Reinforcement Learning
Lecture 10: Deep Reinforcement Learning [1]

Lecturer: Prof. Dr. Mathias Niepert

Institute for Parallel and Distributed Systems
Machine Learning and Simulation Lab

University of Stuttgart
Germany

imprs-is

July 6, 2023

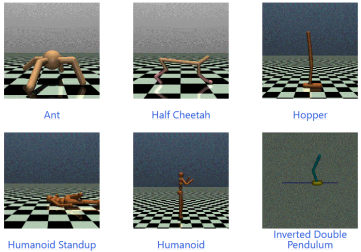
[1]

Introduction

Scaling-up reinforcement learning

- Sparse rewards
  - e.g. gridworld
- Large *state* spaces:
  - Go: $\log_{10}|S| = \log_{10}(3^{19\times 19}) \approx 170 > 82$
  - Camera images , e.g. $\log_{10}|S| = \log_{10}((256^3)^{1280\times 720}) \gg 82$ (3 color channels, 8 bits each)
  - Continuous spaces: e.g. inverted pendulum, mobile robot, etc.

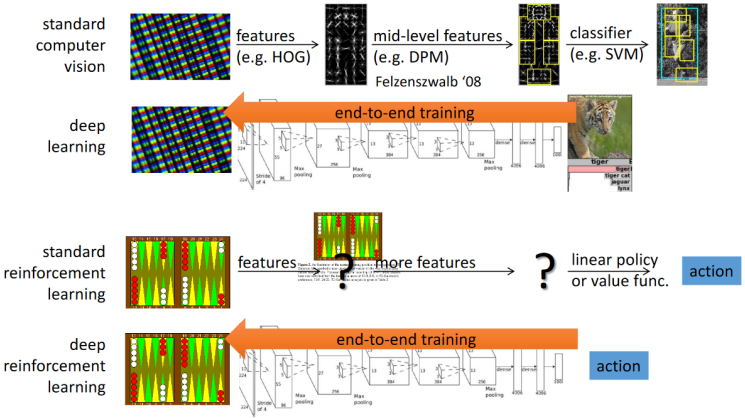
MuJoCo



nb. of atoms in the universe: $N = 10^{82}$ ($\log_{10} N = 82$)

https://www.youtube.com/watch?v=iaF43Ze1oeI&ab_channel=PeterPastor

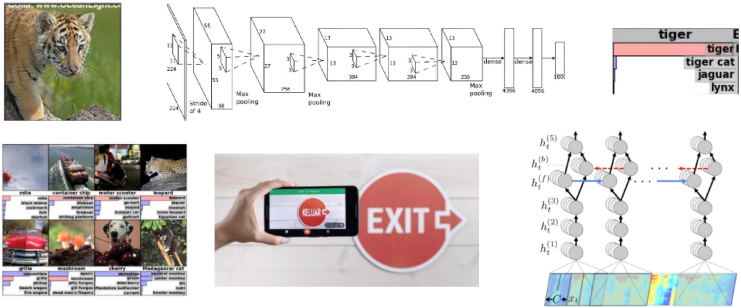
What is deep RL, and why should we care?



HOG: Histogram of Oriented Gradients, DPM: Deformable Part Models

Slide source: `http://rail.eecs.berkeley.edu/deeprlcourse/`

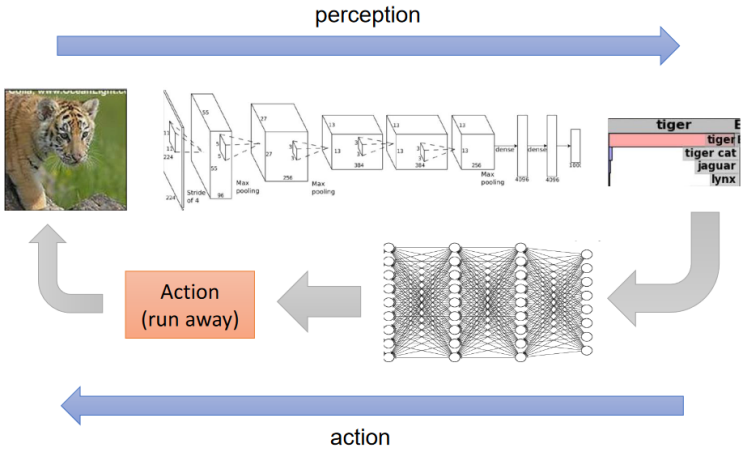
Deep learning helps us handle unstructured environments



These are all **open world** examples

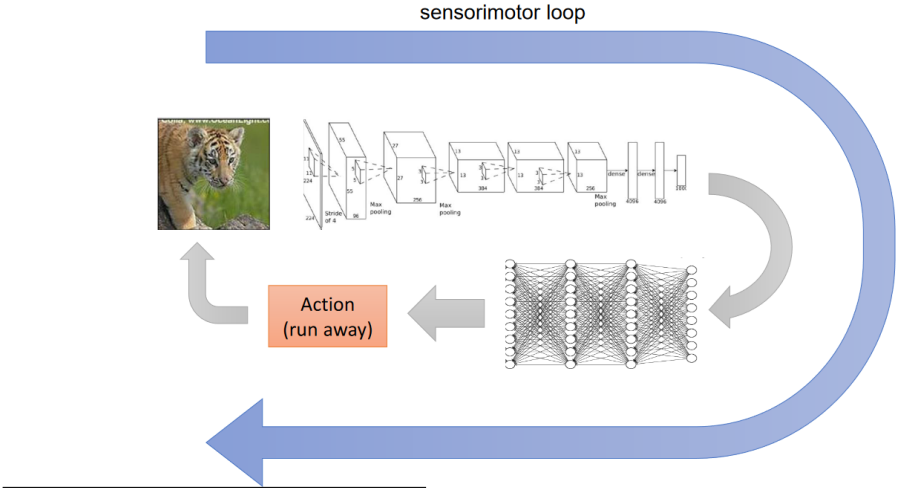
Slide source: `http://rail.eecs.berkeley.edu/deeprlcourse/`

What does end-to-end learning mean for sequential decision making?

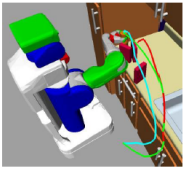


Slide source: `http://rail.eecs.berkeley.edu/deeprlcourse/`

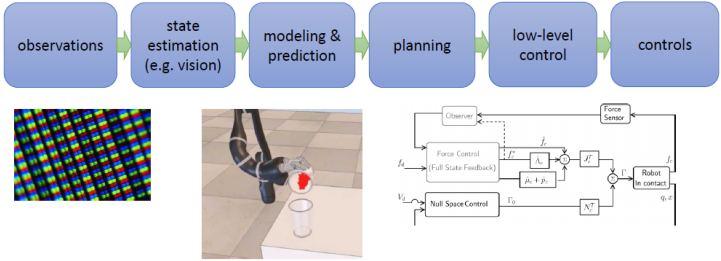
What does end-to-end learning mean for sequential decision making?



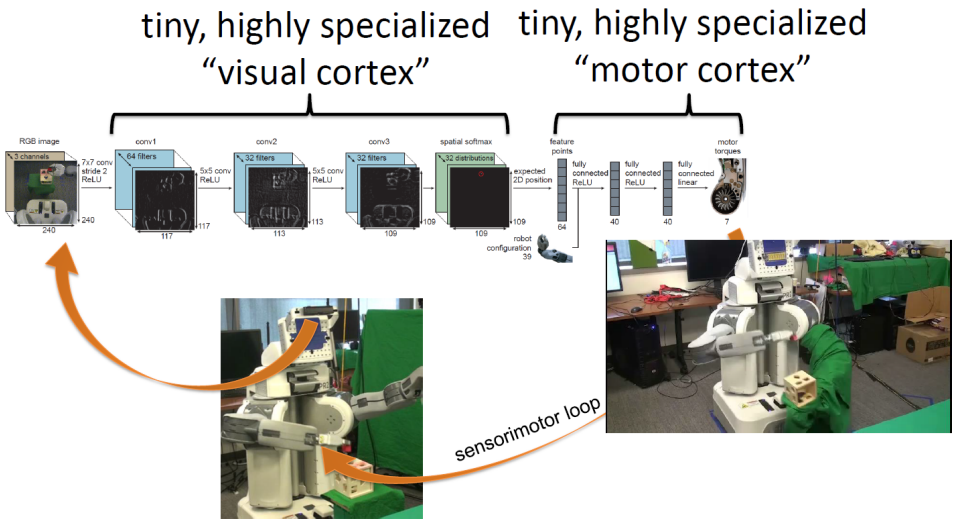
Slide source: `http://rail.eecs.berkeley.edu/deeprlcourse/`



robotic control pipeline

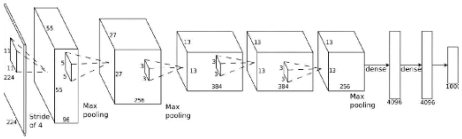


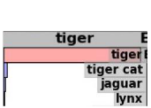
Slide source: `http://rail.eecs.berkeley.edu/deeprlcourse/`



Slide source: `http://rail.eecs.berkeley.edu/deeprlcourse/`

Why should we study this **now**?

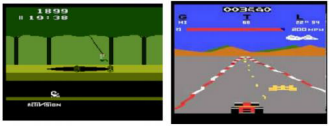




1. Advances in deep learning
2. Advances in reinforcement learning
3. Advances in computational capability

Slide source: `http://rail.eecs.berkeley.edu/deeprlcourse/`

Why should we study this **now**? (state-of-the-art)



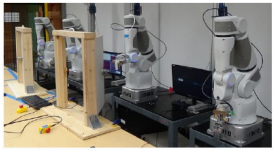
Atari games:

Q-learning:

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, et al. "Playing Atari with Deep Reinforcement Learning". (2013).

Policy gradients:

J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. "Trust Region Policy Optimization". (2015).
V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, et al. "Asynchronous methods for deep reinforcement learning". (2016).



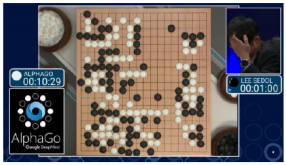
Real-world robots:

Guided policy search:

S. Levine*, C. Finn*, T. Darrell, P. Abbeel. "End-to-end training of deep visuomotor policies". (2015).

Q-learning:

D. Kalashnikov et al. "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". (2018).



Beating Go champions:

Supervised learning + policy gradients + value functions + Monte Carlo tree search:

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, et al. "Mastering the game of Go with deep neural networks and tree search". Nature (2016).

Slide source: `http://rail.eecs.berkeley.edu/deeprlcourse/`

Outline

Value Based

Naive deep Q-learning

- Q-learning update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(\underbrace{r + \gamma \max_a Q(s',a)}_{\text{TD off-policy target}} - Q(s,a) \right)$$

- Q is represented by a neural network with weights $\boldsymbol{w}$: $\hat{q}(s,a,\boldsymbol{w})$
- Loss is the mean-squared TD-error:

$$\mathcal{L}(\boldsymbol{w}) = \mathbb{E}\left[\left(r + \gamma \max_a \hat{q}(s',a,\boldsymbol{w}) - \hat{q}(s,a,\boldsymbol{w}]) \right)^2 \right]$$

- Minimize sample errors with SGD

Recap: Stochastic Gradient Descent (SGD)

- *Mean squared Value Error*: $\mathcal{L}(\boldsymbol{w}) = \sum_{s \in S} \mu(s) \left[q_\pi(s,a) - \hat{q}(s,a,\boldsymbol{w})\right]^2$
- Adjust $\boldsymbol{w}$ to reduce the error on sample $(S_t, A_t) \mapsto q_\pi$:

$$\begin{aligned}
\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \frac{1}{2}\alpha_t(\boldsymbol{\nabla}\mathcal{L}_t)(\boldsymbol{w}_t) \\
&= \boldsymbol{w}_t - \frac{1}{2}\alpha_t \boldsymbol{\nabla} \underbrace{[q_\pi - \hat{q}_{\boldsymbol{w}_t}]^2}_{\textbf{squared sample error}} \\
&= \boldsymbol{w}_t + \alpha_t [q_\pi - \hat{q}] \boldsymbol{\nabla}_{\boldsymbol{w}} \hat{q}
\end{aligned}$$

- α_t is a step size parameter
- why not use $\alpha = 1$, thus eliminating full error on sample?

Recap: problems with Q-learning stability

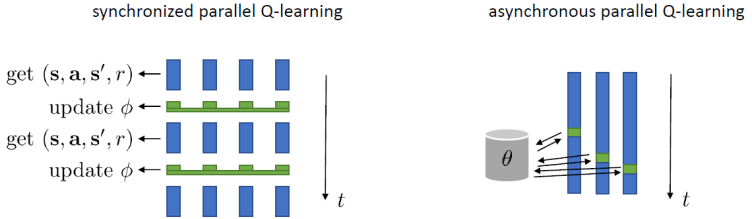
Naive Q-learning with neural networks oscillates or diverges:

1. Data is non i.i.d.!
   - trajectories
   - samples are **correlated (generated by interaction)**
2. Policy changes rapidly with slight changes to Q-values
   - policy may oscillate
3. Reward range is unknown
   - gradients can be large
   - instabilities during back-propagation
4. Maximization bias

Correlated samples in online Q-learning

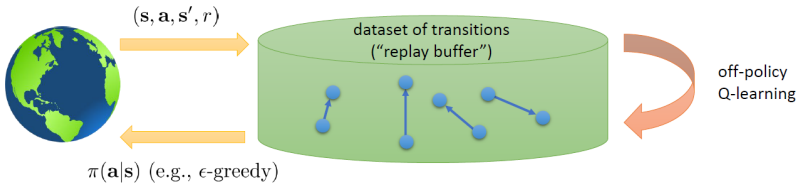
Online Q-learning algorithm with function approximation

1. Take some action A_t and observe (S_t, A_t, R_t, S_{t+1})
2. $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \Big(\underbrace{R_t + \gamma \max_{A'} \hat{q}_{\boldsymbol{w}}(S_{t+1}, A')}_{\text{TD target}} - \hat{q}_{\boldsymbol{w}}(S_t, A_t) \Big) \boldsymbol{\nabla}_{\boldsymbol{w}} \hat{q}_{\boldsymbol{w}}(S_t, A_t)$



Another solution: replay buffers

Take some action A_t and observe (S_t, A_t, R_t, S_{t+1})
for $i = 0, 1, \ldots, K$ **do** ▷ Sample mini-batch
  $\boldsymbol{y}_i \leftarrow R_i + \gamma \max_{A'_i} \hat{q}_{\boldsymbol{w}}(S_i, A'_i)$ ▷ TD target
  $\boldsymbol{w} \leftarrow \arg\min_{\boldsymbol{w}} \frac{1}{2} \sum_i \left\| \hat{q}_{\boldsymbol{w}}(S_i, A_i) - \boldsymbol{y}_i \right\|^2$ ▷ Gradient descent
end for



Target Network : $\nabla \mathcal{L}(\boldsymbol{w}) \approx \mathbb{E}\left[\left(R_i + \gamma \max_{A'} \hat{q}_{\boldsymbol{w}} - \hat{q}_{\boldsymbol{w}}\right) \nabla_{\boldsymbol{w}} \hat{q}_{\boldsymbol{w}}\right]$

while alive **do**
  Save target network parameters $\boldsymbol{w}' \leftarrow \boldsymbol{w}$
  for $n = 0, 1, \ldots, N$ **do**
    Take action A_t and observe (S_t, A_t, R_t, S_{t+1}), add to buffer $\mathcal{B}$
    for $k = 0, 1, \ldots, K$ **do**
      Sample a transition (S_i, A_i, R_i, S_{i+1}) from $\mathcal{B}$
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \sum_i \left(R_i + \gamma \max_{A'_i} \hat{q}_{\boldsymbol{w}'}(S_{i+1}, A'_i) - \hat{q}_{\boldsymbol{w}}(S_i, A_i) \right) \nabla_{\boldsymbol{w}} \hat{q}_{\boldsymbol{w}}(S_i, A_i)$
    end for
  end for
end while

Deep Q-networks (DQN)

Deep Q-networks (DQN) address instabilities through:

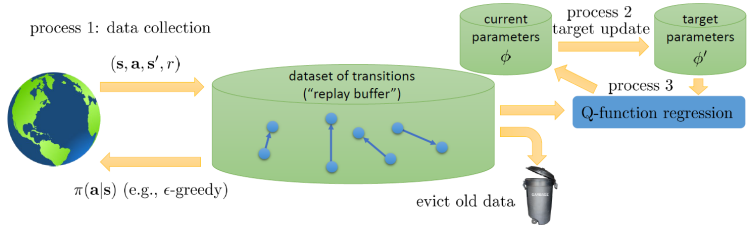
- **Experience replay**
  - store transitions (S_t, A_t, R_t, S_{t+1})
  - sample random mini-batches
  - removes correlation, restores i.i.d. property
- **Target network**
  - second q network (second set of parameters)
  - fixed parameters in target network
  - periodically update target network parameters
- **Reward clipping/normalization**
  - clip rewards to $r \in [-1, 1]$
  - batch normalization

Fitted Q-iteration and Q-learning

while alive **do**
  Save target network parameters $\boldsymbol{w}' \leftarrow \boldsymbol{w}$
  for $n = 0, 1, \ldots, N$ **do**
    Take action A_t and observe (S_t, A_t, R_t, S_{t+1}), add to buffer $\mathcal{B}$
    for $k = 0, 1, \ldots, K$ **do**
      Sample a transition (S_i, A_i, R_i, S_{i+1}) from $\mathcal{B}$
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \sum_i \left(R_i + \gamma \max_{A'_i} \hat{q}_{\boldsymbol{w}'}(S_{i+1}, A'_i) - \hat{q}_{\boldsymbol{w}}(S_i, A_i) \right) \boldsymbol{\nabla}_{\boldsymbol{w}} \hat{q}_{\boldsymbol{w}}(S_i, A_i)$
    end for
  end for
end while

DQN : N = 1, K = 1

A more general view



- Online Q-learning: evict immediately, process 1, process 2, and process 3 all run at the same speed
- DQN: process 1 and process 3 run at the same speed, process 2 is slow
- Fitted Q-iteration: process 3 in the inner loop of process 2, which is in the inner loop of process 1

Slide source: `http://rail.eecs.berkeley.edu/deeprlcourse/`

Q-learning with continuous actions

- What's the problem with continuous actions?

$$\pi(a \mid s) = \begin{cases} 1 & \text{if} \quad a = \arg\max_{a'} \hat{q}(s, a') \\ 0 & \text{otherwise} \end{cases}$$

$$\boldsymbol{y}_i = R_i + \gamma \max_{A'_t} \hat{q}(S_i, A'_i)$$

- How do we perform the max?
- Option 1: optimization
  - gradient based optimization (e.g., SGD) a bit slow in the inner loop
  - action space typically low-dimensional – what about stochastic optimization?
- Option 2: use function class that is easy to optimize (e.g. quadratic)

Q-learning with continuous actions

- Option 3: learn an approximate maximizer
  DDPG (Lillicrap et al., ICLR 2016)
- Train another network $\mu_{\boldsymbol{\theta}}$ such that $\mu_{\boldsymbol{\theta}}(s) \approx \arg\max_a \hat{q}_{\boldsymbol{w}}(s, a)$
- How?

$$\mu_{\boldsymbol{\theta}} \leftarrow \arg\max_{\boldsymbol{\theta}} \hat{q}_{\boldsymbol{w}}, \quad \text{with} \quad \frac{\mathrm{d}\hat{q}_{\boldsymbol{w}}}{\mathrm{d}\boldsymbol{\theta}} = \frac{\partial \hat{q}_{\boldsymbol{w}}}{\partial a} \frac{\mathrm{d}a}{\mathrm{d}\boldsymbol{\theta}}$$

- The new target is:

$$\boldsymbol{y}_i = R_i + \gamma \hat{q}(S_i, \mu_{\boldsymbol{\theta}}(S_i))$$

David Silver, et al. "Deterministic policy gradient algorithms." International conference on machine learning. PMLR, 2014.

Q-learning with continuous actions

- Option 3: learn an approximate maximizer
  DDPG (Lillicrap et al., ICLR 2016)

while alive **do**
  Take action A_t and observe (S_t, A_t, R_t, S_{t+1}), add to buffer $\mathcal{B}$
  Sample a mini-batch (S_i, A_i, R_i, S_{i+1}) from $\mathcal{B}$ uniformly
  Compute $y_i = R_i + \gamma \hat{q}_{\boldsymbol{w}'}(S_i, \mu_{\boldsymbol{\theta}'}(S_i))$ with target $\hat{q}_{\boldsymbol{w}'}$ and $\mu_{\boldsymbol{\theta}'}$
  $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \sum_i \left(\hat{q}_{\boldsymbol{w}}(S_i, A_i) - y_i \right) \boldsymbol{\nabla}_{\boldsymbol{w}} \hat{q}_{\boldsymbol{w}}(S_i, A_i)$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \beta \sum_j \frac{\partial \hat{q}_{\boldsymbol{w}}}{\partial A_j} (S_j, \mu(S_j)) \frac{\mathrm{d}\mu}{\mathrm{d}\boldsymbol{\theta}} (S_j)$
end while

Lillicrap et al., Continuous control with deep reinforcement learning, ICLR 2016.

Policy Optimization

Recap: Monte–Carlo policy gradient (REINFORCE)

- Update parameters $\boldsymbol{\theta}$ by stochastic gradient ascent
  - using the policy gradient theorem
  - using return G_t as an unbiased sample of $q_\pi(s_t, a_t)$

for all episodes **do**
  Sample $\{\tau\}$ following $\pi(.|., \boldsymbol{\theta})$
  $\boldsymbol{\nabla_\theta} J \approx \sum_i \left(\sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi(a_t \mid s_t, \boldsymbol{\theta}) \left(\sum_{k=t+1}^{T} R_k \right) \right)$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{\nabla_\theta} J(\boldsymbol{\theta})$
end for

Why does policy gradient work?

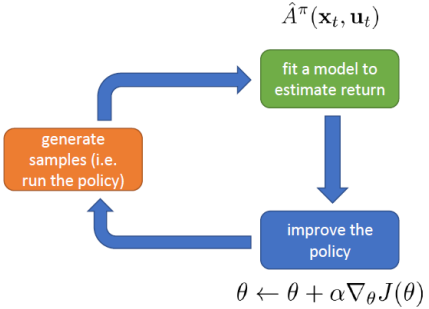
$$\nabla_{\theta} J \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(a_{i,t} \mid s_{i,t}) \hat{A}^{\pi}_{i,t}$$

1. Estimate $\hat{A}^{\pi}$ for current policy π
2. Use $\hat{A}^{\pi}$ to get an *improved* policy π'

look familiar?

Policy iteration algorithm

1. Evaluate$\hat{A}^{\pi}$
2. Set $\pi \leftarrow \pi'$



Policy gradient as policy iteration

What we want is to find $\boldsymbol{\theta}' \leftarrow \arg\max_{\boldsymbol{\theta}'} J(\boldsymbol{\theta}')$

$$
\begin{aligned}
J(\boldsymbol{\theta}') - J(\boldsymbol{\theta}) &= \mathbb{E}_{\tau \sim p_{\boldsymbol{\theta}'}(\tau)} \left[\sum_t \gamma^t A^{\pi_{\boldsymbol{\theta}}}(s_t, a_t) \right] \\
&= \sum_t \mathbb{E}_{s_t \sim p_{\boldsymbol{\theta}'}(s_t)} \left[\mathbb{E}_{a_t \sim \pi_{\theta'}} \left[\gamma^t A^{\pi_{\boldsymbol{\theta}}}(s_t, a_t) \right] \right] \\
&= \sum_t \mathbb{E}_{s_t \sim p_{\boldsymbol{\theta}'}(s_t)} \left[\mathbb{E}_{a_t \sim \pi_{\theta}} \left[\frac{\pi_{\boldsymbol{\theta}'}(a_t \mid s_t)}{\pi_{\boldsymbol{\theta}}(a_t \mid s_t)} \gamma^t A^{\pi_{\boldsymbol{\theta}}}(s_t, a_t) \right] \right]
\end{aligned}
$$

We can compute the expectation using importance sampling for the action, but is it ok to use $p_{\boldsymbol{\theta}}$ for the state?

How do we optimize the objective?

- We can guarantee monotonic improvement

$$\boldsymbol{\theta}' \leftarrow \arg\max_{\boldsymbol{\theta}'} \sum_t \mathbb{E}_{s\sim p_{\boldsymbol{\theta}}(s_t)} \left[\mathbb{E}_{a_t\sim\pi_\theta} \left[\frac{\pi_{\boldsymbol{\theta}'}(a_t \mid s_t)}{\pi_{\boldsymbol{\theta}}(a_t \mid s_t)} \gamma^t A^{\pi_{\boldsymbol{\theta}}}(s_t, a_t) \right] \right]$$
$$\text{such that } D_{KL}(\pi_{\boldsymbol{\theta}'} || \pi_{\boldsymbol{\theta}}) \leq \epsilon$$

- For small enough ϵ, this is guaranteed to improve $J(\boldsymbol{\theta}') - J(\boldsymbol{\theta})$
- This can be enforced using Lagrange multipliers (see TRPO).

Schulman, John, et al. "Trust region policy optimization." (TRPO) International conference on machine learning. PMLR, 2015.

Natural Gradient Descent

Coming back to our optimization problem

$$\begin{aligned} \arg\min_{\theta} \quad & f(\theta_t) + \nabla f(\theta_t)^T(\theta - \theta_t) \\ \text{subject to} \quad & D_{KL}[p(x|\theta)||p(x|\theta')] = \epsilon^2 \end{aligned} \tag{1}$$

Solving this problem by using Lagrange multipliers.

$$\begin{aligned} \theta_{t+1} &= \text{argmin}_\theta f(\theta_t) + \nabla f(\theta_t)^\top(\theta - \theta_t) + \lambda(D_{KL}[p(x|\theta)||p(x|\theta')] - \epsilon^2) \\ &\approx \text{argmin}_\theta f(\theta_t) + \nabla f(\theta_t)^\top(\theta - \theta_t) + \lambda(\frac{1}{2}\delta\theta^\top \boldsymbol{F}(\theta_t)\delta\theta - \epsilon^2) \end{aligned}$$

$\boldsymbol{F}(\theta_t)$ is the Fisher-information matrix

Natural Gradient Descent

Setting it to zero and solving for $\delta\theta$,

$$\delta\theta = -\frac{1}{\lambda}\boldsymbol{F}^{-1}\nabla f(\theta)$$

Therefore, the natural gradient is defined as

$$\hat{\nabla} f(\theta_t) = \boldsymbol{F}^{-1}(\theta_t)\nabla f(\theta_t) \tag{2}$$

Summary

- We have seen deep reinforcement learning algorithms
- Large capacity models with a lot of parameters are used for the policy and the Q functions
- We need to train them with SGD and i.i.d. data
- One way to get this is by using a replay buffer
- In order to get better guaranties for the policy gradient method we saw how to constraint the update to remain within a trust region