ECE7121 Learning-based control – 2025 Fall

Policy gradient

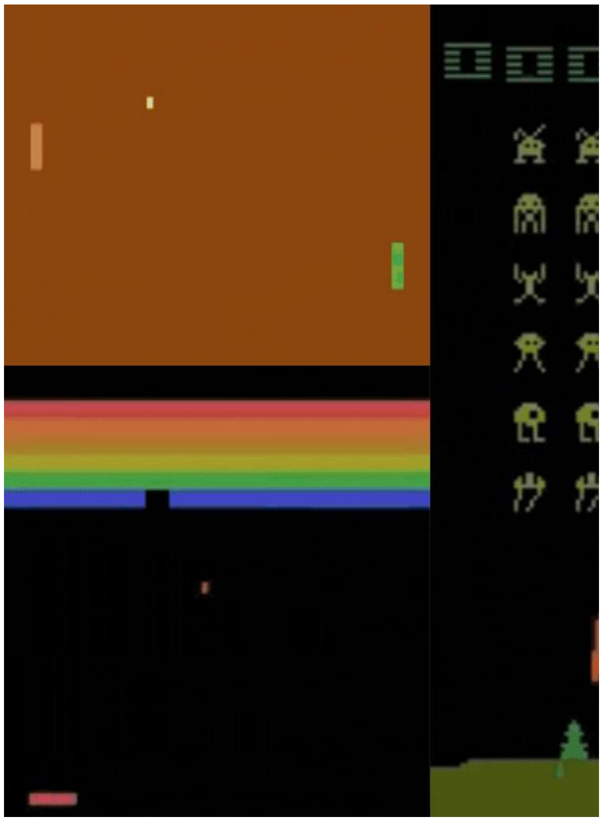
INHA UNIVERSITY

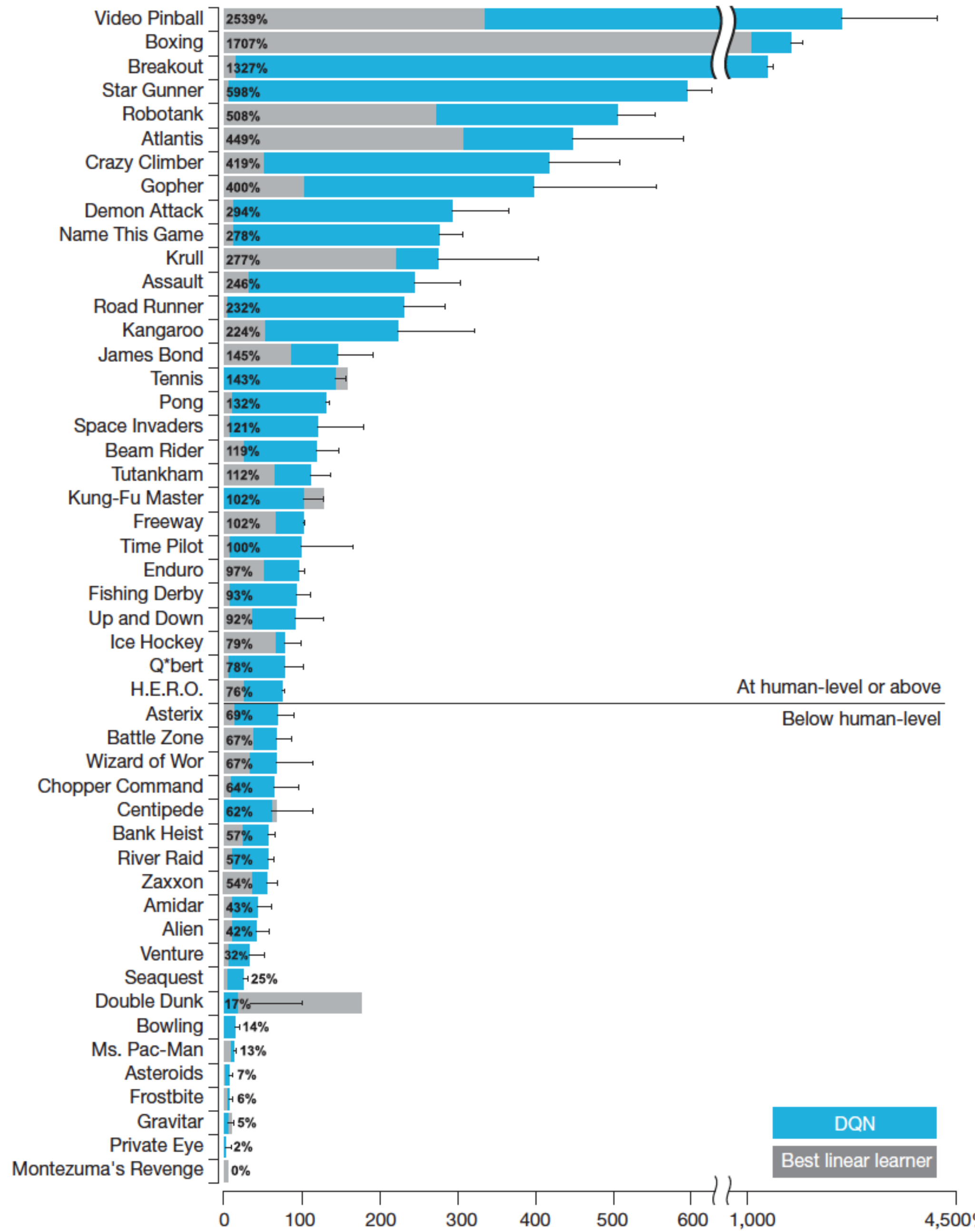
Overview

- DQN
- REINFORCE
- Off-policy policy gradient

Deep Q-learning

> Deep Q-Network (DQN)





Deep Q-learning

- Epsilon greedy policy gives a and observe (s, a, s', r)
- Q-learning

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a} Q(s',a') - Q(s,a))$$

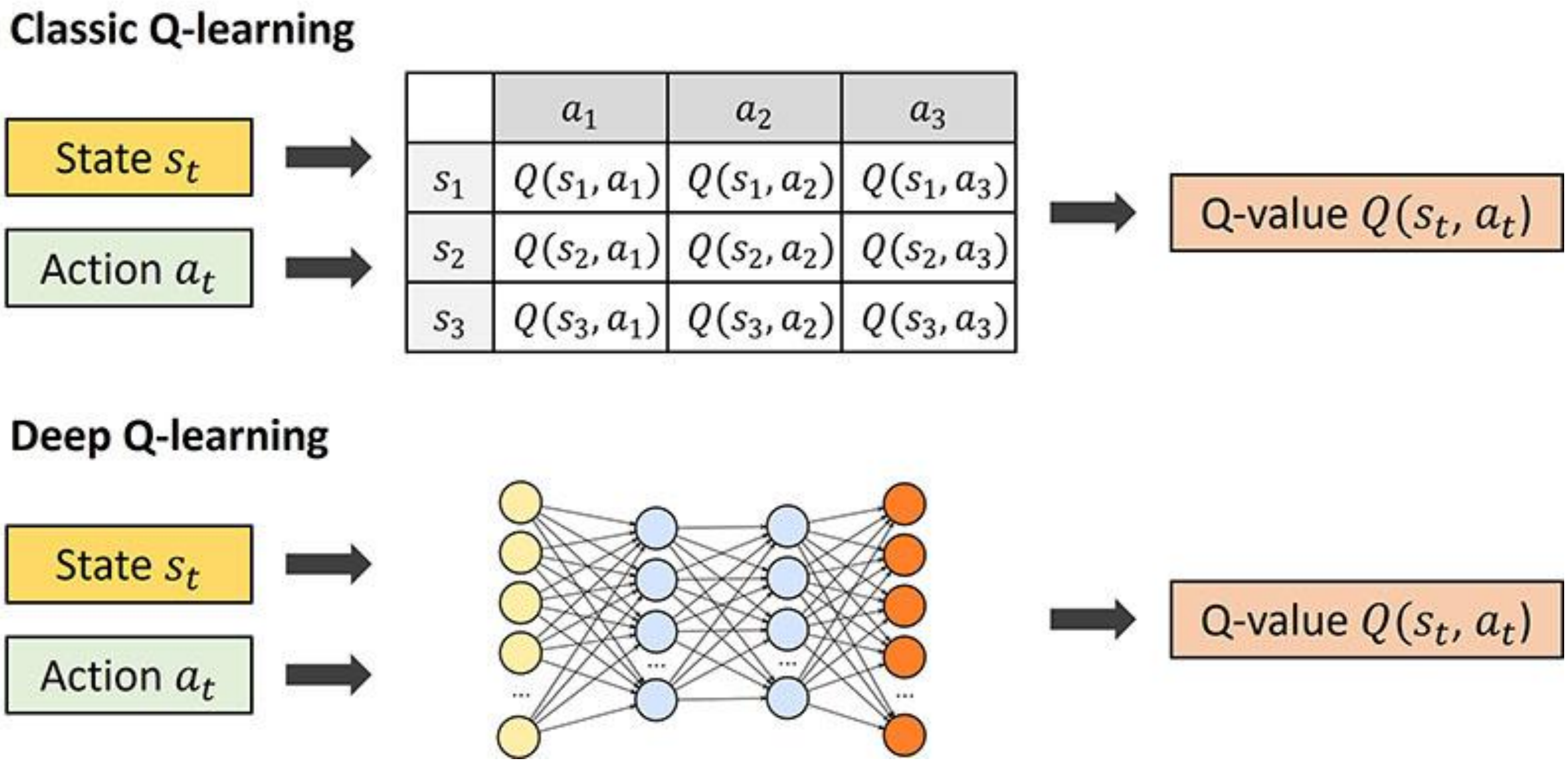
- DQN, Q function is represented by neural network
  - $Q \sim Q_\phi$
  - sample from the buffer and compute the target

$$y_i = r(s_i, a_i) + \gamma \max_{a} Q_{\phi^-}(s_i', a_i')$$

  - update

$$\phi \leftarrow \mathrm{argmin}_\phi \sum_i \left\| Q_\phi(s_i, a_i) - y_i \right\|^2$$

Deep Q-learning



Deep Q-learning

> Convergence guarantee under some assumptions
  - data collection policy has good coverage
  - realizability: Q_ϕ can represent Q
  - ϕ has some good property

> Online update
  - $\phi \leftarrow \mathrm{argmin}_\phi \sum_i \left\| Q_\phi(s_i, a_i) - y_i \right\|^2$
  - $\phi \leftarrow \phi - \eta \sum_i \big(Q_\phi(s_i, a_i) - y_i \big) \nabla_\phi Q_\phi(s_i, a_i)$

> DQN paper ideas (DeepMind 2013, later 2015 nature)
  - Replay buffer: Q-learning is off-policy! save all transition data (s, a, s', r)
  - target network: use two Q-networks, periodically update parameters

Deep Q-learning

- Practical tips (general RL tips)
  - it can take a while to converge (be patient)
  - high exploration initially, then reduce
  - test on known / easy tasks first
  - use / add more advanced features
  - use a well tested library (e.g., stable-baselines3)

DQN extensions

> DDQN

> Prioritized experience replay

> Dueling architecture

> **Q-learning with continuous actions -> DDPG**

Q-learning with continuous actions

> How to solve max or argmax with continuous actions?

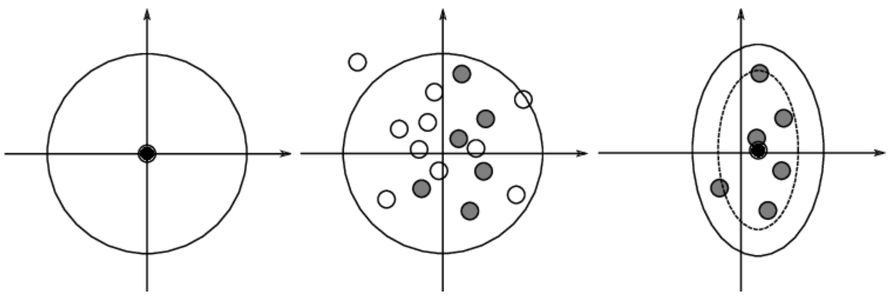
$$\mathbb{E}_{(s,a,s',r)}[\left(Q_\phi(s,a) - r(s,a) - \gamma \max_{a'} Q_{\phi^-}(s',a')\right)^2]$$

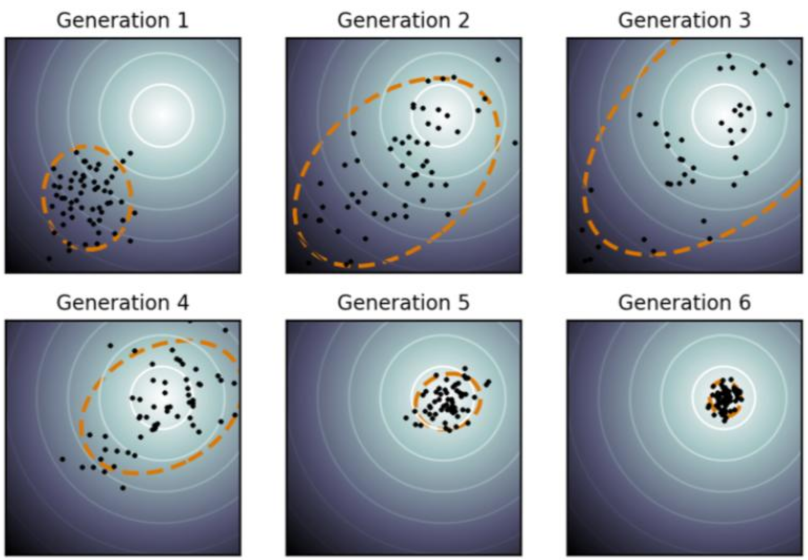
> Option 1: discretization
  - not scalable

> Option 2: optimization
  - gradient-based optimization (GD, SGD) on $Q_\phi(s',a')$
  - stochastic optimization
    - Cross-entropy method (CEM): simple iterative stochastic optimization
    - CMA-ES: more complex iterative stochastic optimization
  - slow and stuck in the local minimum

Stochastic optimization

- CEM / CMA-ES





Q-learning with continuous actions

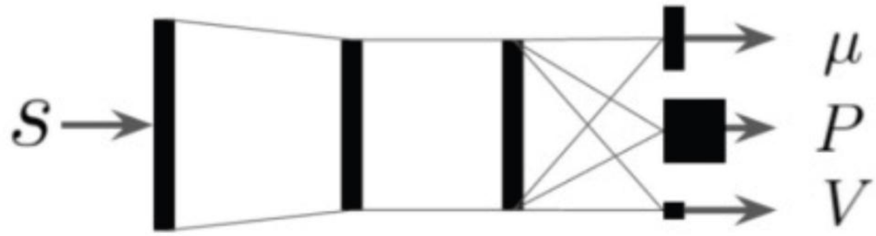
> How to solve max or argmax with continuous actions?

$$\mathbb{E}_{(s,a,s',r)}[\left(Q_\phi(s,a) - r(s,a) - \gamma \max_{a'} Q_{\phi^-}(s',a')\right)^2]$$

> Option 3: structured Q functions that are easy to optimize
- example: normalized advantage functions (NAF)
- problem: less expressive power

$$Q_\phi(\mathbf{s},\mathbf{a}) = -\frac{1}{2}(\mathbf{a} - \mu_\phi(\mathbf{s}))^T P_\phi(\mathbf{s})(\mathbf{a} - \mu_\phi(\mathbf{s})) + V_\phi(\mathbf{s})$$

$$\arg\max_{\mathbf{a}} Q_\phi(\mathbf{s},\mathbf{a}) = \mu_\phi(\mathbf{s}) \qquad \max_{\mathbf{a}} Q_\phi(\mathbf{s},\mathbf{a}) = V_\phi(\mathbf{s})$$



Q-learning with continuous actions

> How to solve max or argmax with continuous actions?

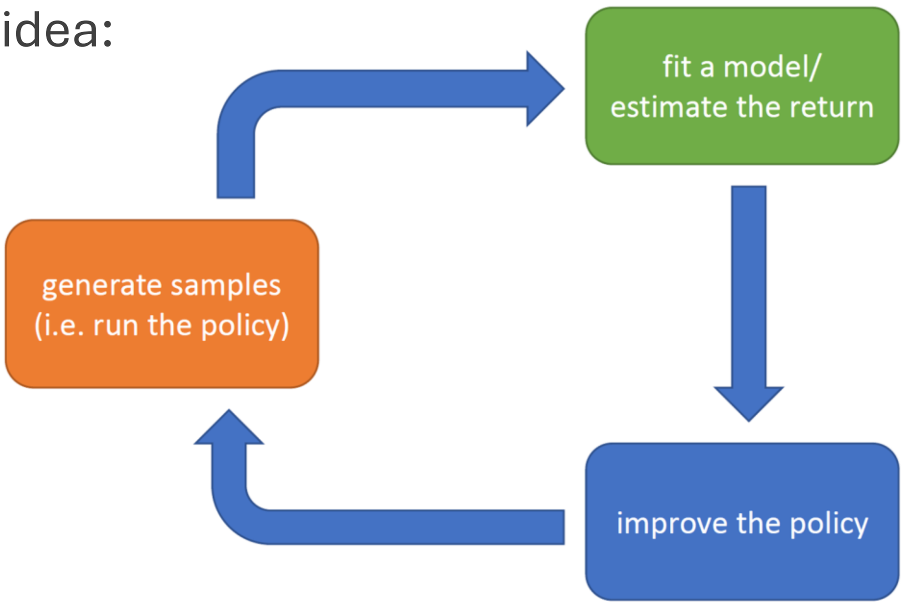
$$\mathbb{E}_{(s,a,s',r)}[\left(Q_\phi(s,a) - r(s,a) - \gamma \max_{a'} Q_{\phi^-}(s',a')\right)^2]$$

> **Option 4: learn an approximate optimizer**

- DDPG (deep deterministic policy gradient)
- Main idea: train another DNN $\mu_\theta(s)$ such that $\mu_\theta(s) \approx \arg\max_a Q_\phi(s,a)$
- $\mu_\theta(s)$ is essentially a policy
- How? gradient ascent on $Q_\phi(s',a')$, using a chain rule $\frac{dQ_\phi}{d\phi} = \frac{dQ_\phi}{da}\frac{da}{d\theta}$
- Policy gradient: can we directly optimize $\mu_\theta(s)$?

Summary of Q-learning

- Key idea:



$$Q_\phi(s,a) \leftarrow r + \gamma \max_a Q_\phi(s',a')$$

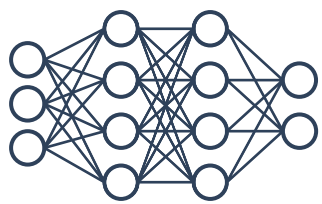
$$a = argmax_a Q_\phi(s,a)$$

- Q-learning is off-policy and model-free
- Q-learning doesn’t explicitly learn a policy
- Q-learning could be hard with continuous actions
- Online Q-learning with neural network approximations: DQN

Why policy gradient?

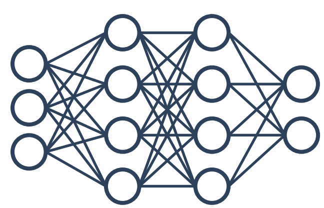
> Key philosophy: why learn value function when all we need is a policy?
  - In some cases, policy might be easier to learn!
  - ex) In LQR, the optimal policy is linear while the optimal value is quadratic

> DQN:

$s, a \longrightarrow$  $\longrightarrow Q_\phi(s, a) \longrightarrow a = arg\max_a Q(s, a)$

ϕ

> Policy gradient: directly optimize the policy π_ϕ via $\nabla_\phi J(\phi)$

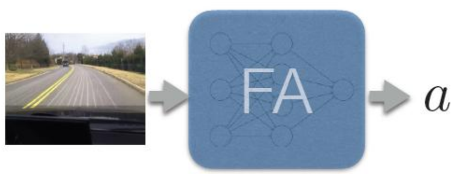
$s \longrightarrow$  $\longrightarrow \pi_\phi(a|s) \longrightarrow a$

ϕ

Policy functions

> Advantages over value based methods
  - effective in high-dimensional or continuous action spaces
  - can learn stochastic policies

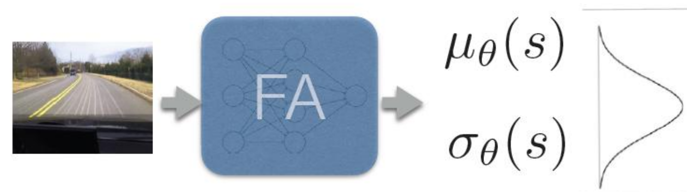
deterministic continuous policy



$$a = \pi_\theta(s)$$

e.g. outputs a steering angle directly

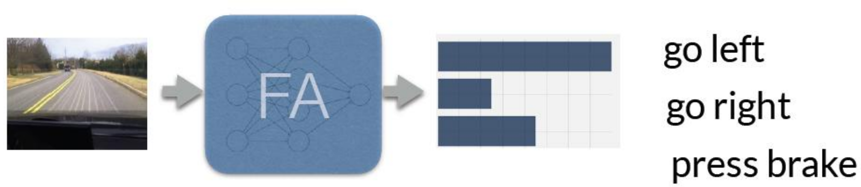
stochastic continuous policy



$$a \sim \mathcal{N}(\mu_\theta(s), \sigma_\theta^2(s))$$

FA for stochastic multimodal continuous policies is an active area of research

(stochastic) policy over discrete actions



Outputs a distribution over a discrete set of actions

Policy gradient theorem

- Policy gradient: directly optimize the policy π_ϕ via $\nabla_\phi J(\phi)$
  - $p_\phi(s_1, a_1, \dots, s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi_\phi(a_t|s_t) p(s_{t+1}|s_t, a_t)$

$$\phi^* = argmax_\phi \underbrace{\mathbb{E}_{\tau \sim p_\phi(\tau)}\left[\sum_t r(s_t, a_t)\right]}_{J(\phi)}$$

- Given some policy π_ϕ, how to evaluate its $J(\phi)$
- Simple idea: Monte-Carlo sampling
  - sample N trajectories using π_ϕ
  - Monte-Carlo estimation of $J(\phi)$, $J(\phi) = \frac{1}{N} \sum_i \sum_t r(s_{i,t}, a_i, t)$

- Can we estimate $\nabla_\phi J(\phi)$?

Policy gradient theorem

> How to compute $\nabla_\phi J(\phi)$?
  - $\nabla_\phi J(\phi)$ is hard to compute: ϕ influences both the action selections and the distribution of states in which those selections are made

> Policy gradient theorem:

$$\nabla_\phi J(\phi) = E_{\tau \sim p_\phi(\tau)}[R(\tau) \sum_t \nabla_\phi log\pi_\phi(a_t|s_t)]$$

$$R(\tau) = \sum_t r(s_t, a_t)$$

Policy gradient theorem

> Derivation

- $J(\phi) = \mathbb{E}_{\tau \sim p_\phi(\tau)}[\sum_t r(s_t, a_t)] = \int p_\phi(\tau) R(\tau) d\tau$
- $\nabla_\phi J(\phi) = \int \nabla_\phi p_\phi(\tau) R(\tau) d\tau = \int p_\phi(\tau) \nabla_\phi \log p_\phi(\tau)\, R(\tau) d\tau$
  $= \mathbb{E}_{\tau \sim p_\phi(\tau)}[\nabla_\phi \log p_\phi(\tau) R(\tau)]$

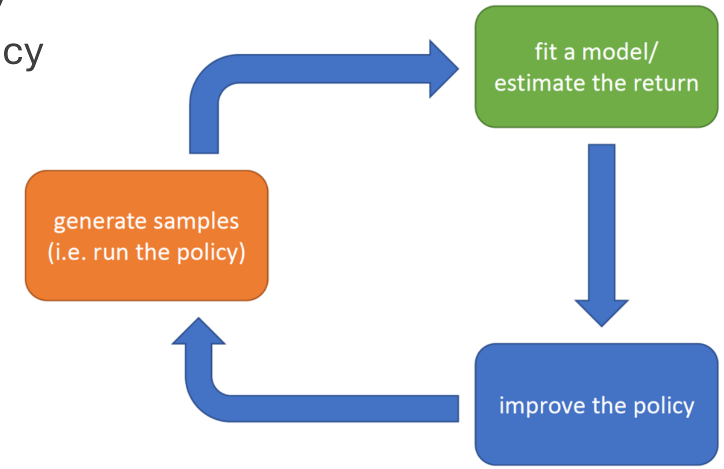
  since $p_\phi(\tau) \nabla_\phi \log p_\phi(\tau) = p_\phi(\tau) \frac{\nabla_\phi p_\phi(\tau)}{p_{\phi(\tau)}} = \nabla_\phi p_\phi(\tau)$

- $$\begin{aligned}\nabla_\phi \log p_\phi(\tau) &= \nabla_\phi \log p(s_1) \prod_{t=1}^{T} \pi_\phi(a_t|s_t) p(s_{t+1}|s_t, a_t) \\ &= \nabla_\phi \log p(s_1) + \sum_{t=1}^{T} \log \pi_\phi(a_t|s_t) + \log p(s_{t+1}|s_t, a_t) \\ &= \nabla_\phi \sum_{t=1}^{T} \log \pi_\phi(a_t|s_t) \\ &= \sum_{t=1}^{T} \nabla_\phi \log \pi_\phi(a_t|s_t)\end{aligned}$$

- $\nabla_\phi J(\phi) = \mathbb{E}_{\tau \sim p_\phi(\tau)}[\sum_{t=1}^{T} \nabla_\phi \log \pi_\phi(a_t|s_t)\, R(\tau)]$

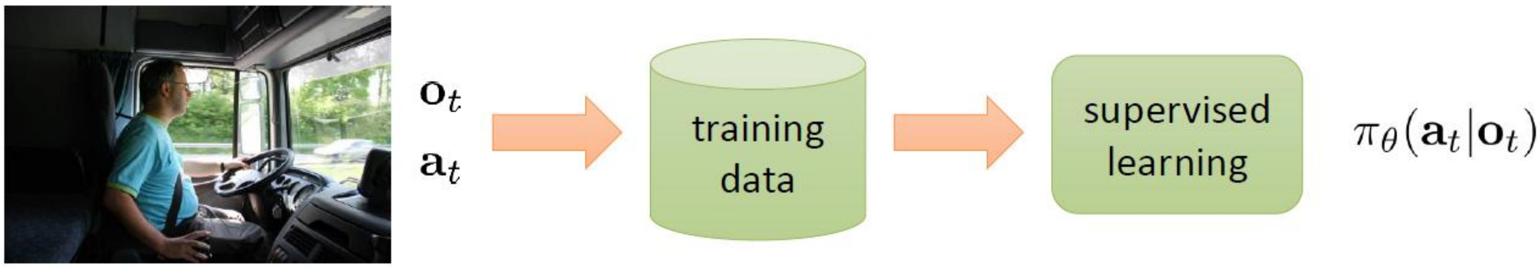
REINFORCE

- REINFORCE = Monte-Carlo PG
  - estimate the expectation by Monte-Carlo sampling
  - $\nabla_\phi J(\phi) = \mathbb{E}_{\tau \sim p_\phi(\tau)}\left[\sum_{t=1}^{T} \nabla_\phi \log \pi_\phi(a_t|s_t)\, R(\tau)\right]$
    $\approx \frac{1}{N}\sum_i R(\tau^i)(\sum_t \nabla_\phi \log \pi_\phi(a_{i,t}|s_{i,t})\,)$
  - Algorithm (Sutton et al., NeurIPS 1998)
    1. sample $\{\tau^i\}$ from $\pi_\phi(a_t|s_t)$ (run the policy
    2. estimate $\nabla_\phi J(\phi)$
    3. $\phi \leftarrow \phi + \alpha \nabla_\phi J(\phi)$ and go to 1



Understanding REINFORCE

> In imitation learning



maximum likelihood: $\nabla_\phi J_{ML}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}(\sum_{t=1}^{T} \nabla_\phi \log \pi_\phi(a_{i,t}|s_{i,t}))$

- maximizing log likelihood of the policy π_ϕ on expert data

> Policy gradient:

- imitation gradient, but weighted by reward

$$\nabla_\phi J(\phi) \approx \frac{1}{N}\sum_{i=1}^{N} R(\tau^i)(\sum_{t=1}^{T} \nabla_\phi \log \pi_\phi(a_{i,t}|s_{i,t}))$$

Understanding REINFORCE

> Policy gradient:

$$\nabla_\phi J(\phi) \approx \frac{1}{N} \sum_{i=1}^{N} R(\tau^i) \left(\sum_{t=1}^{T} \nabla_\phi \log \pi_\phi(a_{i,t} | s_{i,t}) \right)$$

- with out expert data: signal is from $R(\tau^i)$
- Good stuff (bigger $R(\tau^i)$) is made more likely
- Bad stuff is made less likely
- formalization of trial-and-error

- It is on-policy ($\mathbb{E}_{\tau \sim p_\phi(\tau)}$) and model-free (no use of $p(s_{t+1} | s_t, a_t)$)
- Markov property is not used $\Rightarrow$ it can be used in POMDP

Understanding REINFORCE

> Policy gradient:

$$\nabla_\phi J(\phi) \approx \frac{1}{N} \sum_{i=1}^{N} R(\tau^i) \left(\sum_{t=1}^{T} \nabla_\phi \log \pi_\phi (a_{i,t} | s_{i,t}) \right)$$

> Weakness
- It has high variance because it relies on full-trajectory
- There might be good samples, but have $R\left(\tau^i\right) = 0$
- It requires waiting until the end of the episodes

Reducing variance

- $\nabla_\phi J(\phi) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_t \nabla_\phi \log \pi_\phi\left(a_{i,t} \middle| s_{i,t}\right) \sum_t r(s_{i,t}, a_{i,t})\right)$

- Causality: policy at time t' cannot affect reward $t < t'$

$$\nabla_\phi J(\phi) \approx \frac{1}{N}\sum_{i=1}^{N} \sum_t \nabla_\phi \log \pi_\phi(a_{i,t}|s_{i,t}) \underbrace{\left(\sum_{t'=t} r\left(s_{i,t}, a_{i,t}\right)\right)}_{\substack{\textit{Rewards to go} \\ Q^\pi(s,a)}}$$

- Another form of PG theorem, using Q^π to estimate return

$$\nabla_\phi J(\phi) = \mathbb{E}_{s\sim p_\phi, a\sim\pi_\phi}\left[\sum_{t=1}^{T} \nabla_\phi \log \pi_\phi(a_t|s_t)\, Q^\pi(s,a)\right]$$

Reducing variance

> Add a bias term does not affect gradient

$$
\begin{aligned}
&\mathbb{E}_{s\sim p_\phi, a\sim\pi_\phi}\left[\sum_{t=1}^{T} \nabla_\phi \log \pi_\phi(a_t|s_t)\, b(s)\right] \\
&= \iint p_\phi(s)\pi_\phi(a|s)\,\frac{\nabla_\phi \pi_\phi(a|s)}{\pi_\phi(a|s)}\, b(s)\, dads \\
&= \iint p_\phi(s)\nabla_\phi \pi_\phi(a|s) b(s)\, dads \\
&= \int p_\phi(s) b(s) \nabla_\phi \int \pi_\phi(a|s) da\, ds \\
&= \int p_\phi(s) b(s) \nabla_\phi 1 ds = 0
\end{aligned}
$$

> Therefore

$$
\nabla_\phi J(\phi) = \mathbb{E}_{s\sim p_\phi, a\sim\pi_\phi}\left[\sum_{t=1}^{T} \nabla_\phi \log \pi_\phi(a_t|s_t)\, (Q^\pi(s,a) - b(s))\right]
$$

- average reward or state value function can be used for $b(s)$
- optimal bias is the expected reward weighted by gradient magnitudes

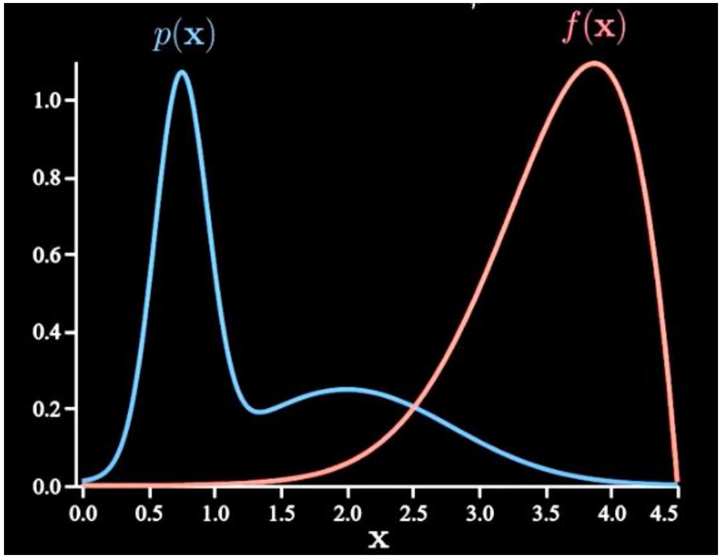
Off-policy PG

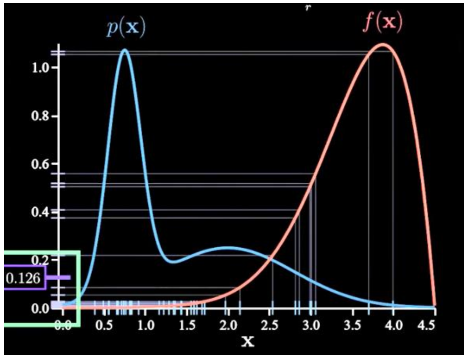
- Neural networks change only a little bit with each gradient step
  - since we have a tons of parameters
- On-policy learning can be extremely inefficient!
  - cannot reuse old data

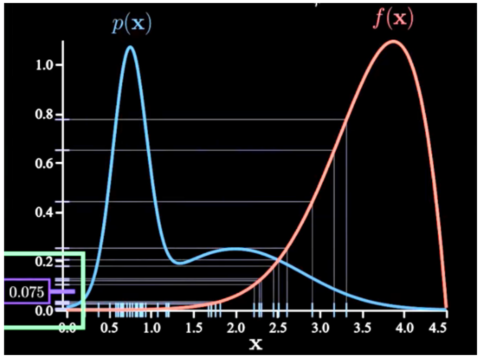
- Algorithm
  1. sample $\{\tau^i\}$ from $\pi_\phi(a_t|s_t)$ (run the policy)
  2. estimate $\nabla_\phi J(\phi)$
  3. $\phi \leftarrow \phi + \alpha \nabla_\phi J(\phi)$ and go to 1 $\leftarrow$ we change ϕ here

Importance sampling

> How can we use the data from other policies?
  - $$\begin{aligned}\mathbb{E}_{x\sim p(x)}[f(x)] &= \int p(x)f(x)dx \\ &= \int \frac{q(x)}{q(x)}p(x)f(x)dx \\ &= \int q(x)\frac{p(x)}{q(x)}f(x)dx \\ &= \mathbb{E}_{x\sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right]\end{aligned}$$

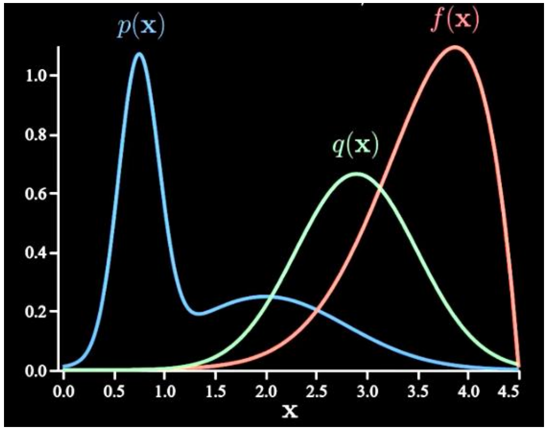


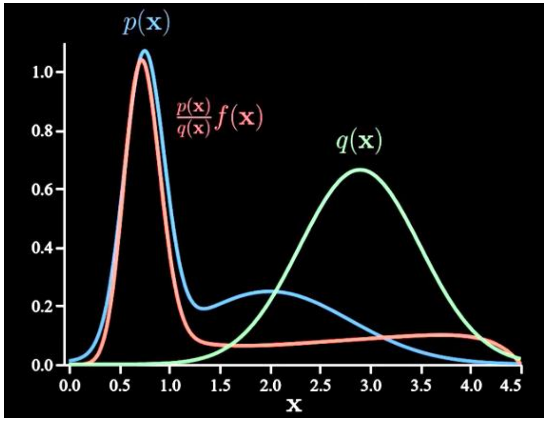


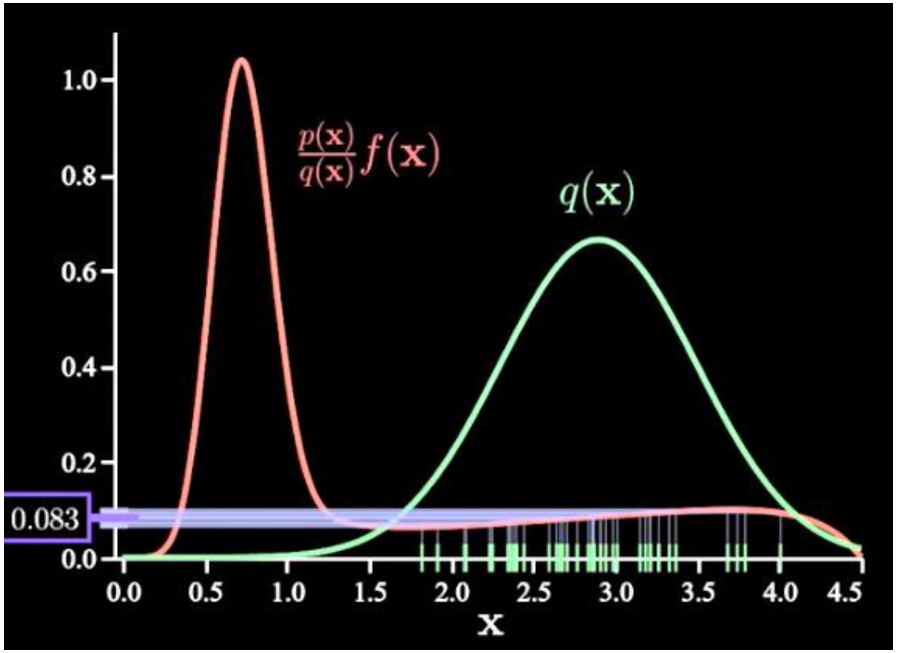


Importance sampling

> How can we use the data from other policies?
  - $\mathbb{E}_{x\sim p(x)}[f(x)] = \mathbb{E}_{x\sim q(x)}\left[\frac{p(x)}{q(x)} f(x)\right]$







Off-policy PG

- How can we use the data from other policies?
  - $\mathbb{E}_{x\sim p(x)}[f(x)] = \mathbb{E}_{x\sim q(x)}\left[\frac{p(x)}{q(x)} f(x)\right]$

- Likewise, we have samples from different policy $p_{\phi'}(\tau)$
  - $J(\phi') = \mathbb{E}_{\tau\sim p_{\phi'}(\tau)}[R(\tau)] = \mathbb{E}_{\tau\sim p_{\phi}(\tau)}\left[\frac{p_{\phi'}(\tau)}{p_{\phi}(\tau)} R(\tau)\right]$
  - $\nabla_{\phi'} J(\phi') = \mathbb{E}_{\tau\sim p_{\phi}(\tau)}\left[\nabla_{\phi'} \frac{p_{\phi'}(\tau)}{p_{\phi}(\tau)} R(\tau)\right]$

$$= \mathbb{E}_{\tau\sim p_{\phi}(\tau)}\left[\frac{p_{\phi'}(\tau)}{p_{\phi}(\tau)} \nabla_{\phi'} \log p_{\phi'}(\tau)\, R(\tau)\right]$$

$$= \mathbb{E}_{\tau\sim p_{\phi}(\tau)}\left[\frac{p_{\phi'}(\tau)}{p_{\phi}(\tau)} \left(\sum_t \nabla_{\phi'} \log \pi_{\phi'}(a_t|s_t)\right) \left(\sum_t r(s_t, a_t)\right)\right]$$

Off-policy PG

- $\nabla_{\phi'} J(\phi') = \mathbb{E}_{\tau \sim p_\phi(\tau)}\left[\frac{p_{\phi'}(\tau)}{p_\phi(\tau)}\left(\sum_t \nabla_{\phi'} \log \pi_{\phi'}(a_t|s_t)\right)\left(\sum_t r(s_t, a_t)\right)\right]$

- $\frac{p_{\phi'}}{p_\phi} = \frac{p(s_1)\prod_t \pi_{\phi'}(a_t|s_t) p(s_{t+1}|s_t, a_t)}{p(s_1)\prod_t \pi_\phi(a_t|s_t) p(s_{t+1}|s_t, a_t)} = \frac{\prod_t \pi_{\phi'}(a_t|s_t)}{\prod_t \pi_\phi(a_t|s_t)}$

- $\nabla_{\phi'} J(\phi') = \mathbb{E}_{\tau \sim p_\phi(\tau)}\left[\underline{\left(\frac{\prod_t \pi_{\phi'}(a_t|s_t)}{\prod_t \pi_\phi(a_t|s_t)}\right)}\left(\sum_t \nabla_{\phi'} \log \pi_{\phi'}(a_t|s_t)\right)\left(\sum_t r(s_t, a_t)\right)\right]$
- Causality

$$= \mathbb{E}_{\tau \sim p_\phi(\tau)}\left[\underline{\left(\frac{\prod_t \pi_{\phi'}(a_t|s_t)}{\prod_t \pi_\phi(a_t|s_t)}\right)}\left(\sum_t \nabla_{\phi'} \log \pi_{\phi'}(a_t|s_t)\right)\left(\sum_{t'=t} r(s_t, a_t)\right)\right]$$

- Problem: orange is exponential in horizon, can become very small or very large. Also hard to measure

Off-policy PG

> Reducing variance

- $\mathbb{E}_{\tau\sim p_\phi(\tau)}\left[\underline{\left(\frac{\prod_t \pi_{\phi'}(a_t|s_t)}{\prod_t \pi_\phi(a_t|s_t)}\right)}\left(\sum_t \nabla_{\phi'} \log \pi_{\phi'}(a_t|s_t)\right)\left(\sum_{t'=t} r(s_t, a_t) - b\right)\right]$

> Instead of using trajectories, consider the expectation over timesteps

- $\nabla_{\phi'} J(\phi') \approx \frac{1}{N}\sum_i \sum_t \left(\frac{\pi_{\phi'}(a_{i,t}|s_{i,t})}{\pi_\phi(a_{i,t}|s_{i,t})}\right)\left(\nabla_{\phi'} \log \pi_{\phi'}(a_{i,t}|s_{i,t})\right)\left(\left(\sum_{t'=t} r(s_{i,t}, a_{i,t}\right) - b\right)$

- this would be the common final form
- we can take multiple gradients steps on the same batch

> Algorithm

1. sample $\{\tau^i\}$ from $\pi_\phi(a_t|s_t)$ (run the policy
2. estimate $\nabla_\phi J(\phi)$
3. $\phi \leftarrow \phi + \alpha \nabla_\phi J(\phi)$ and go to 1

can take multiple gradient steps

Off-policy PG

- What if our policy changes a lot before sampling new data?
  - Data no longer reflects states that policy will visit
  - gradient estimate less accurate

- Can we constrain the policy not stray too far during gradient update?
  - One common choice: $\mathbb{E}_{s \sim \pi_\phi}\left[D_{KL}\left(\pi_{\phi'}(\cdot \mid s) \parallel \pi_\phi(\cdot \mid s)\right)\right] \leq \delta$
  - We will learn in the future

Review

- Since the gradient has high variance,
  - consider using much larger batches
  - tuning the learning rates is challenging
  - Adaptive step size rules like Adam can be used

- Policy gradient is on-policy
  - off-policy variant using importance sampling
  - Some approximations are made by ignoring parts of the state
  - We can apply multiple gradients updates

- Intuition
  - do more high reward stuff, less low reward stuff
  - Gradient will be very noisy, best with large batch sizes and dense rewards