CS 4803-DL / 7643-A: LECTURE 24
DANFEI XU

Topics:

- Reinforcement Learning Part 2
  - **Deep Q Learning (cont.)**
  - **Policy Gradient**
  - **Actor-Critic**
  - **Advanced Policy Gradient Methods**
  - **Applications**

RL: Sequential decision making in an environment with evaluative feedback.

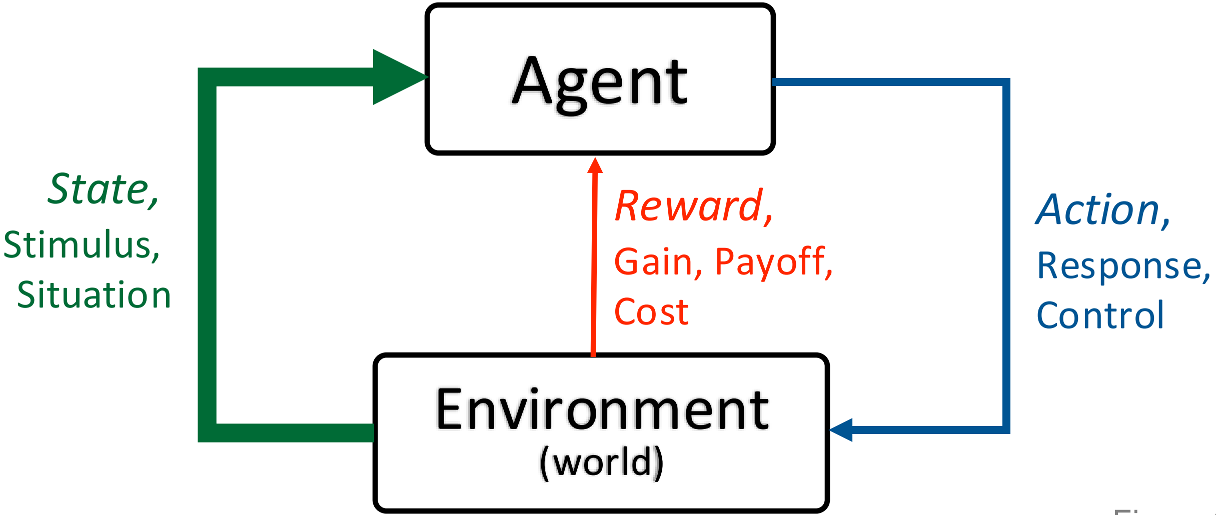


Figure Credit: Rich Sutton

- **Environment** may be unknown, non-linear, stochastic and complex.
- **Agent** learns a **policy** to map states of the environments to actions.
  - Seeking to maximize cumulative reward in the long run.

- **MDPs**: Theoretical framework underlying RL
- An MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{T}, \gamma)$
  - $\mathcal{S}$: Set of possible states
  - $\mathcal{A}$: Set of possible actions
  - $\mathcal{R}(s, a, s')$: Distribution of reward
  - $\mathbb{T}(s, a, s')$: Transition probability distribution, also written as p(s'|s,a)
  - γ : Discount factor
- **Experience**: $\ldots s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, r_{t+2}, s_{t+2}, \ldots$
- **Markov property**: Current state completely characterizes state of the environment
- **Assumption**: Most recent observation is a sufficient statistic of history

$$p\left(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, \ldots S_0 = s_0\right) = p\left(S_{t+1} = s' | S_t = s_t, A_t = a_t\right)$$

Markov Decision Processes (MDPs)

Bellman equation:

$$V^*(s) = \max_a \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma V^*(s')\right]$$

Goal: Learn a value function V that correctly maps states to optimal values.

Facts:

- If a value function V is correct, then this equation should hold exactly.
- If the value function is incorrect, we can use this equation to update the value estimate.

$$\boxed{V^{i+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma V^i(s')\right]}$$

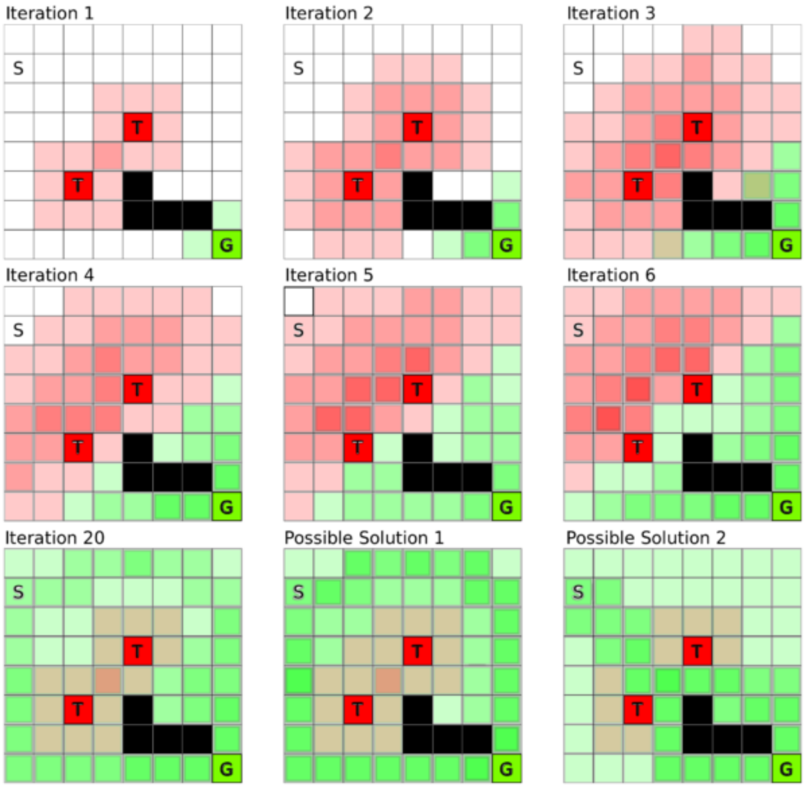
Value Iteration

$$V^{i+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma V^i(s')\right]$$

Initialize Value Function table

For each iteration i:

- For each state s:
    - For each action a:
        - Get reward $r(s,a)$
        - For each possible future states s':
            - Get current $V(s')$ from table
        - Compute the expectation term
    - Select the highest future value for a
    - Update new $V(s)$



This algorithm looks familiar …

It's dynamic programming!

https://developer.nvidia.com/blog/deep-learning-nutshell-reinforcement-learning/

Algorithm: Value Iteration

- Initialize values of all states to arbitrary values, e.g., all 0's.
- While not converged:
  - For each state: $\boxed{V^{i+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma V^i(s')\right]}$
- Repeat until convergence (no change in values)

$$V^0 \rightarrow V^1 \rightarrow V^2 \rightarrow \quad \cdots \rightarrow V^i \rightarrow \ldots \quad \rightarrow V^*$$

Time complexity per iteration $O(|\mathcal{S}|^2|\mathcal{A}|)$

Q-Learning

- We'd like to do Q-value updates to each Q-state:

$$Q'(s_t, a_t) \cong \sum_{s'} T(s_{t+1} | s_t, a_t)[r_t + \gamma \max_a Q(s_{t+1}, a)]$$

  - But can't compute this update without knowing the transition function and enumerate all possible next states s'!

- Instead, *approximate* the expectation (sum over next states) with (lots of) experience samples
  - Take an action in the environment following *policy* $\mathrm{argmax}_{\mathrm{a}} Q(s, a)$
  - receive a sample transition (s_t, a_t, r_t, s_{t+1})
  - This sample suggests: $Q(s_t, a_t) \cong r_t + \gamma \max_a Q(s_{t+1}, a)$
  - Keep a running average to approximate the expectation:

$$Q'(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1}, a)]$$

Old estimates

New estimates

Q-Learning: a model-free method for RL

Idea: represent the Q value table as a parametric function $Q_\theta(s,a)$!

How do we learn the function?

$$Q'(s_t,a_t) = (1-\alpha)Q(s_t,a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1},a)]$$
$$= Q(s_t,a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t))$$

Now, at optimum, $Q(s_t,a_t) = Q'(s_t,a_t) = Q^*(s_t,a_t)$; This gives us:

$$0 = 0 + \alpha(r_t + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t))$$

Learning problem:

$$\mathrm{argmin}_\theta || \underbrace{r_t + \gamma \max_a Q_\theta(s_{t+1},a)}_{\text{Target Q value}} - Q_\theta(s_t,a_t))\,||$$

- **Q-Learning** with **linear function approximators**

$$Q(s, a; w, b) = w_a^\top s + b_a$$

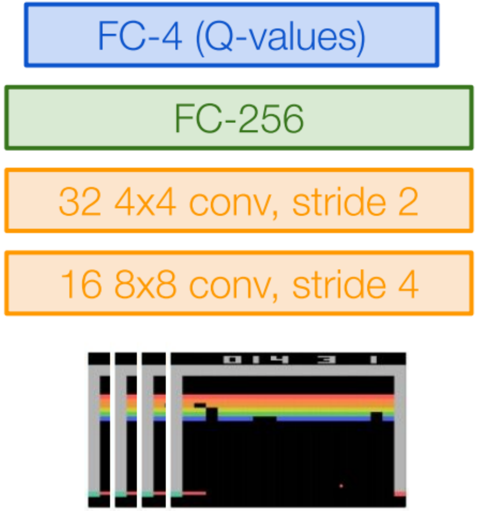
  - Has some theoretical guarantees

- **Deep Q-Learning**: Fit a **deep Q-Network**

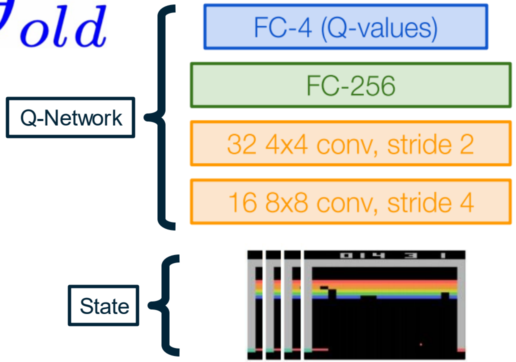
$$Q(s, a; \theta)$$

  - Works well in practice
  - Q-Network can take arbitrary input (e.g. RGB images)
  - Assume discrete action space (e.g., left, right)

Value per action dim



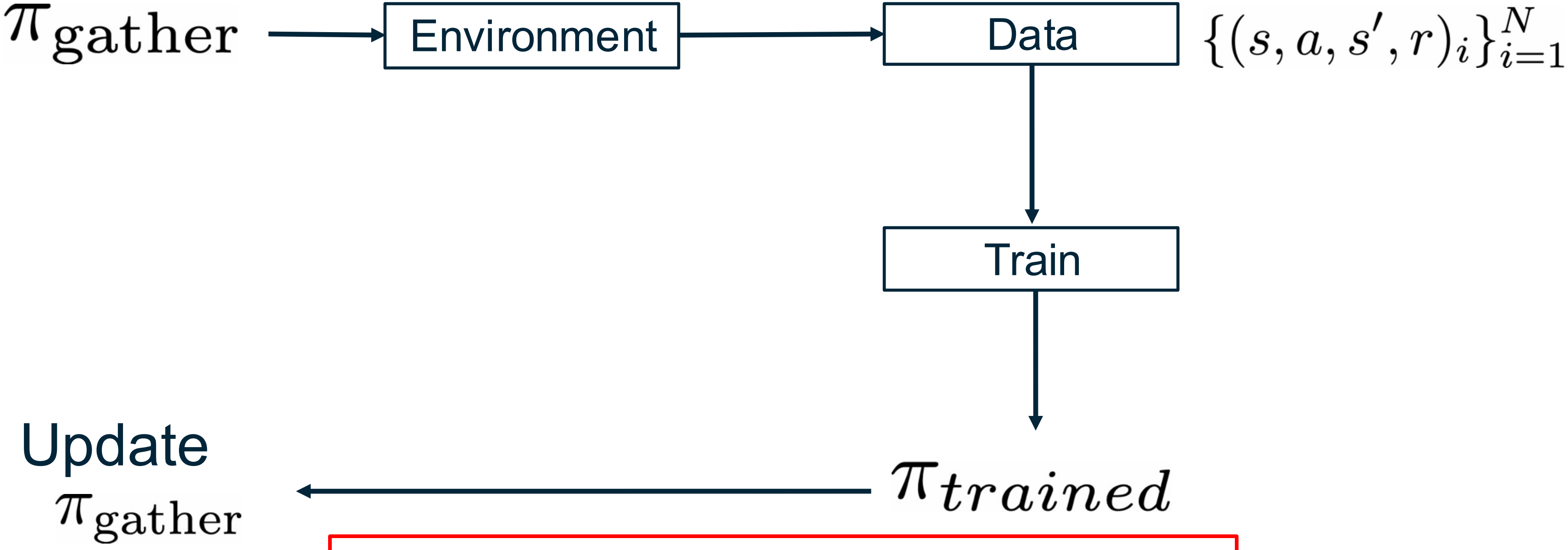
- Minibatch of $\{(s, a, s', r)_i\}_{i=1}^{B}$
- Forward pass: $\underset{B \times D}{\boxed{\text{State}}} \rightarrow \boxed{\text{Q-Network}} \rightarrow \underset{B \times n_{actions}}{\boxed{\text{Q-Values per action}}}$
- Compute loss: $$\Big(\underbrace{Q_{new}(s, a)}_{\theta_{new}} - (r + \gamma \max_{a} \underbrace{Q_{old}(s', a)}_{\theta_{old}})\Big)^2$$
- Backward pass: $$\frac{\partial Loss}{\partial \theta_{new}}$$



Deep Q-Learning

$$\text{MSE Loss} := \left(Q_{new}(s,a) - (r + \max_a Q_{old}(s',a)) \right)^2$$

- In practice, for stability:
  - Freeze Q_{old} and update Q_{new} parameters
  - Set $Q_{old} \leftarrow Q_{new}$ at regular intervals or update as running average
    - $\theta_{old} = \beta\theta_{old} + (1-\beta)\theta_{new}$



Challenge 1: Exploration vs Exploitation

Challenge 2: Non iid, highly correlated data

How to gather experience?

- What should π_{gather} be?
  - Greedy? -> no exploration, always choose the most confident action
    $\arg\max_a Q(s, a; \theta)$
- An exploration strategy:
  - ϵ-greedy

$$a_t = \begin{cases} \arg\max_a Q(s, a) & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$$

Georgia Tech

- Correlated data: addressed by using experience replay
    - A replay buffer stores transitions (s, a, s', r)
    - Continually update replay buffer as game (experience) episodes are played, older samples discarded
    - Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples
- Larger the buffer, lower the correlation

Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for

Experience Replay

Epsilon-greedy

Q Update

Deep Q-Learning Algorithm

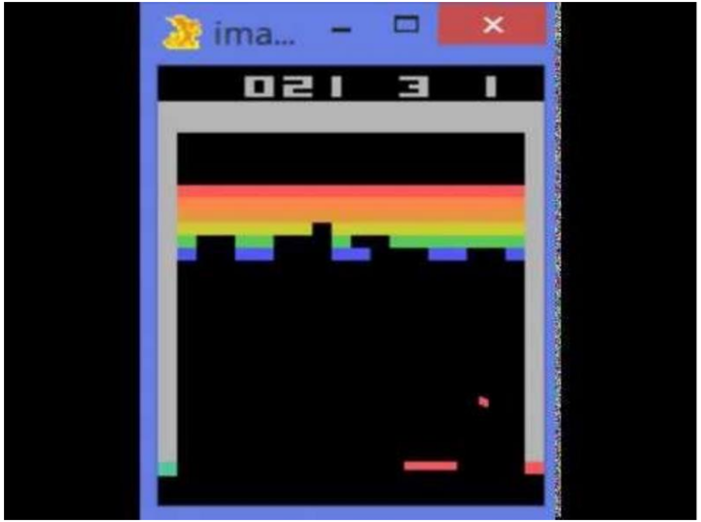
Georgia Tech

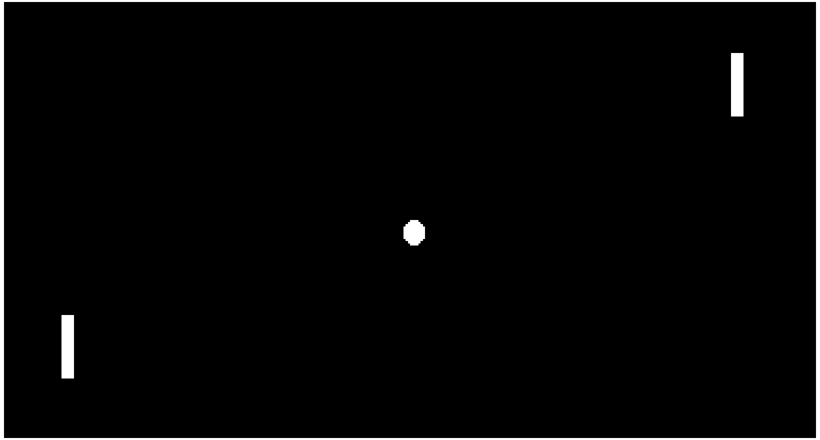
Atari Games

- **Objective**: Complete the game with the highest score
- **State**: Raw pixel inputs of the game state
- **Action**: Game controls e.g. Left, Right, Up, Down
- **Reward**: Score increase/decrease at each time step

Figures copyright Volodymyr Mnih et al., 2013. Reproduced with permission.

Atari Games





https://www.youtube.com/watch?v=V1eYniJ0Rnk

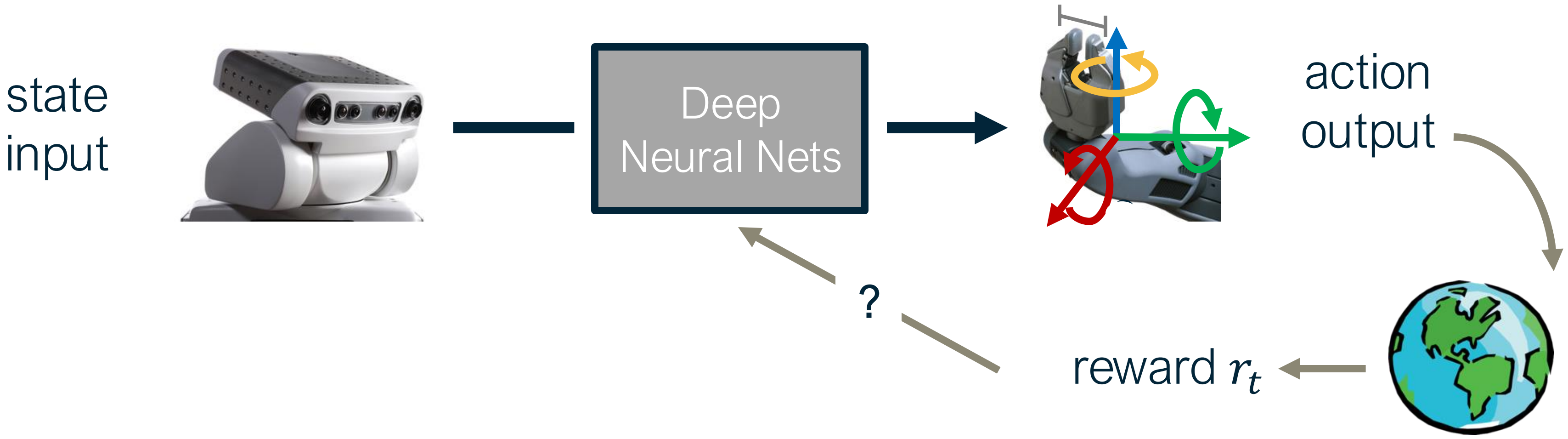
Summary: Value-based RL

- Solving an MDP by modeling / learning the values (Q and V) of an optimal policy
- Examples: Value iteration, Q learning, DQN, SARSA, TD(0), …
- Pros:
  - Conceptually simple
  - Efficient in discrete action space
- Cons:
  - Handling continuous / large action space is challenging.
  - A *proxy* of what we actually want (a policy)

Different RL Paradigms

- **Value-based RL**
  - (Deep) Q-Learning, approximating $Q^*(s, a)$ with a deep Q-network
- **Policy-based RL**
  - Directly approximate optimal policy π^* with a parametrized policy π_θ^*
- **Model-based RL**
  - Approximate transition function $T(s', a, s)$ and reward function $\mathcal{R}(s, a)$
  - Plan by looking ahead in the (approx.) future!

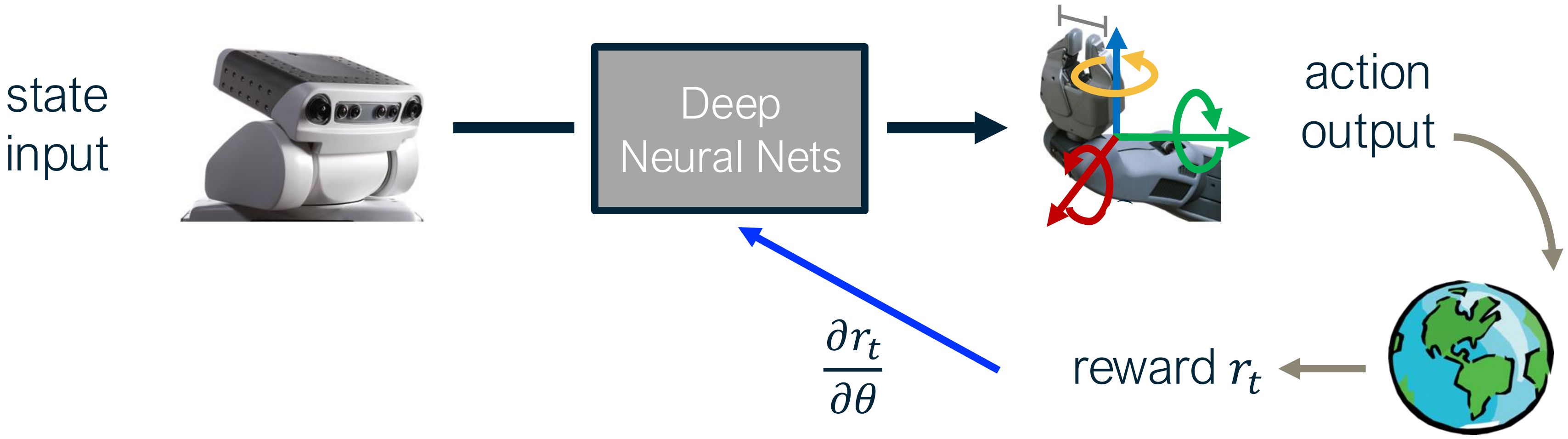
Deep Learning for Decision Making



Problem: we don't know the correct action label to supervise the output!

All we know is the step-wise task reward

Deep Learning for Decision Making

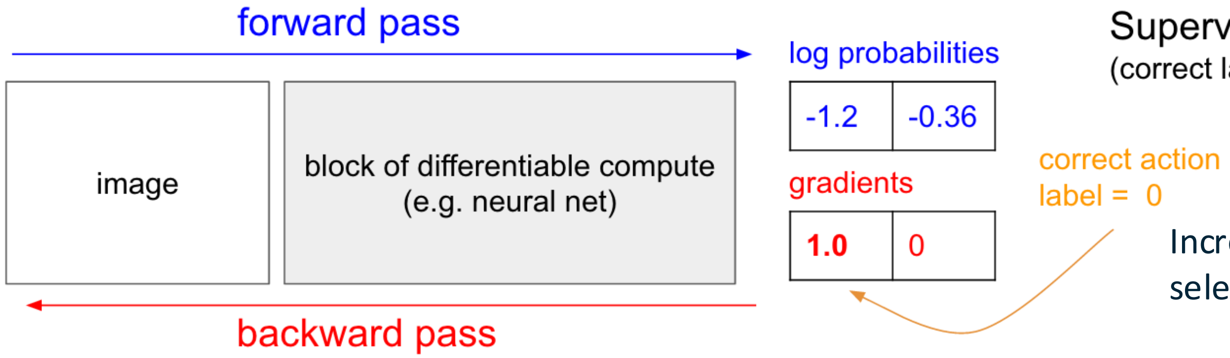


Problem: we don't know the correct action label to supervise the output!

All we know is the step-wise task reward

Can we directly backprop reward???

Policy Gradient: Just backprop from reward (sort of)!

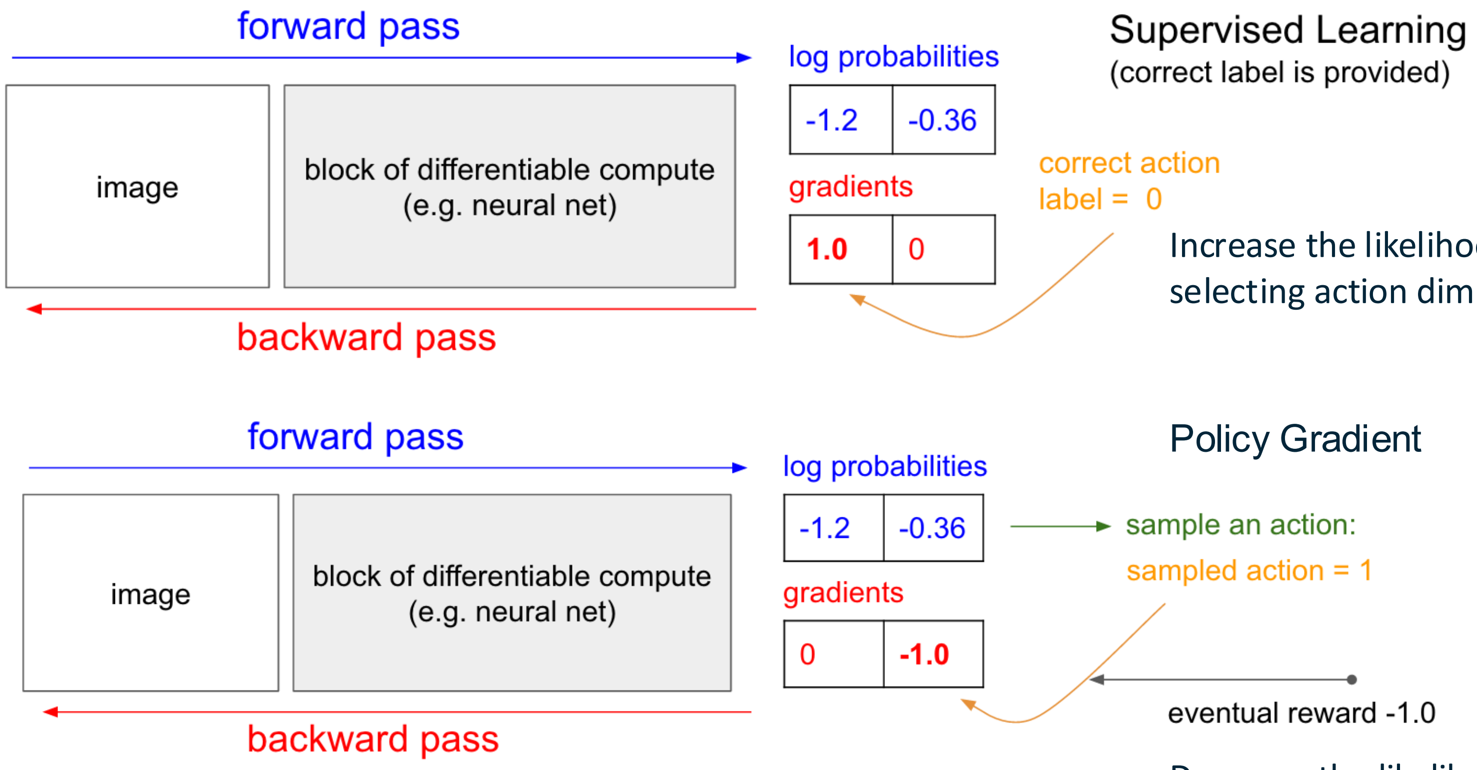


Supervised Learning
(correct label is provided)

Increase the likelihood of selecting action dim = 0!

Image Source: http://karpathy.github.io/2016/05/31/rl/

Policy Gradient: Just backprop from reward (sort of)!



Increase the likelihood of selecting action dim = 0!

Decrease the likelihood of selecting action dim = 1!

Image Source: http://karpathy.github.io/2016/05/31/rl/

Brief derivation of policy gradient (REINFORCE)

Let $\tau = (s_0, a_0, \dots s_T, a_T)$ denote a trajectory

Brief derivation of policy gradient (REINFORCE)

Let $\tau = (s_0, a_0, \ldots s_T, a_T)$ denote a trajectory

- Distribution of trajectories given a policy parameterized by θ is:

$$
\begin{aligned}
p_\theta(\tau) &= p_\theta\left(s_0, a_0, \ldots s_T, a_T\right) \\
&= p(s_0) \prod_{t=0}^{T} p_\theta\left(a_t \mid s_t\right) \cdot p\left(s_{t+1} \mid s_t, a_t\right)
\end{aligned}
$$

Brief derivation of policy gradient (REINFORCE)

Let $\tau = (s_0, a_0, \dots s_T, a_T)$ denote a trajectory

- Distribution of trajectories given a policy parameterized by θ is:

$$\begin{aligned} p_\theta(\tau) &= p_\theta(s_0, a_0, \dots s_T, a_T) \\ &= p(s_0) \prod_{t=0}^{T} p_\theta(a_t \mid s_t) \cdot p(s_{t+1} \mid s_t, a_t) \end{aligned}$$

- Optimization objective:

$$\boxed{\arg\max_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} [\mathcal{R}(\tau)]}$$

Brief derivation of policy gradient (REINFORCE)

Let $\tau = (s_0, a_0, \ldots s_T, a_T)$ denote a trajectory

- Distribution of trajectories given a policy parameterized by θ is:

$$\begin{aligned} p_\theta(\tau) &= p_\theta\left(s_0, a_0, \ldots s_T, a_T\right) \\ &= p(s_0) \prod_{t=0}^{T} p_\theta\left(a_t \mid s_t\right) \cdot p\left(s_{t+1} \mid s_t, a_t\right) \end{aligned}$$

- Optimization objective:

$$\boxed{\arg\max_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\mathcal{R}(\tau)\right]}$$

- What we need (policy gradient):

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)}[\mathcal{R}(\tau)]$$

Brief derivation of policy gradient (REINFORCE)

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)}[\mathcal{R}(\tau)]$$

$$= \nabla_\theta \int \pi_\theta(\tau)\mathcal{R}(\tau) d\tau$$

Expectation as integral

$$= \int \nabla_\theta \pi_\theta(\tau)\mathcal{R}(\tau) d\tau$$

Exchange integral and gradient

$$= \int \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau)\mathcal{R}(\tau) d\tau$$

Log derivative rule: $\frac{d\log(u)}{dx} = \frac{du}{udx}$

With $u = \pi_\theta(\tau)$ and $x = \theta$, we have:

$$\nabla_\theta \pi_\theta(\tau) = \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau)$$

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)\mathcal{R}(\tau)]$$

Brief derivation of policy gradient (REINFORCE)

$$\pi_\theta(\tau) = p(s_0) \prod_{t=0}^{T} p_\theta(a_t \mid s_t) \cdot p(s_{t+1} \mid s_t, a_t)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[\underbrace{\nabla_\theta \log \pi_\theta(\tau)}\mathcal{R}(\tau)]$$

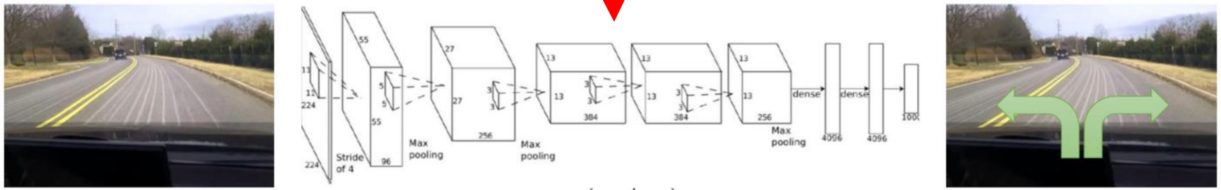
$$\nabla_\theta \left[\cancel{\log p(s_0)} + \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \sum_{t=1}^{T} \cancel{\log p(s_{t+1} \mid s_t, a_t)} \right]$$

∇_θ doesn't depend on initial state or transition probabilities!

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot \sum_{t=1}^{T} \mathcal{R}(s_t, a_t) \right]$$

Log action probs across steps

Episodic reward



$\mathbf{s}_t$ $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ $\mathbf{a}_t$

Can use continuous action space!

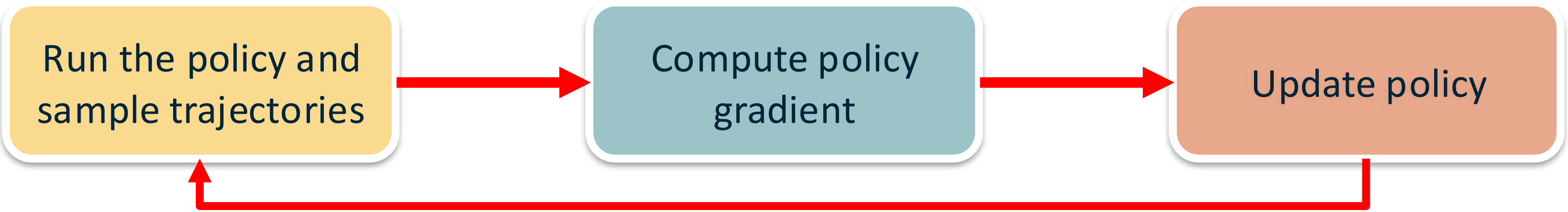
Policy gradient: algorithm sketch

- Sample trajectories $\tau_i = \{s_1, a_1, \ldots s_T, a_T\}_i$ by acting according to π_θ

- Compute policy gradient as

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i^N \left[\sum_{t=1}^T \nabla_\theta \log \pi_\theta \left(a_t^i \mid s_t^i\right) \cdot \sum_{t=1}^T \mathcal{R}\left(s_t^i \mid a_t^i\right) \right]$$

- Update policy parameters: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$



Slide credit: Sergey Levine

Policy gradient intuition

$\log \pi_\theta(a|s)$

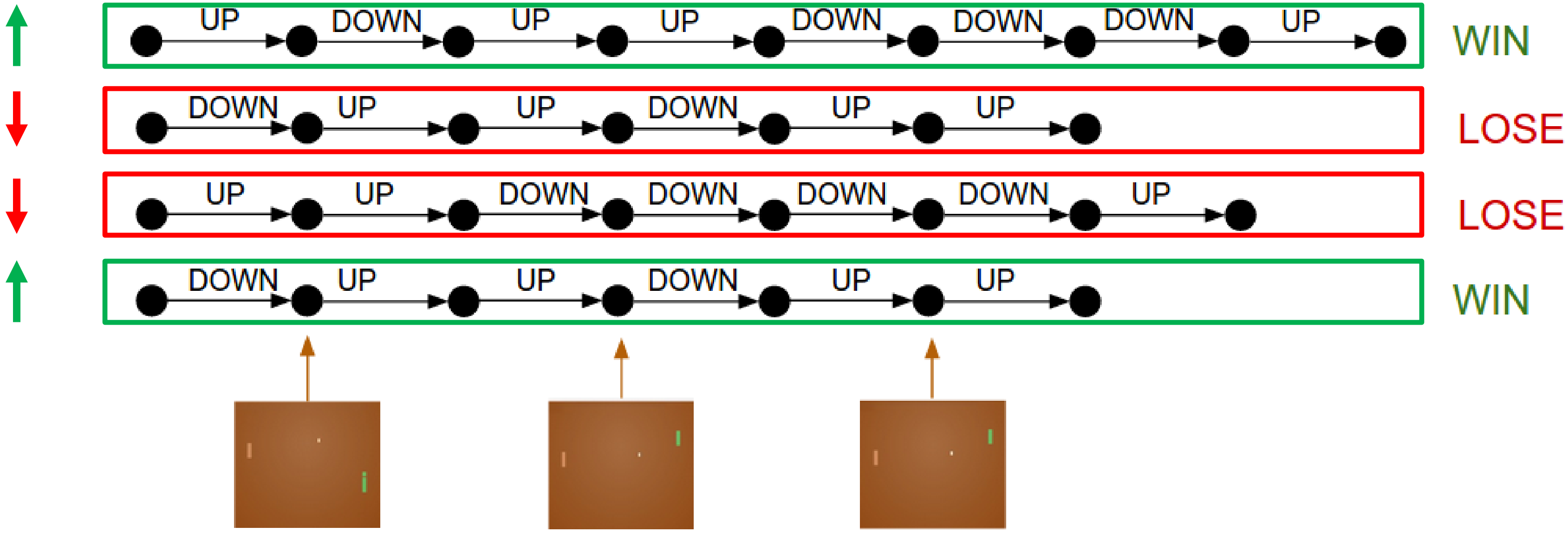


Image Source: http://karpathy.github.io/2016/05/31/rl/

Issues with Policy Gradients

- **Credit assignment is hard!**
  - Which specific action led to increase in reward
  - Suffers from high variance → leading to unstable training

Can we do better?

What if instead of just reward per episode, we know the expected future return of taking an action? (This should remind you of something …)

Q value function $Q(s, a)$!

Actor-Critic

- Learn both policy and Q function
  - Use the “actor” to sample trajectories
  - Use the Q function to “evaluate” or “critic” the policy

Actor-Critic

- Learn both policy and Q function
  - Use the "actor" to sample trajectories
  - Use the Q function to "evaluate" or "critic" the policy

- REINFORCE: $\nabla_\theta J(\pi_\theta) = \mathbb{E}_{a \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) \mathcal{R}(s, a) \right]$

- Actor-critic: $\nabla_\theta J(\pi_\theta) = \mathbb{E}_{a \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a) \right]$

Actor-Critic

- Initialize θ (policy network) and β (Q network)

Actor-Critic

- Initialize θ (policy network) and β (Q network)
- For each step:
  - sample action $a \sim \pi_\theta(\cdot\,|s)$, take action to get s' and r

Actor-Critic

- Initialize θ (policy network) and β (Q network)
- For each step:
  - – sample action $a \sim \pi_\theta(\cdot\,|s)$, take action to get s' and r
  - – evaluate "actor" using "critic" $Q_\beta(s,a)$ and update policy:

$$\theta \leftarrow \theta + \alpha \nabla_\theta (\log \pi_\theta(a|s) Q_\beta(s,a))$$

Actor-Critic

- Initialize θ (policy network) and β (Q network)
- For each step:
  - sample action $a \sim \pi_\theta(\cdot\,|s)$, take action to get s' and r
  - evaluate "actor" using "critic" $Q_\beta(s,a)$ and update policy:

$$\theta \leftarrow \theta + \alpha \nabla_\theta (\log \pi_\theta(a|s) Q_\beta(s,a))$$

  - Update "critic":
    - Q-learning using $\mathrm{argmin}_\beta [Q_\beta(s,a) - (r + Q(s', a \sim \pi_\theta(s')))]$

Actor-Critic

- Initialize θ (policy network) and β (Q network)
- For each step:
  - – sample action $a \sim \pi_\theta(\cdot\,|s)$, take action to get s' and r
  - – evaluate "actor" using "critic" $Q_\beta(s,a)$ and update policy:

$$\theta \leftarrow \theta + \alpha \nabla_\theta (\log \pi_\theta(a|s) Q_\beta(s,a))$$

  - – Update "critic":
    - Q-learning using $\mathrm{argmin}_\beta [Q_\beta(s,a) - (r + Q(s', a \sim \pi_\theta(s'))]$

Note the difference to DQN: $\left(Q_{new}(s,a) - (r + \gamma \max_a Q_{old}(s',a))\right)^2$

Actor-Critic

Actor-critic Policy Gradient: $\nabla_\theta J(\pi_\theta) = \mathrm{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_\beta(s,a)]$

Actor-Critic

Actor-critic Policy Gradient: $\nabla_\theta J(\pi_\theta) = \mathrm{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_\beta(s,a)]$

Consider a situation where $Q_\beta(s,a_1) = 10.1$ and $Q_\beta(s,a_2) = 10.5$

Actor-Critic

Actor-critic Policy Gradient: $\nabla_\theta J(\pi_\theta) = \mathrm{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_\beta(s,a)]$

Consider a situation where $Q_\beta(s, a_1) = 10.1$ and $Q_\beta(s, a_2) = 10.5$

– Good news: s is a great state to be in!

Actor-Critic

Actor-critic Policy Gradient: $\nabla_\theta J(\pi_\theta) = \mathrm{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_\beta(s,a)]$

Consider a situation where $Q_\beta(s,a_1) = 10.1$ and $Q_\beta(s,a_2) = 10.5$

– Good news: s is a great state to be in!

– Bad news: hard to tell the policy to prefer a_2 over a_1

Actor-Critic

Actor-critic Policy Gradient: $\nabla_\theta J(\pi_\theta) = \mathrm{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_\beta(s,a)]$

Consider a situation where $Q_\beta(s,a_1) = 10.1$ and $Q_\beta(s,a_2) = 10.5$

– Good news: s is a great state to be in!

– Bad news: hard to tell the policy to prefer a_2 over a_1

Idea: use *advantage function* $A(s,a) = Q(s,a) - V(s)$

Actor-Critic

Actor-critic Policy Gradient: $\nabla_\theta J(\pi_\theta) = \mathrm{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_\beta(s,a)]$

Consider a situation where $Q_\beta(s,a_1) = 10.1$ and $Q_\beta(s,a_2) = 10.5$

– Good news: s is a great state to be in!

– Bad news: hard to tell the policy to prefer a_2 over a_1

Idea: use *advantage function* $A(s,a) = Q(s,a) - V(s)$

- $A(s,a)$: How much better is taking action a over the average value at state s

Actor-Critic

Actor-critic Policy Gradient: $\nabla_\theta J(\pi_\theta) = \mathrm{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_\beta(s,a)]$

Consider a situation where $Q_\beta(s,a_1) = 10.1$ and $Q_\beta(s,a_2) = 10.5$

– Good news: s is a great state to be in!

– Bad news: hard to tell the policy to prefer a_2 over a_1

Idea: use *advantage function* $A(s,a) = Q(s,a) - V(s)$

- $A(s,a)$: How much better is taking action a over the average value at state s
- Say $V(s) = 10.0$, we have $A(s,a_1) = 0.1$ and $A(s,a_2) = 0.5$

Advantage Actor-Critic (A2C)

Advantage Actor-critic Gradient: $\nabla_\theta J(\pi_\theta) = \mathrm{E}_{a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) A(s,a)]$

Asynchronous Methods for Deep Reinforcement Learning (Minh et al., 2016)

Advantage Actor-Critic (A2C)

Advantage Actor-critic Gradient: $\nabla_\theta J(\pi_\theta) = \mathrm{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) A(s,a)]$

Problem: need to learn both Q and V to calculate A

Asynchronous Methods for Deep Reinforcement Learning (Minh et al., 2016)

Advantage Actor-Critic (A2C)

Advantage Actor-critic Gradient: $\nabla_\theta J(\pi_\theta) = \mathrm{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) A(s,a)]$

Problem: need to learn both Q and V to calculate A

Idea: use state value of experience sample to approximate Q:

Given experience (s, a, r, s')

$$A(s,a) = Q(s,a) - V(s) \cong r + V(s') - V(s)$$

Asynchronous Methods for Deep Reinforcement Learning (Minh et al., 2016)

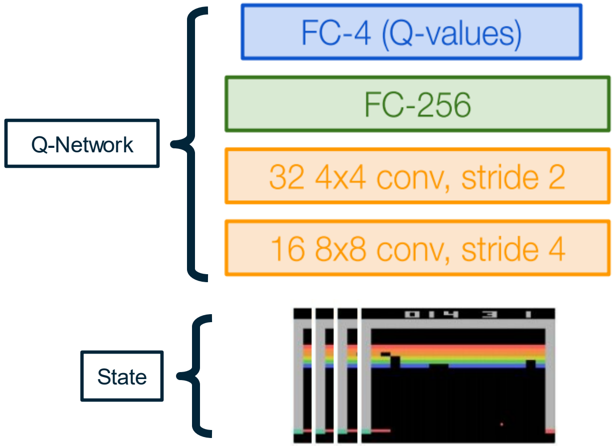
Policy Gradient Methods

- REINFORCE: $\nabla_\theta J(\pi_\theta) = \mathrm{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) R(s,a)]$
- Actor-critic (AC): $\nabla_\theta J(\pi_\theta) = \mathrm{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)]$
- Advantage Actor-critic (A2C): $\nabla_\theta J(\pi_\theta) = \mathrm{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) A(s,a)]$

Asynchronous Methods for Deep Reinforcement Learning (Minh et al., 2016)

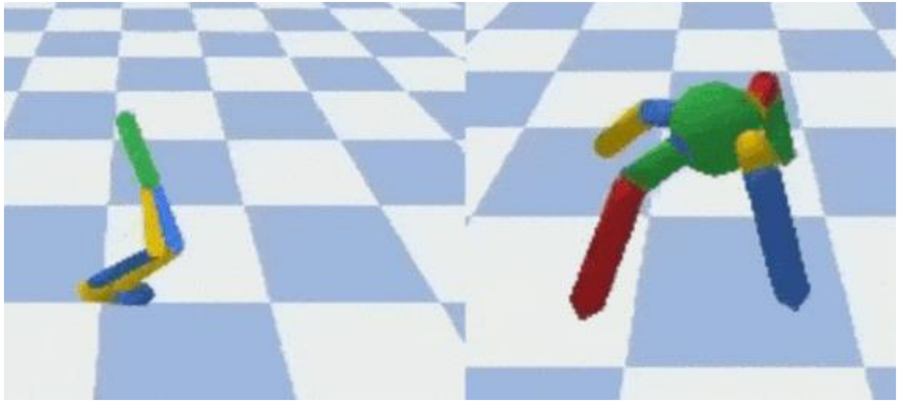
Welcome to continuous control!

DQN: limited to discrete action space



$$\nabla_\theta J(\pi_\theta) = \mathrm{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) A(s,a)]$$

Policy net can output anything!



Policy Gradient Methods

- REINFORCE: $\nabla_\theta J(\pi_\theta) = \mathrm{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) R(s,a)]$
- Actor-critic (AC): $\nabla_\theta J(\pi_\theta) = \mathrm{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)]$
- Advantage Actor-critic (A2C): $\nabla_\theta J(\pi_\theta) = \mathrm{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) A(s,a)]$

Common Policy Gradient methods are *on-policy.*

On-policy vs. off policy algorithms

- REINFORCE: $\nabla_\theta J(\pi_\theta) = \mathrm{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) R(s,a)]$

We are taking expectation wrt the *policy being learned*

Cannot use replay buffer, since the experience data is an outdated policy.

- Less data-efficient: cannot reuse old data
- Less stable to train: explore may lead to bad on-policy data -> immediate performance degradation.
- Correlated samples in training data.

Example of an *off-policy* learning algorithm: DQN

$$Q'(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

Bellman equation is true for all transitions!

Deep Deterministic Policy Gradient (DDPG)

A direct adaptation of DQN for continuous action space

Learning the critic (value function): bellman consistency

$$\min_\beta [Q_\beta(s,a) - \left(r + \max_a Q(s',a)\right)]$$

Q: What’s the problem with this objective?

Difficult to compute for continuous action space!

Deep Deterministic Policy Gradient (DDPG)

A direct adaptation of DQN for continuous action space

Learning the critic (value function): bellman consistency

$$\min_\beta [Q_\beta(s,a) - \left(r + \max_a Q(s',a)\right)]$$

Idea: approximate with a deterministic policy $\max_a Q(s',a) \approx Q(s',\pi(s))$

Deep Deterministic Policy Gradient (DDPG)

A direct adaptation of DQN for continuous action space

Learning the critic (value function): bellman consistency

$$\min_{\beta}[Q_{\beta}(s,a) - \left(r + Q_{old}(s', \pi(s')\right)]$$

Deterministic policy gradient theorem (*off-policy*)

Gradient of Q wrt to action

$$\nabla_{\theta} J(\pi_{\theta}) \approx \mathrm{E}_{s\sim\rho^*}[\nabla_{\theta} \log \pi_{\theta}(s) \nabla_a Q(s,a)]$$

We are taking expectation wrt a behavior policy (replay buffer)

Learning the actor (policy model):

$$\max_{\theta} \mathrm{E}_{s\sim\rho^*}[Q_{\beta}(s, \pi_{\theta}(s))]$$

Just back prop to policy from the value function!

A2C vs. DDPG

- Two related families of algorithms.
- A2C is on-policy. Learn advantage-based critic. Train policy through the policy gradient theorem (REINFORCE).
- DDPG is off-policy (train on replay buffer). Learn value-based critic. Train policy through direct backpropagation from critic to actor based on the deterministic policy gradient theorem.
- **Drawback**: DDPG is deterministic and often struggles with exploration.

Continuous control with deep reinforcement learning, Lilicrap et al., 2015

Advanced policy gradient methods

Soft Actor Critic (Haarnoja, 2018)

Entropy-regularized RL: achieve high reward while being as random as possible

$$\pi^* = \arg\max_{\pi} \mathop{\mathrm{E}}_{\tau\sim\pi} \left[\sum_{t=0}^{\infty} \gamma^t \Big(R(s_t, a_t, s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right) \Big) \right],$$

Bellman equation with entropy-regularized RL:

$$Q^{\pi}(s,a) \approx r + \gamma \left(Q^{\pi}(s', \tilde{a}') \boxed{- \alpha \log \pi(\tilde{a}'|s')}\right), \qquad \tilde{a}' \sim \pi(\cdot|s').$$

Approx. entropy of the policy

Advanced policy gradient methods

Soft Actor Critic (Haarnoja, 2018)

Learning the policy model:

$$V^\pi(s) = \mathop{\mathrm{E}}_{a\sim\pi}\left[Q^\pi(s,a)\right] + \alpha H\left(\pi(\cdot|s)\right)$$
$$= \mathop{\mathrm{E}}_{a\sim\pi}\left[Q^\pi(s,a) - \alpha \log \pi(a|s)\right].$$

Requires integrating a distribution!

Reparameterization trick (truncated Gaussian):

$$\tilde{a}_\theta(s,\xi) = \tanh\left(\mu_\theta(s) + \sigma_\theta(s) \odot \xi\right), \qquad \xi \sim \mathcal{N}(0, I).$$

Backprop through the value function (same as DDPG):

$$\mathop{\mathrm{E}}_{a\sim\pi_\theta}\left[Q^{\pi_\theta}(s,a) - \alpha \log \pi_\theta(a|s)\right] = \mathop{\mathrm{E}}_{\xi\sim\mathcal{N}}\left[Q^{\pi_\theta}(s,\tilde{a}_\theta(s,\xi)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s,\xi)|s)\right]$$

Advanced policy gradient methods

Trust Region Policy Gradient (TRPO, Schulman 2017)

Advanced policy gradient methods

Trust Region Policy Gradient (TRPO, Schulman 2017)

- Issue with vanilla actor critic: policy may receive huge update!
  - Big parameter update -> drastic change in behavior -> may stuck in low-reward region!

Advanced policy gradient methods

Trust Region Policy Gradient (TRPO, Schulman 2017)

- Issue with vanilla actor critic: policy may receive huge update!
  - Big parameter update -> drastic change in behavior -> may stuck in low-reward region!

Advanced policy gradient methods

Trust Region Policy Gradient (TRPO, Schulman 2017)

- Issue with vanilla actor critic: policy may receive huge update!
  - Big parameter update -> drastic change in behavior -> may stuck in low-reward region!

- Idea: constrain the update to a *trust region* using off-policy policy gradient

$$J(\theta) = \mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}},a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}\hat{A}_{\theta_{\text{old}}}(s,a)\right]$$

Advanced policy gradient methods

Trust Region Policy Gradient (TRPO, Schulman 2017)

- Issue with vanilla actor critic: policy may receive huge update!
  - Big parameter update -> drastic change in behavior -> may stuck in low-reward region!

- Idea: constrain the update to a *trust region* using off-policy policy gradient

$$J(\theta) = \mathbb{E}_{s \sim \rho^{\pi_{\theta_{\text{old}}}}, a \sim \pi_{\theta_{\text{old}}}} \Big[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} \hat{A}_{\theta_{\text{old}}}(s,a) \Big]$$

Subject to:

$$\mathbb{E}_{s \sim \rho^{\pi_{\theta_{\text{old}}}}} [D_{\text{KL}}(\pi_{\theta_{\text{old}}}(.|s) \| \pi_\theta(.|s)] \leq \delta$$

Advanced policy gradient methods

Trust Region Policy Gradient (TRPO, Schulman 2017)

- Issue with vanilla actor critic: policy may receive huge update!
  - Big parameter update -> drastic change in behavior -> may stuck in low-reward region!

- Idea: constrain the update to a *trust region* using off-policy policy gradient

$$J(\theta) = \mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}}, a\sim\pi_{\theta_{\text{old}}}}\Big[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}\hat{A}_{\theta_{\text{old}}}(s,a)\Big]$$

Subject to:

$$\mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}}}[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(.|s)\|\pi_\theta(.|s))] \leq \delta$$

Optimizing this objective requires calculating Hessian (second-order optimization)!

Advanced policy gradient methods

Proximal Policy Optimization (PPO, Schulman 2017)

Issue with TRPO: objective too complicated! Requires second-order optimization (calculating Hessian).

Advanced policy gradient methods

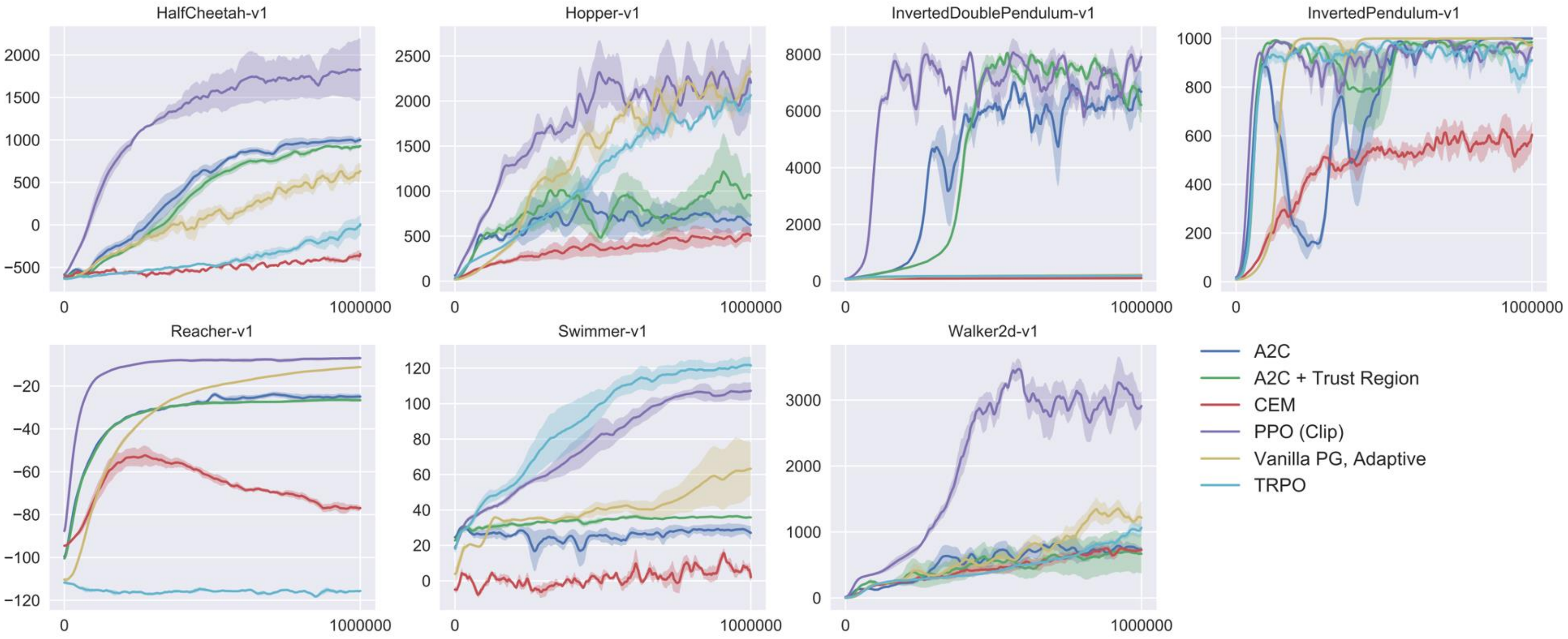
Proximal Policy Optimization (PPO, Schulman 2017)

Issue with TRPO: objective too complicated! Requires second-order optimization (calculating Hessian).

Idea: Approximate trust-region constraint with a penalty term

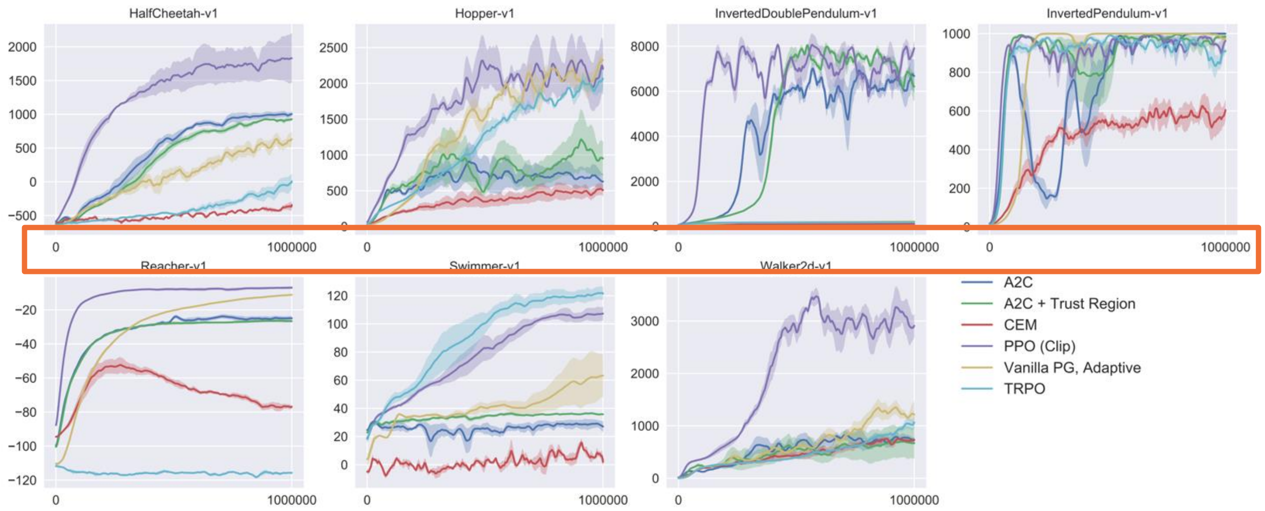
$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] - \beta\hat{\mathbb{E}}_t[\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]]$$

Advanced policy gradient methods



Schulman 2017

But Deep RL is still pretty expensive to train …



Idea: transfer policy trained in simulation (cheap) directly to the real world (expensive)!

Simulation to Real World Transfer (Sim2Real)

Issue: simulators is a *very crude* approximation of the real world!

Simulation to Real World Transfer (Sim2Real)

Issue: simulators is a *very crude* approximation of the real world!

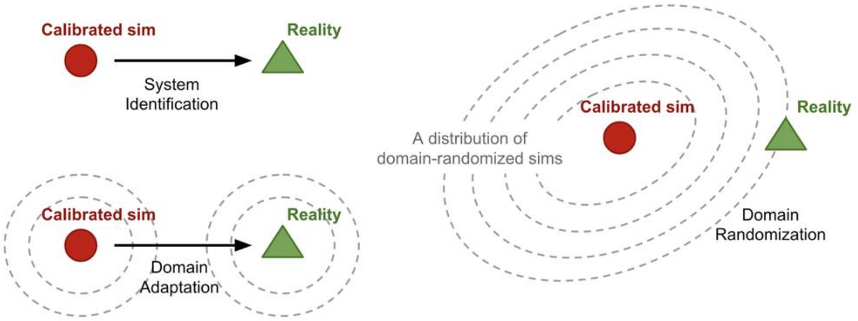
Potential gaps (not an exhaustive list):

- Position, shape, and color of objects,
- Material texture,
- Lighting condition,
- Other measurement noise,
- Position, orientation, and field of view of the camera in the simulator.
- Mass and dimensions of objects,
- Mass and dimensions of robot bodies,
- Damping, kp, friction of the joints,
- Gains for the PID controller (P term),
- Joint limit,
- Action delay,

Simulation to Real World Transfer (Sim2Real)

Issue: simulators is a *very crude* approximation of the real world!

Idea: domain randomization

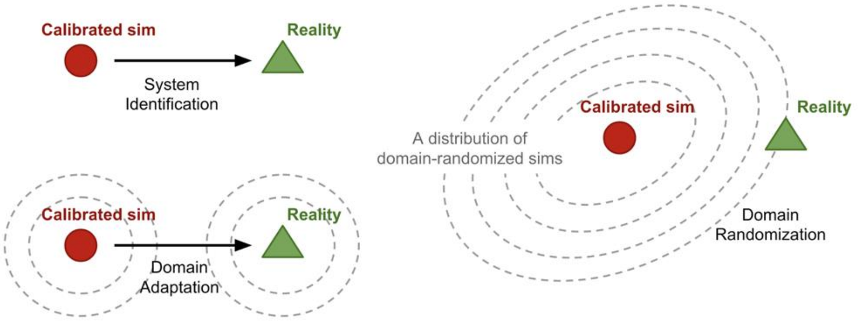


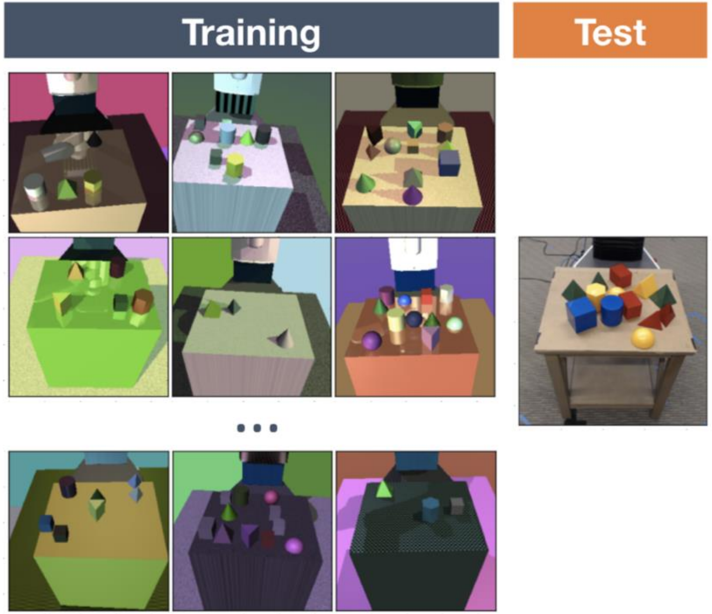
https://lilianweng.github.io/posts/2019-05-05-domain-randomization/

Simulation to Real World Transfer (Sim2Real)

Issue: simulators is a *very crude* approximation of the real world!

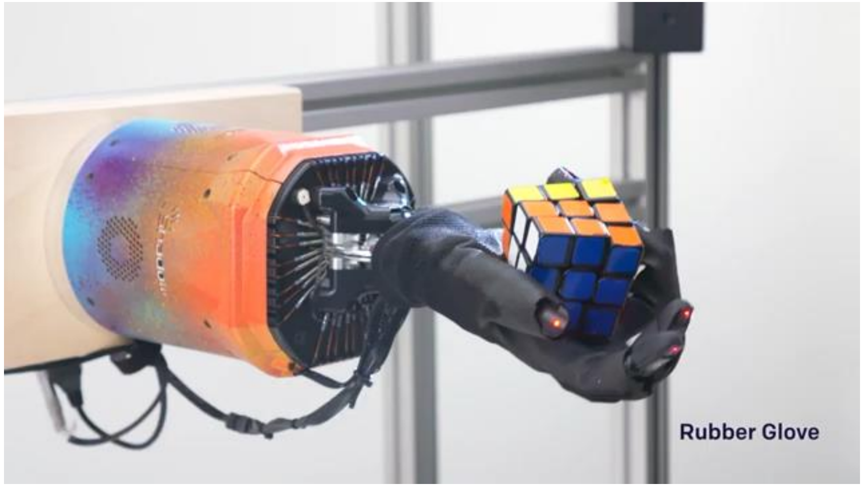
Idea: domain randomization





https://lilianweng.github.io/posts/2019-05-05-domain-randomization/

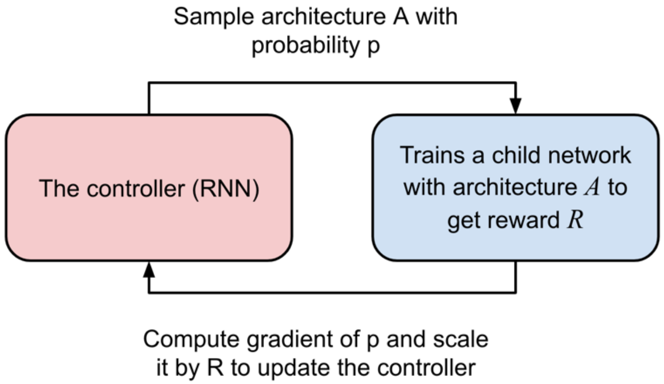
Deep RL for Robotics



Source: OpenAI

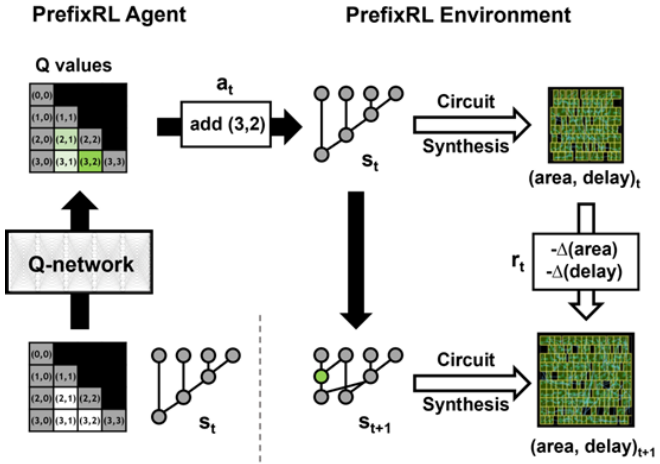
Source: ETH Zurich

Deep RL beyond robotics / games …



Neural Architecture Search

Zoph and Le, 2016



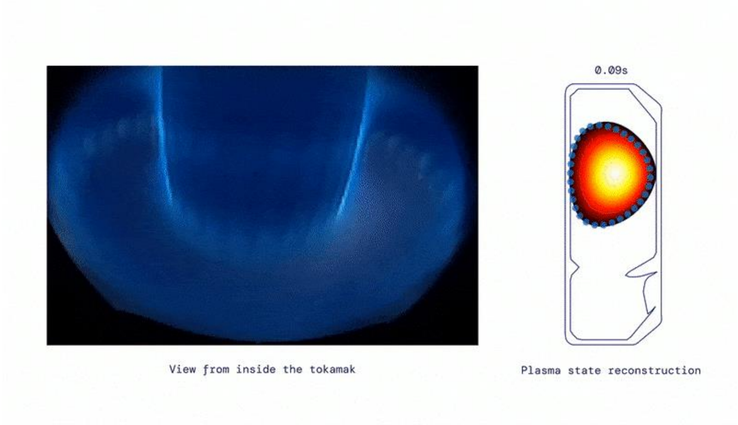
Chip Design

Roy, 2022

Deep RL beyond robotics / games …

Data Center Cooling

Lazic, 2018



Plasma Control (nuclear fusion)

Degrave, 2022

Summary

- It turns out we *can* directly backprop from reward (sort of)!
- Naïve policy gradient (REINFORCE) has high variance due to the use of episodic reward. Credit assignment is hard.
- Use Action Value Function (Q) instead!
  - Actor-Critic: learn Q value function jointly with policy
  - Advantage Actor-Critic: estimate advantage A using V value function
  - Deep Deterministic Policy Gradient for off-policy learning
  - SAC for off-policy learning with stochastic policy model
- Other advanced policy gradient methods: TRPO, PPO
- Still pretty expensive to train! Mostly used for application that can be simulated.