End-to-End Robotic Reinforcement Learning without Reward Engineering

Presented by Utku Bozdoğan

Authors

Avi Singh, Larry Yang, Kristian Hartikainen, Chelsea Finn, Sergey Levine

University of California, Berkeley




Outline

- Introduction
- Preliminaries
  - SAC
  - RAQ
  - VICE
- Off-Policy VICE-RAQ with SAC
- Experiment Results
  - Simulated Experiments
  - Real-World Experiments
- Discussion

Introduction

- Real world tasks often involve high dimensional observations, like images
- Obtaining rewards from pixels is difficult, and often requires task-specific engineering
- This paper’s goal is to solve real world robotics task from pixel-level observations in an end-to-end fashion:
  - Without task-specific systems to compute rewards
  - With minimal human intervention

Deep Neural Networks with Reinforcement Learning

➔ Uses raw RGB image input

➔ No reward function engineering or extra sensors

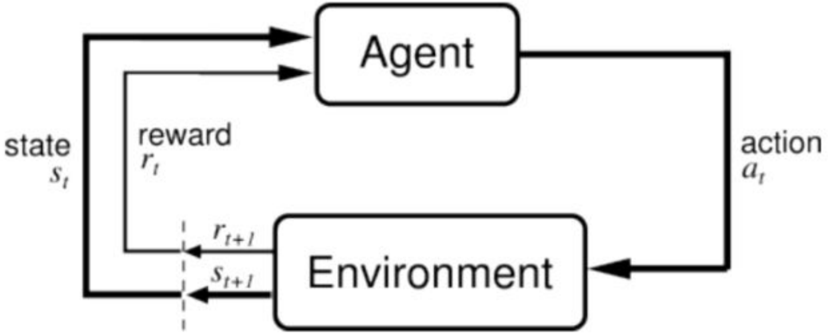
➔ Human only provides modest number of successful labels initially and then responds to the occasional queries of the robot. (of RGB images)

https://sites.google.com/view/reward-learning-rl/

RL Overview

An MDP is defined by:

- Set of states S
- Set of actions A
- Transition function $P(s' \mid s, a)$
- Reward function $R(s, a, s')$
- Start state s_0
- Discount factor γ
- Horizon H



Left: https://sites.google.com/view/deep-rl-bootcamp/lectures
Right: From Sutton and Barto, Reinforcement Learning: An Introduction

RL Overview

1. Initialization
   $v(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\quad \Delta \leftarrow 0$
   $\quad$ For each $s \in \mathcal{S}$:
   $\quad\quad temp \leftarrow v(s)$
   $\quad\quad v(s) \leftarrow \sum_{s'} p(s'|s, \pi(s)) \Big[r(s, \pi(s), s') + \gamma v(s') \Big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |temp - v(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad temp \leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s'} p(s'|s,a) \Big[r(s,a,s') + \gamma v(s') \Big]$
   $\quad$ If $temp \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return v and π; else go to 2

Figure 4.3: Policy iteration (using iterative policy evaluation) for v_*. This algorithm has a subtle bug, in that it may never terminate if the policy continually switches between two or more policies that are equally good. The bug can be fixed by adding additional flags, but it makes the pseudocode so ugly that it is not worth it. :-)

finding optimal value function

Initialize array v arbitrarily (e.g., $v(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
$\quad \Delta \leftarrow 0$
$\quad$ For each $s \in \mathcal{S}$:
$\quad\quad temp \leftarrow v(s)$
$\quad\quad v(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma v(s')]$
$\quad\quad \Delta \leftarrow \max(\Delta, |temp - v(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
$\quad \boxed{\pi(s) = \arg\max_a \sum_{s'} p(s'|s,a) \Big[r(s,a,s') + \gamma v(s') \Big]}$

Figure 4.5: Value iteration.

one policy update (extract policy from the optimal value function

Figures are from Sutton and Barto's book: *Reinforcement Learning: An Introduction*

RL Overview

We want to optimize long term future (predicted) rewards, which has a degree of uncertainty.

Let us start with the defined objective function $J(\theta)$. We can expand the expectation as:

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}|\pi_\theta]$$
$$= \sum_{t=i}^{T-1} P(s_t, a_t|\tau) r_{t+1}$$

where i is an arbitrary starting point in a trajectory, $P(s_t, a_t|\tau)$ is the probability of the occurrence of s_t, a_t given the trajectory τ.

Differentiate both sides with respect to policy parameter θ:

$$\text{Using} \quad \frac{d}{dx} log f(x) = \frac{f'(x)}{f(x)},$$

$$\nabla_\theta J(\theta) = \sum_{t=i}^{T-1} \nabla_\theta P(s_t, a_t|\tau) r_{t+1}$$
$$= \sum_{t=i}^{T-1} P(s_t, a_t|\tau) \frac{\nabla_\theta P(s_t, a_t|\tau)}{P(s_t, a_t|\tau)} r_{t+1}$$
$$= \sum_{t=i}^{T-1} P(s_t, a_t|\tau) \nabla_\theta log P(s_t, a_t|\tau) r_{t+1}$$
$$= \mathbb{E}[\sum_{t=i}^{T-1} \nabla_\theta log P(s_t, a_t|\tau) r_{t+1}]$$

However, during, learning, we take random samples of episodes instead of computing the expectation, so we can replace the expectation with

$$\nabla_\theta J(\theta) \sim \sum_{t=i}^{T-1} \nabla_\theta log P(s_t, a_t|\tau) r_{t+1}$$

From here, let us take a more careful look into $\nabla_\theta log P(s_t, a_t|\tau)$. First, by definition,

$$P(s_t, a_t|\tau) = P(s_0, a_0, s_1, a_2, ..., s_{t-1}, a_{t-1}, s_t, a_t|\pi_\theta)$$
$$= P(s_0)\pi_\theta(a_1|s_0)P(s_1|s_0, a_0)\pi_\theta(a_2|s_1)P(s_2|s_1, a_1)\pi_\theta(a_3|s_2)$$
$$...P(s_{t-1}|s_{t-2}, a_{t-2})\pi_\theta(a_{t-1}|s_{t-2})P(s_t|s_{t-1}, a_{t-1})\pi_\theta(a_t|s_{t-1})$$

RL Overview

Differentiating and taking log

If we *log* both sides,

$$\begin{aligned} logP(s_t,a_t|\tau) &= log(P(s_0)\pi_\theta(a_1|s_0)P(s_1|s_0,a_0)\pi_\theta(a_2|s_1)P(s_2|s_1,a_1)\pi_\theta(a_3|s_2)...\\ &\quad P(s_{t-1}|s_{t-2},a_{t-2})\pi_\theta(a_{t-1}|s_{t-2})P(s_t)log\pi_\theta(a_t|s_{t-1})\\ &= logP(s_0)+log\pi_\theta(a_1|s_0)+logP(s_1|s_0,a_0)+log\pi_\theta(a_2|s_1)\\ &\quad +logP(s_2|s_1,a_1)+log\pi_\theta(a_3|s_2)+...+logP(s_{t-1}|s_{t-2},a_{t-2})\\ &\quad +log\pi_\theta(a_{t-1}|s_{t-2})+logP(s_t|s_{t-1},a_{t-1})+log\pi_\theta(a_t|s_{t-1}) \end{aligned}$$

Then, differentiating $logP(s_t,a_t|\tau)$ with respect to θ yields:

$$\begin{aligned} \nabla_\theta logP(s_t,a_t|\tau) = &\nabla_\theta logP(s_0)+\nabla_\theta log\pi_\theta(a_1|s_0)+\nabla_\theta logP(s_1|s_0,a_0)\\ &+\nabla_\theta log\pi_\theta(a_2|s_1)+\nabla_\theta logP(s_2|s_1,a_1)+\nabla_\theta log\pi_\theta(a_3|s_2)+\\ &...+\nabla_\theta logP(s_{t-1}|s_{t-2},a_{t-2})+\nabla_\theta log\pi_\theta(a_{t-1}|s_{t-2})+\\ &\nabla_\theta logP(s_t|s_{t-1},a_{t-1})+\nabla_\theta log\pi_\theta(a_t|s_{t-1}) \end{aligned}$$

However, note that the $P(s_t|s_{t-1},a_{t-1})$ is not dependent on the policy parameter θ, and is solely dependant on the environment on which the reinforcement learning is acting on; it is assumed that the state transition is unknown to the agent in model free reinforcement learning. Thus, the gradient of it with respect to θ will be 0. How convenient! So:

$$\begin{aligned} \nabla_\theta logP(s_t,a_t|\tau) &= 0+\nabla_\theta log\pi_\theta(a_1|s_0)+0+\nabla_\theta log\pi_\theta(a_2|s_1)+0+\nabla_\theta log\pi_\theta(a_3|s_2)+\\ &\quad ...+0+\nabla_\theta log\pi_\theta(a_{t-1}|s_{t-2})+0\\ &= \nabla_\theta log\pi_\theta(a_1|s_0)+\nabla_\theta log\pi_\theta(a_2|s_1)+\nabla_\theta log\pi_\theta(a_3|s_2)+\\ &\quad ...+\nabla_\theta log\pi_\theta(a_{t-1}|s_{t-2})+log\pi_\theta(a_t|s_{t-1})\\ &= \sum_{t'=0}^{t}\nabla_\theta log\pi_\theta(a_{t'}|s_{t'}) \end{aligned}$$

Plugging this into our $\nabla_\theta J(\theta)$ yields:

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_{t=0}^{T-1} r_{t+1}\nabla_\theta P(s_t,a_t|\tau)\\ &= \sum_{t=0}^{T-1} r_{t+1}(\sum_{t'=0}^{t}\nabla_\theta log\pi_\theta(a_{t'}|s_{t'})) \end{aligned}$$

RL Overview

Including the discount factor:

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t]$$

Lets expand that!

$$\begin{aligned}
\nabla_\theta J(\theta) &= \sum_{t=0}^{T-1} r_{t+1}(\sum_{t'=0}^{t} \nabla_\theta log\pi_\theta(a_{t'}|s_{t'})) \\
&= r_1(\sum_{t'=0}^{0} \nabla_\theta log\pi_\theta(a_{t'}|s_{t'})) + r_2(\sum_{t'=0}^{1} \nabla_\theta log\pi_\theta(a_{t'}|s_{t'})) \\
&\quad + r_3(\sum_{t'=0}^{2} \nabla_\theta log\pi_\theta(a_{t'}|s_{t'})) + ... + r_{T-1}(\sum_{t'=0}^{T-1} \nabla_\theta log\pi_\theta(a_{t'}|s_{t'})) \\
&= r_1 \nabla_\theta log\pi_\theta(a_0|s_0) + r_2(\nabla_\theta log\pi_\theta(a_0|s_0) + \nabla_\theta log\pi_\theta(a_1|s_1)) \\
&\quad + r_3(\nabla_\theta log\pi_\theta(a_0|s_0) + \nabla_\theta log\pi_\theta(a_1|s_1) + \nabla_\theta log\pi_\theta(a_2|s_2)) \\
&\quad + ... + r_T(\nabla_\theta log\pi_\theta(a_0|s_0) + \nabla_\theta log\pi_\theta(a_1|s_1) + ... + \nabla_\theta log\pi_\theta(a_{T-1}|s_{T-1})) \\
&= \nabla_\theta log\pi_\theta(a_0|s_0)(r_1 + r_2 + ... + r_T) + \nabla_\theta log\pi_\theta(a_1|s_1)(r_2 + r_3 + ... + r_T) \\
&\quad + \nabla_\theta log\pi_\theta(a_2|s_2)(r_3 + r_4 + ... + r_T) + ... + \nabla_\theta log\pi_\theta(a_{T-1}|s_{T-1}) r_T \\
&= \sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t)(\sum_{t'=t+1}^{T} r_{t'})
\end{aligned}$$

Simplifying the term $\sum_{t'=t+1}^{T} r_{t'}$ to G_t, we can derive the policy gradient

$$\sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t) G_t$$

Incorporating the discount factor $\gamma \in [0, 1]$ into our objective (in order to weight immediate rewards more than future rewards):

$$J(\theta) = \mathbb{E}[\gamma^0 r_1 + \gamma^1 r_2 + \gamma^2 r_3 + ... + \gamma^{T-1} r_T | \pi_\theta]$$

We can perform a similar derivation to obtain

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t)(\sum_{t'=t+1}^{T} \gamma^{t'-t-1} r_{t'})$$

and simplifying $\sum_{t'=t+1}^{T} \gamma^{t'-t-1} r_{t'}$ to G_t,

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t) G_t$$

RL Overview

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

Policy Gradient

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})\right)$$

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

https://medium.com/@jonathan_hui/rl-policy-gradients-explained-9b13b688b146

Actor-Critic Method

Merges policy and value iteration to achieve better results

Actor performs policy iteration, takes state and outputs the best action

Critic performs value iteration, takes state and action from actor and outputs value.

Algorithm 1 Q Actor Critic

Initialize parameters s, θ, w and learning rates α_θ, α_w; sample $a \sim \pi_\theta(a|s)$.
for $t = 1 \ldots T$: **do**
  Sample reward $r_t \sim R(s,a)$ and next state $s' \sim P(s'|s,a)$
  Then sample the next action $a' \sim \pi_\theta(a'|s')$
  Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s,a)\nabla_\theta \log \pi_\theta(a|s)$; Compute the correction (TD error) for action-value at time t:
    $\delta_t = r_t + \gamma Q_w(s',a') - Q_w(s,a)$
  and use it to update the parameters of Q function:
    $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s,a)$
  Move to a $\leftarrow a'$ and s $\leftarrow s'$
end for

$$\nabla_\theta J(\theta) \sim \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)(r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t))$$

$$= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)$$

Advantage

Maximum Entropy RL

The policy in MaxEntRL is incentivized to explore more widely.

Also, the policy can capture multiple modes of near-optimal behavior. In problem settings where multiple actions seem equally attractive, the policy will commit equal probability mass to those actions.

Only a small change in the equation, to include the entropy regularization term:

$$J(\pi) = \sum_{t=0}^{T} E_{\tau \sim \pi} \left[r(\mathbf{s}_t, \mathbf{a}_t) - \log \pi(\mathbf{a}_t | \mathbf{s}_t) \right] . \qquad (1)$$

SAC - Soft Actor-Critic

Off-policy SAC is a Maximum Entropy RL algorithm and is preferred in this paper because:

- It tends to produce stable and **robust** policies for real-world reinforcement learning, and
- Maximum entropy RL makes it straightforward to integrate their method with VICE.

Algorithm 1 Soft actor-critic (SAC)

Initialize policy π, critic Q
Initialize replay buffer $\mathcal{R}$
for each iteration **do**
    for each environment step **do**
        $\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
        $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
        $\mathcal{R} \leftarrow \mathcal{R} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
    for each gradient step **do**
        Sample from $\mathcal{R}$
        Update π and Q according to Haarnoja et al. [15]

Soft Critic Update

- Updating the value network:ψ

$$\hat{\nabla}_\psi J_V(\psi) = \nabla_\psi V_\psi(\mathbf{s}_t)(V_\psi(\mathbf{s}_t) - Q_\theta(\mathbf{s}_t, \mathbf{a}_t) + \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t))$$

- Updating the Q-function:

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(\mathbf{s}_t, \mathbf{a}_t)(Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - r(\mathbf{s}_t, \mathbf{a}_t) + \gamma V_{\bar{\psi}}(\mathbf{s}_{t+1}))$$

where ψ is an exponentially moving average ψ.

Soft Actor Update

- Policy parameters can be learned by minimizing the expected KL divergence.

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}}\left[KL\left(\pi_\phi(.|\mathbf{s}_t) \,\middle\|\, \frac{\exp(\frac{1}{\alpha}Q_\theta(\mathbf{s}_t, .))}{Z_\theta(\mathbf{s}_t)}\right)\right].$$

- Rewriting:

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}}[\mathbb{E}_{\mathbf{a}_t \sim \pi_\phi}[\alpha \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t) - Q_\theta(\mathbf{s}_t, \mathbf{a}_t)]] \qquad (2)$$

SAC - Soft Actor-Critic

Algorithm 1 Soft Actor-Critic

Input: θ_1, θ_2, ϕ ▷ Initial parameters
  $\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$ ▷ Initialize target network weights
  $\mathcal{D} \leftarrow \emptyset$ ▷ Initialize an empty replay pool
  for each iteration **do**
    for each environment step **do**
      $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$ ▷ Sample action from the policy
      $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ ▷ Sample transition from the environment
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$ ▷ Store the transition in the replay pool
    end for
    for each gradient step **do**
      $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ ▷ Update the Q-function parameters
      $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ ▷ Update policy weights
      $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$ ▷ Adjust temperature
      $\bar{\theta}_i \leftarrow \tau\theta_i + (1 - \tau)\bar{\theta}_i$ for $i \in \{1, 2\}$ ▷ Update target network weights
    end for
  end for
Output: θ_1, θ_2, ϕ ▷ Optimized parameters

Soft Actor Update

- We set $\mathbf{a}_t = \mathbf{f}_\phi(\epsilon_t, \mathbf{s}_t) = \boldsymbol{\mu}_\phi(\mathbf{s}_t) + \epsilon_t \boldsymbol{\sigma}_\phi(\mathbf{s}_t)$

- We now have:

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}}[\mathbb{E}_{\epsilon \sim \mathcal{N}(0,1)}[\alpha \log \pi_\phi(\mathbf{f}_\phi(\epsilon_t, \mathbf{s}_t)|\mathbf{s}_t) - Q_\theta(\mathbf{s}_t, \mathbf{f}_\phi(\epsilon_t, \mathbf{s}_t))]]$$

- Whose gradient w.r.t. ϕ can be obtained through some maths

$$\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t) + (\nabla_{\mathbf{a}_t} \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t) - \nabla_{\mathbf{a}_t} Q_\theta(\mathbf{s}_t, \mathbf{a}_t)) \nabla_\phi f_\phi(\epsilon_t; \mathbf{s}_t)$$

Deep RL Bootcamp Lecture 7 SVG, DDPG, and Stochastic Computation Graphs (John Schulman)

$$x \sim N(\mu, \Sigma)$$

$$\theta := \{\mu, \Sigma\}$$

$$z \sim N(0, 1)$$

$$x = \mu + \Lambda z$$

16:41 / 1:11:13

Reparameterization Trick

Enabled us to find the gradient. We can safely reparametrize to use a different estimator which is differentiable, since the actual distribution is unchanged.

In SAC, policy is represented as a Gaussian with the mean given by a neural network function of the state, so Pathwise Derivative estimator is a better choice.

Reparameterization trick yields lower variance than Standard likelihood ratio method, so it is preferable.

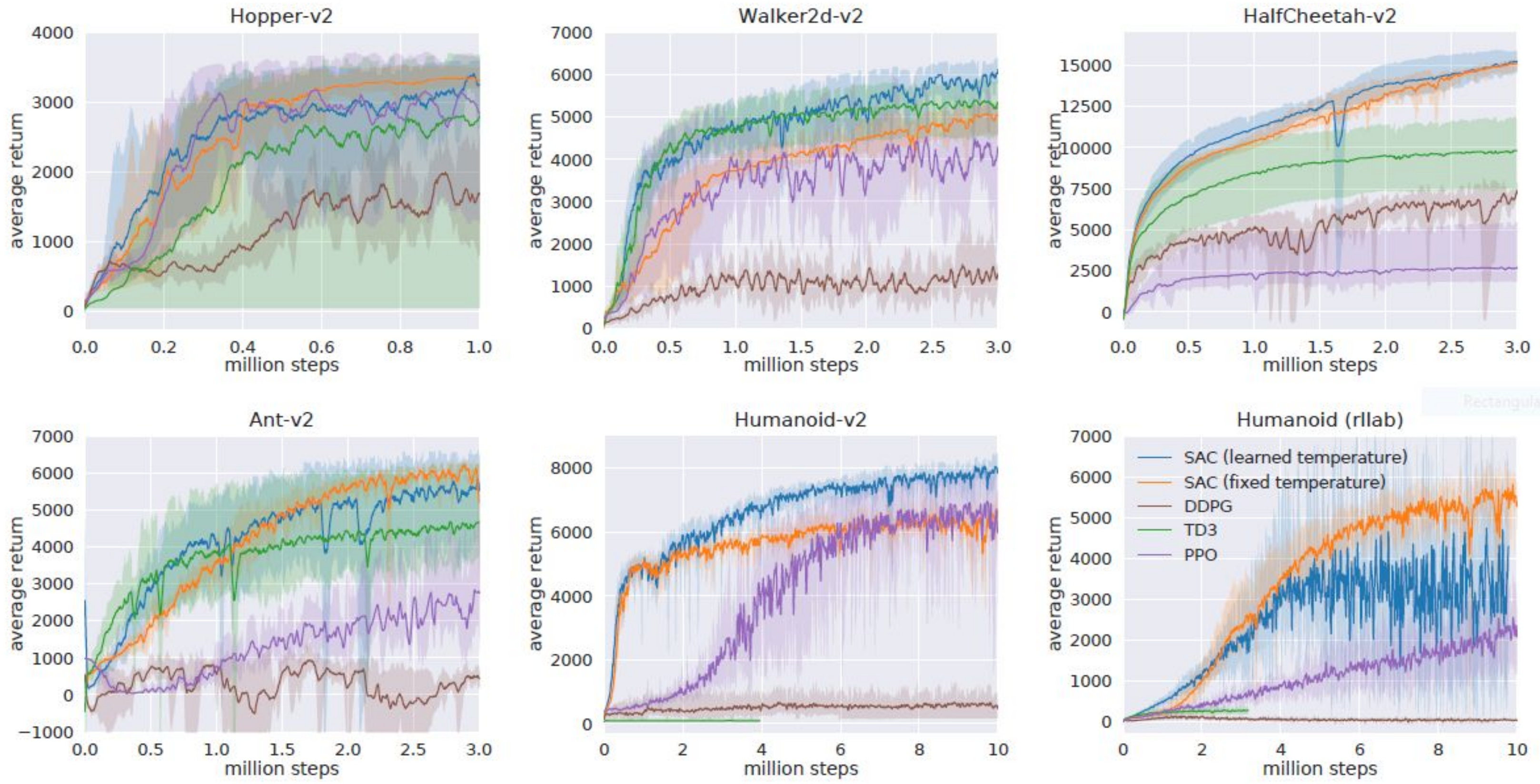


Figure 1: Training curves on continuous control benchmarks. Soft actor-critic (blue and yellow) performs consistently across all tasks and outperforming both on-policy and off-policy methods in the most challenging tasks.

Classifier Based Rewards

- Engineering reward functions for RL algorithms is difficult
- A reasonable alternative is to use a goal classifier, where the user provides a dataset of example states (e.g., images) before training the policy, and a binary classifier is trained to predict whether a given state is a success or a failure.

Algorithm 2 Classifier-based rewards for RL

Require: : $\mathcal{D}_i := \{(\mathbf{s}_n, y_n)\}$

1: Update the parameters of g to minimize $\sum_n \mathcal{L}(g(\mathbf{s}_n), y_n)$

2: Run RL or planning, using reward derived from $\log p_g(y|\mathbf{s})$

Log-probabilities often increase smoothly
as the agent approaches the goal.

RAQ - Reinforcement Learning with Active Queries

Query the user for labels but when?

- Can only query for small number of states to remain efficient. (25 to 75 active queries for a single run)

Which states?

- Need to prevent RL agent exploitation problem, so querying the states with high success probability is a good idea. If the state is actually a failure, the classifier will be updated and will no longer assign high success probability to queried states.

SAC with RAQ

Note that the dataset D includes the negative query results in this case.

But this is not enough, we are using but a fraction of available data.

Algorithm 3 Reinforcement learning with active queries (RAQ)

Require: initial $\mathcal{D} := \{(\mathbf{s}_n, y_n)\}$

Initialize policy π, critic Q

Initialize replay buffer $\mathcal{R}$

for each iteration **do**

  for each environment step **do**

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

    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$

    $\mathcal{R} \leftarrow \mathcal{R} \cup \{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})\}$

  for each gradient step **do**

    Sample from $\mathcal{R}$

    Compute rewards: $r(\mathbf{s}_t) \leftarrow \log p_g(y_t|\mathbf{s}_t)$

    Update π and Q according to Haarnoja et al. [15]

  if active query **then**

    $k \rightarrow \arg\max \log p_g(y_t|\mathbf{s}_t)$ for all t since the last query

    if $\mathbf{s}_k$ is a successful outcome **then**

      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_k, 1)\}$

    else

      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_k, 0)\}$

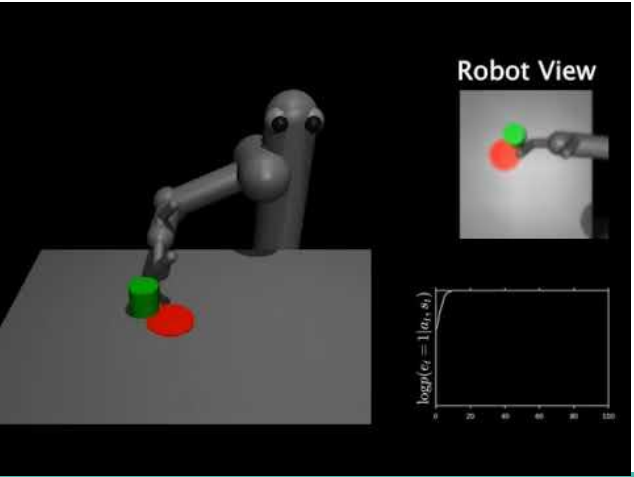
    Update g to minimize $\sum_n \mathcal{L}(g(\mathbf{s}_n), y_n)$

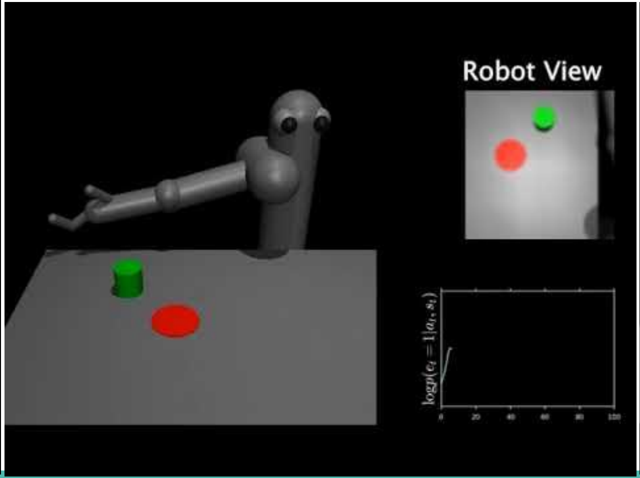
Classifier and VICE on Rewards

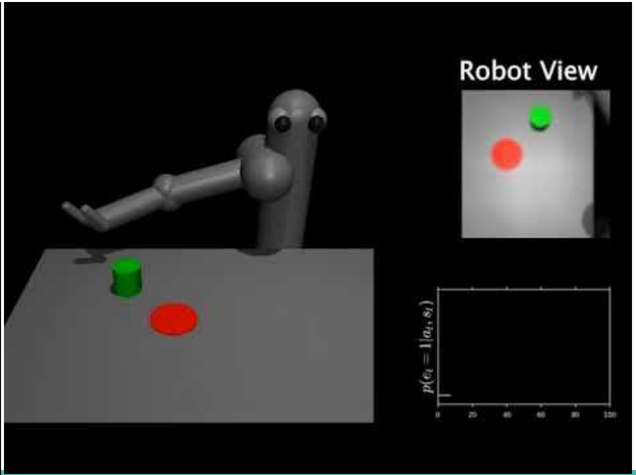
Events classifier all: https://youtu.be/Idyoyb6uSUM

Events VICE all: https://youtu.be/jiDK4_wmqPw

Events binary indicator: https://youtu.be/sbYVf-J35Wg







VICE - Variational Inverse Control with Events

VICE is a classifier-based reward specification framework that uses **on-policy** RL with policy gradients, and generally requires a large number of positive outcome examples. However, VICE can effectively overcome the classifier exploitation problem, and does so by using all of the data collected during RL without making any active queries.

VICE requires the success examples to include both the state s and action a, which is unnatural for the user to provide.

VICE - Variational Inverse Control with Events

We can formulate the problem of learning a policy that succeeds at the task as inference in this graphical model, where the policy corresponds to

$$p(\mathbf{a}_t | \mathbf{s}_t, y_{1:T} = 1)$$

and also corresponds exactly to the maximum entropy objective in entropy regularized objective function in (1).

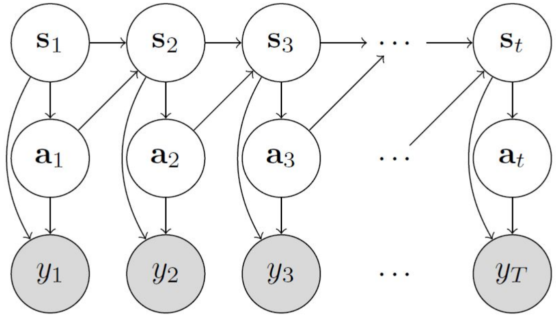


Fig. 2. A graphical model framework for VICE. The node y_t is a binary random variable that denotes whether an event happens at a given time step or not.

Now, an agent's goal is to maximize the probability that one or more events will happen at some point in the future, rather than maximizing cumulative rewards.

VICE - Variational Inverse Control with Events

Learning the event probabilities in VICE corresponds to an optimization that is similar to maximum entropy inverse reinforcement learning.

A scalable way to implement maximum entropy inverse RL is to utilize adversarial inverse reinforcement learning (AIRL)

Can sample negative examples for the discriminator directly from replay buffer without importance weighting, making use of all available data. Active queries from RAQ will also provide further success examples.

Off-Policy VICE-RAQ with SAC

Algorithm 4 Off-Policy VICE-RAQ with soft actor-critic

Require: : $\mathcal{D}_i := \{(\mathbf{s}_n, 1)\}$

Initialize f_ψ (described in Equation 2)
Initialize policy π, critic Q
Initialize replay buffer $\mathcal{R}$
for each iteration **do**
    for each environment step **do**
        $\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
        $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
        $\mathcal{R} \leftarrow \mathcal{R} \cup \{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})\}$
    for each gradient step for f_ψ **do**
        Sample positives from $\mathcal{D}$
        Sample action labels $\mathbf{a}_i^E \sim \pi(\mathbf{a}|\mathbf{s}_i^E)$
        Sample negatives from $\mathcal{R}$
        Update f_ψ using Equation 2 as discriminator
    for each gradient step for the policy π **do**
        Sample from $\mathcal{R}$
        Compute rewards: $r(\mathbf{s}_t) \leftarrow f_\psi(\mathbf{s}_t)$
        Update π and Q according to Haarnoja et al. [15]
    if active query **then**
        $k \rightarrow \arg\max f_\psi(\mathbf{s}_t)$ for all t since the last query
        if $\mathbf{s}_k$ is a successful outcome **then**
            $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_k, 1)\}$

Architecture

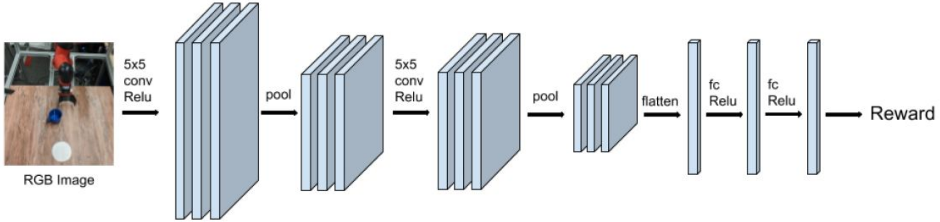


Fig. 3. Our convolutional neural network architecture. The same architecture is used for the policy, critic, and the learned reward function.

It consists of two convolutional layers, each of which is followed by a max-pooling layer, with 8 filters in each of the convolutional layers for simulated tasks, and 32 filters per layer for real world tasks. The flattened output of the convolutional layers is followed by two fully-connected hidden layers with 256 units each. The ReLU non-linearity is applied after each of the convolutional and fully-connected layers. The reward function is also represented using a convolutional neural network with the same architecture.

Mixup Regularization

$$\begin{aligned}\tilde{\mathbf{s}} &= \lambda \mathbf{s}_i + (1-\lambda)\mathbf{s}_j \\ \tilde{y} &= \lambda y_i + (1-\lambda) y_j\,, \end{aligned} \tag{3}$$

Enables smoother transitions between different classes by encouraging linear behavior, so the learned reward function is smoother and yields better results.

Simulation Experiments

Videos: https://sites.google.com/view/reward-learning-rl/

Results:

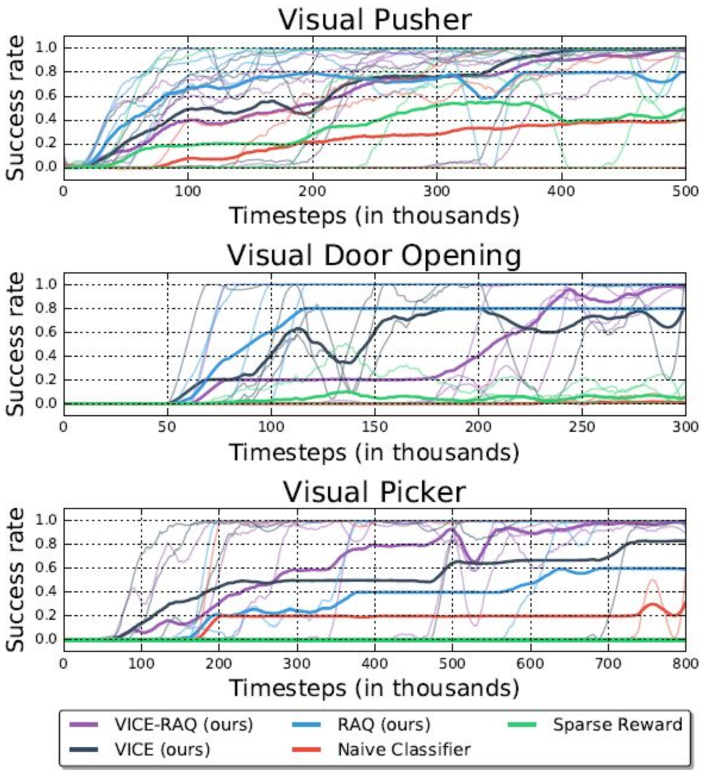


Fig. 5. Results on simulated tasks. Each method is run with five different random seeds for each task. The lines in bold indicate the mean across five runs, while the faint lines depict the individual random seeds for each method. We observe that VICE-RAQ achieve the best performance on all tasks, with RAQ being comparable to VICE-RAQ on the Visual Pusher task. We also notice that both RAQ and VICE have significant variance among runs, while VICE-RAQ achieves relatively low variance towards the end of the learning process.

Real-World Experiments

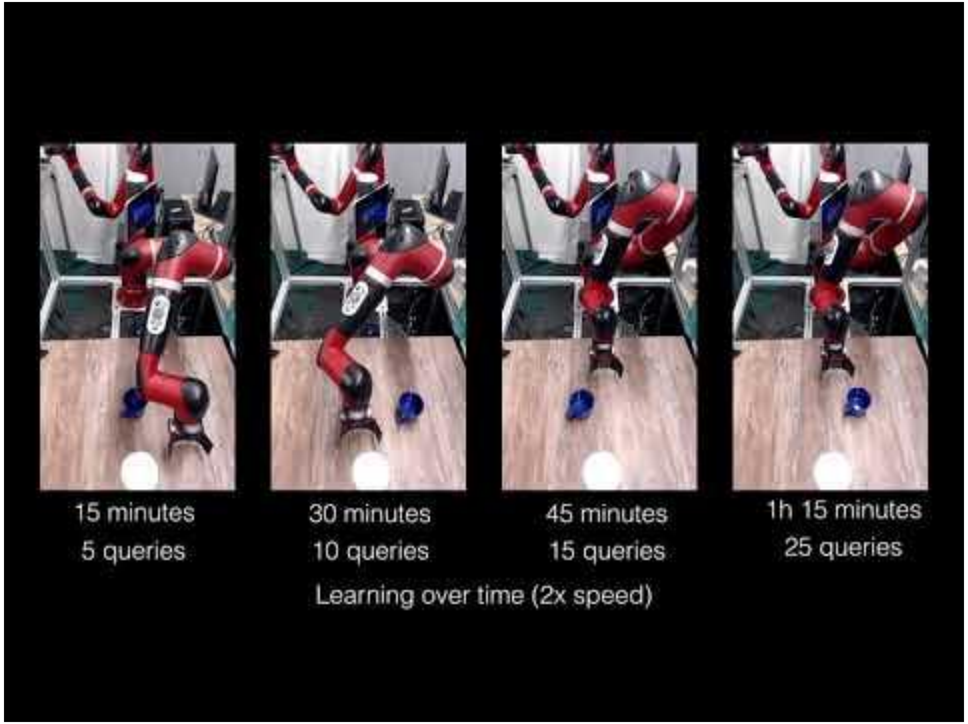
Video:

https://youtu.be/9pWJzb4G-CA

Results:

	VICE-RAQ (ours)	RAQ (ours)	VICE (ours)	Naïve Classifier
visual pushing	100%	60%	0%	0%
visual draping	100%	100%	100%	0%
visual bookshelf	100%	0%	60%	0%

Fig. 7. Results on the real world robot experiments. For all tasks, the reported numbers are success rates, indicating whether or not the object was successfully pushed to the goal, whether the cloth was successfully draped over the able, and whether the book was placed correctly on the shelf, averaged across 10 trials. In all cases, VICE-RAQ succeeds at learning the task, while VICE and RAQ succeed at some tasks while failing at others.



Discussion

- For the Visual Pushing task, VICE-RAQ obtains a success rate of 100%, while RAQ only obtains a success rate of 60%. Both off-policy VICE and naive classifier fail to solve this task. This indicates that including active queries in the classifier training process is helpful for obtaining good rewards, both with and without VICE.
- For the Visual Draping task, we observe that all of our reward-learning methods (off-policy VICE, VICE-RAQ and RAQ) are able to solve the task, and only the naive classifier baseline fails. (Baseline with hand-defined reward function also failed at draping task)
- The number of queries required at each training iteration

Other Sources

- https://stackoverflow.com/questions/37370015/what-is-the-difference-between-value-iteration-and-policy-iteration
- https://youtu.be/CLZkpo8rEGg
- https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63
- https://sergioskar.github.io/Actor_critics/
- https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html
- https://youtu.be/jmMsNQ2eug4

- Justin Fu, Avi Singh, Dibya Ghosh, Larry Yang, and Sergey Levine. Variational inverse control with events: A general framework for data-driven reward definition. In Advances in Neural Information Processing Systems, 2018.
- Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. 2018.