Reinforcement Learning

Alireza Kazemipour

CSICC 2021, K. N. Toosi University of Technology

Lab: Deep Learning (Tools and Applications)

What is RL?

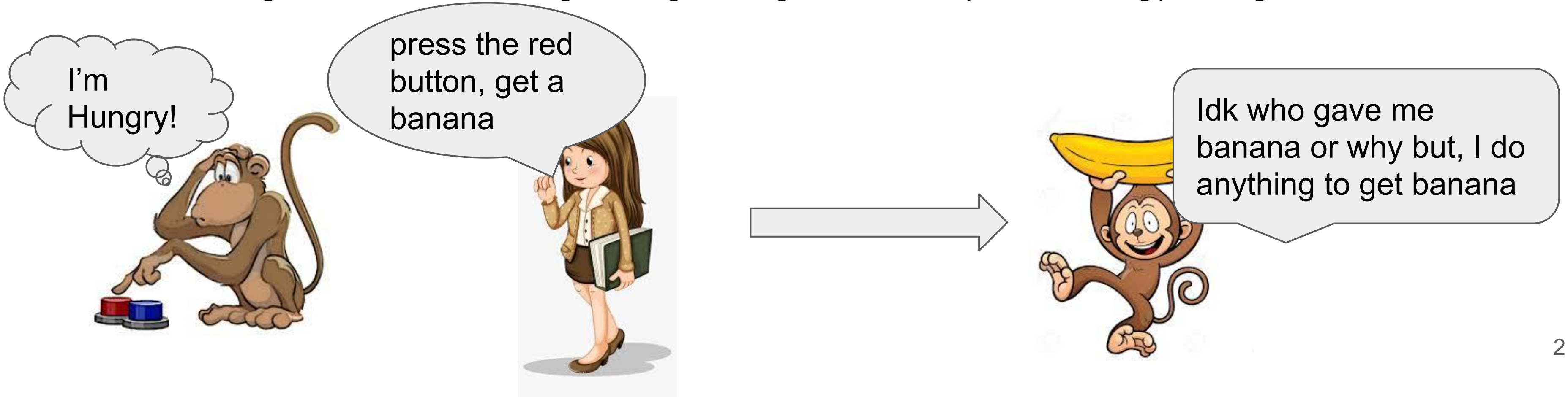
- Mathematical formalism for learning-based <u>decision making</u>.

  What for? To reach a <u>goal</u>!

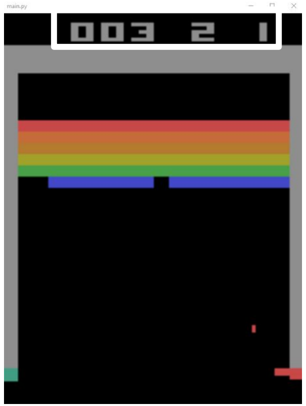
  ?

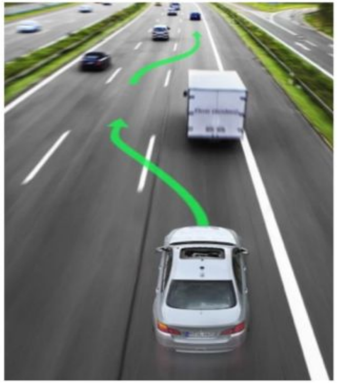
- What's a goal? How do we define a goal?

  Using some sort of signals guiding towards (describing) the goal.



Guidance signal = Reward





What's the reward?

More on this in Inverse RL.

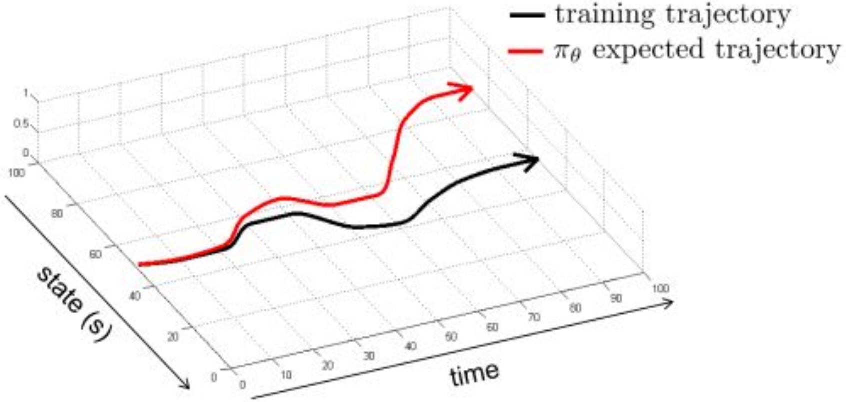
- It's just getting complicated, wasn't Supervised Learning better? At least, Goals and the procedure of making decisions are Clear! 😍
  NO!

Not that easy!!!

Supervised Learning of Behaviors(Imitation Learning)

- It worked but does it work in all situations (Snows, Rain, etc.) too? **No!**

Distributional Shift

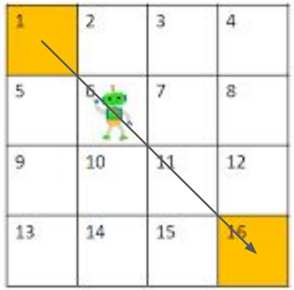


Let's see what RL does! ⇦ **<u>Okay let's collect more data</u>.** ⇦

Kidding, right? 😐

End to End Learning for Self-Driving Cars, Bojarski et al., 2016

Using rewards



Sparse reward env: If you reach location 16, you get +1 reward otherwise, 0.

Dense reward env: $reward = \sqrt{x^2 + y^2}$

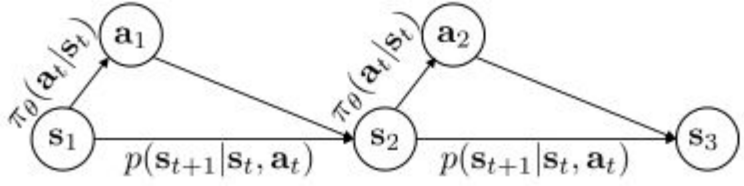
In any settings:

$$\theta = \arg\max_\theta \mathbb{E}_{\tau \sim P_\theta(\tau)} [\textstyle\sum_t r(s_t, a_t)]$$

Break down the objective

$$\theta = \boxed{\arg\max_\theta} \mathbb{E}_{\tau\sim \underline{P_\theta(\tau)}}\left[\sum_t r(s_t, a_t)\right]$$

Let's do it.



$$MDP = (S, A, \gamma, R, p)$$

$$P_\theta(\tau) = P_\theta(s_1, a_1, \ldots, s_T, a_T) = p(s_1)\prod_{t=1}^{T} \pi_\theta(a_t|s_t)\underline{p(s_{t+1}|s_t, a_t)}$$

$$\tau := 0, \ldots, T$$

Model-Based RL

Important Parameters

- $V^{\pi}(s_t) = \sum_{t=t'}^{T} \mathbb{E}_{P_\theta(s_{t'}, a_{t'})}[r(s_{t'}, a_{t'})|s_t] : Total\ reawrds\ from\ s_t$

- $Q^{\pi}(s_t, a_t) = \sum_{t=t'}^{T} \mathbb{E}_{P_\theta(s_{t'}, a_{t'})}[r(s_{t'}, a_{t'})|s_t, a_t] : Total\ reawrds\ from\ taking\ action\ a_t\ in\ s_t$

- $V^{\pi}(s_t) = \mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)}[Q^{\pi}(s_t, a_t)] :\ V^{\pi}(s_t)\ is\ the\ mean\ of\ total\ rewards\ of\ taking\ different\ actions\ in\ s_t$

- RL’s objective = $\mathbb{E}_{s_1 \sim P(s_1)}[V^{\pi}(s_1)]$ but $Q^{\pi}(s_t, a_t)$ is more expressive.

Q-Learning

- Intuition: the bigger $Q^\pi(s_t, a_t)$ is, the better a_t is in s_t:

$$Q^\pi(s_t, a_t) = \sum_{t=t'}^{T} \mathbb{E}_{P_\theta(s_{t'}, a_{t'})}[r(s_{t'}, a_{t'})|s_t, a_t]$$

$$= r(s_t, a_t) + \underbrace{\mathbb{E}_{s_{t'+1} \sim p(s_{t'+1}|s_t, a_t)}[V^\pi(s_{t'+1})]}_{\approx \max_{a_{t'+1}} Q(s_{t'+1}, a_{t'+})} = r(s_t, a_t) + \max_{a_{t'+1}} Q(s_{t'+1}, a_{t'+1})$$

$$\boxed{Q^\pi(s_t, a_t) = r(s_t, a_t) + \max_{a_{t'+1}} Q(s_{t'+1}, a_{t'+1})}$$

- $r(s_t, a_t)$ in sparse environments is not in touch!
- $t + 1$does not exist for the last timestep.

Recall:

$$\theta = \arg\max_\theta \mathbb{E}_{\tau \sim P_\theta(\tau)}[\textstyle\sum_t r(s_t, a_t)] \quad \pi(a_t|s_t) = \begin{cases} 1 & a_t = \arg\max Q(s_t, a_t) \\ 0 & Otherwise \end{cases}$$

Q-Learning

$$Q^{\pi}(s_t, a_t) = r(s_t, a_t) + \max_{a_{t'+1}} Q(s_{t'+1}, a_{t'+1})$$

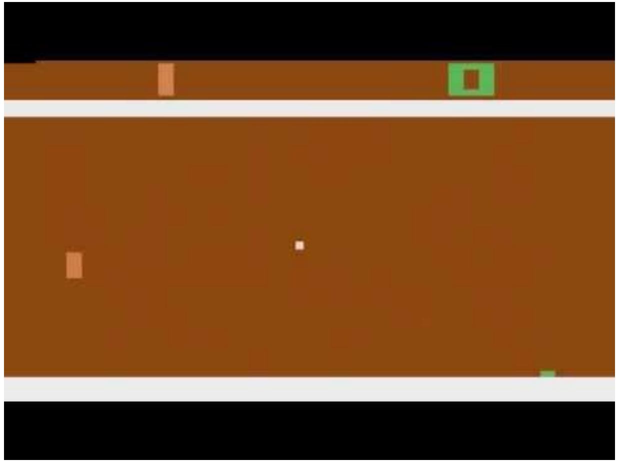
So there must be a memory that stores Q-Values of each (s_t, a_t) pairs.

But what if s_t is an image or a high-dimension quantity? Use Deep Learning. 😎

Deep Q-Networks (DQN)

DQN

1. Initialize replay buffer D. (Make data IID and enables usage of past experiences).
2. Initialize $Q(s_t, a_t)$ & $\bar{Q}(s_t, a_t)$ *such that* $\theta = \bar{\theta}$. (Reduce non-stationary target problem)
3. With prob ϵ select a_t otherwise, $a_t = \arg\max_{a_t} Q(s_t, a_t)$
4. Execute a_t and observe r_t and s_{t+1}.
5. Store (s_t, a_t, r_t, s_{t+1}) in D.
6. Sample random a mini-batch from D.
7. $y = \begin{cases} r & \text{Terminal s} \\ r + \gamma \max_{a_{t+1}} \bar{Q}(s_{t+1}, a_{t+1}) & \text{otherwise} \end{cases}$
8. Perform a Gradient-Descent step on $(y - Q(s_t, a_t))^2$
9. Every C steps



Human-level control through deep reinforcement learning, Mnih et al., 2015

Rainbow

1. Initialize replay buffer D.
2. Initialize $Q(s_t, a_t)$ & $\bar{Q}(s_t, a_t) such\ that\ \theta = \bar{\theta}$.
3. With prob ϵ select a_t otherwise, $a_t = \arg\max_{a_t} Q(s_t, a_t)$
4. Execute a_t and observe r_t and s_{t+1}.
5. Store (s_t, a_t, r_t, s_{t+1}) in D.
6. Sample random a mini-batch from D.
7. $y = r + \gamma \max_{a_{t+1}} \bar{Q}(s_{t+1}, a_{t+1})$
8. Perform a Gradient-Descent step on $(y - Q(s_t, a_t))^2$
9. Every C steps

1. Initialize replay buffer D.
2. $Q(s, a) - V(s) = A(s, a)$: $Better\ measure\ about\ a.$ (**Dueling**)
3. $\epsilon - Greedy$ is naive: Use **NoisyNets**
4. Same as DQN
5. Bootstrap r_t (**N-Step** returns) to reduce Variance.
6. Imbalance dataset: Use **PER** (Priortized Experience Replay).
7. $\begin{cases} (1)\ r + \gamma Q(s_{t+1}, \arg\max_{a_{t+1}} \bar{Q}(s_{t+1}, a_{t+1})) & \text{Reduce Over Estimation. (\textbf{Double})} \\ (2)\ Z(X, A) = R(X, A) + \gamma Z(X', A') & \text{Learn Distribution not EV for stability. (\textbf{C51})} \end{cases}$
8. Same as DQN
9. Same as DQN

Performance would be again on Atari Domain.

Rainbow: Combining Improvements in Deep Reinforcement Learning, Hessel et al., 2017

Policy Gradient

$$\theta = \boxed{\arg\max_\theta} \underbrace{\mathbb{E}_{\tau \sim P_\theta(\tau)} \left[\sum_t r(s_t, a_t)\right]}_{J(\theta)}$$

- It's an analytical expression, so let's find its maximum directly:

$$J(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)}[r(\tau)] = \int P_\theta(\tau) r(\tau) d\tau$$

$$\nabla_\theta J(\theta) = \int \underline{\nabla_\theta P_\theta(\tau)} r(\tau) d\tau = \int P_\theta(\tau) \nabla_\theta \log P_\theta(\tau) r(\tau) d\tau = \mathbb{E}_{\tau \sim P_\theta(\tau)}[\nabla_\theta \log P_\theta(\tau) r(\tau)]$$

$$\nabla_\theta P_\theta(\tau) = P_\theta(\tau) \frac{\nabla_\theta P_\theta(\tau)}{P_\theta(\tau)} = P_\theta(\tau) \nabla_\theta \log P_\theta(\tau)$$

$$P_\theta(\tau) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t | s_t) p(s_{t+1} | s_t, a_t)$$

$$\log P_\theta(\tau) = \log p(s_1) + \sum_{t=1}^{T} \pi_\theta(a_t | s_t) + \sum_{t=1}^{T} p(s_{t+1} | s_t, a_t)$$

$$\nabla_\theta \log P_\theta(\tau) = \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t | s_t)$$

$$\boxed{\nabla_\theta J_\theta(\theta) = \mathbb{E}_{\tau \sim P_\theta}\left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t | s_t)\right)\left(\sum_{t=1}^{T} r(s_t, a_t)\right)\right]}$$

Reinforce Algorithm

Actor-Critic

$$\nabla_\theta J_\theta(\theta) = \mathbb{E}_{\tau\sim P_\theta}[(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t))\underline{(\sum_{t=1}^{T} r(s_t,a_t))}]$$

High variance! 🙁

+10 +10 +15 <u>-100</u> +100 +10

Causality trick: Samples from t' can't affect rewards at t when $t < t'$.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim P_\theta(\tau)}[\overbrace{(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t))}^{\text{Maximum likelihood}}\overbrace{\underbrace{(\sum_{t=t'}^{T} r(s_{t'},a_{t'}))}_{Q(s_t,a_t)}}^{\text{How much to increase prob}}$$

What if the agent worked good but even better than the **<u>expected</u>**? $Q(s_t,a_t) - \underbrace{V(s_t)}_{V(s_t)} = A(s_t,a_t)$

$$\boxed{\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim P_\theta(\tau)}[\sum_{t=1}^{T} \nabla_\theta \log \underbrace{\boxed{\pi_\theta}}_{\text{Actor}}(a_t|s_t)A(s_t,a_t)]}$$

$Q(s_t,a_t) = r(s_t,a_t) + V(s_{t+1})$

$A(s_t,a_t) = r(s_t,a_t) + V(s_{t+1}) - \overbrace{\boxed{V(s_t)}}^{\text{Critic}}$

Asynchronous Advantage Actor-Critic (A3C)

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
    Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
    Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
    $t_{start} = t$
    Get state s_t
    repeat
        Perform a_t according to policy $\pi(a_t|s_t;\theta')$
        Receive reward r_t and new state s_{t+1}
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    until terminal s_t **or** $t - t_{start} == t_{max}$
    $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
    for $i \in \{t-1, \dots, t_{start}\}$ **do**
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
        Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i;\theta'_v))^2 / \partial \theta'_v$
    end for
    Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

Asynchronous Methods for Deep Reinforcement Learning, Mnih et al., 2016

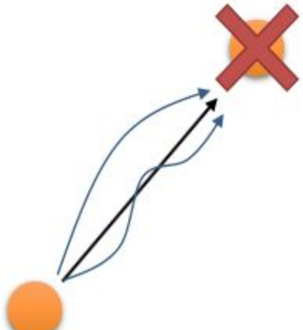
RL as Inference

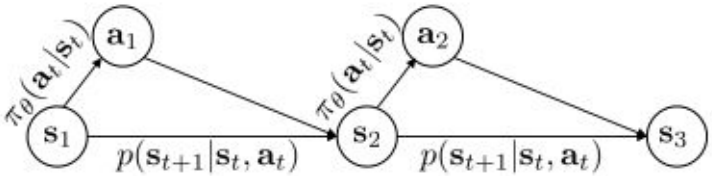
- Is $\theta = \arg\max_\theta \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[\sum_t r(s_t, a_t)\right]$ a good objective?
- Are humans' behavior in harmony with the objective?
- Does this objective include laziness, tiredness, distraction?

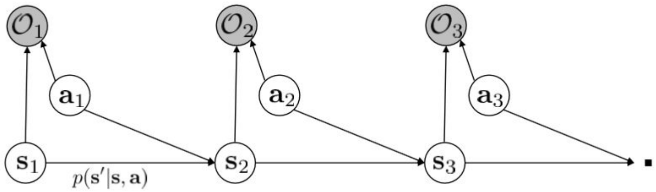
❑ Humans and animals have an important presumption:

<u>Some mistakes matter less than others</u>

So they may not be optimal! And the objective ignores the notion of optimality!







$$P(s_{1:T}, a_{1:T}|O_{1:T})$$

RL as Inference

There is a problem!

$$P(s_{1:T}, a_{1:T} | O_{1:T}) = p(s_1) \prod_{t=1}^{T} \overbrace{\pi(a_t | s_t, O_{1:T})}^{\text{This is what we want}} \underbrace{p(s_{t+1} | s_t, a_t, O_{1:T})}_{\text{Dynamics has changed}}$$

Policy: *Given that you got high reward, what was your action prob*?

$$p(s_{t+1} | s_t, a_t, O_{1:T}) \neq p(s_{t+1} | s_t, a_t)$$

<u>*Given that you got high reward, what was your transition probability*?</u>

We do not want this! Why? The game of lottery: If you’ve won lottery what was your transition prob while winning the lottery without any knowledge about victory is **so unlikely**!

RL as Inference

We want another distribution $q(s_{1:T}, a_{1:T})$ that is close to $P(s_{1:T}, a_{1:T} | O_{1:T})$ but has dynamics $p(s_{t+1} | s_t, a_t, O_{1:T})$. How? Use variational Inference: $[A(x) = \int A(x|z)B(z)dz]$

$x = O_{1:T}$ and $z = (s_{1:T}, a_{1:T})$, find $q(z)$ to approximate $P(z|x)$.

$$q(s_{1:T}, a_{1:T}) = p(s_1) \prod_{t=1}^{T} q(a_t | s_t) p(s_{t+1} | s_t, a_t)$$

Okay, what do we want? More **optimality**!

Variational lower bound

$$q(s_{1:T}, a_{1:T}) = p(s_1) \prod_{t=1}^{T} q(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

Okay, what do we want? More **optimality**! $Let: \; P(O_t|s_t, a_t) = exp(r(s_t, a_t))$

We know for any random X and Z and distributions p and q:

$$\log p(x) \geq \mathbb{E}_{z\sim p(z)}[\log p(x, z) - \log q(z)]$$

So:

$$\log P(O_{1:T}) \geq \mathbb{E}_{(s_{1:T}, a_{1:T})\sim q}[\log p(s_1) + \sum_{t=1}^{T} \log p(s_{t+1}|s_t, a_t) + \sum_{t=1}^{T} \log p(O_{t+1}|s_t, a_t) - \log p(s_1) - \sum_{t=1}^{T} \log p(s_{t+1}|s_t, a_t) - \sum_{t=1}^{T} \log q(a_t|s_t)]$$

$$\log P(O_{1:T}) \geq \mathbb{E}_{(s_{1:T}, a_{1:T})\sim q}[\sum_{t=1}^{T} r(s_t, a_t) - \log q(a_t|s_t)]$$

$$\boxed{\log P(O_{1:T}) \geq \mathbb{E}_{(s_{1:T}, a_{1:T})\sim q}[\sum_{t=1}^{T} r(s_t, a_t) + H(q(a_t|s_t))]}$$

New Objective: Maximize reward and maximize action entropy!

New Objective

$$q(s_{1:T}, a_{1:T}) = p(s_1) \prod_{t=1}^{T} \underbrace{q(a_t|s_t)}_{\text{Policy}} p(s_{t+1}|s_t, a_t)$$

$$\theta = \arg\max_\theta \sum_t \mathbb{E}_{(s_t,a_t)\sim q}[r(s_t, a_t) + H(q(a_t|s_t))]$$

$$q(s_t|a_t) = \arg\max_\theta \sum_t \mathbb{E}_{(s_t,a_t)\sim q}[r(s_t, a_t) + H(q(a_t|s_t))] = \arg\max_\theta \underbrace{\sum_t \mathbb{E}_{(s_t,a_t)\sim q}[r(s_t, a_t) - \log q(a_t|s_t)]}_{\text{Let's find the max}}$$

Base case t = T:

Recall: $\boxed{D_{KL}(q||p) = \mathbb{E}_{x\sim q(x)}[log\frac{q(x)}{p(x)}] = \mathbb{E}_{x\sim q(x)}[logq(x)] - \mathbb{E}_{x\sim q(x)}[\log p(x)]}$

$$\mathbb{E}_{(s_T,a_T)\sim q}[r(s_T, a_T) - \log q(a_T|s_T)] = \mathbb{E}_{s_T\sim p(S_T)}[\underbrace{-D_{KL}(q(a_T|s_T)||\frac{exp(r(s_T,a_T))}{\int exp(r(s_T,a_T))da_T}}_{\text{Let's find the min}})]$$

$$q(a_T|s_T) = \frac{exp(r(s_T,a_T))}{\int exp(r(s_T,a_T))da_T} = exp(r(s_T, a_T) - V(s_T))$$

Let define: $V(s_T) = \log \int exp(\underbrace{Q(s_T, a_T)}_{Q(s_T, a_T) = r(s_T, a_T)})da_T$

New Objective

$$q(a_T|s_T) = \frac{exp(r(s_T,a_T))}{\int exp(r(s_T,a_T))da_T} = exp(r(s_T,a_T) - V(s_T)$$

$$\mathbb{E}_{(s_T,a_T)\sim q}[r(s_T,a_T) - \log q(a_T|s_T)] = \mathbb{E}_{s_T\sim q(s_T)}[\mathbb{E}_{a_T\sim q(a_T|s_T)}[V(s_T)]]$$

For any t:

$$\begin{aligned} q(a_t|s_t) &= \arg\max \mathbb{E}_{s_t\sim q(s_t)}[\mathbb{E}_{a_t\sim q(a_t|s_t)}[\sum_t r(s_t,a_t) + H(q(a_t|s_t))] \\ &= \arg\max \mathbb{E}_{s_t\sim q(s_t)}[\mathbb{E}_{a_t\sim q(a_t|s_T)}[r(s_t,a_t) + H(q(a_t|s_t)) + \mathbb{E}_{p(s_{t+1}|s_t,a_t)}[V(s_{t+1})]]] \\ &= \arg\max \mathbb{E}_{s_t\sim q(s_t)}[\mathbb{E}_{a_t\sim q(a_t|s_T)}[Q(s_t,a_t) + \underbrace{H(q(a_t|s_t))}_{-\log q(a_t|s_t)}]] \end{aligned}$$

Same as t = T:

$$\boxed{q(s_t|a_t) = \frac{exp(Q(s_t,a_t))}{\int exp(Q(s_t,a_t))da_t} = exp(Q(s_t,a_t) - V(s_t))}$$

$$\boxed{V(s_t) = \log \int exp(Q(s_t,a_t))da_t}$$

$$\boxed{Q(s_t|a_t) = r(s_t,a_t) + \mathbb{E}[V(s_{t+1})]}$$

Same as regular RL!

Soft Q-Learning

1. $\pi(a|s) = exp(Q_\phi(s,a) - V(s)) = exp(A(s,a))$

2. $V(s_t) = \log \int exp(Q(s_t, a_t))da_t$

3. Take action a, observe transition and add it to buffer

4. Sample a mini-batch

5. $y = r + \underbrace{\gamma softmax}_{2.}{}_{a_{t+1}} Q_{\bar{\phi}}(s_{t+1}, a_{t+1})$

6. Do gradient descent on $(y - Q(s_t, a_t))^2$

7. Every C steps $\bar{\phi} = \phi$

Soft Policy Gradient

$$J(\theta) = \sum_t \mathbb{E}_{(s_t,a_t)\sim\pi(s_t,a_t)}\underbrace{[r(s_t,a_t) \overbrace{- \log \pi(a_t|s_t)}^{H(\pi(a_t|s_t))}]}_{\text{Just a new reward function}}$$

Same as before:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim P_\theta(\tau)}[(\textstyle\sum_{t=1}^T \nabla_\theta log\pi_\theta(a_t|s_t))(\sum_{t=1}^T r_{new}(s_t,a_t))]$$

But: $\log \pi(a_t|s_t) = Q(s_t,a_t) - V(s_t)$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim P_\theta(\tau)}[\textstyle\sum_{t=1}^T(\nabla_\theta Q(a_t|s_t) - \nabla_\theta V(s_t))(r(s_t,a_t) - \log \pi(a_t|s_t) + \underbrace{\textstyle\sum_{t'=t+1}^T r(s_{t'},a_{t'}) - \log \pi(a_{t'}|s_{t'}) - 1}_{V(s_{t+1})})]$$

$$\boxed{= \mathbb{E}_{\tau\sim P_\theta(\tau)}[\sum_{t=1}^T(\nabla_\theta Q(a_t|s_t) - \nabla_\theta V(s_t))(r(s_t,a_t) - Q(s_t,a_t) + V(s_t) + V(s_{t+1}) - 1)]}$$

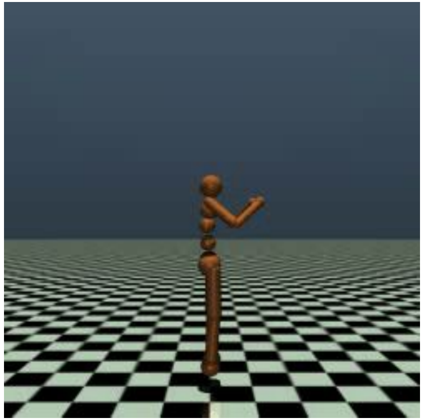
Soft Actor-Critic

$$\arg\min \mathbb{E}_{s_t\sim\pi(s_t)}[\mathbb{E}_{a_t\sim\pi(a|s_t)}[D_{KL}(\pi(s_t,a_t)||\frac{exp(Q(s_t,a_t))}{\int exp(Q(s_t,a_t))dt}]]$$

$$\arg\min \mathbb{E}_{(s_t,a_t)\sim\pi_\phi}[\log \pi_{\underbrace{\phi}_{\text{NN}}}(a_t|s_t) - Q_{\underbrace{\theta}_{\text{NN}}}(s_t,a_t)] : \text{Target } \phi$$

$$Q_\theta(s_t,a_t) = r(s_t,a_t) + \gamma V_{\underline{\psi}}(s_{t+1}) : \text{Target } \theta$$

$$V_\psi(s_t) = Q(s_t,a_t) - \log \pi_\phi(a_t|s_t) : \text{Target } \psi$$

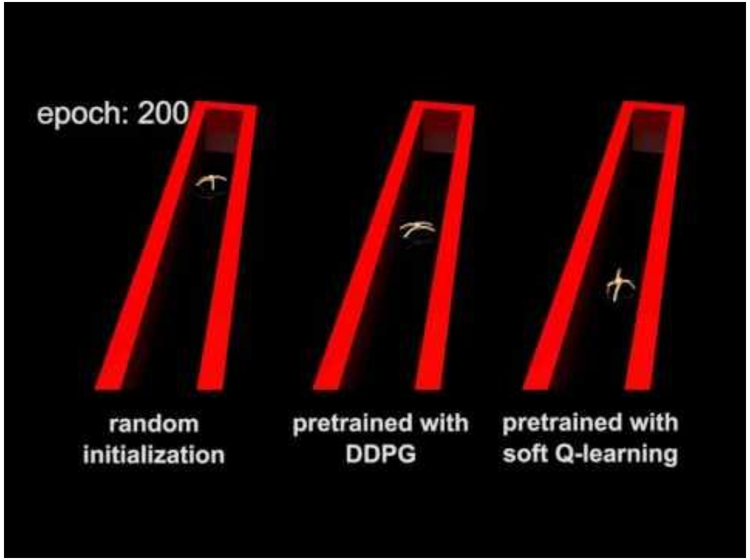


Initialize parameter vectors ψ, $\bar{\psi}$, θ, ϕ.
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t,\mathbf{a}_t,r(\mathbf{s}_t,\mathbf{a}_t),\mathbf{s}_{t+1})\}$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1,2\}$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$
  end for
end for

Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, Haarnoja et al., 2018

RL as Inference

So what!?

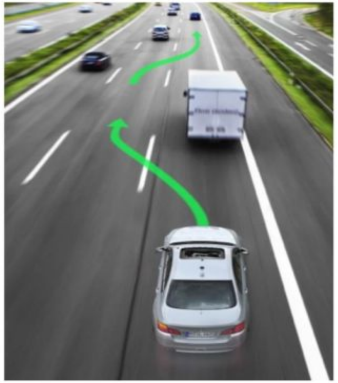


And: **Inverse RL** - **Transfer Learning in RL**

Reinforcement Learning with Deep Energy-Based Policies, Haarnoja et al., 2017

Inverse RL

Okay, does world behave in such an ideal mechanism?



IRL: Learn the reward function from observing an expert then use RL to learn a policy.

But what's the difference between IRL & Imitation Learning?

What's the reward? **But the goal is clear, to overtake the car**! Still Idk the reward. 😢

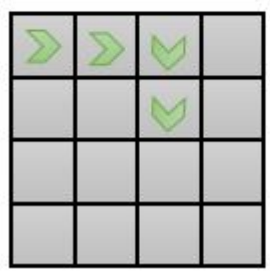
IRL:
1. Copy the intention of the expert
2. Might take different actions

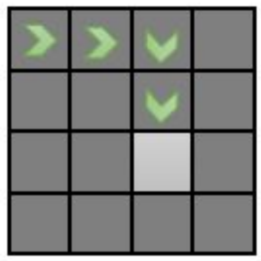
Imitation Learning:
1. Copy the actions of the expert
2. No reasoning about outcome of actions

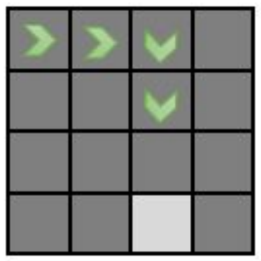
IRL is not easy

Let’s learn the reward and now we know that the expert might not be **optimal** too!

The hard part is that many reward functions may describe an unique task!







Does it really matter as far as it describes the task?

IRL's General Procedure

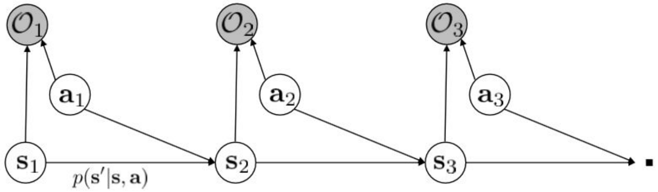
Given:

1. States s ∈ S, actions a ∈ A
2. (Sometimes) transitions P(s'| s, a)
3. Sample $\{\tau\}$ from $\pi^*(\tau)$

- ❑ Learn $r_\psi(s,a)$: ψ reward parameters
- ❑ Use RL to learn $\pi^*(s|a)$

Learn a reward function

We use expert's data so, take into account optimality!



We defined before: $P(O_t|s_t, a_t) = exp(r(s_t, a_t))$

$$P(\tau|O_{1:T}, \psi) \propto P(\tau)exp(\sum_t r_\psi(s_t, a_t))$$

We're given $\{\tau_i\}$ from $\pi^*(\tau)$ so, let's make reward function behave as expert's data is more likely to be optimal than any other possible rewards.

$$L = \max \mathbb{E}_{\tau_i \sim \pi^*(\tau)}[\log P(\tau_i|O_{1:T}, \psi) = \max \mathbb{E}_{\tau_i \sim \pi^*(\tau)}[r_\psi(\tau_i)] - \boxed{logZ}$$

$$Z = \int P(\tau)exp(r_\psi(\tau))d\tau$$

Derive a reward function

$$L = \max \mathbb{E}_{\tau_i \sim \pi^*(\tau)} [\log P(\tau_i | O_{1:T}, \psi) = \max \mathbb{E}_{\tau_i \sim \pi^*(\tau)} [r_\psi(\tau_i)] - logZ$$

$$Z = \int P(\tau) exp(r_\psi(\tau)) d\tau$$

$$\nabla_\psi L = \mathbb{E}_{\tau_i \sim \pi^*(\tau)} [\nabla_\psi r_\psi(\tau_i)] - \underbrace{\frac{1}{Z} \int P(\tau) exp(r_\psi(\tau))}_{P(\tau | O_{1:T}, \psi)} \nabla_\psi r_\psi(\tau) d\tau$$

It comes from our current policy.

$$\boxed{\nabla_\psi L = \mathbb{E}_{\tau_i \sim \pi^*(\tau)} [\nabla_\psi r_\psi(\tau)] - \mathbb{E}_{\tau \sim P(\tau | O_{1:T}, \psi)} [\nabla_\psi r_\psi(\tau)]}$$

IRL's General Procedure

Given:

1. States s ∈ S, actions a ∈ A
2. (Sometimes) transitions P(s'| s, a)
3. Sample $\{\tau\}$ from $\pi^*(\tau)$

- ❑ Learn $r_\psi(s,a)$: $\nabla_\psi L = \mathbb{E}_{\tau_i \sim \pi^*(\tau)}[\nabla_\psi r_\psi(\tau)] - \mathbb{E}_{\tau \sim P(\tau|O_{1:T},\psi)}[\nabla_\psi r_\psi(\tau)]$
- ❑ Use RL to learn $\pi^*(s|a)$: $J(\theta) = \mathbb{E}_{(s_{1:T},a_{1:T})\sim q}[\sum_{t=1}^{T} r(s_t,a_t) + H(q(a_t|s_t))]$

Seems so familiar! 🤔

IRL and GANs

Initial policy

π

Demos are made more likely, samples less likely

$$\nabla_\psi L = \mathbb{E}_{\tau_i \sim \pi^*(\tau)}[\nabla_\psi r_\psi(\tau)] - \mathbb{E}_{\tau \sim P(\tau|O_{1:T}, \psi)}[\nabla_\psi r_\psi(\tau)]$$

Human Demos

Samples from $\pi_\theta(\tau)$

Samples from $\pi^*(\tau)$

$$J(\theta) = \mathbb{E}_{(s_{1:T}, a_{1:T}) \sim q}[\sum_{t=1}^{T} r(s_t, a_t) + H(q(a_t|s_t))]$$

Policy changes to make it harder to distinguish samples from demos.

Transfer Learning in RL

Generally, Transfer Learning is assumed an open problem in RL! Why?

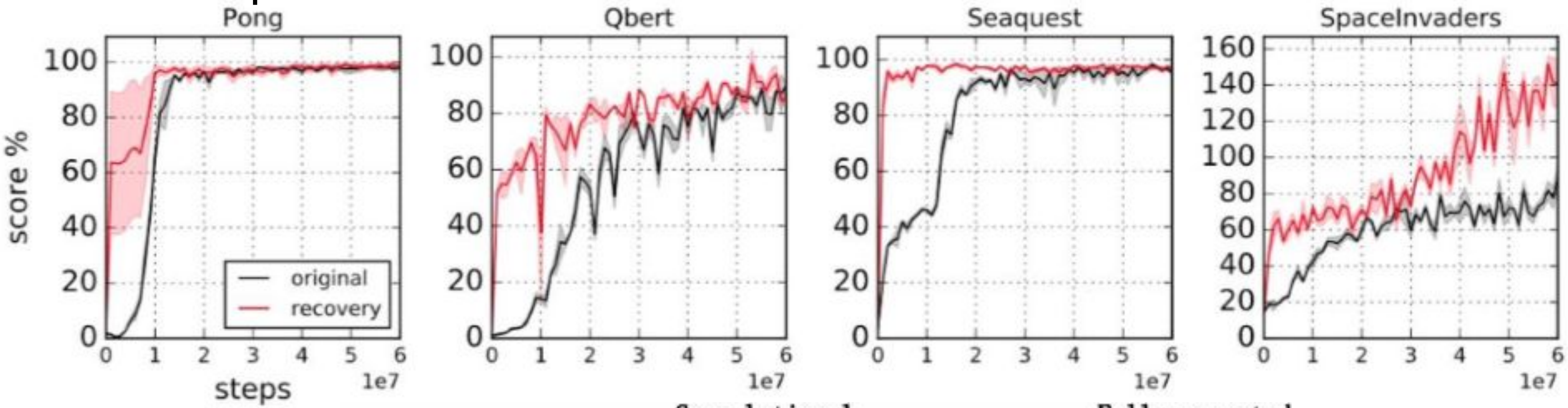
- Domain shift
- Difference in the MDP
- Fine-tuning issues

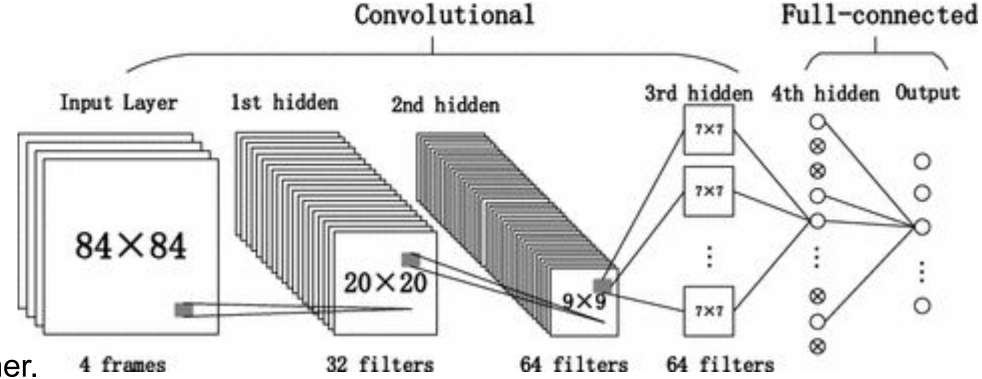
But, there are some solutions:

- Train on one task, transfer to a new one
  - Transfer visual representations
- Train on many tasks, transfer to a new one
- Transfer Models & Value Functions

Transfer Learning in RL

- Transfer representations





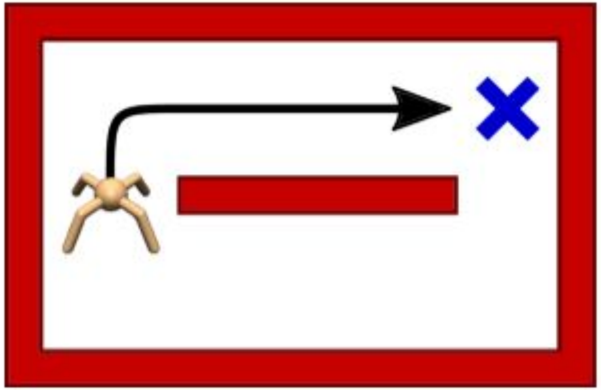
Loss is its own Reward, Shelhamer.

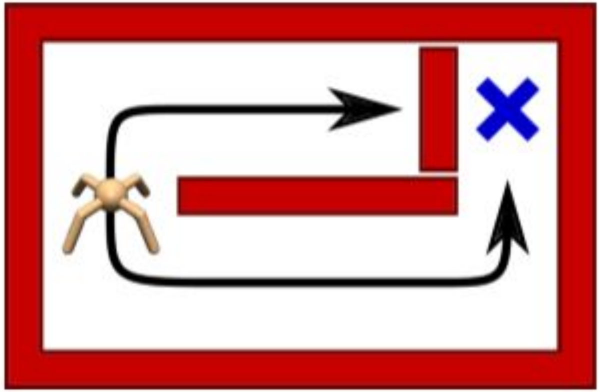
Transfer Learning in RL

- Train on many tasks, transfer to a new one

$$J(\theta) = \mathbb{E}_{(s_{1:T}, a_{1:T}) \sim q}\left[\sum_{t=1}^{T} r(s_t, a_t) + \underline{H(q(a_t|s_t))}\right]$$

Act as random as possible while collecting high reward!





What we did not cover

1. Model-Based RL
2. Meta-RL
3. Advanced Policy Gradient methods: TRPO, PPO
4. Differences between Off-Policy and On-policy Learning
5. Exploration in RL: Count-based exploration, Novelty-seeking in RL

Practical tips

1. Always use PyTorch. Trust me. :)
2. Monitor losses **ONLY** for being sure that your NNs have not diverged, not anything else.
3. Usually Colab is sufficient but, paperspace is also available and even better.
4. Be clever; Rainbow utilized a replay buffer with size 10^6 including images! Store *np.unit8* instead of *float* for images to have more memory then, transform them to float at the last step(which is your input to your CNN).
5. Benefit from *Object Oriented Programing*.
6. Transfer of data between RAM and GPU is expensive. Be careful to transfer completely your batch once to reduce latency!

References

1. CS 285 at UC Berkeley
2. *Human-level control through deep reinforcement learning*, Mnih et al., 2015
3. *Deep Reinforcement Learning with Double Q-learning*, Van Hasselt et al., 2015
4. *Dueling Network Architectures for Deep Reinforcement Learning*, Wang et al., 2015
5. *Prioritized Experience Replay*, Schaul et al., 2015
6. *A Distributional Perspective on Reinforcement Learning*, Bellemere et al., 2017
7. *Noisy Networks for Exploration*, Fortunato et al., 2017
8. *Rainbow: Combining Improvements in Deep Reinforcement Learning*, Hessel et al., 2017
9. *End to End Learning for Self-Driving Cars*, Bojarski et al., 2016
10. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*, Haarnoja et al., 2018
11. *Asynchronous Methods for Deep Reinforcement Learning*, Mnih et al., 2016
12. Reinforcement Learning with Deep Energy-Based Policies, Haarnoja et al., 2017