Robotic Learning Notes

Patrick Yin

Updated August 31, 2021

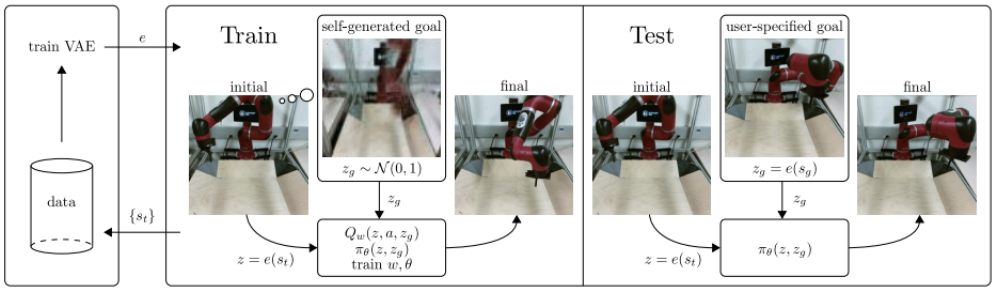
Contents

1 Visual Reinforcement Learning with Imagined Goals

Paper here.

1.1 Summary

This paper introduces self-supervised goal-conditioned reinforcement learning with their method, reinforcement learning with imagined goals (RIG).



RIG involves a VAE with encoder $q_\phi(s)$ and decoder $p_\psi(z)$, goal-conditioned policy $\pi_\theta(z, z_g)$, and goal-conditioned Q-function $Q_w(z, a, z_g)$. Here, s is a state observation (i.e. image), z is its latent encoding of s, z_g is the latent encoding of the sampled goal, and a is the action taken.

1.1.1 Variational Autoencoder Training

We first train the β-VAE by executing a random policy, collecting state observations, and optimizing

$$\mathcal{L}(\psi, \phi; s^{(i)}) = -\beta D_{KL}(q_\phi(z|s^{(i)})\|p(z)) + \mathbb{E}_{q_\phi(z|s^{(i)})}[\log p_\psi(s^{(i)}|z)]$$

where $p(z)$ is a prior which we take to be unit Gaussian. Our encoding is the mean of the encoder: $z = \mu_\phi(s)$.

1.1.2 Goal-conditioned RL Training

Next, we train our Q-function and policy in this latent space. We use twin delayed deep deterministic policy gradients (TD3) to train our policy and q-function following the Bellman error

$$\mathcal{E}(w) = \frac{1}{2}\|Q_w(s, a, g) - (r(s', g) + \gamma \max_{a'} Q_{\bar{w}}(s', a', g))\|^2$$

where the reward is determined with negative Mahalanobis distance in the latent space between the latent encoding the state and the goal: $r(z, z_g) = -\|z - z_g\|_A$. We can set matrix A to be the precision matrix of q_ϕ, but in practice $A = \mathbf{I}$

works better. We also fine-tune the VAE here to randomly generated state observations and state observations collected during explorations so it can be exposed to new states it wasn't trained on previously.

1.1.3 Latent Goal Relabeling

We can enable sample-efficient learning by using the VAE to relabel goals. Given the (s, a, s') in our dataset, we encode the state observations, sample z_g from our VAE prior $p(z)$, and compute $r(z, z_g) = -\|z - z_g\|_A$ to produce (s, a, s', g, r), giving us more data to train on without sampling new datapoints. Half ot the goals are generated this way. The other half has its goals generated using states seen along the trajectory as in hindsight experience replay (HER).

1.1.4 Automated Goal-Generation for Exploration

We sample from the VAE prior to obtain plausible goals and give this to our policy $\pi(z, z_g)$ to collect data. This is essentially a self-supervised "practice" phase during training.

1.1.5 Algorithm Summary

Given VAE encoder q_ϕ, VAE decoder p_ψ, policy π_θ, goal-conditioned value function Q_w, we have RIG:

Algorithm 1 RIG: Reinforcement learning with imagined goals

Collect $\mathcal{D} = \{s_{(i)}\}$ using any exploration policy
Train β-VAE on $\mathcal{D}$
Fit prior $p(z)$ to latent encodings $\{\mu_\phi(s^{(i)})\}$
for $n = 0, ..., N-1$ episodes **do**
  Sample latent goal from prior $z_g \sim p(z)$.
  Sample initial state $s_0 \sim E$
  for $t = 0, ..., H-1$ steps **do**
    Get action $a_t = \pi_\theta(e(s_t), z_g) + \text{noise}$
    Get next state $s_{t+1} \sim p(\cdot|s_t, a_t)$
    Store (s_t, a_t, s_{t+1}, z_g) into replay buffer $\mathcal{R}$
    Sample transition $(s, a, s', z_g) \sim \mathcal{R}$
    Encode $z = e(s)$, $z' = e(s')$
    With probability 0.5, replace z_g with $z'_g \sim p(z)$
    Compute reward $r = -\|z' - z_g\|$
    Train Q-function and policy on (z, a, z', z_g, r) with an RL algorithm (TD3 used in paper)
  end for
  for $t = 0, ..., H-1$ steps **do**
    for $i = 0, ..., k-1$ steps **do**
      Sample future state s_{h_i}, $t < h_i \leq H-1$
      Store $(s_t, a_t, s_{t+1}, e(s_{h_i}))$ into $\mathcal{R}$
    end for
  end for
  Fine-tune β-VAE every K episodes on mixture of D and R
end for

1.2 Q&A

1.2.1 Basic

- When calculating reward, how does Mahalanobis distance work if both the state and goal latent encoding are points, not distributions?
  - VAE outputs a distribution.
- When calculating reward, what is the precision matrix of a VAE encoder? How was $r(s, g) = -\|z - z_g\|_A \propto \sqrt{\log e_\phi(z_g|s)}$ derived?
  - Precision matrix is 1 over the covariance matrix. Not used anymore. Just set to identity.
- During VAE fine-tuning, what does it mean when it says state observations are randomly generated?
  - Manually random set initial state and add to replay buffer, then pull from replay buffer.

- During latent goal relabeling, what is a fitted prior? How does this help the distribution of latents match the prior (i.e. gaussian)? Why does it have to match?
  - Point is that extra dimensions that don't have meaning aren't used and set to zero during generation. No learning here, just a one-step gaussian fit.

1.2.2 Trivial

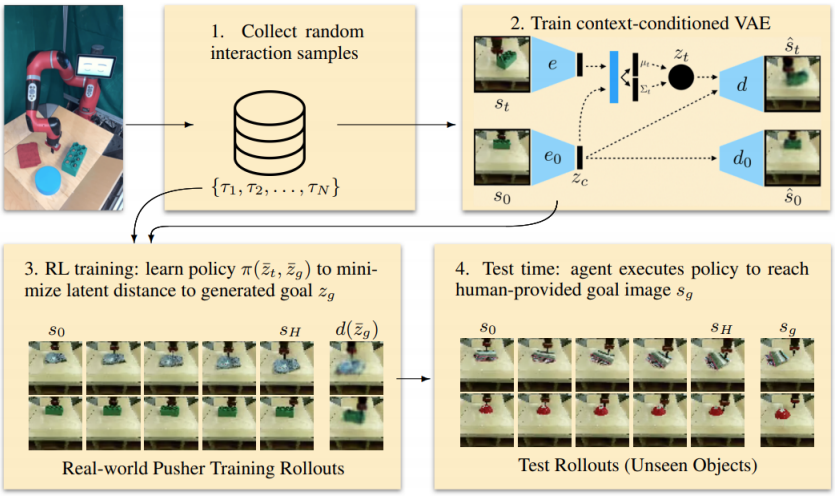
- When getting action from policy, why do we add noise?
  - For exploration.

2 Contextual Imagined Goals for Self-Supervised Robotic Learning

Paper here.

2.1 Summary

This paper introduces context-conditioned goal sampling. In other words, the VAE used in RIG is conditioned on the initial state, which prevents the generation of impossible goals and thus improves learning (i.e. a robot cannot push a red puck to a desired location if the red puck doesn't exist).



2.1.1 Context-Conditioned VAE (CC-VAE) Training

We use a modified CVAE which takes in the initial state s_0 as input, which we call the "context" of the rollout. This modified CVAE is called a context-conditioned VAE (CC-VAE). The CC-VAE takes in z_c as input c, which is s_0 encoded with convolutional encoder e_0. The loss function for training is

$$\mathcal{L}_{\text{CC-VAE}} = \mathcal{L}_{\text{CVAE}} + \log p(s_0|z_c)$$

$$\mathcal{L}_{\text{CVAE}} = -\mathbb{E}_{q_\phi(z|s,c)}[\log p(s|z,c)] + \beta D_{KL}(q_\phi(z|s,c)\|p(z))$$

Essentially, this is the CVAE loss with an additional term for reconstructing the initial state. Due to the information bottleneck on z_t, the optimal solution would encode as much information as possible in z_c, while only encoding information about state changes from initial state within z_t.

2.1.2 Context-Conditioned Reinforcement Learning with Imagined Goals (CC-RIG) Training

In CC-RIG, we essential just do RIG again but now with the CC-VAE instead of a VAE. We compute encoding of s_0, $z_c = e_0(s_0)$. Let $\bar{z}$ denote concatenated (z, z_c) and $\mu(s, s_0)$ denote the mean of $q_\phi(z|s_t, s_0)$. Given our encoders $\mu(s_t, s_0)$, $e_0(s_0)$, policy $\pi_\theta(\bar{z}, \bar{z}_g)$, goal-conditioned value function $Q_w(\bar{z}, \bar{z}_g)$, and dataset of trajectories $\mathcal{D} = \{\tau^{(i)}\}$. In practice, we sample $z_g \sim N(0, I)$ as in RIG.

Algorithm 2 Context-Conditioned RIG

Train CC-VAE on $\mathcal{D}$
for $n = 0, ..., N-1$ episodes **do**
  Sample latent goal from prior $z_g \sim p(z)$, $\bar{z}_g = (z_g, z_c)$
  Sample initial state $s_0 \sim p(s_0)$
  Encode $z_c = e_0(s_0)$
  for $t = 0, ..., H-1$ steps **do**
    Observe s_t and encode $\bar{z}_t = (\mu(s_t, s_0), z_c)$
    Get action $a_t = \pi_\theta(\bar{z}_t, \bar{z}_g) + \text{noise}$
    Get next state $s_{t+1} \sim p(\cdot|s_t, a_t)$
    Store $(\bar{z}_t, a_t, \bar{z}_{t+1}, \bar{z}_g)$ into replay buffer $\mathcal{R}$
    Sample transition $(\bar{z}, a, \bar{z}', \bar{z}_g) \sim \mathcal{R}$
    Compute reward $r = -\|\bar{z}' - \bar{z}_g\|$
    Train Q-function and policy on $(\bar{z}, a, \bar{z}', \bar{z}_g, r)$ with an RL algorithm (TD3 used in paper)
  end for
end for

During testing, given goal image s_g, we encode it as $z_g = \mu(s_g, s_0)$ and execute policy with latent goal $\bar{z}_g$.

2.2 Q&A

2.2.1 Basic

- Are latent relabeling, fitted priors, adding future-state data, VAE fine-tuning still done?
  - Latent relabeling/future-state data is done. Fitted priors and fine-tuning no.

3 AWAC: Accelerating Online Reinforcement Learning with Offline Datasets

Paper here.

3.1 Summary

Advantage weighted actor critic (AWAC) is an actor critic algorithm that leverages a combination of prior demonstration data (which could be sub-optimal) and online experience. The challenge to learning from offline data and subsequent online fine-tuning is

- **Data Efficiency**: Pre-training with imitation learning and fine-tuning with on-policy RL algorithms has two drawbacks: the prior data may not be optimal so imitation learning may be ineffective, and on-policy fine-tuning is data inefficient because it doesn't reuse prior data in the RL stage.
- **Bootstrapping Error**: Off-policy actor-critic methods struggle with off-policy bootstrapping error accumulation. Q-estimates are not fully accurate for bootstrapping, especially when the actions are not in data distribution. The policy exploits overestimated Q-values, making estatimed Q values worse.
- **Non-stationary Behavior Models**: Prior offline RL algorithms address the bootstrapping problem by constraining the policy π close to behavior policy π_β (actions present in the replay buffer). Many offline RL algorithms explicitly fit a parametric model to samples from the replay buffer for the distribution π_β. However, fitting an accurate behavorial model as data is collected online during fine-tuning is a challenging research problem. We require an off-policy RL algorithm that constrains the policy to prevent offline stability and error accumulation, but is not so conservative that it prevents online fine-tuning due to imperfect behavior modeling.

AWAC avoids these issues. For data efficiency, the algorithm trains a critic with DP. To avoid bootstrapping error while avoiding modeling the data distribution, we optimize

$$\arg\max_{\pi} \mathbb{E}_{a\sim\pi(\cdot|s)}[A^{\pi_k}(s,a)] \text{ s.t. } D_{KL}(\pi(\cdot|s)\|\pi_\beta(\cdot|s)) \leq \epsilon, \int_a \pi(a|s)da = 1$$

The Lagrangian is

$$\mathcal{L}(\pi,\lambda,\alpha) = \mathbb{E}_{a\sim\pi(\cdot|s)}[A^{\pi_k}(s,a)]+\lambda(\epsilon-D_{KL}(\pi(\cdot|s)\|\pi_\beta(\cdot|s)))+\alpha(1-\int_a \pi(a|s)da)$$

Differentiating with respect to π gives

$$\frac{\partial\mathcal{L}}{\partial\pi} = A^{\pi_k}(s,a) - \lambda\log\pi_\beta(a|s) + \lambda\log\pi(a|s)_\lambda - \alpha$$

Setting $\frac{\partial \mathcal{L}}{\partial \pi}$ to zero and solving for π gives the closed form solution

$$\pi^*(a|s) = \frac{1}{Z(s)} \pi_\beta(a|s) \exp\Big(\frac{1}{\lambda} A^{\pi_k}(s,a)\Big)$$

We now want to project our optimal solution into our parameter space, which we will do by minimizing the KL-divergence of π_θ from π^* under $p_{\pi_\beta}(s)$:

$$\arg\min_\theta \mathbb{E}_{p_{\pi_\beta}(s)}[D_{KL}(\pi^*(\cdot|s)\|\pi_\theta(\cdot|s))] = \mathbb{E}_{p_{\pi_\beta}(s)}\Big[\mathbb{E}_{\pi^*(\cdot|s)}[-\log \pi_\theta(\cdot|s)]\Big]$$

We choose the reverse KL direction because it allows us to optimize θ as MLE problem rather than sampling actions from a policy that may be out of distribution for the Q function. We can know compute the policy update by sampling directoy from β:

$$\theta_{k+1} = \arg\max_\theta \mathbb{E}_{s,a\sim\beta}\Big[\log \pi_\theta(a|s) \exp\Big(\frac{1}{\lambda} A^{\pi_k}(s,a)\Big)\Big]$$

This actor update resembles weighted behavior cloning, where targets are obtained by reweighting the state-action pairs observed in the current dataset with predicted advantages from the learned critic. With this, we now have our AWAC algorithm

Algorithm 3 AWAC

Dataset $\mathcal{D} = \{(s, a, s', r)_j\}$
Initialize buffer $\beta = \mathcal{D}$
Initialize π_θ, Q_ϕ
for iteration $i = 1, 2, ...$ **do**
  Sample batch $(s, a, s', r) \sim \beta$
  $y = r(s,a) + \gamma \mathbb{E}_{s',a'}[Q_\phi(s', a')]$
  $\phi \leftarrow \arg\min_\phi \mathbb{E}_{\mathcal{D}}[(Q_\phi(s,a) - y)^2]$
  $\theta \leftarrow \arg\max_\theta \mathbb{E}_{s,a\sim\beta}[\log \pi_\theta(a|s) \exp(\frac{1}{\lambda} A^{\pi_k}(s,a))]$
  if $i >$ num_offline_steps **then**
    $\tau_1, ..., \tau_K \sim p_\theta(\tau)$
    $\beta \leftarrow \beta \cup \{\tau_1, ..., \tau_K\}$
  end if
end for

3.2 Q&A

3.2.1 Basic

- For deriving AWAC, when setting $\frac{\partial \mathcal{L}}{\partial \pi}$ to zero, I got

$$\pi^*(a|s) = \frac{1}{Z(s)} \pi_\beta(a|s) \exp\Big(-\frac{1}{\lambda} A^{\pi_k}(s,a)\Big)$$

  . What am I missing?

– Probably something wrong with Lagrangian. Flipped sign or something. I'll check this later.

4 What Can I Do Here? Learning New Skills by Imagining Visual Affordances

Paper here.

4.1 Summary

This paper introduces visual affordance learning (VAL), which is essentially AWAC + a scaled up CC-RIG to allow for offline training + online fine-tuning, more expressive generative models, and learning a diverse set of skills. VAL consists of three learning phases: an affordance learning phase to learn affordances from the prior data, an offline behavior learning phase to learn behaviors from prior data, and an online behavior learning phase where an agent actively interacts with the test environment using affordances and learns potential behaviors in the new environment.

4.1.1 Affordance Learning

We use a lower-dimensional latent space to encode the image observations in order to make goal generation easier. Given such a latent space, we can then learn affordances by training a conditional model $p(z_t|z_0)$ to generate plausible outcomes of an initial state. For the latent variable generative model $p(s_t|z_t)$, we use a deterministic VQVAE $z_t = \phi(s_t)$. For the conditional model, we use a conditional PixelCNN in the latent space.

4.1.2 Offline Behavior Learning

We use AWAC to optimize $\pi(a|z, z_g)$ with reward function $r(z, z_g) = -\mathbf{1}_{\|z-z_g\|>\epsilon}$. We also relabel z_g with future hindsight experience with 40% probability and sample from $p(z_t|z_0)$ with 40% probability.

4.1.3 Online Behavioral Learning

During the online fine-tuning phase, we sample from the affordance module $z_g \sim p_\theta(\cdot|z_0)$ and roll out our policy $\pi(a|z, z_g)$. We then iterate between improving the policy with offline RL and collecting exploration data, and appending it to the replay buffer.

4.1.4 Algorithm Summary

Given dataset $\mathcal{D}$, policy $\pi(a|z, z_g)$, Q-function $Q(z, a, z_g)$, RL algorithm $\mathcal{A}$, replay buffer $\mathcal{R}$, relabeling strategy $p_{RS}(z)$, and environment family $p(\mathcal{E})$

Algorithm 4 Visual Affordance Learning

Learn encoder $\phi(z|s)$ by generative model of $\mathcal{D}$
Learn affordances $p(z_t|z_0)$ by generative model of $\mathcal{D}$
Add latent encoding of $\mathcal{D}$ to the replay buffer
Initialize π and Q by running $\mathcal{A}$ offline
Sample $\mathcal{E}_{new} \sim p(\mathcal{E})$, $\mathcal{E}_{new} = (p_{new}(s_0), p_{new}(s_{t+1}|s, a))$
for $1, ..., N_{episodes}$ **do**
  Sample initial state $s_0 \sim p_{new}(s_0)$
  Sample goal $z_g \sim p(z_t|z_0)$
  for t = 0,...,H **do**
    Sample $a_t \sim \pi(\cdot|z_t, z_g)$
    Sample $s_{t+1} \sim p_{new}(\cdot|s_t, a_t)$
  end for
  Store trajectory $(z_1, a_1, ..., z_H)$ in replay buffer $\mathcal{R}$
  for $1, ..., N_{train\_steps}$ **do**
    Sample transition (z_t, a_t, z_{t+1}, z_g)
    Relabel with $z'_g \sim p_{RS}(z_g)$ and recompute reward
    Update π and Q with relabeled transition using $\mathcal{A}$
  end for
end for

5 Bridge Data: Boosting Generalization of Robotic Skills with Cross-Domain Datasets

Paper here.

5.1 Summary

This is a dataset paper which includes a diverse 'bridge' dataset of 4,700 human demonstrations of a robot performing 33 common kitchen tasks across 3 toy kitchens and 3 toy sinks with varying lighting, robot positions, and backgrounds. The role of the bridge dataset is to boost generalization for policies:

1. **Transfer with matching behaviors**: user collects some data in their target domain for tasks that are also present in the bridge data, and uses the bridge data to boost performance and generalization of these tasks
2. **Zero-shot transfer with target support**: user utilizes some data of one task in their target domain to import other tasks that are present in the bridge data without additionally collecting new demonstration of them in the target domain
3. **Boosting generalization of new tasks**: user provides some data for a new task that is not present in the bridge data, and then utilizes the bridge to boost generalization and performance

6 C-Learning: Learning to Achieve Goals via Recursive Classification

Paper here.

6.1 Summary

C-learning is a goal-conditioned RL algorithm that learns a conditional probability density function over future states by training a classifier to predict whether an observation comes from the future.

6.1.1 Monte Carlo C-Learning

We want to model the future γ-discounted state density function:

$$p^\pi_+(s_{t+}|s_t, a_t) \triangleq (1-\gamma)\sum_{\triangle=1}^{\infty} \gamma^\triangle p^\pi_\triangle(s_{t+\triangle} = s_{t+}|s_t, a_t)$$

We will first derive an on-policy Monte Carlo C-learning algorithm. Given a distribution over state action pairs, $p(s_t, a_t)$, we define the marginal future state distribution $p(s_{t+}) = \int p^\pi_+(s_{t+}|s_t, a_t)p(s_t, a_t)ds_t da_t$. The classifier will take in (s_t, a_t, s_{t+}) can predict whether s_{t+} was sampled from $p^\pi_+(s_{t+}|s_t, a_t)$ ($F = 1$) or $p(s_{t+})$ ($F = 0$). The Bayes optimal classifier is

$$p(F = 1|s_t, a_t, s_{t+}) = \frac{p^\pi_+(s_{t+}|s_t, a_t)}{p^\pi_+(s_{t+}|s_t, a_t) + p(s_{t+})}$$

Thus, our future state density function estimate is

$$f^\pi_\theta(s_{t+}|s_t, a_t) = \frac{C^\pi_\theta(F = 1|s_t, a_t, s_{t+})}{C^\pi_\theta(F = 0|s_t, a_t, s_{t+})}p(s_{t+})$$

Our policy will choose action a_t that maximizes this density, and this solution doesn't depend on marginal $p(s_{t+})$. After sampling state-action pair $(s_t, a_t) \sim p(s_t, a_t)$, we sample $s^{(1)}_{t+} \sim p^\pi_+(s_{t+}|s_t, a_t)$ with label $F = 1$ and $s^{(0)}_{t+} \sim p^\pi_+(s_{t+})$ with label $F = 0$. We then train a classifier to maximize log likelihood

$$\mathcal{F}(\theta) \triangleq \mathbb{E}_{\substack{s_t,a_t\sim p(s_t,a_t) \\ s^{(1)}_{t+}\sim p^\pi_+(s_{t+}|s_t,a_t)}} [\log C^\pi_\theta(F = 1|s_t, a_t, s^{(1)}_{t+})] + \mathbb{E}_{\substack{s_t,a_t\sim p(s_t,a_t) \\ s^{(0)}_{t+}\sim p_+(s_{t+})}} [\log C^\pi_\theta(F = 0|s_t, a_t, s^{(0)}_{t+})]$$

To sample $p^\pi_+(s_{t+}|s_t, a_t)$, we use ancestral sampling with $\triangle \sim \text{GEOM}(1-\gamma)$. This is on-policy because $p^\pi_+(s_{t+}|s_t, a_t)$ depends on π and s_{t+}.

With this we have our Monte Carlo C-learning algorithm: given input trajectories $\{\tau_i\}$, $p(s,a) \sim \text{Unif}(\{s,a\}_{(s,a)\sim\tau})$, and $p(s_+) \sim \text{Unif}(\{s_t\}_{s_t\sim\tau, t>1)}$,

Algorithm 5 Monte Carlo C-Learning

1: **while** not converged **do**
2: Sample $s_t, a_t \sim p(s,a), s_{t+}^{(0)} \sim p(s_{t+}), \triangle \sim \text{GEOM}(1-\gamma)$
3: Set goal $s_{t+}^{(1)} \leftarrow s_{t+\triangle}$
4: $\mathcal{F}(\theta) \leftarrow \log C_\theta^\pi(F=1|s_t, a_t, s_{t+}^{(1)}) + \log C_\theta^\pi(F=0|s_t, a_t, s_{t+}^{(0)})$
5: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{F}(\theta)$
6: **end while**
7: **return** classifier C_θ

6.1.2 Off-Policy C-Learning

On-policy learning precludes the ability to readily share experience across tasks because we cannot use experience collected when commanding one goal to learn a classifier for another goal. Note that

$$p_+^\pi(s_{t+} = s_{t+}|s_t, a_t) = (1-\gamma)p(s_{t+1} = s_{t+}|s_t, a_t) + \gamma \mathbb{E}_{\substack{p(s_{t+1}|s_t,a_t) \\ \pi(a_{t+1}|s_{t+1})}} \left[p_+^\pi(s_{t+} = s_{t+}|s_{t+1}, a_{t+1}) \right]$$

We can now rewrite our classification objective as

$$\begin{aligned}\mathcal{F}(\theta, \pi) = \mathbb{E}_{\substack{p(s_t,a_t),p(s_{t+1}|s_t,a_t) \\ \pi(a_{t+1}|s_{t+1}),p_+^\pi(s_{t+}|s_{t+1},a_{t+1})}} & [(1-\gamma)\log C_\theta^\pi(F=1|s_t, a_t, s_{t+1}) + \gamma \log C_\theta^\pi(F=1|s_t, a_t, s_{t+})] \\ & + \mathbb{E}_{p(s_t,a_t),p(s_{t+})}[\log C_\theta^\pi(F=0|s_t, a_t, s_{t+})]\end{aligned}$$

We can estimate expectations that use $p_+^\pi(s_{t+}|s_{t+1}, a_{t+1})$ by sampling from $s_{t+} \sim p(s_{t+})$ and then weighting samples by importance weight

$$w(s_{t+1}, a_{t+1}, s_{t+}) \triangleq \frac{p_+^\pi(s_{t+}|s_{t+1}, a_{t+1})}{p(s_{t+})} = \frac{C_\theta^\pi(F=1|s_{t+1}, a_{t+1}, s_{t+})}{C_\theta^\pi(F=0|s_{t+1}, a_{t+1}, s_{t+})}$$

The new objective is then

$$\begin{aligned}\mathcal{F}(\theta, \pi) = \mathbb{E}_{\substack{p(s_t,a_t),p(s_{t+1}|s_t,a_t) \\ p(s_{t+}),\pi(a_{t+1}|s_{t+1})}} & [(1-\gamma)\log C_\theta^\pi(F=1|s_t, a_t, s_{t+1}) \\ & + \gamma \lfloor w(s_{t+1}, a_{t+1}, s_{t+}) \rfloor_{sg} \log C_\theta^\pi(F=1|s_t, a_t, s_{t+}) \\ & + \log C_\theta^\pi(F=0|s_t, a_t, s_{t+})]\end{aligned}$$

where $\lfloor \cdot \rfloor_{sg}$ indicates the gradients of the importance-weighted objective should not depend on the gradients of the importance weights.

We now have our off-policy C-learning algorithm: given transitions $\{s_t, a, s_{t+1}\}$ and policy π_ϕ

Algorithm 6 Off-Policy C-Learning

while not converged **do**
  Sample $(s_t, a_t, s_{t+1}) \sim p(s, a, s_{t+1}), s_{t+} \sim p(s_{t+}), a_{t+1} \sim \pi_\phi(a_{t+1}|s_t, a_t)$
  $w \leftarrow \text{stop\_grad}\left(\frac{C^\pi_\theta(F=1|s_{t+1}, a_{t+1}, s_{t+}}{C^\pi_\theta(F=0|s_{t+1}, a_{t+1}, s_{t+}}\right)$
  $\mathcal{F}(\theta, \pi) \leftarrow (1-\gamma)\log C^\pi_\theta(F=1|s_t, a_t, s_{t+1}) + \log C^\pi_\theta(F=0|s_t, a_t, s_{t+}) + \gamma w \log C^\pi_\theta(F=1|s_t, a_t, s_{t+})$
  $\theta \leftarrow \theta - \eta\nabla_\theta \mathcal{F}(\theta, \pi)$
end while
return classifier C^π_θ

6.2 Goal-Conditioned C-Learning

With commanded goal $s_{t+} = g \in S$, we have objective

$$\begin{aligned}\mathcal{F}(\theta, \pi) = \mathbb{E}_{\substack{p(s_t,a_t), p(s_{t+1}|s_t,a_t) \\ p(s_{t+}), \pi(a_{t+1}|s_{t+1}, g=s_{t+})}} & [(1-\gamma)\log C^\pi_\theta(F=1|s_t, a_t, s_{t+1}) \\ & + \gamma \lfloor w(s_{t+1}, a_{t+1}, s_{t+}) \rfloor_{sg} \log C^\pi_\theta(F=1|s_t, a_t, s_{t+}) \\ & + \log C^\pi_\theta(F=0|s_t, a_t, s_{t+})]\end{aligned}$$

The difference between this objective and the off-policy one is that the next action is sampled from a goal-conditioned policy. The density function obtained represents the future state density of s_{t+} given commanded goal $g = s_{t+}$. We now need to optimize our policy w.r.t the learned density function by maximizing the policy's probability of reaching the commanded goal:

$$\mathcal{G}(\phi) = \max_\phi \mathbb{E}_{\pi_\phi(a_t|s_t,g)}[\log C^\pi_\theta(F=1|s_t, a_t, s_{t+}=g)]$$

With this we have the goal-conditioned C-learning algorithm: given transitions $\{s_t, a, s_{t+1}\}$,

Algorithm 7 Goal-Conditioned C-Learning

while not converged **do**
  Sample $(s_t, a_t, s_{t+1}) \sim p(s, a, s_{t+1}), s_{t+} \sim p(s_{t+}), a_{t+1} \sim \pi_\phi(a_{t+1}|s_t, a_t, s_{t+})$
  $w \leftarrow \text{stop\_grad}\left(\frac{C^\pi_\theta(F=1|s_{t+1}, a_{t+1}, s_{t+}}{C^\pi_\theta(F=0|s_{t+1}, a_{t+1}, s_{t+}}\right)$
  $\mathcal{F}(\theta, \pi) \leftarrow (1-\gamma)\log C^\pi_\theta(F=1|s_t, a_t, s_{t+1}) + \log C^\pi_\theta(F=0|s_t, a_t, s_{t+}) + \gamma w \log C^\pi_\theta(F=1|s_t, a_t, s_{t+})$
  $\theta \leftarrow \theta - \eta\nabla_\theta \mathcal{F}(\theta, \pi)$
  $\mathcal{G}(\phi) \leftarrow \mathbb{E}_{\pi_\phi(a_t|s_t, g=s_{t+})}[\log C^\pi_\theta(F=1|s_t, a_t, s_{t+})]$
  $\phi \leftarrow \phi + \eta\nabla_\phi \mathcal{G}(\phi)$
end while
return policy π_ϕ

6.3 Q&A

6.3.1 Basic

- How does on-policy Monte Carlo C-learning depend on the commanded goal, and as a result cannot share experience across tasks?
- How were the objectives for C-learning and Q-learning derived? How does this motivate hypothesis 1 and 2?
- How were remark 1, 2, and 3 proved/derived?

6.3.2 Trivial

- Why isn't there a policy gradient descent step in Monte Carlo and Off-Policy C-Learning?

7 Persistent Reinforcement Learning via Subgoal Curricula

Paper here.

7.1 Summary

Value-accelerated Persistent Reinforcement Learning (VaPRL) generates a curriculum of initial states such that the agent can bootstrap on the success of easier tasks to efficiently learn harder tasks.

7.1.1 Persistent RL

Persistent RL distinguishes between training and evaluation objectives, where the evaluation objective measures the performance of the desired behavior while the training objective enables us to acquire those behaviors. Naively optimizing r in "long" episode horizons deteriorates performance, we we use a surrogate reward function

$$\tilde{r}_t(s_t, a_t) = \begin{cases} r(s_t, a_t) & t = [1, H_E], [2H_E + 1, 3H_E], ... \\ r_p(s_t, a_t) & [H_E + 1, 2H_E], ... \end{cases}$$

Here our reward alternates between task-reward r for H_E steps and r_p which encourages initial state distribution recovery for H_E steps. This is only suitable for reverse environments, where agent can continue to make progress on the task and not get stuck. The training objective is then

$$J_T(\pi) = \mathbb{E}_{s_0 \sim \tilde{p}, a_t \sim \pi(\cdot|s_t, G(s_t, p_g)), s_{t+1} \sim p(\cdot|s_t, a_t)}[\sum_{t=1}^{H_T} \gamma^t r(s_t, G(s_t, p_g))]$$

where the goal generator G is used to generate a curriculum of goals.

7.1.2 VaPRL

We want an increasingly-difficult curriculum so that the policy is eventually able to reach goal g from initial state distribution p. We propose using subgoal $C(g)$ such that

$$C(g) = \arg\min_s \mathcal{X}_p(s) \text{ s.t. } V^\pi(s, g) \geq \epsilon$$

where $\mathcal{X}_p(s)$ is a user-specified distance function between state s and initial state distribution p. The value represents the ability of the policy to reach from g from s. The goal generator $G(s_t, p_g)$ is then

$$G(s_t, p_g) = g \text{ s.t. } \begin{cases} g_{task} \sim p_g, g \leftarrow C(g_{task}) & \text{if } switch(s_t, g) = subgoal \\ r_p(s_t, a_t) & \text{elif } switch(s_t, g) = taskgoal \end{cases}$$

We use $\mathcal{X}_p(s) = -\mathbb{E}_{s_0 \sim p} V^\pi(s, s_0)$. Since calculating the min for $C(g)$ is intractable, we minimize $C(g)$ over a randomly sampled subset of the data collect by policy π by enumeration. We also relabel every trajectory collected in the training environment with N goals sampled randomly from the set of goals that may be part of the curriculum. With this, we have the VaPRL algorithm: given initial state(s) $\mathcal{D}_p$, N (number of goals for relabeling), and demos $\mathcal{D}$,

Algorithm 8 Value-accelerated Persistent Reinforcement Learning (VaPRL)

Initialize replay buffer $\mathcal{B}$, $\pi(a|s, g)$, $Q^\pi(s, a, g)$
$\mathcal{B} \leftarrow \mathcal{B} \cup \mathcal{D}$
relabel_demos($\mathcal{B}$)
while not done **do**
  $s \sim \tilde{p}$
  for H_T steps **do**
    $g \leftarrow G(s, p_g)$
    $a \sim \pi(\cdot|s, g)$, $s' \sim p(\cdot|s, a)$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s, a, s', g, r(s', g)\}$
    for $i \leftarrow 1, i \leq N$ **do**
      $\tilde{g} \sim \mathcal{D} \cup p_g$
      $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s, a, s', \tilde{g}, r(s', \tilde{g}))\}$
    end for
    update π, Q^π
    $s \leftarrow s'$
  end for
end while

7.2 Q&A

7.2.1 Basic

- Does VaPRL use surrogate reward?
- What are issues with VaPRL is solving reset-free problem?