A ENVIRONMENT DETAILS

In this section, we introduce the details of the environments used in the experiments. We build all the Meta-RL environments by modifying the origin environments of OpenAI Gym[1]. We describe these environments as follows:

- Walker2d-params: This environment is built by modifying the OpenAI gym Walker2d, in which a two-dimensional two-legged robot should move forward. For each task, the dynamics parameters of the environment are randomly initialized. The horizon length is set as 200. For the experiment, we sample 40 training tasks and 10 test tasks.
- Cheetah-vel-ood: This environment is built by modifying the OpenAI gym Half Cheetah. In the modified environment, a 2-dimensional robot with nine links should achieve a target velocity running forward. For a training task, the target velocity is sampled uniformly from [0, 2.5]. For a test task, the target velocity is sampled uniformly from [2.5, 3.0]. The horizon length is set as 200. For the experiment, we sample 50 training tasks and 15 test tasks.
- Hopper-params: This environment is built by modifying the OpenAI gym Hopper, in which a two-dimensional one-legged robot should move forward. For each task, the dynamics parameters of the environment are randomly initialized. The horizon length is set as 200. For the experiment, we sample 40 training tasks and 10 test tasks.
- LunarLander-params: This environment is built by modifying the OpenAI gym Lunar Lander, in which a rocket should land on the landing pad. For each task, we randomly initialize the gravity. The horizon length is set as 1000. For the experiment, we sample 40 training tasks and 10 test tasks.
- InvDoublePend-params: This environment is built by modifying the OpenAI gym Inverted Double Pendulum. In this environment, there is a cart that can move linearly. A pole is fixed on it, and a second pole is fixed on the other end of the first pole. For each task, the dynamics parameters of the environment are randomly initialized. The horizon length is set as 200. For the experiment, we sample 40 training tasks and 10 test tasks.
- Cartpole-fl-ood: This environment is built by modifying the OpenAI gym Cart Pole[2]. In the environment, there is a cart that can move linearly with a pole fixed on it. For each task, we randomly initialize the force magnitude and the length of the pole. For a training task, the force magnitude and the length of the pole are sampled uniformly from $[7.5, 12.5]$ and $[0.3, 0.7]$. For a test task, the force magnitude and the length of the pole is sampled uniformly from $[5, 7.5] \cup [12.5, 15]$ and $[0.2, 0.3] \cup [0.7, 0.8]$. The horizon length is set as 200. For the experiment, we sample 40 training tasks and 10 test tasks.

B IMPLEMENTATION DETAILS

In this section, we first introduce the implementation details of the symbolic operators. Then we provide more details of training and the pseudo-code of the training process.

B.1 SYMBOLIC OPERATORS

To ensure the numerical stability of the proposed symbolic learning framework, we regularize the operators and employ a penalty term to keep the input from the "forbidden" area. We show the operators and the corresponding penalty terms as follows:

- Multiplying operator:

$$y = min(max(x_1, -100), 100) * min(max(x_2, -100), 100),$$

the penalty term can be formulated as:

$$L_{mul} = max(x_1 - 100, 0) + max(-100 - x_1, 0) + max(x_2 - 100, 0) + max(-100 - x_2, 0)$$

[1] We build our environments based on the random-param-env https://github.com/dennisl88/rand_param_envs.

[2] We use a continuous version from an open-source implementation https://gist.github.com/iandanforth.

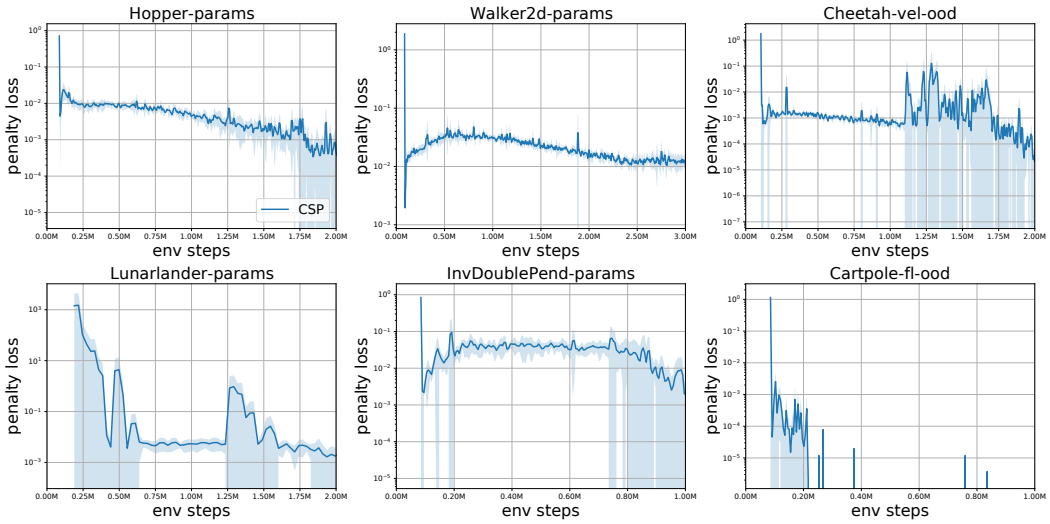

Figure 1: Learning curves of the penalty loss function. The shaded area spans one standard deviation.

- Division operator:

$$y = \begin{cases} 0, & x_2 < 0.01, \\ \dfrac{x_1}{x_2}, & x_2 \geq 0.01. \end{cases}$$

  the penalty term can be formulated as:

$$L_{div} = max(0.01 - x_2, 0)$$

- Sine operator: $y = sin(x)$, the penalty term is set as zero: $L_{sin} = 0$.
- Cosine operator: $y = cos(x)$, the penalty term is set as zero: $L_{cos} = 0$.
- Exponential operator:

$$y = exp(min(max(x, -10), 4)),$$

  the penalty term can be formulated as:

$$L_{exp} = max(x - 4, 0) + max(-10 - x, 0)$$

- Log operator:

$$y = log(max(x, 0.001)),$$

  the penalty term can be formulated as:

$$L_{log} = max(0.001 - x, 0)$$

- Identity operators: $y = x$, the penalty term is set as zero: $L_{identity} = 0$.
- Condition operator: $y = sigmoid(x_1) * x_2 + (1 - sigmoid(x_1)) * x_3$, the penalty ter is set as zero: $L_{condition} = 0$.

In practice, we only use the identity operator in the plain structure. For all the environments, we use the same symbolic network structure described 4.1. Especially, for Cheetah-vel-ood, we add one Condition operator in each layer. We think the Condition operator will be useful for environments where the reward function changes. During training, we involve a penalty loss function $\mathcal{L}_{penalty}$ which is the sum of the penalty terms of regularized operators in symbolic networks:

$$\mathcal{L}_{penalty}(\theta_\Phi, \theta_\Psi) = \sum_{i=1}^{i=M} \sum_{j=1}^{j=L} \sum_{k=1}^{k=N_j} \mathcal{L}_{g_{i,j,k}}(x_{i,j,k}), \tag{1}$$

Algorithm 1 The training process of CSP.

Input: Batch of training tasks $\{\kappa_i\}_{i=1,...m}$. The number of iterations of temperature and target L_0 norm schedule t_e. Target temperature τ_t. The target L_0 norm of the mask at the end of training l_e. The scale of the penalty loss α_1. The scale of the loss to regularize the sum of score α_2. The scale of the loss for the KL divergence β.

Initialize replay buffers $\mathcal{B}_i$ for each training tasks.
for training iteration $t = 0$ to $T - 1$ **do**
  $\tau = (1 - \tau_t) * (1 - \frac{min(t,t_e)}{t_e}) + \tau_t$
  for each task κ_i **do**
    Initialize context $c_i = \{\}$
    for $k = 0$ to $K - 1$ **do**
      Sample $z \sim q_{\theta_q}(z|c_i)$
      Generate symbolic policy S with $\Phi(z)$ and $\Psi(z, \tau)$
      Collect data with $a \sim N(S(s), F(s, z))$ and add to buffer $\mathcal{B}_i$.
      Update $c_i = \{(s_j, a_j, s'_j, r_j)\}_{j=1,\cdots,N} \sim \mathcal{B}_i$
    end for
  end for
  for steps in training step **do**
    Sample a batch of tasks.
    for each κ_i in the batch **do**
      Sample context c_i and RL Batch b_i from the buffer $\mathcal{B}_i$
      Sample context variables $z \sim q_{\theta_q}(z|c_i)$
      $l_{target} = l_e + (1 - l_e) * \left(1 - \frac{min(t,t_e)}{t_e}\right)^2$
      Calculate loss for the critic: $\mathcal{L}^i_{critic} = \mathcal{L}^{sac}_{critic}(z, b_i)$
      Calculate loss for the Actor:
      $\mathcal{L}^i_{actor} = \mathcal{L}^{sac}_{actor}(z, b_i) + \alpha_1 \mathcal{L}_{penalty}(z, b_i) + \alpha_2 \mathcal{L}_{select}(z, b_i, l_{target})$
      Calculate the KL divergence $\mathcal{L}^i_{KL} = D_{KL}(q_{\theta_q}(z|c_i)|N(0, 1))$
    end for
    Update the Critic with $\sum_{i=1}^{i=m} \mathcal{L}^i_{critic}$
    Update the context encoder with $\sum_{i=1}^{i=m} (\mathcal{L}^i_{critic} + \beta \mathcal{L}^i_{KL})$
    Update the path collector and the parameter generator with $\sum_{i=1}^{i=m} \mathcal{L}^i_{actor}$
  end for
end for

where θ_Φ, θ_Ψ is the parameters of the parameter generator and the path selector, M is the dimension of action, L is the number of layers in a symbolic network, N_j is the number of regularized operators in layer j, $x_{i,j,k}$ is the input of operator $g_{i,j,k}$. We show the learning curves of the penalty loss function in Figure 1. During the training process, for all environments, the penalty loss function remains on a very small order of magnitude, which indicates that most of the operators in the symbolic network work the same as the original unregularized operators.

B.2 Training Details

In practice, we build up our off-policy learning framework on top of the soft actor-critic algorithm (SAC)(Haarnoja et al., 2018) following PEARL. To construct a stochastic policy, we also employ a small neural network F to output the standard deviation. The neural network has two hidden layers with 64 hidden units. Note that this neural network is only used during training. During the evaluation, we only use the produced symbolic policy to infer the action. Besides, to limit the score s produced by $\Psi(z)$ in the range of $(0, 1)$, we employ the $sigmoid$ tanh function. We initialize the bias of the last layer in $\Psi(z)$ as 3.0. As $sigmoid(3.0) = 0.9526$, we will have most of the paths of the symbolic network active at the beginning of training and ensure that the input of the sigmoid function is not too large to prevent the gradient from disappearing during training. For the details of the training process, we give the pseudo-code in Algorithm 1.

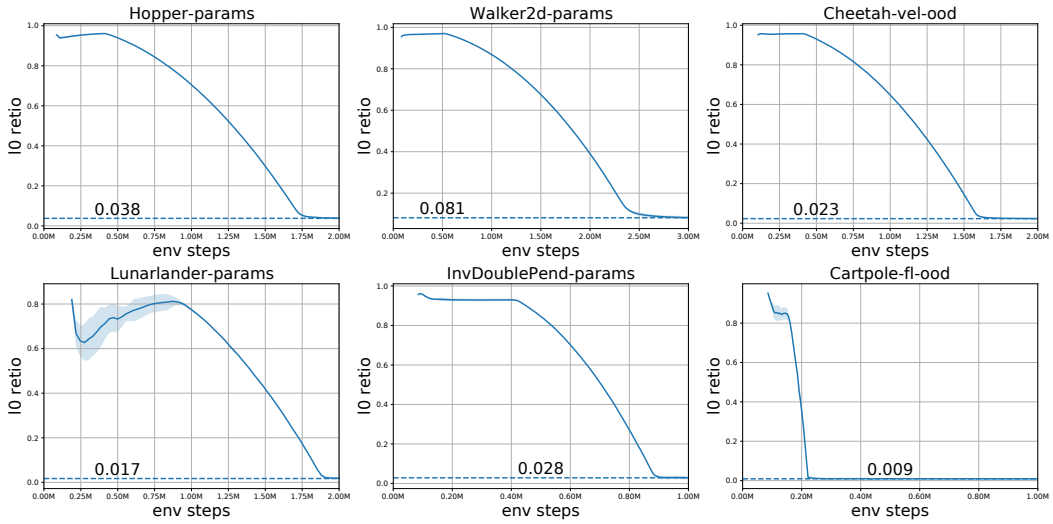

Figure 2: Learning curves of the average L_0 norm ratio of the mask for the sampled tasks.

C Experiment Details

C.1 Hyperparameters

In this section, we give the main hyperparameters for our experiments. We show the common hyperparameters of CSP in Table 1. We also list the environment specific hyperparameters in Table 2. The meta batchsize is the number of sampled tasks per training step. We set it according to the number of training tasks. For the scale of $\mathcal{L}_{select}$, we choose the best one from $\{0.1, 0.15, 0.2, 0.25, 0.5, 1.0, 2.0\}$. The target l_0 norm ratio is the ratio of the target L_0 norm of the mask at the end of training l_e to the number of parameters of the symbolic network. We set the value according to the complexity of the task and do not tune the value. We show the learning curves of the average L_0 norm ratio of the mask for the sampled tasks in Figure 2. The L_0 norm ratio at the end of training is always higher than the target l_0 norm ratio. Thus, the L_0 norm ratio at the end of training is more affected by the scale of $\mathcal{L}_{select}$. The schedule iterations mean the number of iterations of temperature and target L_0 norm schedule. We end the schedule near the end of training but for Cartpole-fl-ood in which the policy converges quickly, we reduce the number of schedule iterations. We run all the experiments with five random seeds and average the results to plot the learning curve.

C.2 Platform and Device

The implementation of our CSP is based on the pytorch(Paszke et al., 2019). Besides, we train the proposed CSP with Nvidia V100 GPU. When evaluating the inference time, we use Intel(R) Xeon(R) Gold 5218R @ 2.10GHz CPU.

D Experiments Results for Single Task RL

The gradient-based symbolic policy learning framework can also be used for single task reinforcement learning. For single task RL, we remove the parameter generator and the path selector. We define the parameters of symbolic network $\boldsymbol{w}$ and the score $\boldsymbol{s}$ as parameters that can be updated with gradient. The score $\boldsymbol{s}$ are all initialized as one and clipped to [0,1] after each training step. We compare the symbolic policy with the original neural network SAC and show the results in Figure 3. We also plot the learning curves of the average L_0 norm of the mask for the sampled tasks in Figure 4. The results show that the compact symbolic policy achieves comparable or higher performance compared with the neural network policy.

Table 1: Hyperparameters for the CSP.

Parameter	Value
optimizer	Adam(Kingma & Ba, 2015)
number of samples per minibatch	256
scale of the reward	5
learning rate	$3 \cdot 10^{-4}$
scale of the kl divergence loss	1
discount	0.99
sac target smoothing coefficient	0.005
target temperature	0.2
training steps per iteration	2000
scale of the penalty loss	1

Table 2: Environment Specific Hyperparameters.

Environment	Meta batchsize	Scale of $\mathcal{L}_{select}$	Target l_0 norm ratio	Schedule iterations
Walker2d-params	10	0.25	0.01	450
Hopper-params	10	0.25	0.01	300
InvDoublePend-params	10	2.0	0.01	150
Cartpole-fl-ood	10	0.25	0.002	25
Lunarlander-g	10	0.25	0.01	60
Cheetah-vel-ood	16	2.0	0.01	300

E DISCUSSION

In this paper, we propose to learn a contextual symbolic policy for Meta-RL. In our gradient-based learning framework, we train the contextual symbolic policy efficiently without any pre-trained model. For unseen tasks, the CSP can produce symbolic policies which achieve higher generalization performance. Besides, the compact symbolic policies are more efficient to be deployed and easier to understand compared with pure neural network policy. However, there are some limitations. For too complex tasks, CSP may generate complex symbolic policies, which may be hard to understand directly. A solving strategy is to learn a modularized symbolic policy and analyze each module to understand the whole policy. For tasks with high dimension observation like images, the CSP can not directly generate a symbolic policy. But we can employ a neural network to extract the environment variables and generate symbolic policy based on these environment variables. We leave these improvements talked about above in the future work.

REFERENCES

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

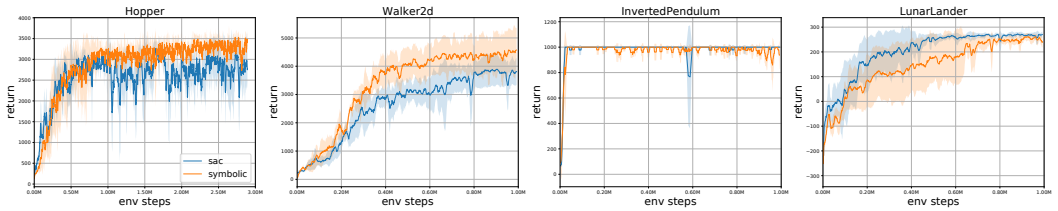

Figure 3: Comparison of the symbolic policy and the neural network policy.

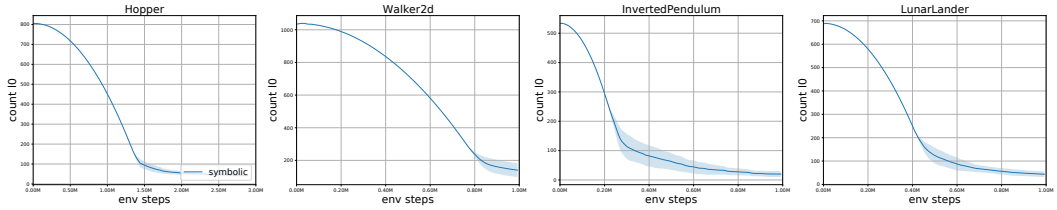

Figure 4: Learning curves of the average L_0 norm of the mask for the sampled tasks.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.