Hyperparameter Auto-tuning in Self-Supervised Robotic Learning

Jiancong Huang1, Juan Rojas2, Matthieu Zimmer3, Hongmin Wu4, Yisheng Guan1, and Paul Weng3,5

Abstract—Policy optimization in reinforcement learning requires the selection of numerous hyperparameters across different environments. Fixing them incorrectly may negatively impact optimization performance leading notably to insufficient or redundant learning. Insufficient learning (due to convergence to local optima) results in under-performing policies whilst redundant learning wastes time and resources. The effects are further exacerbated when using single policies to solve multi-task learning problems. In this paper, we study how the Evidence Lower Bound (ELBO) used in Variational Auto-Encoders (VAEs) is affected by the diversity of image samples. Different tasks or setups in visual reinforcement learning incur varying diversity. We exploit the ELBO to create an auto-tuning technique in self-supervised reinforcement learning. Our approach can auto-tune three hyperparameters: the replay buffer size, the number of policy gradient updates during each epoch, and the number of exploration steps during each epoch. We use the state-of-the-art self-supervised robotic learning framework (Reinforcement Learning with Imagined Goals (RIG) using Soft Actor-Critic) as baseline for experimental verification. Experiments show that our method can auto-tune online and yields the best performance at a fraction of the time and computational resources. Code, video, and appendix for simulated and real-robot experiments can be found at www.JuanRojas.net/autotune

I. INTRODUCTION

Most Reinforcement Learning (RL) algorithms depend on many hyperparameters and their performances can be very sensitive to them [1], [2]. Hyperparameter tuning traditionally requires extensive computing resources and is challenging in practice, especially in multi-task settings since good hyperparameter values may be task-dependent. Recently, Zahavy et al. [3] made progress on self-tuning all the differentiable hyperparameters of an actor-critic loss function by using meta-gradients. Our work instead focuses on tuning three hyperparameters, which are: the replay buffer size $N_r$, the number of gradient updates in each epoch $N_\theta$, and the number of exploration steps in each epoch $N_e$. In contrast to [3], our hyperparameter auto-tuning technique is more broadly applicable than tuning specific algorithm parameters in various Monte Carlo RL. Different environment’s diversity may need different $N_{r,\theta,e} = (N_r, N_\theta, N_e)$. The poor optimization of these hyperparameters can negatively affect an agent’s learning progress in a cumulative way. For instance, an insufficient number of policy gradient update steps will result in a sub-optimal policy and thus underusing (wasting) some of the collected samples. Consequently, the policy used by the agent at the next epoch is suboptimal to explore and learn further. On the other hand, choosing too large a number for a hyperparameter results in a redundant number of policy gradient update steps resulting in wasted time and computing resources. In this paper, we contribute auto-tuning $N_{r,\theta,e}$ without extensive search for each new task.

In self-supervised reinforcement learning, Reinforcement learning with Imagined Goals (RIG) [4], [5], advanced learning general-purpose skills in task-agnostic environments by using RGB images as observations (eliminating the use for instrumentation) and by not specifically designing a reward function for each different task [4]–[7]. RIG leverages the self-generated goals to collect vast skills from the environment. In RIG, an agent learns to reach goals produced by a goal-generator that is fine-tuned (online) with the replay buffer’s collected samples. As such, exploiting the goal-generator’s performance to auto-tune parameters for policy learning is a promising direction.

To create the connection between the policy learning and the goal-generator, previous work has used the reward value to select the intermediate difficult samples to update the goal-generator (with GAN [8]) to learn goal-policy with increasing difficulty [7]. In RIG [5], [6], Variational Auto-Encoders (VAEs) are used as self goal-generators instead. Other research works have used ELBO in VAEs to measure the novelty of visited states as intrinsic rewards to enhance exploration [9], [10].

In this paper, we propose to use the ELBO signal with VAE-based goal-generators to estimate the diversity of the observations in the replay buffer and then auto-tune three hyperparameters to optimize learning:

- Tunes $N_r$ so as not to lose (or forget) transitions nor waste memory resources.
- Tunes $N_\theta$ to optimally update the policy at each epoch.
- Tunes $N_e$ to sample goals and interact with the environment with sufficient steps in each epoch.

To this end, we first estimate how the changing diversity of training samples affects the ELBO. The $-\text{ELBO}$ is the loss function of VAEs (details in Appendix VI-B). The value of the $-\text{ELBO}$ is directly proportional to the diversity in the training samples, which is also directly proportional to the diversity of the imagined goals for RIG. Consequently, we can exploit the $-\text{ELBO}$ to auto-tune $N_{r,\theta,e}$ which depend on the diversity of the goals. Our framework is summarized in Algo. [1]
The contributions of our work are threefold: (i) we identify that the loss function of VAEs has a lower bound related to the diversity of the samples; (ii) to avoid suboptimal learning or wasted computer resources, we propose a methodology that auto-tunes three hyperparameters $N_{r, \theta, e}$; (iii) we experimentally validate our approach on diverse domains, and we additionally report competitive performances in more general (curriculum learning and multitask) settings.

II. BACKGROUND

A. Visual Reinforcement Learning with Imagined Goals.

In RL, given a state observation $s \in S$, an agent decides which action $a \in A$ to take, where $S$ is the state observation space and $A$ is the action space. To quantify the goodness of an agent’s behavior, a reward function $R : S \to \mathbb{R}$ yields a reward signal at each time step depending on the observation. Hence, the agent has to learn a policy $\pi : S \to \Delta(A)$ to maximize an expected cumulative rewards.

In goal-conditioned RL [11], the policy must also take a goal $g \in \mathcal{G}$ into account, where $\mathcal{G}$ is the goal space. The policy definition becomes $\pi : S \times \mathcal{G} \to \Delta(A)$ and the reward function $R : S \times \mathcal{G} \to \mathbb{R}$ indicates the proximity to the goal.

In self-generated goal RL, goals are generated by the agent instead of being provided by the environment. More specifically, in RIG [5], [6], an observation and a goal belong to the same space $\mathcal{S} = \mathcal{G}$, they represent the current observed image and the target image respectively. To simplify the problem, instead of working in the high-dimensional observation space, the input is compressed into a latent space $Z$ via an encoder $p_{\phi} : S \to \Delta(Z)$ with parameters $\phi$. As the observation and goal spaces are equivalent, both can be encoded. Given an observation $s \in S$ and a goal $g \in \mathcal{G}$, we denote $z_s \sim p_{\phi}(s)$ and $z_g \sim p_{\phi}(g)$ to refer to their associated latent representations. Therefore, the policy can be decomposed as $\pi(s, g) = \pi_\theta(z_s, z_g)$ with $\pi_\theta : Z \times Z \to \Delta(A)$ being a parametric policy defined on the latent space where $\theta$ represents the parameters (weights of a neural network) to learn. They also assume that reward function is known to the agent and defined over the latent space as follows: $r(z_s, z_g) = -||z_s - z_g||_2$. In this paper, we use the L2 norm following [5] instead of the Mahalanobis distance mentioned in [6]. Thus, the best parameters $\theta^*$ for the policy are defined as:

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{z_s \sim \mathcal{N}(0,1)} \left[ \mathbb{E}_{z_g} \left[ \sum_{i} \gamma^i r(z_s^i, z_g^i) \right] \right].$$

where $I$ is the identity matrix of equal dimension as $Z$, self-generated goals $z_g$ are acquired by sampling a unit Gaussian, and $\gamma \in [0, 1)$ is a discount factor.

B. Evidence Lower Bound of Variational Auto-Encoders.

$\beta$-Variational Auto-Encoders ($\beta$-VAEs) learn structured latent representations of high dimensional data with the help of the following loss function [12], [13]:

$$\mathcal{L}_{\psi, \phi} = \mathbb{E}_{p(s)} \left[ \beta KL(p_\phi(z|s)||q(z)) - \mathbb{E}_{p_\phi(z|s)} \log p_\psi(s|z) \right].$$

The loss function, structured with the encoder $p_\phi(z|s)$, decoder $p_\psi(s|z)$, and prior $q(z)$ fixed with standard normal distribution, represents the negative normalized evidence lower bound (−ELBO) of the marginal likelihood of data points [14]. The term $\beta KL(p_\phi(z|s)||q(z))$ is the Kullback–Leibler divergence [15] between $p_\phi(z|s)$ and $q(z)$, with a penalty value $\beta \geq 1$. The second term $\mathbb{E}_{p_\phi(z|s)} \log p_\psi(s|z)$ is the reconstruction log-probability. The loss function in Eq. (2) is minimized by simultaneously optimizing with respect to $\phi$ and $\psi$. Note that decreasing the KL-divergence between $p_\phi(z|s)$ and $q(z)$ also decreases the reconstruction log-probability, which in turn may increase the loss function. In summary, it is through this process that VAE learns a latent representation that is maximally expressive about reconstruction and maximally compressive about the samples.

C. Information Theory and Goal-generator.

In information theory, the entropy of a random sample $S$ is defined as the amount of uncertainty. The entropy is maximum when the sample is uniformly distributed and its uncertainty is greatest. When referring to a set of samples, this measure of uncertainty can also be conceptualized as its diversity. In our work, the samples are the visual observations of RIG as well as the inputs to the VAE goal-generator. As such, the mutual information between input samples $S$ and output latent representation $Z$ is defined as $I(S,Z) = \mathbb{E}_{p(z|s)} \log \frac{\mathbb{E}_{p(z|s)} \log p(z)}{\mathbb{E}_{p(z)} p(z)}$ and represents the reduction of uncertainty about $S$ given $Z$ and vice-versa.

D. The Problem of choosing $N_{r, \theta, e}$.

In RIG [6], the training progress is separated by sequential epochs (or cycles). Each and every epoch has an exploration phase, a policy update phase, a policy evaluation phase, and a VAE fine-tuning phase as delineated in Alg. 1. RIG uses fixed values for the hyperparameters $N_{r, \theta, e}$.

The number of gradient updates, $N_\theta$, defines how many times the policy $\pi_\theta$ is updated by sampling transitions from the replay buffer in one epoch. If $N_\theta$ is too small, the agent will learn sub-optimally given the transitions stored in the replay buffer and use a sub-optimal policy in the next epoch. If $N_\theta$ is too large, unnecessary computational resources and time will be used to compute gradient updates that are redundant. The value of $N_\theta$ should be such that we could sample all the different transitions for gradient update at least one time.

The hyperparameter $N_e$, which corresponds to the number of exploration steps, specifies how many times the agent interacts with the environment (and stores transitions into the replay buffer) in each epoch. In self-generated goal RL, different environments may need different number of exploration steps at each epoch since the diversity of the goals is different. A sufficiently large value for $N_e$ guarantees to interact with the environment with all possible goals to get enough novel transitions for policy updates. If $N_e$ is limited (insufficient number of exploration steps), we will optimize the current policy $\pi_\theta$, with only a limited number of collected transitions, which may lead to a sub-optimal policy. On the
other hand, if $N_e$ is excessive, the superfluous number of exploration steps will be extremely time-costly, particularly in real-world setups. Such a problem is one of the main hurdles in the wider adoption of deep reinforcement learning methods in real-world scenarios. The value of $N_e$ should be such that the agent interacts with all the possible goals at least one time.

Regarding the replay buffer size, $N_r$, this hyperparameter controls how many transitions $\tau = (s, a, s', z_g)$ experienced during the exploration phase across all epochs are saved in the replay buffer $R$. If $N_r$ is insufficient, it is possible that the replay buffer is filled. If so, once novel transitions are stored, they will replace previously stored experiences leading to forgotten behaviors from previous exploration steps. On the other hand, if $N_r$ is redundant, unnecessary memory resources are used to maintain the large buffer. The value of $N_r$ should at least be equal to the product of $N_e$ and the trajectory length $l$ to exploit on-policy samples. Note that under the RIG setup, we only consider the behaviors for policy update that belong to a single (current) epoch.

III. AUTO-TUNING THE HYPERPARAMETERS

A. ELBO and Datasets with Changing Diversity

In information theory, we can use variational approximation to approximate the upper bound of $I(S; Z)$ as below (details in the Appendix VI-A):

$$I(S; Z) = H(S) - H(S|Z)$$

$$= \mathbb{E}_{p(s, z)} \log \frac{p(z|s)}{p(z)} \leq \mathbb{E}_{p(s, z)} \log \frac{p(z|s)}{q(z)}.$$  \hspace{1cm} (3)

It follows that:

$$H(S) \leq \mathbb{E}_{p(s, z)} \log \frac{p(z|s)}{q(z)} + H(S|Z)$$

$$= \mathbb{E}_{p(s, z)} \log \frac{p(z|s)}{q(z)} - \mathbb{E}_{p(z, s)} \log p(s|z)$$

$$= \mathbb{E}_{p(s)} \left[ KL(p(z|s)||q(z)) - \mathbb{E}_{p(z|s)} \log p(s|z) \right].$$  \hspace{1cm} (4)

Then, we approximate the intractable distributions $p(z|s)$ and $p(s|z)$ with $p_\phi(z|s)$ and $p_\psi(s|z)$. After combining the penalty $\beta$, the relationship between the diversity of the samples $H(S)$ and the VAE’s loss function is as below:

$$H(S) \leq \mathbb{E}_{p(s)} \left[ \beta KL(p_\phi(z|s)||q(z)) - \mathbb{E}_{p_\psi(z|s)} \log p_\psi(s|z) \right]$$

$$= -\text{ELBO}.$$  \hspace{1cm} (5)

We can see that the RHS of Eq. (5) is the loss function of the $\beta$-VAE [12], [13], [16] and has a lower bound $H(S)$. The learning process, as described in Sec. II-B, is the same process mentioned earlier that attempts to converge to the lower bound $H(S)$. From Eq. (5) it is possible to explain why the ELBO tends to stable values during VAE training when the diversity of the samples does not change. Compared to the previous equilibrium state, if the diversity of the samples increases, the KL-divergence term increases and the log-probability term decreases and vice-versa.

B. Auto-tuning of the Hyperparameters

In RIG [6], most of the next observations $s'$ in $\tau_i$ become the $s$ at the next transition $\tau_{i+1}$. Suppose the maximum size of the replay buffer is sufficient and always update the VAE model with sufficient steps during each epoch. If so, the environment’s observable diversity would be equal to the diversity of all the next observation $s'$ from all the transitions $\tau$ in the replay buffer. This diversity can be measured by $H(R^{s'})$, which is the entropy of the observable environment’s diversity. According to Sec. II-C, $H(R^{s'})$ is the measure of the distribution of all states in the replay buffer (it does not measure the number of states). Now, consider an RL batch size $b = 1$. We can use a hyperparameter $\xi$ to approximate the number of different states $s'$ in the replay buffer with $\xi H(R^{s'})$, where $H(R^{s'})$ is the lower bound of the loss function in Eq. [5] which can be approximated with $H(R^{s'}) < -\text{ELBO}_i$ in VAEs optimization.

The goal-generator is updated by sampling from the replay buffer during each training epoch. The images used to fine-tune the goal-generator are the next states $s'$ of each transition $\tau = (s, a, s', z_g)$. Note that the diversity of the input (visual state) is equal to the diversity of the reconstruction (generated goal) from the VAE. As such, the number of the various generated goals is equal to the number of different states and can also be approximated by $\xi H(R^{s'})$. In RIG, the latent goal $z_g \sim N(0, I)$ is used instead of the generated goal. For the policy $\pi_\theta$, the number of different latent goals should be equal to the number of different states.

For the replay buffer size, we at least need a size of $N_{ri} = l\xi \text{ELBO}_{i-1} \geq lN_{ei}$, where $l$ is the maximum number of trajectory length for each exploration step (equal to a full trajectory).

As for policy gradient updates, an agent randomly samples transitions from the replay buffer for gradient updates at each epoch. An agent needs to take at least $N_{\theta_i}$ steps in current epoch $i$ more than the number of different novel transitions $(\tau_i, \ldots, \tau_{i-1})$ stored in the replay buffer from the last epoch to explore. The number of different novel transitions in replay buffer should at least more than the number of different goals. We can tune $N_{\theta_i}$ in each epoch according to the number of different goals with $N_{\theta_i} = -\xi \text{ELBO}_{i-1} \geq \xi H(R^{s_{i-1}})$.

As for the number of exploration steps, recall that an agent takes exploration steps to generate goals and interact with the environment. As such, the least number of exploration steps is defined by $N_{ei} = -\xi \text{ELBO}_{i-1} \geq \xi H(R^{s_{i-1}})$.

Given the aforementioned reasons, it is thus possible to use a single hyperparameter $\xi$ with $-\text{ELBO}$ to approximate the true number of different states in the replay buffer as well as the number of different generated goals. This approximation does directly capture the utility of each $N_{\theta_i, e}$. Finally, although value $\xi$ could be optimized, we empirically show that setting it to 1 is sufficient ($b = 1024$). Although our auto-tuning method is generic, we formulate it (Algo. 1) in the RIG framework for concreteness and for the experimental
in the degree of a sample's diversity thus corroborating our samples with different hyperparameters \(\beta\). Note how a change in the \(-ELBO\) value is accompanied by a change in the degree of a sample’s diversity thus corroborating our proof in Sec. III.

Secondly, we continue to demonstrate this relationship by using simulated and real robot environments with visual observations. For example, different camera view angles and different workspaces (See detail in Fig. 9 at Appendix). Here, we change diversity of observation by manually changing the workspaces of the robot arm, or change the camera’s view angle (programmatically in simulation and manually in real world). In Fig. 3 we see the larger workspace of robot arm and the camera view angle (can captures more movable features) both experience a larger \(-ELBO\) value.

B. Ablation Studies of HyperParameters Auto-tuning

In this section, we conduct ablation studies to investigate the performance of only optimizing a single hyperparameter at-a-time. We run the Visual Push, Pickup, and Multi-object Push tasks of [5], [6] as shown in Fig. 1. See Tab. I for more detail about the setup of robot environments. In these experiments, the agent controls a simulated robot arm using only RGB image observations without access to any ground truth reward. The policy learning process is based on RIG (Sec. III) and Algo. 1. During training, the agent’s goal is to learn all possible skills from the environment. In policy evaluation, the aim is to achieve a user-specified goal given an initial state. The user-specified goal \(g\) can for instance be an image of the robot arm positioned somewhere relative to a puck (see Fig. 10 in the Appendix).

For the experiments below, we conduct redundant hyper-parameters \(N_{r,\theta,e}\) that achieve the best task-performance. We use these results to compare with the performance of our auto-tuning algorithm with time-consuming or resource-consuming. Task-performance metrics are set as in [6] and include: the Image Distance, Puck Distance, and Total Pickups according to the task. All experiments are averaged more than five runs per method and we report the mean ± standard deviation curves across metrics.

1) Auto-tuning of the replay buffer size: To test the performance of auto-tuning the replay buffer size, we measure the resident size of the memory (RES). RES accurately represents the amount of actual physical memory consumed by a process. Fig. 4a shows the image distance error and RES plots for different tasks. Two sets of hyperparameter values were used to compare performance. The first one is a redundancy (\(N_r=300,000\)) which led to the best performance metric. The second one is an insufficiency (\(N_r=1,000\)). In Fig. 4a we see that our algorithm auto-tuning the buffer size adaptively and converges to a RES near 6GB. Compared to the redundant buffer size, the performance metric with our auto-tuning agent is often indistinguishable. Critically however, our approach generally used half or less of the memory resources. For the insufficient replay buffer size, transitions were forgotten through the updating of the agent leading to suboptimal performance. From Fig. 4a (in the Appendix), we can see that size of the replay buffer quickly increased before converging to a size of 50,000. This behavior is indicative of the way the diversity in the sampling goals changed throughout the experiment: quickly increasing early in exploration because the VAEs model not learning well, then converging to near-steady value.

2) Auto-tuning of the number of policy gradient updates:

Here, we study the effects of tuning number of policy update \(N_\theta\). Unlike the previous setup, to directly compare

Fig. 1: Visualization of the simulated and real-robot environments. In both cases, the RGB frames are reshaped, flattened and normalized into 6912 x 1 as the state \( s \) for the RIG framework. The action is defined as the end-effector’s 3D positions and the gripper’s 1D binary control variable if used.

Fig. 2: Experimental results for the MNIST and Fashion-MNIST datasets with changing diversity. The plot shows values for -ELBO, \( \text{KL}[p_\theta(z|s)||q(z)] \), and \( -\mathbb{E}_{p_\theta(z|s)} \log p_\psi(s|z) \) averaged across more than 6 different seeds. Testing was conducted across three different combinations of \( \beta \) and \( z \) values: \((\beta=5,z=4)(\beta=1,z=4)(\beta=1,z=2)\). Note when the diversity of the training samples does not change, the KL-divergence value converges to equilibrium. If the diversity of the samples increases, \( \text{KL}[p_\theta(z|s)||q(z)] \) also increases but the \( \mathbb{E}_{p_\theta(z|s)} \log p_\psi(s|z) \) decreases and vice-versa.

Fig. 3: We compute -ELBO for visual robotic manipulation tasks with RIG. The diversity of the environment is modified by changing the physical workspace or the camera view angle. In (a), –ELBO increases when changes the camera view angle to capture more visual features of the robot arm in simulation. In (b) and (c), –ELBO increases when the workspace enlarged both in simulation and real world.

3) Auto-tuning of the number of exploration steps: Two sets of exploration values were used to compare performance. The redundant one used \( N_e=3,000 \) and the other one was \( N_e=10 \). Once again task-performance metrics was indistinguishable. On the other hand, as shown in Fig. 4b, our auto-

...
(a) Auto-tuning $N_e$ in different manipulation environments. We use the image distance between the initial state and the goal from the evaluation phase to compare across different setups.

(b) Auto-tuning $N_\theta$ in different manipulation environments. Performance metrics and total numbers of gradient updates for the whole learning process are shown.

(c) Auto-tuning $N_\alpha$ in different manipulation environments. Performance metrics and the total numbers of gradient updates for the whole learning process are shown.

Fig. 4: Ablative studies in the auto-tuning hyperparameters.

tuning algorithm only had to explore around 0.3, 0.5, and 0.3 times the number of exploration steps as baseline allowing it to optimize its policy much faster. As similar with auto-tuning the replay buffer size, auto-tuning adapted to the diversity of generated goals and produced a quick uptick in the number of exploration steps before it converged to steady-state.

C. Coverage performance of auto-tuning

The RIG framework is able to obtain all possible goals from the environment, not only from the task-relevant goal [4], [5]. In this experiment, we compare the goal-coverage performance of auto-tuning with the fixed value setups in the push environment. To this end, we fixed $N_{r, \theta, \alpha}$ to the redundant levels introduced in the previous experiments. We then sampled goals randomly from the environment to evaluate the obtained policy. From Fig. 5 we see that the auto-tuning method coverage is qualitatively similar to the baseline.

D. Real-world performance of auto-tuning

We compare use auto-tuning and just fixed $N_{r, \theta, \alpha}$ in real-robot. In Fig 6, compare with use fixed number, auto-tuning can tune $N_{r, \theta, \alpha}$ to achieve the better performance in response to different workspaces of arm without hyper-search. In smaller diversity environment, both auto-tuning and fixed setup are all learn equally fast. When the diversity of the environment increases by increasing the arm’s workspace, the −ELBO increased. For auto-tuning, $N_{r, \theta, \alpha}$ increase following and can learn faster than the fixed $N_{r, \theta, \alpha}$ setups when the environment become more diverse.

E. Auto-tuning in curriculum and multitask setups

We study the compound effect of combining the auto-tuning $N_{r, \theta, \alpha}$ under a task curriculum setup and a multitask setup (environment details in Tab. II of Appendix)

1) Curriculum Problem: We first set out to measure the performance gain of the combined auto-tuning under a curriculum setup. The latter is setup in such a way that we (programmatically in simulation and manual in real-robot experiments) change the environments’ visual diversity. The simulated curriculum begins with a reach task, followed by a push after 99 epochs, followed by a multi-object push after 199 epochs. On the other hand, the real-robot curriculum begins with reach, followed by a manual change in camera angle view after 20 epochs followed by the addition of objects that render the environment images more diverse after 30 epochs. Video of how the environment changes in the real-world environment is available on the project page.

Our results are visualized in Fig. 7 As with previous experiments, the −ELBO automatically changes in response to modifications in the environment prompted by the curriculum. After each epoch, $N_{r, \theta, \alpha}$ were calculated directly from −ELBO. It is evident that −ELBO consistently adapted to increases in visual diversity for both simulated and real-robot tasks.
2) Multitask Problem: In multitask setup, tasks are randomly initial in different epoch including Reach, Single Object Push, and Multi-object Push. Up-to-date, there is no clear notion of optimally setting the value of \( N_{r,\theta,e} \). As such, we wanted to measure the potential gains that arise from combining our optimal auto-tuning performance in significantly more challenging environments and compare against various sets of (fixed) hyperparameter combinations. Recall that the auto-tuning chooses the hyperparameters according to the diversity of the generating goals.

For our baseline, we create for combinations of fixed values for \( N_{r,\theta,e} \). We list them in the following format: \( \{ N_r, N_\theta, N_e \} \). We choose values that range from low values to high values, hyperparameter values are set as follows: \( \{ 1k, 100, 10 \} \), \( \{ 100k, 1k, 1k \} \), \( \{ 200k, 2k, 2k \} \), and \( \{ 300k, 3000, 3000 \} \). For auto-tuning, we use same hyperparameters set as ablative study. All different hyperparameters sets runs were conducted using the same Intel Xeon ES-2620 based processor with an NVIDIA Titan RTX GPU.

In Fig. 8 auto-tuning achieve the optimal performance directly, in terms of learning efficiency and time consuming.

V. RELATED WORK

A. Self-generative goal agents.

RIG’s the goal-generator resulted in agents learning a diverse set of tasks without a specified reward function in an RGB based MDP [4]–[6]. Such goal generators can be trained online [5]–[7] or offline [4]. This work studies online methods as offline methods are unable to recognize dynamic environment changes in future episodes. Also, self-generated goal agents can be easily combined with modern RL algorithms like PPO [7], TD3 [6], and SAC [5]. Nevertheless, previous works still use a fixed number of \( N_{r,\theta,e} \) in off-policy learning regardless of how diversity of the explored environment. Our work instead exploits the negative ELBO to tune \( N_{r,\theta,e} \) flexibly, resulting in a more efficient policy optimization procedure.

B. Goal-generator and information theory estimation.

Goal generators like VAEs are trained in an unsupervised manner [6]. VAEs can use information theory for estimation and optimization [19]–[21]. However, previous works analysed datasets with fixed diversity whose sample entropy did not change throughout the training [6], [12]. Instead, we estimate how a dataset’s changing diversity affects the ELBO of the VAEs.

C. Tuning hyperparameters in iterative learning process.

With regard to model optimization, a fixed number of iterative updates is commonly used [22]. The number of iterations is typically a fixed hyperparameter because the diversity of the samples not change [12] and few studies have learned to optimize the number. However, such fixed
Fig. 8: Auto-tuning result for the multitask setup across different combinations of fixed hyperparameters setups. Auto-tuning adapts $N_{r,\theta,e}$ sufficiently to obtain optimal performance in half the time.

parameterization is ineffective in learning robotic tasks where the diversity of the environment changes online. There has been much work in deterministic scheduling. Some decay the learning rate of gradient descent at every epoch [23]. Pong et al. [5] decrease the number of VAEs iterations according to a schedule. These methods empirically derived the schedule rather than learning it according to the varying diversity of an environment.

VI. CONCLUSION

Hyperparameter auto-tuning in self-supervised robot learning makes RIG easier to apply in practice to different domains with the auto-tuning of three hyperparameters $N_{r,\theta,e}$ instead of relying on costly hyperparameter optimization. Hyperparameters $N_{r,\theta,e}$ are common in various RL framework, so our proposition can be extended to various VAE-based RL algorithms. To choose adequate hyperparameter values $N_{r,\theta,e}$, we estimated and verified that the VAE’ loss function – ELBO has a lower bound related to the diversity of the training dataset. Moreover, the ELBO varies according to the change of diversity, which is so far un-studied. A nice property of our method is that it does not require any much additional computational cost, since it directly uses the metric already computed during the VAE evaluation phase in RIG. Experiments demonstrated that auto-tuning can save time and resources by adaptively selecting $N_{r,\theta,e}$ according to the diversity of goals of a given task. Designing deep RL algorithms with hyperparameters that do not need to be hand-tuned is an important step to make RL more practical. Future investigation to autotune other hyperparameters would be worthwhile.

ACKNOWLEDGMENT

This work is supported by Key R&D Program of Guangdong Province (Grant No. 2019B090915001, 2019A050510040), Guangzhou Basic and Applied Basic Research Project (Grant No. 202002030237), and NSFC grants (51975126, 51905105, 61950410758).

REFERENCES

[1] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meyer, “Deep reinforcement learning that matters,” arXiv preprint arXiv:1709.06560, 2018.

[2] “A closer look at deep policy gradients,” https://openreview.net/forum?id=rxyxEEHPS, accessed: 2020-10-14.

[3] T. Zahavy, Z. Xu, V. Veeriah, M. Hessel, J. Oh, H. van Hasselt, D. Silver, and S. Singh, “Self-tuning deep reinforcement learning,” arXiv preprint arXiv:2002.12928, 2020.

[4] A. Nair, S. Bahl, A. Khazatsky, V. Pong, G. Berseth, and S. Levine, “Contextual imagined goals for self-supervised robotic learning,” in International Conference on Machine Learning, 2019.

[5] V. H. Pong, M. Dalal, S. Lin, A. Nair, S. Bahl, and S. Levine, “Skew-fit: State-covering self-supervised reinforcement learning,” CoRR, 2019.

[6] A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine, “Visual reinforcement learning with imagined goals,” in Advances in Neural Information Processing Systems, 2018.

[7] C. Florensa, D. Held, X. Geng, and P. Abbeel, “Automatic goal generation for reinforcement learning agents,” in International Conference on Machine Learning, 2018.

[8] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, “Generative adversarial nets,” in Advances in Neural Information Processing Systems, 2014.

[9] M. Klissarov, R. Islam, K. Khetarpal, and D. Precup, “Variational state encoding as intrinsic motivation in reinforcement learning,” Presented at the Task-Agnostic Reinforcement Learning Workshop at Proceedings of the International Conference on Learning Representations, 2019.

[10] Y. Kim, W. Nam, H. Kim, J.-H. Kim, and G. Kim, “Curiosity-bottleneck: Exploration by distilling task-specific novelty,” in International Conference on Machine Learning, 2019.

[11] T. Schaul, D. Horgan, K. Gregor, and D. Silver, “Universal value function approximators,” in Proceedings of the 32nd International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, P. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 1312–1320. [Online]. Available: http://proceedings.mlr.press/v37/schaul15.html

[12] P. K. Diederik, M. Welling, et al., “Auto-encoding variational bayes,” in Proceedings of the International Conference on Learning Representations, 2014.

[13] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, “beta-vae: Learning basic visual concepts with a constrained variational framework,” Proceedings of the International Conference on Learning Representations, 2017.

[14] A. Alemi, B. Poole, I. Fischer, J. Dillon, R. A. Saurous, and K. Murphy, “Fixing a broken elbo,” in International Conference on Machine Learning, 2018.

[15] S. Kuliback, Information theory and statistics. Courier Corporation, 1997.

[16] C. P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner, “Understanding disentangling in beta-vae,” arXiv preprint arXiv:1804.03599, 2018.

[17] L. Deng, “The mnist database of handwritten digit images for machine learning research [best of the web],” IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 141–142, 2012.

[18] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,” arXiv preprint arXiv:1708.07747, 2017.

[19] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, “Infogan: Interpretable representation learning by information maximizing generative adversarial nets,” in Advances in Neural Information Processing Systems, 2016.

[20] B. Poole, S. Ozair, A. Van Den Oord, A. Alemi, and G. Tucker, “On variational bounds of mutual information,” in International Conference on Machine Learning, 2019.

[21] S. Zhao, J. Song, and S. Ermon, “Infovae: Balancing learning and inference in variational autoencoders,” in Proceedings of the AAAI Conference on Artificial Intelligence, 2019.

[22] C. D. Meyer, Matrix analysis and applied linear algebra. Siam, 2000.
[23] I. G. Goodfellow, Y. Bengio, and A. C. Courville, “Deep learning.” *Nature*, vol. 521, pp. 436–444, 2015.
APPENDIX

A. Theoretical Detail

The upper bound of MI:
\[
I(S; Z) = H(S) - H(S|Z) = H(Z) - H(Z|S)
\]
\[
= \mathbb{E}_{p(s,z)} \log \frac{p(z)}{p(s)}
\]
\[
= \mathbb{E}_{p(s,z)} \log \frac{p(z)q(z)}{q(z)p(z)}
\]
\[
= \mathbb{E}_{p(s,z)} \log \frac{p(z)q(z) - \mathbb{E}_{p(z)} [KL(p(z)||q(z))]}{q(z)}
\]
\[
\leq \mathbb{E}_{p(s,z)} \log \frac{p(z)}{q(z)}
\]

B. ELBO and the loss function of VAEs

The potential target of VAEs is to infer the intractable \(p(z|s)\) use \(p_\phi(z|s)\), measure with \(KL(p_\phi(z|s)||p(z|s))\). KL divergence metric tells us how different of \(p(z|s)\) and \(p_\phi(z|s)\) is.

\[
KL(p_\phi(z|s)||p(z|s))
\]
\[
= \mathbb{E}_{p(z|s)} \log \frac{p_\phi(z|s)}{p(z|s)}
\]
\[
= \mathbb{E}_{p(z|s)} \log p_\phi(z|s) - \mathbb{E}_{p(z|s)} \log p(z|s)
\]
\[
= \mathbb{E}_{p(z|s)} \log p_\phi(z|s) - \mathbb{E}_{p(z|s)} \log \left( \frac{p(s|z)p(z)}{p(s)} \right)
\]
\[
= \mathbb{E}_{p(z|s)} \left( \log p_\phi(z|s) - \log p(s|z) - \log p(z) + \log p(s) \right),
\]

where has:
\[
\mathbb{E}_{p(z|s)} \log p(s) - KL(p_\phi(z|s)||p(z|s))
\]
\[
= \mathbb{E}_{p(z|s)} \left( \log p(s|z) - \log p_\phi(z|s) + \log p(z) \right)
\]
\[
= \mathbb{E}_{p(z|s)} \log p(s|z) - KL[p_\phi(z|s)||\log p(z)].
\]

Because \(\mathbb{E}_{p(z|s)} \log p(s)\) is a fix negative value, we have:
\[
- KL(p_\phi(z|s)||p(z|s)) \leq \mathbb{E}_{p(z|s)} \log p(s|z) + KL[p_\phi(z|s)||p(z)].
\]

Finally, combine with approximation, we can have
\[
- \mathbb{E}_{p(s)} KL(p_\phi(z|s)||p(z|s))
\]
\[
\leq -\mathbb{E}_{p(s)}(\mathbb{E}_{p_\phi(z|s)} \log p_\phi(s|z) + KL[p_\phi(z|s)||q(z)])
\]
\[
= -ELBO.
\]

Minimize the KL-divergence \(KL(p_\phi(z|s)||p(z|s))\) is equal to maximize the ELBO: \(-\mathbb{E}_{p(s)}(\mathbb{E}_{p_\phi(z|s)} \log p_\phi(s|z) + KL[p_\phi(z|s)||q(z)])\). And the loss function of VAEs in Eq. \[2\] is the \(-\text{ELBO}\) that going to minimize the KL-divergence \(KL(p_\phi(z|s)||p(z|s))\) equally.

C. Environment Detail

In the simulation, the agent controls a 7-dof Sawyer arm in the MuJoCo environment. Two tasks are considered: (i) pushing single and multiple objects on a table and (ii) object pickup. Regarding the action space, simulated actions for the pickup task consist of motions along the YZ plane along with gripper control. Whilst for object push, the action space consists of motions in the XYZ plane.

The real-world experiments are conducted on an Elfin 6-dof arm and only included the reach task as shown in Fig. \[4\] The action space is XY. Different with the simulated environment, we programmatically wait 2.5 seconds to guarantee the action completion and allow the camera to obtain a stable image after publishing an action to robot.

D. Implementation Detail

Hyperparameter and environmental settings are shown in Tab. \[1\] We use the \(-\text{ELBO}\) statistics in last testing phase to create auto-tuning methodology without any additional computation cost across experiments. We use the samples from the replay buffer for testing the VAEs. Auto-tuning in different manipulation environments. Different colors represent different numbers of gradient updates. Performance metrics and completion times for the whole learning process are showing.

E. Additional Examples and Results

Examples of different observation` s sivity due to the different workspaces are showing in Fig. \[5\].
TABLE I: RIG hyperparameters for the visual robotic control task. The rest of the hyperparameters are the same as opensource core with the traditional RIG-SAC.

| Hyperparameters                      | Simulated Push | Simulated Pickup | Simulated Multi-Object Push | Real Reach |
|--------------------------------------|----------------|-----------------|-----------------------------|------------|
| Algorithm                            | SAC            |                 |                             |            |
| Q network hidden sizes               | 400,300        | 400,300         |                             |            |
| Policy network hidden sizes          | 400,300        |                 |                             |            |
| Q network and policy activation      | ReLU           |                 |                             |            |
| Exploration Noise                    | None           |                 |                             |            |
| RL Batch Size b                      | 1024           |                 | Auto-tuning, 50              |            |
| Discount Factor                      | 0.99           |                 | Auto-tuning, 200             |            |
| Max TimeSteps Until Done             | 50             |                 |                             |            |
| Exploration And Evaluation Steps     | Auto-tuning, 500| Auto-tuning     |                             |            |
| Steps to Initialize the Replay Buffer| 10000          | 100             |                             |            |
| Replay Buffer Size                   | Auto-tuning    |                 |                             |            |
| VAEs Batch Size                      | 64             |                 |                             |            |
| Latency Dimension Size               | 1              |                 |                             |            |
| Use Skew-Fit                         | None           |                 |                             |            |
| VAEs Training Schedule               | Always train with 500 steps | | | |
| VAEs Testing Epochs                  | Average of 10 epoch | | | |
| Encoder Hidden Sizes                 | Kernel sizes=(5,3,3), Channels=(16,32,64), Strides=(3,2,2) | | | |
| Decoder Hidden Sizes                 | Kernel Sizes=(3,3), Channels=(32,16), Strides=(2,2) | | | |
| Latest Decoder Activation            | Gaussian (Identity) | | | |
| Pretrained VAEs                      | None           |                 |                             |            |
| Observation Image Sizes              | 3×48×48        |                 |                             |            |
| Object Shape                         | Puck | Box (2.5×1.5×1.5, but use 3.5 ball as visual-mark) | Two Cylinders | Nothing |
| Arm Workspace                        | 10×10×50(cm³)  | 10×10×8(cm³)    | 40×10(cm³)                  | 190×100×5(cm³) |
| Action Workspace                     | 10×10×50(cm³)  | 10×10×8(cm³)    | 40×10(cm³)                  | 190×100×5(cm³) |
| Goal Space                           | 30×20(cm³)     | 10×10×8(cm³)    | 20×20×50(cm³)               | 190×100(cm³) |

TABLE II: Environment detail of curriculum setup and multitask setup experiments.

| Environment Details | Simulated Curriculum | Real Curriculum | Real Multitask |
|---------------------|----------------------|-----------------|---------------|
| Arm Workspace       | 10×10×50(cm³)        | 10×10×50(cm³)   | 190×100×5(cm³) |
| Action Workspace    | 10×10×50(cm³)        | 190×100×5(cm³)  | 190×100×5(cm³) |
| Goal Workspace      | 10×10(cm³)           | 190×100(cm³)    | 190×100(cm³)   |

Fig. 10: We use the image from RIG paper directly. The user-specified goal $g$ can be an image having a puck where in a desired position in the environment.

Fig. 11: The size of replay buffer choose from auto-tuning.