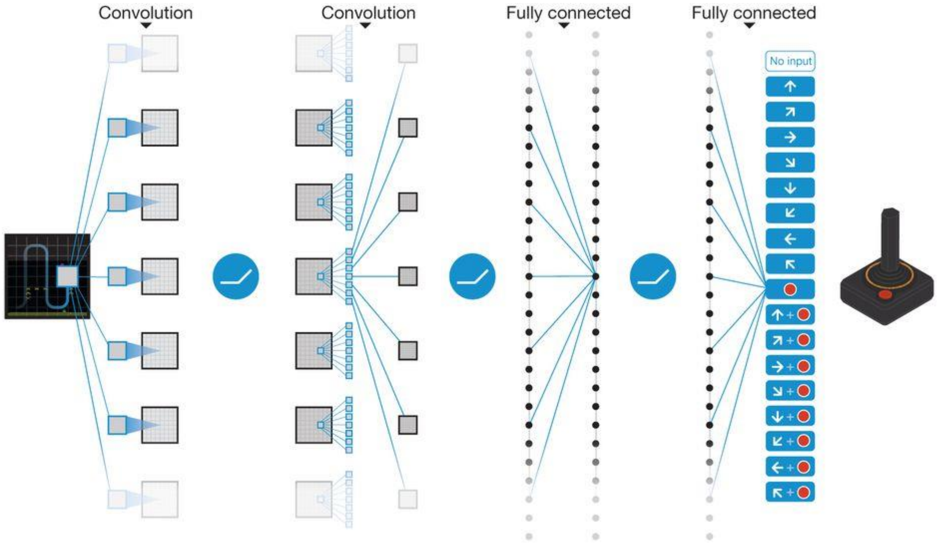
# COMS W4733: Computational Aspects of Robotics

## Lecture 27: Reinforcement Learning




Instructor: Tony Dear

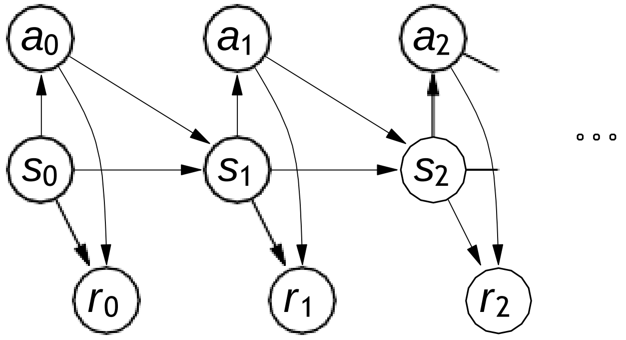
# Reinforcement Learning

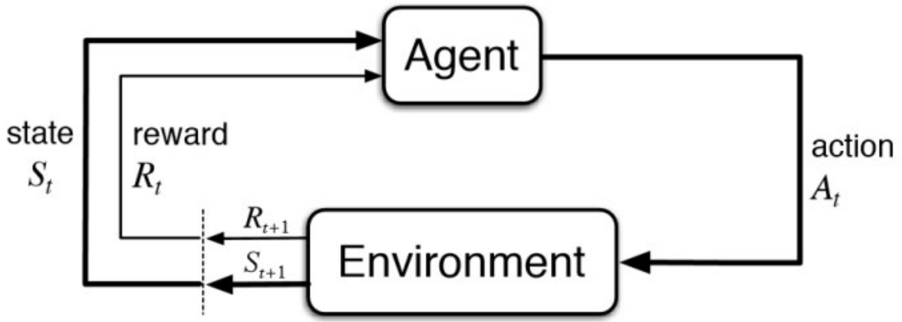
- Even if we have an idea of the models that a robot can use, finding actions, strategies, or policies can still be hard!

- We've seen different algorithms: IK, motion planning, estimation, etc.
- All useful in different contexts with different limitations

- Instead of specifying strategies, let robots *learn* from data
- Data comes from interactions with environment
- Robot has access to **state**, **actions**, **rewards**

# Components of RL




- Transition model: $P(s_{t+1}|s_t, a_t)$
- Reward model: $P(r_t|s_t, a_t)$
- Agent policy: $\pi(a_t|s_t)$

Joint MDP model:

$$P(s_{0:T}, a_{0:T}, r_{0:T}; \pi) =$$

$$P(s_0) \prod_{t=0}^{T} P(a_t|s_t; \pi) P(r_t|s_t, a_t) P(s_{t+1}|s_t, a_t)$$

# State-Values and Action-Values

- Rewards: how good the current state and action are
- May be temporally or spatially separated, sparse, or inconsistent
- Key idea: A reward is the result of current and past states/actions

- **State value function**:

$$V^*(s) = \max_{\pi} E\left[\sum_{i=1}^{T} \gamma^{i-1} r_i\right]$$

**State value** is the expected sum of discounted rewards starting from that state *under the best policy*

- **State-action value (q-value) function**:

$$Q^{\pi}(s,a) = E\left[\sum_{i=1}^{T} \gamma^{i-1} r_i\right]$$

**Q-value** is the expected sum of discounted rewards starting from a state and *taking specified action*

# RL Paradigms

- In general a robot will *explore* its state space by taking different actions, receiving rewards, and saving experiences

- From experience data $D = (s_t, a_t, r_t)$:
  - **Model-based RL**: Learn transition and reward models, get value function (dynamic programming), extract policy
  - **Model-free RL**: Learn value function directly, extract policy
- From demonstration data $D = (s_{0:T}, r_{0:T})$:
  - **Learning from demonstration** (LfD): Learn policy directly
  - **Inverse RL**: Learn latent rewards, get value function (dynamic programming), extract policy

# Learning from Demonstration

- AKA imitation learning
- Idea: No need to learn from scratch, humans can provide examples
  - Particularly useful if policy or reward is hard to specify

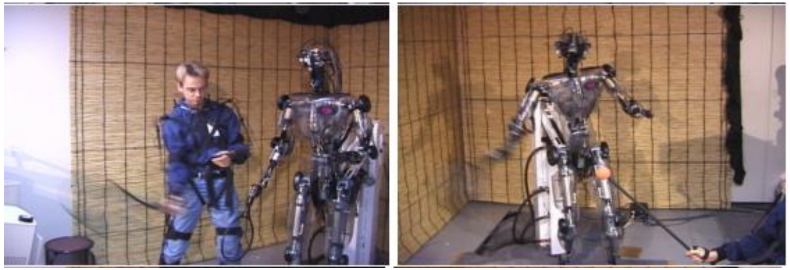
- Inputs: State-action pairs $(s_t, a_t)$—supervised!
- Output of learning: A policy $\pi$

- Demonstration modes: Teleoperation, kinesthetic teaching, camera recordings
- How to learn? Low-level skills or high-level action compositions

# Sensors

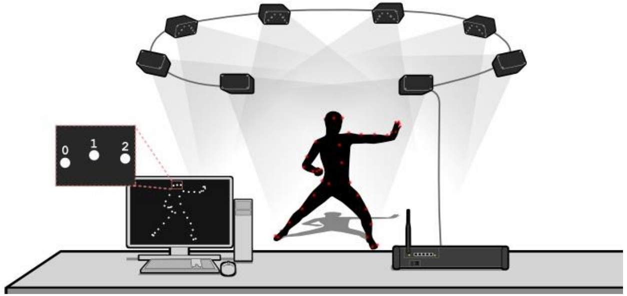
- Cameras, depth sensors, visual fiducials


- Wearable sensors: accelerometers, pressure sensors



Sarcos Sensuit

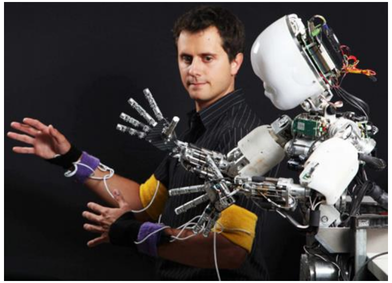
- Motion capture




Phasespace

# Correspondences

- Mimicry and shadowing
- Teleoperation
- Kinesthetic teaching
- User feedback



Calinon et al. 2010

Embodied Intelligence

Kormushev et al. 2010

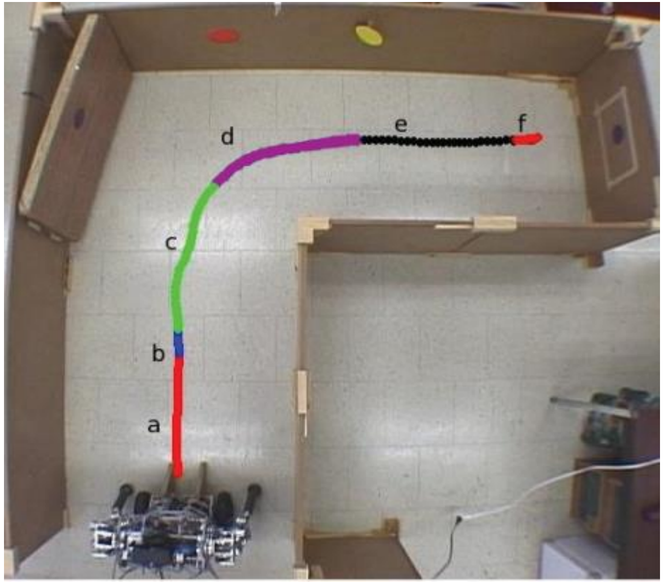

# Learning Process

- **Low-level** learning of individual motions
- Controller sees primitive motion patterns, learns corresponding parameters
- Methods: Gaussian processes, Gaussian mixture models, SVMs

- Force control tasks, particularly with the help of haptics and tactile sensing

- **High-level** learning of action composition
- One approach: First learn primitive motions, then stitch them together
- Alternatively, learn to *segment* a complicated task into actions

# Example: Skill Segmentation




| # | Abstraction | Description | Trajectories Required |
|---|---|---|---|
| a | torso-purple | Drive to door. | 2 |
| b | hand-purple | Push the door open. | 1 |
| c | torso-orange | Drive toward wall. | 1 |
| d | torso-yellow | Turn toward the end wall. | 2 |
| e | torso-purple | Drive to the panel. | 1 |
| f | hand-purple | Press the panel. | 3 |

Konidaris et al. 2012

# LfD Considerations

- Human/robot correspondences may not be straightforward
- Robot controller may not be straightforward

- Difficult to improve on the expert or learn for new situations
- Difficult to deal with deviations or errors

- Lots of rich training data required, significant burden on the expert
- Some tasks are difficult to demonstrate in meaningful way
- Humans are prone to error, impatience, fatigue

# Inverse Reinforcement Learning

- Idea: Again start with state-action pairs, but learn a reward function first and then use it to find a policy

- Why? Assume that reward function is the best representation of the task
- Reward ($R^*$) is often more succinct and robust than optimal policy ($\pi^*$)

- Ng and Russell (2000): If $\pi^*$ is known, then a parameterization (i.e., features) of $R$ can be learned using convex optimization
- Otherwise we can use sampled expert trajectories and assume some parameterization of $R$

# IRL Mathematical Formulation

- We want to find a reward function $R^*$ such that

$$E\left[\sum_{i=0}^{\infty}\gamma^i R^*(s_i,a_i);\pi^*\right] \geq E\left[\sum_{i=0}^{\infty}\gamma^i R^*(s_i,a_i);\pi\right], \forall\pi$$

- General strategy: Assume $R^*$ is a linear combination of features

$$E\left[\sum_{i=0}^{\infty}\gamma^i R(s_i,a_i);\pi\right] = E\left[\sum_{i=0}^{\infty}\gamma^i w^T\phi(s_i,a_i);\pi\right] = w^T\mu(\pi)$$

"Feature expectations"

- Max margin optimization problem

$$(w^*)^T\mu(\pi^*) \geq (w^*)^T\mu(\pi), \forall\pi \quad \Rightarrow \quad \min_w \|w\|_2^2 \text{ subject to } (w^*)^T\mu(\pi^*) \geq (w^*)^T\mu(\pi), \forall\pi$$

Can also modify to include slack variables for *expert suboptimality*

# Apprenticeship via IRL

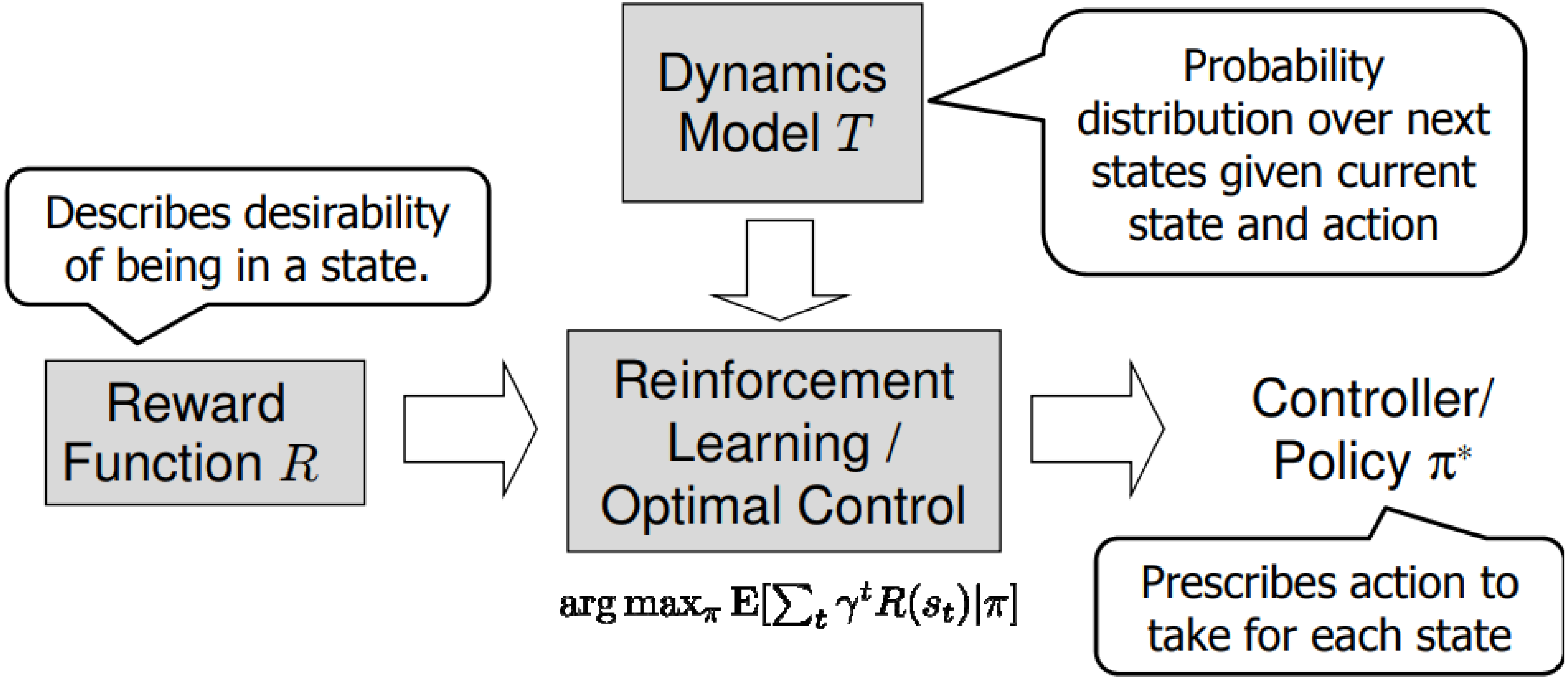
- Once we have a reward function, we can take the next step and get a policy

- Idea: Iterate between learning reward function and optimal policy
- Fix reward and optimize policy; fix policy and find optimal reward function

- Improves on LfD and IRL in that agents can actually perform tasks beyond what they observed

- Can inform reward function if we have some idea of policy beforehand

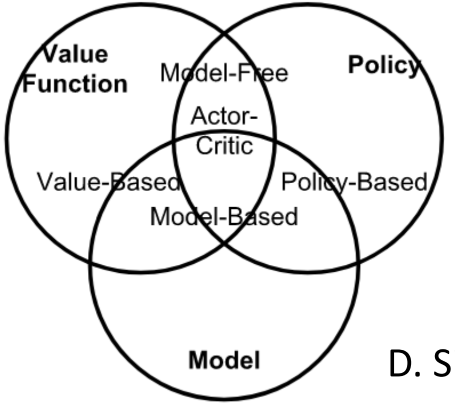
# Reinforcement Learning



# Learning From Experiences

- In contrast to LfD or IRL, robot must actively interact with environments
- Rewards are known; new challenge is to maximize rewards or learn a policy while simultaneously exploring

- Many different techniques for RL
  - Value iteration
  - Policy gradient
  - Actor-critic
  - Model-based RL




D. Silver

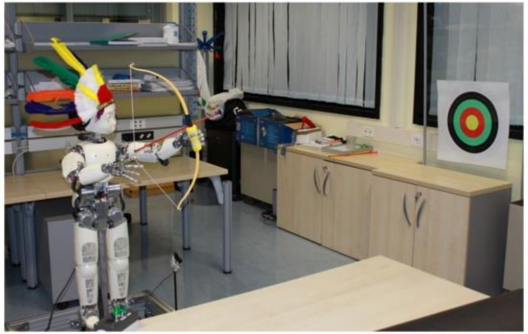
# Reward Functions

- Coming up with a correct and meaningful reward function is challenging
- Often very task-specific, may also encode different costs
- Examples: Rewards based on positions, orientations, velocities

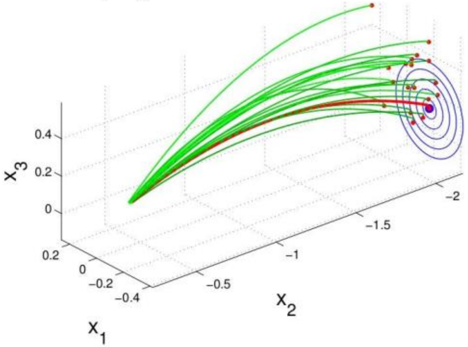
Pancake flipping

$$R(\tau) = w_1 \left[ \frac{\arccos(v_0 . v_{t_f})}{\pi} \right] + w_2 e^{-||x^p - x^F||} + w_3 x_3^M$$

Archery



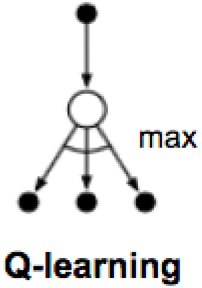
$$R(\tau) = e^{-||\hat{r}_T - \hat{r}_A||}$$



# Value Iteration Methods

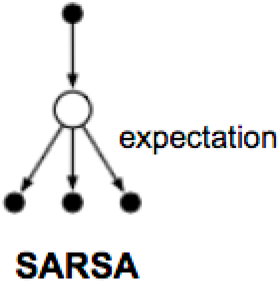
- Idea: Learn state or state-action value functions as we accumulate rewards
- **Q-learning**: Off-policy

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha)\underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}}(r_t + \gamma \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of future value}})$$




- **SARSA** (state-action-reward-state-action): On-policy

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}))$$




- Considerations: Learning rate, discount factor, initial Q-values

# Q-Function Approximation

- Previous description only works if state/action space are small and discrete
- Instead of a table of values, learn a parameterized *function*

- Ex: Weighted linear combination of features $\boldsymbol{f}(s)$, e.g. Gaussian RBFs

$$\hat{Q}_\theta(s_t, a_t) = \theta_0 + \sum_{k=1}^{K} \theta_k f_k(s_t, a_t)$$

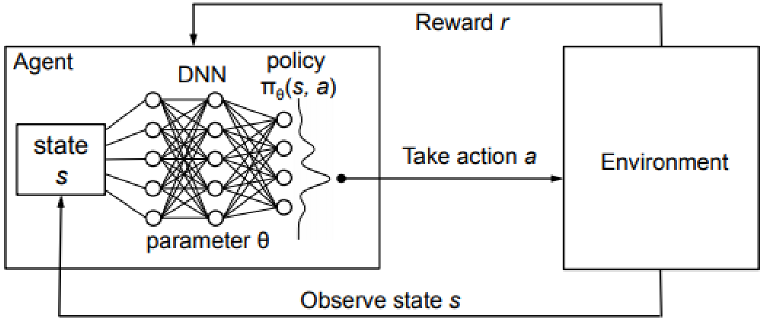
- Update rule based on gradient descent

$$\theta_k \leftarrow \theta_k + \alpha[r(s_t, a_t) - Q(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a)] \frac{\partial \hat{Q}_\theta}{\partial \theta_k}$$

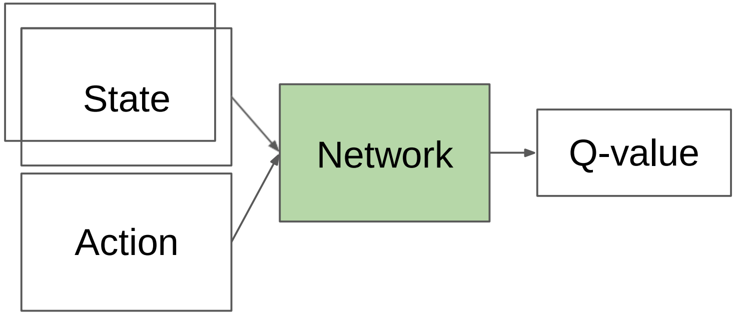
# Other Function Approximations

- We can also have more complex, *nonlinear* function approximators
- Or *non-parametric* approximations, e.g. Gaussian process regression

- Recent success with approximations using deep neural networks (Atari)




Mao et al. 2016



# Policy Gradient Methods

- Another approach: Directly learn and update policy $\pi(a_t|s_t)$ instead of values
- Values can be used to optimize policy, but we won't use it to select actions
- More effective for high-dim/continuous action spaces, stochastic actions

- Policy objective function: $J(\theta) = E_{\pi_\theta}\left[\sum_t r(s_t, a_t)\right]$

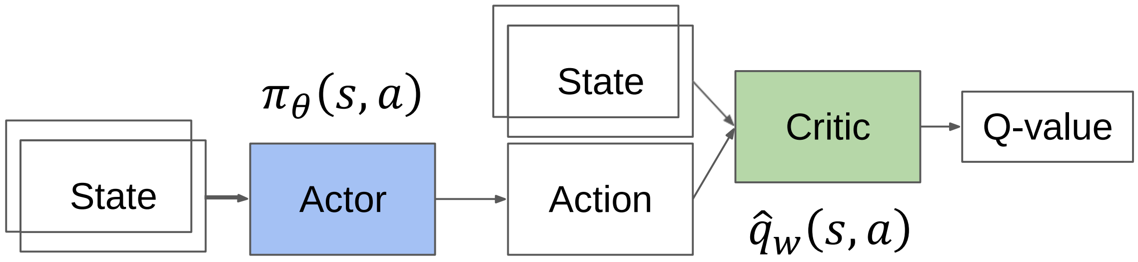
- To update policy parameters $\theta$, iteratively run the policy and perform gradient ascent using $\theta \leftarrow \theta + \alpha \nabla J(\theta)$ (**REINFORCE** algorithm)

- Can find an analytical approximation for $\nabla J(\theta)$ that depends on $\nabla_\theta \log \pi_\theta$

# Actor-Critic Methods

- Policy gradient methods can be unstable and slow due to high variances when computing the gradients

- **Actor-critics** iteratively combine value estimation with policy updates
- After each update, the *critic* estimates an update to (Q-)value function
- Then the *actor* updates policy distribution, e.g. using a policy gradient
- Both can be function-approximated



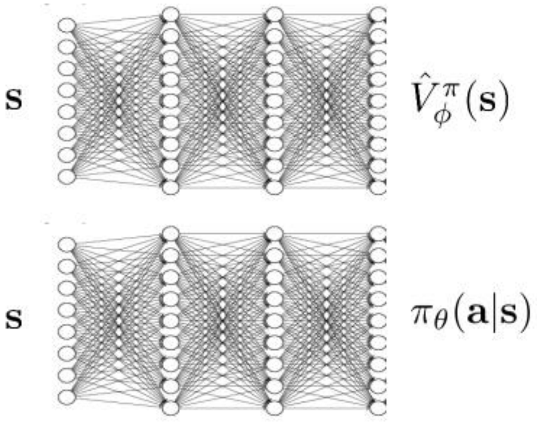
# Actor-Critic Methods

- Policy update (policy gradient):

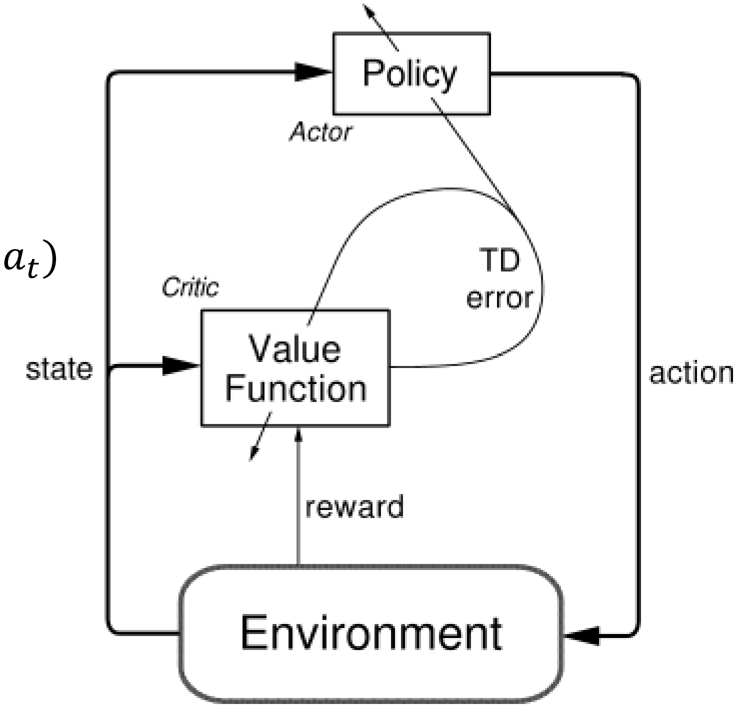
$$\Delta\theta = \alpha \nabla_\theta (\log \pi_\theta(s,a)) \hat{q}_w(s,a)$$

- Value update:

$$\Delta w = \beta\big(R(s,a) + \gamma \hat{q}_w(s_t,a_t) - \hat{q}_w(s_{t+1},a_{t+1})\big)\nabla_w \hat{q}_w(s_t,a_t)$$




Example implementation: separate networks



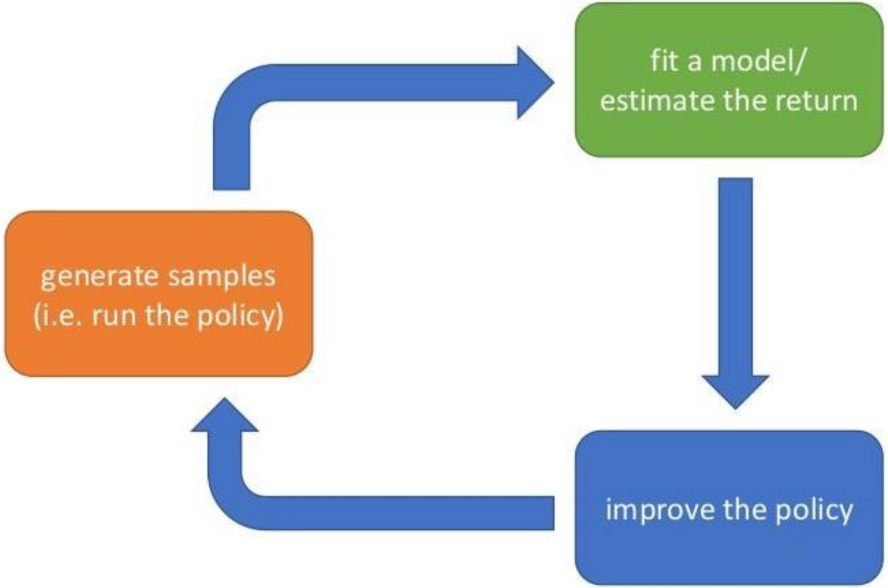
# RL Considerations in Robotics

- High-dimensional state and action spaces
- Real-world samples can be unreliable, expensive, and slow
- Difficult to explore environment freely and fully

- Can use models, but risk under-modeling nonlinearities and uncertainties

- Specifying goals and rewards may not be straightforward
- Discretization of state/action spaces needs to be meaningful
- Prior knowledge may be useful

# Summary of RL Methods




compute $\hat{Q} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ (MC policy gradient)

fit $Q_\phi(\mathbf{s}, \mathbf{a})$ (actor-critic, Q-learning)

estimate $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ (model-based)

$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$ (policy gradient)

$\pi(\mathbf{s}) = \arg\max Q_\phi(\mathbf{s}, \mathbf{a})$ (Q-learning)

optimize $\pi_\theta(\mathbf{a}|\mathbf{s})$ (model-based)

# References


- B. Argall et al., “A survey of robot learning from demonstration,” 2009

- A. Ng and S. Russell, “Algorithms for inverse reinforcement learning,” 2000
- P. Abbeel and A. Ng, “Apprenticeship learning via inverse reinforcement learning,” 2004

- L. Kaelbling et al., “Reinforcement learning: A survey,” 1996
- J. Kober et al., “Reinforcement learning in robotics: A survey,” 2013
- V Mnih et al., “Human-level control through deep reinforcement learning,” 2015

- R. Sutton and A. Barto, “Reinforcement Learning: An Introduction” 2e, 2018 (book)