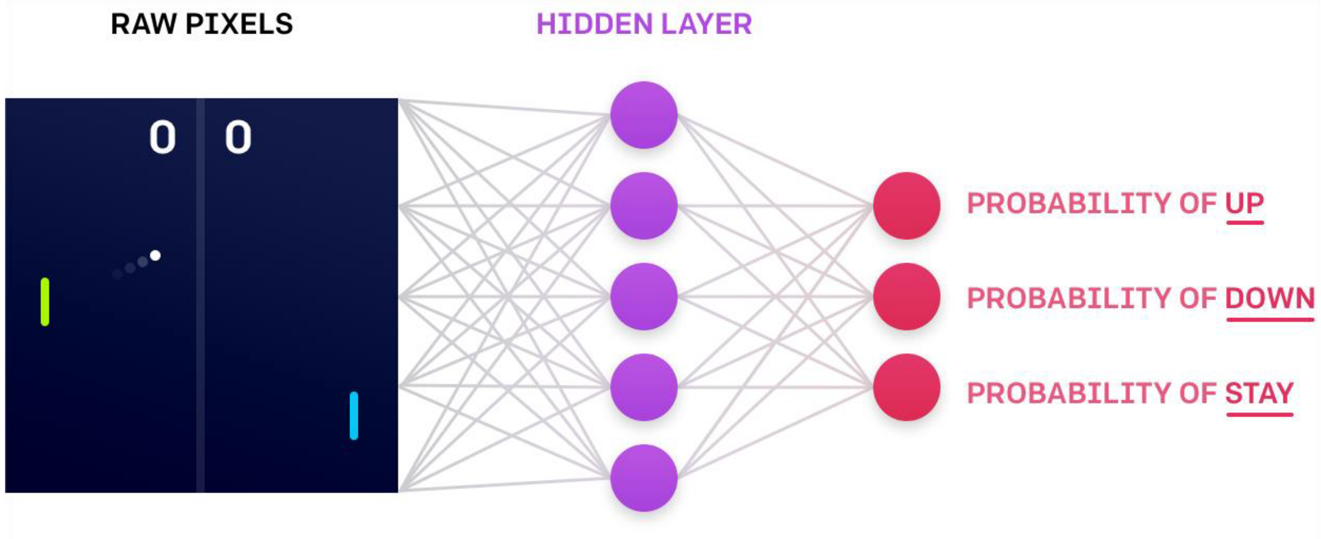
Task Example: CartPole

- Balance a pole attached to a cart by a joint for as long as possible.
  - S: position + velocity of cart and pole
  - A: push left or right
  - $F(s'|s, a)$: simulated physics (deterministic)
  - $\rho_0(s)$: cart and pole with near 0 velocity (slightly random)
  - $R(s)$: 1 if the pole is still standing
  - γ : 1



Policy

- A policy $\pi(a \mid s)$ is the agent's behaviour
- Maps states to actions (or a distr. of actions)
  - Deterministic policy
  - Stochastic policy



Goal of the Agent

Learn the policy $\pi(a \mid s)$ that maximizes *expected discounted reward.*

Expected discounted reward: $J(\pi) = \mathbb{E}_{s_t, a_t, r_t \sim \pi, F}\left[\sum_t \gamma^t r_t \right]$

Approaches to reinforcement learning

- Policy-based RL (focus of the tutorial)
  - Search directly for the optimal policy
  - This is the policy achieving maximum future reward
- Value-based RL (will be discussed briefly)
  - Estimate the optimal value function Q(s, a)
  - This is the maximum value achievable under any policy
- Model-based RL (will be discussed briefly)
  - Build a model of environment
  - Plan (e.g. by lookahead) using model
- State-of-the-art approaches generally combine flavours of all three

Value-based approach (in brief)

- A Q-value function is a prediction of future reward
  - "How much reward will I get from action *a* in state *s*?"
- Q-value function gives expected total reward
  - from state *s* and action *a*
  - under policy π
  - with discount factor γ

$$Q^{\pi}(s, a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \mid s, a\right]$$

- Q-value functions decompose into a Bellman equation:

$$Q^{\pi}(s, a) = \mathbb{E}_{s', a'}\left[r + \gamma Q^{\pi}(s', a') \mid s, a\right]$$

Optimal value function

- An optimal value function is the maximum achievable value

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = Q^{\pi^*}(s,a)$$

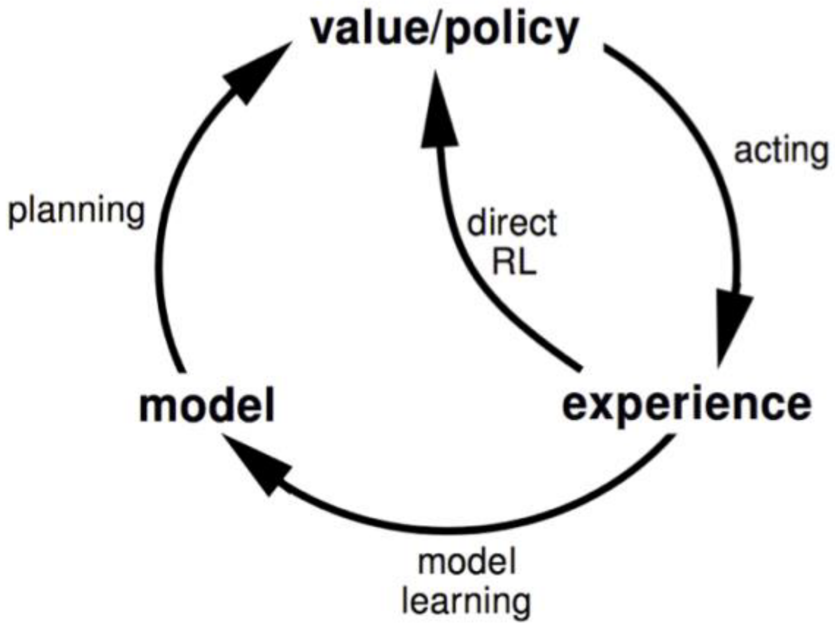
- Once we have optimal Q-value function we can act optimally

$$\pi^*(s) = \operatorname*{argmax}_{a} Q^*(s,a)$$

Model-based approach (in brief)

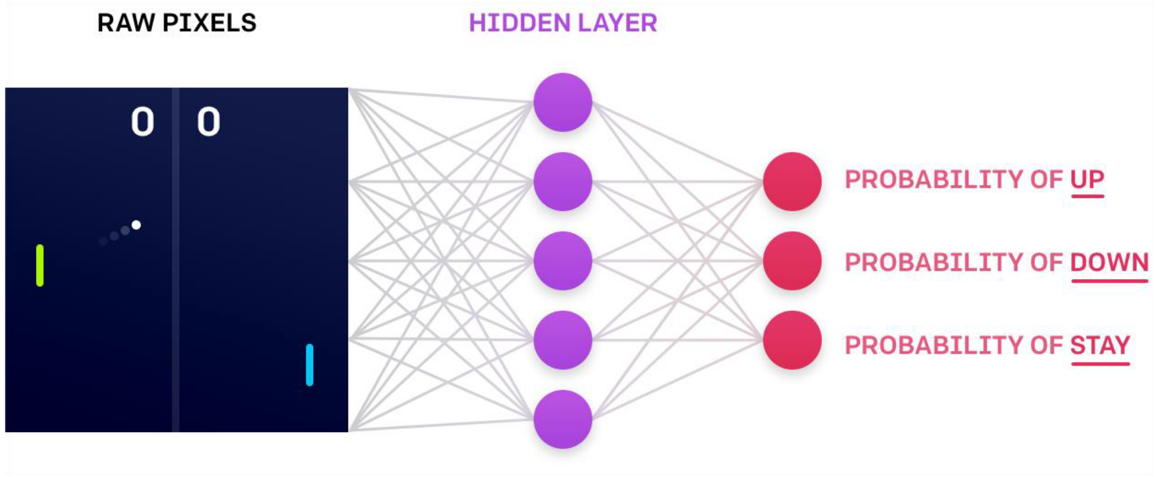
- Learn a model of the environment transitions $F(s' \mid s, a)$ and reward $R(s)$

- Using the model, “imagine” the outcome of each action and choose the best one.



Policy-based Approach (this tutorial)

- Directly search for the best policy π without necessarily modelling *values* or *the environment*.
- Represent a stochastic policy π_θ using continuous parameters θ:
  - In *deep learning*, θ are the neural network weights
  - Can easily handle *discrete* or *continuous* states/actions.



Policy-based Approach (Outline)

- Initialize parameters θ randomly, write expected return $J(\pi_\theta)$ as simply $J(\theta)$.

- **Training Loop**
  1. Collect data D by running π_θ in the environment.
  2. Estimate $\nabla_\theta J(\theta)$ using D
  3. Improve the policy by taking a gradient ascent step
     - $\theta := \theta + \eta \nabla_\theta J(\theta)$
     - Effect: increases $J(\theta)$ (if things go as planned)

Details: Deriving $\nabla_\theta J(\theta)$

- Let $\tau = (s_1, a_1, r_1, \ldots, s_T, a_T, r_T)$ be a random episode under F, π
- Can write $p(\tau) = \rho(s_1)\left[\pi_\theta(a_1 \,|\, s_1)F(s_2 \,|\, s_1, a_1)\ldots\pi_\theta(a_T \,|\, s_T)\right]$
- Rewrite $J(\theta) = \mathbb{E}_{\tau \sim F,\pi}[p(\tau)r(\tau)]$
- $\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim F,\pi}\left[r(\tau)\,\nabla_\theta \log p(\tau)\right]$

  $\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim F,\pi}\left[r(\tau)\sum_t \nabla_\theta \log \pi(a_t \,|\, s_t)\right]$

(I assume $\gamma = 1$ to simplify the derivations a bit)

Details: Estimating $\nabla_\theta J(\theta)$

- REINFORCE / Score Function Estimator

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim F, \pi}\left[r(\tau) \sum_t \nabla_\theta \log \pi(a_t \mid s_t)\right]$$

- Estimate the gradient by averaging over many τ (unbiased!)
- **Intuition**: if τ got high reward $r(\tau)$, "reinforce" the actions on that trajectory
- **Issue:** this estimator has very high variance — need lots of τ to get an accurate gradient estimate

Details: Estimating $\nabla_\theta J(\theta)$

- Lower variance, unbiased gradient estimators exist, and are much more practical.

Policy Gradient with Baseline

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim F, \pi}\Big[\sum_t (r_{t:T}(\tau) - V(s_t))\, \nabla_\theta \log \pi(a_t \mid s_t)\Big]$$

REINFORCE / Score Function Estimator

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim F, \pi}\Big[r(\tau) \sum_t \nabla_\theta \log \pi(a_t \mid s_t)\Big]$$

References

- Sergey Levine, "Policy Search", Deep learning summer school slides, https://dlrlsummerschool.ca/wp-content/uploads/2018/09/levine-policy-search-rlss-2018.pdf
- David Silver, "Deep Reinforcement Learning", ICML 2016 tutorial, https://icml.cc/2016/tutorials/deep_rl_tutorial.pdf
- Sutton, Richard S., and Andrew G. Barto. *Introduction to reinforcement learning*. Vol. 135. Cambridge: MIT press, 1998.