# AMORTIZED POSTERIOR ON LATENT VARIABLES IN GAUSSIAN PROCESS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Deep neural networks have achieved impressive performance on a variety of domains. However, performing tasks in partially observed, dynamic environments is still an open problem. Gaussian Process (GPs) is well-known for capturing uncertainty in model parameters. However, it simply assumes a fixed Gaussian prior on latent variables. Thus, agents are not able to update their beliefs about latent variables as observing data points. Instead, in this paper, we propose to replace the prior with an amortized posterior, which enables quick adaptation, especially to abrupt changes. Experiments show that our proposed method can adjust behaviors on the fly (*e.g.*, blind "Predator" take 56% more chance to approach "Prey"), correct mistakes to escape bad situations (*e.g.*, 25% $\uparrow$ on avoiding repeating to hit objects with negative rewards), and update beliefs quickly (*e.g.*, 9% faster convergence rate on learning new concepts).

## 1 INTRODUCTION

We have witnessed rapid progress in applying deep neural networks to a broad spectrum of tasks such as autonomous driving[7; 28], advertisement recommendation[1], and home assistant robot [18; 26; 13], etc. Without loss of generality, machine learning models can be expressed as $y = f(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta})$, where function $f$ is parameterized by $\boldsymbol{\theta}$, $\mathbf{x}$ is the input, $y$ is the output, and $\mathbf{z}$ is a vector of latent variables. Considering partially-observed navigation tasks, as illustrated in Fig.1, latent variables are (a) *where traps are?* (b) *which category is a treasure?* Moreover, (c) *where is the "Prey"?*. To enable quick adaptation, intelligent models have to capture the uncertainty in both

- Latent variable $\mathbf{z}$. For example, (a) *it is* $80\%$ *that the cell below is a trap*, (b) *the vase is less likely to be a treasure*, and (c) *feel like "Prey" has moved up. It is* $70\%$ *that it is on the right*
- Model parameter $\boldsymbol{\theta}$. Even though the models are confident of $\mathbf{z}$, say *It is* $95\%$ *that the cells on the right and left are traps*, they may still produce incorrect predictions if the situation is less likely seen before, such as the trap has never been placed in that cell.

It is worth noting that, instead of phase shift (train $\rightarrow$ test), the values of latent variables vary either cross episodes or over time steps, which makes quick adaption way more challenging.

Gaussian Process (GPs)[14] is well-known for capturing uncertainty, which specifies Gaussian priors on $\theta$ and $\mathbf{z}$. However, only the posterior $q(\boldsymbol{\theta}|\hat{X}, \hat{\mathbf{y}})$ is updated while $q(\mathbf{z}|\hat{X}, \hat{\mathbf{y}})$ is fixed at $p(\mathbf{z})$ during inference. Therefore, in this paper, we introduce Amortized Posterior on Latent Variables in Gaussian Process (APLV-GP) by replacing prior $p(\mathbf{z})$ with posterior $q(\mathbf{z}|\hat{X}, \hat{\mathbf{y}})$ (green box in Fig.2) which is then used to modulate the representation of data point $\mathbf{x}$ (purple box in Fig.2).

We evaluate our method to sinusoid regression, concept learning, and different reinforcement learning tasks in dynamic and partially observed environments. Experiment results show that our method consistently outperforms baselines. Furthermore, our method can adjust behaviors on the fly (*e.g.*, blind "Predator" take 56% more chance to approach "Prey"), correct mistakes to escape bad situations (*e.g.*, 25% $\uparrow$ on avoiding repeating to hit objects with negative rewards), and update beliefs quickly (*e.g.*, 9% faster convergence rate on learning new concepts).

Our contribution is two-fold. First, we plug the posterior of latent variables into the Gaussian Process, which results in less (or no) efforts to design, search or learn suitable kernels of the Gaussian Process.

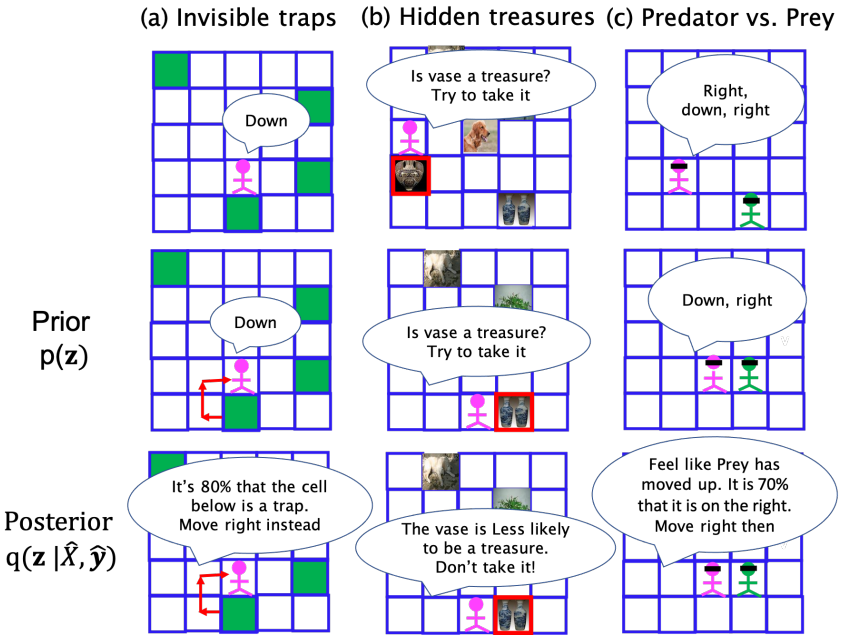



Figure 1: Partially observed navigation tasks. (a) Invisible traps. We randomly place traps (in green), which are invisible to agents, in a maze. Agents receive negative rewards if falling into a trap. (b) Hidden treasures. Objects are visible to agents, but the treasure category ("dog" or "vase"?) is unknown. They receive +1 rewards if taking a treasure, -1 otherwise. (c) "Predator" vs. "Prey". "Predator" and "Prey" are invisible to each other. They are only informed by how close they are (*e.g.*, two blocks away). Notice that the environment is re-initiated in every single episode. Given Hidden treasures as an example, agents perform on a series of episodes {("dog", "**vase**"), ("car", "**cat**"), ("snake", "**lizard**"), ("vase", "**dog**"), ... }, where categories in bold denote treasures. For ("dog", "vase"), "vase" is a treasure in one episode while "dog" becomes a treasure in another. In a particular episode, agents start from a state (1st row) and perform tasks based on the prior (2ed row) or the posterior (3rd row). Based on the prior, agents either repeatedly fall into the same trap or take objects which are not treasures. Moreover, "Predator" sticks to the previous route without being aware of "Prey"'s movement. Instead, performing on the posterior allows agents to walk around traps, avoid taking objects (not treasures), and redirect "Prey" timely.

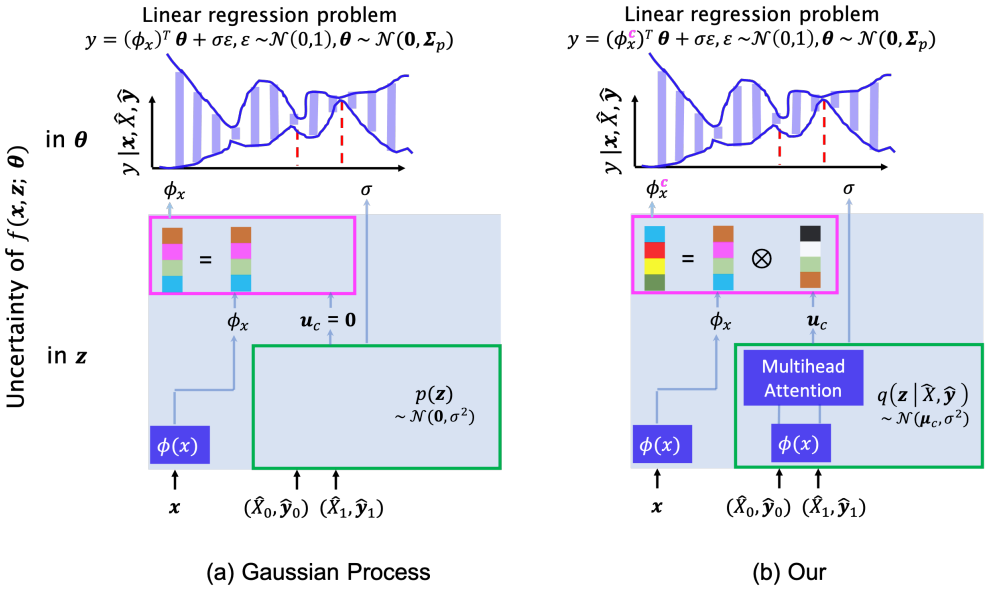
Experiments have proved that the RBF kernel is robust in solving real complex problems. Second, we propose a plug-and-play regression module, which is flexible to integrate into various algorithms. We demonstrate its effectiveness in Q-learning and A2C to solve navigation problems.

## 2 Related work

Our work can be categorized as a meta-learning algorithm using a stochastic process. A couple of works relate to our work in terms of capturing uncertainty, leveraging previous experience, and a variety of tricks to improve the capacity of the stochastic process.

**Probabilistic meta learning** MAML-based algorithms apply a variety of approximation strategies. For example, the gradient update in MAML[3] produces samples from the posterior distribution of parameters. However, it takes time to converge and introduces errors in the few-shot setting. Both Probabilistic MAML [4] and Ravi's work [23] attempt to find a surrogate distribution to approximate the posterior and then average models or using Maximum a Posterior (MAP) at test time. Bayesian MAML [11] attempt to find multiple, diverse MAML models based on Stein Variational Gradient Descent [16]. Hierarchical MAML [36] aims at learning the MAML model for each cluster of tasks. Thus, related tasks in the same cluster share the same model. However, our method is computationally efficient without running multiple models, and it applies exact uncertainty estimation.

**Context-based meta-RL.** Recurrent [2; 34] and recursive [19] meta-RL methods adapt to new tasks by encoding experience into a latent vector on which the policy is conditioned. However, they cannot reason about task uncertainty. Rakelly'work [22] aims to predict hidden task variables given contexts via minimizing the evidence lower bound (ELBO). However, it requires multiple full trajectories (>200 steps) and keeps the hidden variables fixed during each episode. Instead, our method relies




(a) Gaussian Process (b) Our

Figure 2: Overview of our method. To capture uncertainty in $\mathbf{z}$, (a) Gaussian Process specifies a Gaussian prior on latent variables $p(\mathbf{z}) \sim \mathcal{N}(\mathbf{0}, \sigma^2)$ which is constant w.r.t context set $c = \{\hat{X}, \hat{\mathbf{y}}\}$. Instead, (b) we specify a Gaussian posterior $q(\mathbf{z} \mid \hat{X}, \hat{\mathbf{y}}) \sim \mathcal{N}(\boldsymbol{\mu}_c, \sigma^2)$ where mean $\boldsymbol{\mu}_c$ is produced by a neural network which takes context set $c = \{\hat{X}, \hat{\mathbf{y}}\}$ as input. Then, we apply element-wise multiplication on $\phi_{\mathbf{x}}$ and $\boldsymbol{\mu}_c$, where feature $\phi(\mathbf{x})$ could be visual feature, the location of agent in a maze, etc, to produce context-based representation $\phi_{\mathbf{x}}^c$. To capture uncertainty in $\boldsymbol{\theta}$, we follow the same protocol with Gaussian Process according to Eqn.(3) and (4).

on exact uncertainty estimation and focuses on adaptation on a shorter time scale, say in $3 \sim 5$ time steps. Outside of RL, Matching Networks [32; 27] aim at finding nearest neighbors or prototypes. Relation network [29] proposes to classify which data point that a query matches. All these methods target solving discrete problems, *e.g.*, classification problems, and lack uncertainty estimation.

**Neural Process.** NPs[6; 5; 8] propose amortized models to estimate posterior mean and variance via deep neural networks. One advantage is that inference is faster since it only requires a single forward pass. However, the inference is approximated and lacks a theoretical guarantee. Moreover, NPs[10] are known to be underfitting to the context set and data-hungry. In addition, they are designed for supervised learning and do not apply to domains like reinforcement learning. Instead, we apply Bayesian Inference to ensure exact uncertainty estimation and resulting models are data-efficient and fit the observed data points well.

**Gaussian Process.** GPs[14] are probabilistic and data-efficient, which fit in fast adaptation well without suffering from intensive computation. However, it simply assumes a Gaussian prior on latent variables and keeps it fixed as agents observe data points. In addition, it takes effort to search for suitable kernels. Deep Kernel learning [35] attempts to learn more expressive kernels using neural networks. However, the performance is still far more behind the state-of-the-art method. In this paper, we replace the prior of latent variables with an amortized Gaussian posterior, which significantly reduces the efforts for kernel design. Even with a simple RBF kernel, our method can beat several baselines in solving real problems.

## 3 Approach

**Notation.** We let $\mathbf{x} \in \mathbb{R}^d$ denote the input vector, $y$ denote the target value. $X \in \mathbb{R}^{N \times d}$ is the data matrix where $X_i$ is the $i$th data point, and $\mathbf{y} \in \mathbb{R}^N$ is the target vector where $\mathbf{y}_i$ is the target of data point $X_i$. $X_{-i} \in \mathbb{R}^{(N-1) \times d}$ and $\mathbf{y}_{-i} \in \mathbb{R}^{(N-1)}$ are data matrix and target vector by removing the $i$th data point. Plus, $[;]$ denotes the concatenation of vectors and $\otimes$ denotes the element-wise multiplication of vectors.

### 3.1 Amortized Posterior on Latent Variables in Gaussian Process

Given context set $c = \{\hat{X}, \hat{\mathbf{y}}\}$, machine learning models can be expressed as $y = f(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta}), \mathbf{z} \sim q(\mathbf{z} \mid c)$, where function $f$ is parameterized by $\theta$ and $\mathbf{z}$ is a vector of latent variables. The posterior

$q\big(\mathbf{z} \mid c\big)$ is the probability density of $\mathbf{z}$ given context set $c$. When $c = \emptyset$, $q\big(\mathbf{z} \mid c\big) = p(\mathbf{z})$, where $p(\mathbf{z})$ is the prior. According to [12], we reparameterize the posterior as

$$\mathbf{z} = \boldsymbol{\mu}_c + \sigma\epsilon, \text{where } c = h\big(\phi(\hat{X}), \hat{\mathbf{y}}; \boldsymbol{\omega}\big), \epsilon \sim \mathcal{N}(0, 1) \tag{1}$$

where matrix $\phi(\hat{X})$ is the aggregation of columns $\phi_{\hat{X}_i}, \forall i$ and function $\phi(.)$ projects a input vector into feature space. $h(.; \boldsymbol{\omega})$ is a vector-valued function with parameters $\boldsymbol{\omega}$, which takes $\phi(\hat{X})$ and $\hat{\mathbf{y}}$ as inputs and outputs $\boldsymbol{\mu}_c$, the mean of the posterior. $\epsilon$ is Gaussian noise. Further, we plug Eqn.(1) into function $f$ in the following way

$$y = \underbrace{g\Big(\phi(\mathbf{x}), \boldsymbol{\mu}_c; \boldsymbol{\eta}\Big)^T}_{\phi^c_{\mathbf{x}}} \boldsymbol{\theta} + \sigma\epsilon \implies p(y \mid \mathbf{x}; \boldsymbol{\theta}) \sim \mathcal{N}\Big(\big(\phi^c_{\mathbf{x}}\big)^T \boldsymbol{\theta}, \sigma^2\Big) \tag{2}$$

where $g(., .; \boldsymbol{\eta})$ is a function that rewrites feature vector $\phi(\mathbf{x})$ as $\phi^c(\mathbf{x})$ by putting data point $\mathbf{x}$ in the context of $c = \{\hat{X}, \hat{\mathbf{y}}\}$, where we call $\phi^c(\mathbf{x})$ the context-based representation (detailed in Sec.3.2), and $\boldsymbol{\theta}$ is the weight vector.

For any data point $\{\hat{X}_i, \hat{\mathbf{y}}_i\}$ in context set $c$, we define it's own context set as $\{\hat{X}_{\text{-}i}, \hat{\mathbf{y}}_{\text{-}i}\}$. Then, $p(\hat{\mathbf{y}}_i \mid \hat{X}_i; \boldsymbol{\theta}) \sim \mathcal{N}\Big(\big(\phi^c_{\hat{X}_i}\big)^T \boldsymbol{\theta}, \sigma^2\Big)$. To capture the model uncertainty, we put a Gaussian prior on parameter $\boldsymbol{\theta} \sim p(\mathbf{0}, \boldsymbol{\Sigma}_p)$. Thus the target vector $\hat{\mathbf{y}}$ follows a joint Gaussian distribution

$$p(\hat{\mathbf{y}} \mid \hat{X}) \sim \mathcal{N}\Big(\mathbf{0}, \boldsymbol{K}_{\hat{X},\hat{X}} + \sigma^2\Big) \tag{3}$$

where $\boldsymbol{K}_{\hat{X},\hat{X}} = \big(\phi^c_{\hat{X}}\big)^T \boldsymbol{\Sigma}_p \big(\phi^c_{\hat{X}}\big)$ and matrix $\phi^c_{\hat{X}}$ is the aggregation of columns of $\phi^c_{\hat{X}_i}, \forall i$. In Bayesian paradigm, we average over all possible parameter values, weighted by their posterior probability (via Bayes's rule). Thus the predictive distribution becomes

$$\begin{aligned} p(y \mid \mathbf{x}, \hat{X}, \hat{\mathbf{y}}) &= \int_{\boldsymbol{\theta}} p(y \mid \mathbf{x}; \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \hat{X}, \hat{\mathbf{y}}) \mathrm{d}\boldsymbol{\theta} = \int_{\boldsymbol{\theta}} p(y \mid \mathbf{x}; \boldsymbol{\theta}) \frac{p(\hat{\mathbf{y}} \mid \hat{X}; \boldsymbol{\theta}) p(\boldsymbol{\theta})}{\int_\theta p(\hat{\mathbf{y}} \mid \hat{X}; \boldsymbol{\theta}) p(\boldsymbol{\theta})} \mathrm{d}\boldsymbol{\theta} \\ &\sim \mathcal{N}\Big(\boldsymbol{K}_{\mathbf{x},\hat{X}} \big(\boldsymbol{K}_{\hat{X},\hat{X}} + \sigma^2 \mathbf{I}\big)^{-1} \hat{\mathbf{y}}, \boldsymbol{K}_{\mathbf{x},\mathbf{x}} - \boldsymbol{K}_{\mathbf{x},\hat{X}} \big(\boldsymbol{K}_{\hat{X},\hat{X}} + \sigma^2 \mathbf{I}\big)^{-1} \boldsymbol{K}_{\hat{X},\mathbf{x}}\Big) \end{aligned} \tag{4}$$

with $\boldsymbol{K}_{\mathbf{x},\hat{X}} = \big(\phi^c_{\mathbf{x}}\big)^T \boldsymbol{\Sigma}_p \big(\phi^c_{\hat{X}}\big)$ and $\boldsymbol{K}_{\hat{X},\mathbf{x}} = \big(\phi^c_{\hat{X}}\big)^T \boldsymbol{\Sigma}_p \big(\phi^c_{\mathbf{x}}\big)$. According to Gaussian Process, we define kernel function $\boldsymbol{k}(\mathbf{x}, \mathbf{x}') = \psi\big(\phi^c_{\mathbf{x}}\big)^T \psi\big(\phi^c_{\mathbf{x}'}\big)$ where $\psi\big(\phi^c_{\mathbf{x}}\big) = \boldsymbol{\Sigma}_p^{1/2} \phi^c_{\mathbf{x}}$ in Eqn.(3), (4).

**Learning and inference.** During training, we find parameters $\boldsymbol{\omega}, \boldsymbol{\phi}, \boldsymbol{\eta}, , \sigma$ and $\boldsymbol{\Sigma}_p$ (or kernel function $\boldsymbol{k}$) that maximize the joint probability in Eqn.(3). At inference time, models start from $p(y \mid \mathbf{x})$ and adjust their predictions according to Eqn.(4) in a particular episode (or a period of time), as shown in Fig.2.

### 3.2 CONTEXT-BASED REPRESENTATION

The key idea of context-based representation in Eqn.(2) is to modulate feature $\phi(\mathbf{x})$ by the context set $c = \{\hat{X}, \hat{\mathbf{y}}\}$. Inspired by[33; 15; 24], as shown in Fig.2, we have

$$\phi^c_{\mathbf{x}} = \phi(\mathbf{x}) \otimes \boldsymbol{\mu}_c, \text{where } \boldsymbol{\mu}_c = \text{MultiheadAttention}\Big(\big[[\phi(\hat{X}_i); \hat{\mathbf{y}}_i]\big]_{i=1}^N\Big) \tag{5}$$

where function $h$ is parameterized by MultiheadAttention proposed by BERT [31; 17], which takes the sequence $\Big[[\phi(\hat{X}_i); \hat{\mathbf{y}}_i]\Big]_{i=1}^N$ as input and produces hidden vectors $\{\mathbf{h}_i\}_{i=1}^N$ via self-attention. We use the hidden vector after pooling layer as $\boldsymbol{\mu}_c$.

**A toy example.** Let us use a binary classification problem to demonstrate how context-based representation handles scenarios where hidden variables vary their values. We define function $\boldsymbol{\mu}_c = h(\hat{X}, \hat{\mathbf{y}}; \boldsymbol{\omega})$ and Sigmoid classifier as

$$(\boldsymbol{\mu}_c)_j = \begin{cases} 0, & \text{if } \phi(\hat{X})_{j,i} = \phi(\hat{X})_{j,k} \forall i \neq k \\ 1, & \text{if } \frac{\sum_{i, \hat{\mathbf{y}}_i = 1} \phi(\hat{X})_{j,i}}{|\{i, \hat{\mathbf{y}}_i = 1\}|} > 0 \\ -1 & \text{otherwise} \end{cases} \quad \text{and } y = \begin{cases} 1, & \text{if } \sigma\Big((\phi^c_{\mathbf{x}})^T \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\Big) > 0.5 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

and assume observed data points and test data points are

$$\phi(\hat{X}) = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & -2 \end{bmatrix}, \hat{\mathbf{y}} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \text{and } \phi(\mathbf{x}) = \begin{bmatrix} 1 \\ 0 \\ 3 \end{bmatrix} \tag{7}$$

By applying Eqn.(6), we have

$$\boldsymbol{\mu}_c = [0 \quad 0 \quad 1]^T, \phi^c_{\mathbf{x}} = \phi(\mathbf{x}) \otimes \boldsymbol{\mu}_c = [0 \quad 0 \quad 3]^T \implies y = 1 \tag{8}$$

Let us look at the following two scenarios:

• *Label flip.* In this case, hidden variable is which category is positive. In the context of Hidden treasures (in Fig.1), it means "dog" becomes the treasure. We flip labels, *i.e.*, $0 \to 1$ and $1 \to 0$. Then, $\hat{\mathbf{y}} = [0, 0, 1]^T$. Again, applying Eqn.(6), $\boldsymbol{\mu}_c = [0, 0, -1]^T$ and $\phi^c_{\mathbf{x}} = \phi(\mathbf{x}) \otimes \boldsymbol{\mu}_c = [0, 0, -3] \implies y = 0$, which means the label of test data $\mathbf{x}$ is flipped accordingly.

• *Domain shift.* In this case, hidden variable is domain. We assume the first two dimensions of feature $\phi(\mathbf{x})$ are domain features. In Eqn.(7), it is $[1 \quad 0]^T$. we simply change it to $[0 \quad 1]^T$. Thus we have

$$\phi(\hat{X}) = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & -2 \end{bmatrix} \text{ and } \phi(\mathbf{x}) = \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix}$$

Applying Eqn.(6) again, $\boldsymbol{\mu}_c = [0, 0, 1]^T$ and $\phi^c_{\mathbf{x}} = \phi(\mathbf{x}) \otimes \boldsymbol{\mu}_c = 3 \implies y = 1$, which shows that our method is robust to domain shift.

### 3.3 APPLICATIONS

In general, our method is applicable to any regression problems. In this section, we'll plug our method into reinforcement learning algorithms. First, we define state $\mathbf{s}$, action $\mathbf{a}$, next state $\mathbf{s}'$, $r(\mathbf{s}, \mathbf{a})$ as a reward of taking action $\mathbf{a}$ at state $\mathbf{s}$, and short-term memory $c = \{\hat{X}, \hat{\mathbf{y}}\}$. In replay buffer, a single data point is expressed as $\{\mathbf{s}, \mathbf{a}, \mathbf{s}, r, c\}$.

(i) **Off-policy.** Q-learning [20; 9] defines the optimal action-value function $Q(\mathbf{s}, \mathbf{a})$ as the maximum expected rewards by following any arbitrary policy, after observing state $\mathbf{s}$ and taking action $\mathbf{a}$. Q-learning solves a regression problem

$$\text{Minimize } \big(y - Q(\mathbf{s}, \mathbf{a})\big)^2, \text{ where } y = r(\mathbf{s}, \mathbf{a}) + \gamma * \max_a Q(\mathbf{s}', a) \tag{9}$$

We let $\mathbf{x} = [\mathbf{s}; \mathbf{a}]$. According to Eqn.(4), we define the stochastic policy as

$$\pi\big(\mathbf{a} \mid \mathbf{s}\big) = Softmax_{\mathbf{a}}\, Q(\mathbf{s}, \mathbf{a}), \text{ where } Q(\mathbf{s}, \mathbf{a}) \sim y \mid \mathbf{x} = [\mathbf{s}; \mathbf{a}], c = \{\hat{X}, \hat{\mathbf{y}}\} \tag{10}$$

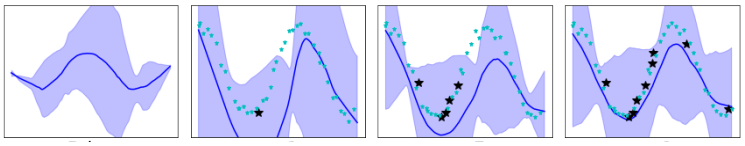
(ii) **On-policy** [21; 25]. It is comprised of two components: "Actor" and "Critic", where "Actor" updates the policy $\pi = p(\mathbf{a} \mid \mathbf{s})$ and "Critic" estimates the value function, such as $V(\mathbf{s})$ or $Q(\mathbf{s}, \mathbf{a})$, which represent accumulated rewards. To enable a quick reaction, we introduce a function $I(\mathbf{s})$, which takes state $\mathbf{s}$ as input and outputs a scalar in range $[0, 1]$, to represent the desire of reaching state $\mathbf{s}$.

$$\text{Minimize } \big(y - I(\mathbf{s})\big)^2, \text{ where } y = r(\tilde{\mathbf{s}}, \mathbf{a}),\ \tilde{\mathbf{s}}, \mathbf{a} \to \mathbf{s} \tag{11}$$

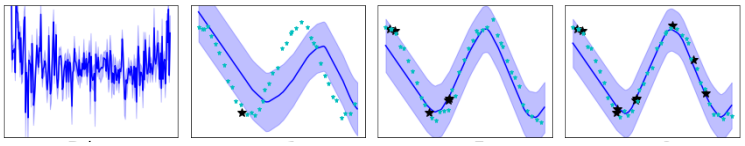
Inspired by [2; 34; 19], we plug $I(\mathbf{s})$ into "Actor". Applying Eqn.(4), we have

$$\pi\big(\mathbf{a} \mid \mathbf{s}, I(\mathcal{S})\big), \text{ where } I(\mathbf{s}) \sim y \mid \mathbf{x} = \mathbf{s}, c = \{\hat{X}, \hat{\mathbf{y}}\} \tag{12}$$

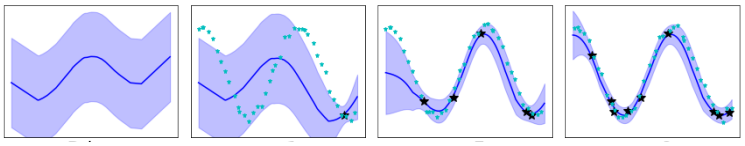
where $\mathcal{S}$ is the state space and $I(\mathcal{S})$ is a map of $I(\mathbf{s}), \forall \mathbf{s} \in \mathcal{S}$. For Invisible traps (in Fig.1), $I(\mathbf{s})$ is the probability of a cell not being a trap. And, for Hidden treasures (in Fig.1), $I(\mathbf{s})$ means the probability of a cell containing a treasure. As agents navigate in a maze, map $I(\mathcal{S})$ is updated accordingly and then guides the agents to decide which action to take. Note that $I(\mathbf{s})$ is complementary to value functions $V(\mathbf{s})$ and $Q(\mathbf{s}, \mathbf{a})$. "Actor" and "Critic" are jointly learned via standard training protocols.



(a) EMAML



(b) NPs



(c) GPs

Figure 3: Sinusoid Regression. Column 1: $y \mid \mathbf{x}$ and column 2-4: $y|\mathbf{x}, \hat{X}, \hat{\mathbf{y}}$. Black star denotes observed data points, the solid blue line denotes the mean of predictions, and cyan star denotes ground truth.

## 4 EXPERIMENTS

In this section, we aim at answering questions: Can agents (i) capture changes in environments and adjust behaviors accordingly? (ii) converge their beliefs faster as observing more data points? And (iii) fit observed data points well and jump out of bad situations? We compare our method with several baselines:

- `NPs`. Neural Processes (NPs) [6; 5; 8] build amortized models for $q(y \mid \mathbf{x}, \hat{X}, \hat{\mathbf{y}}) \sim \mathcal{N}(\mu_y, \sigma_y^2)$ where $\mu_y$ and $\sigma_y$ are predicted by neural networks. They are originally applied in supervised learning. Similar to Sec.3.3, we extend it to reinforcement learning.
- `EMAML`. We train 3 MAML [3] models with random seeds and average predictions at test time.
- `GPs` Gaussian Process with RBF kernel $\boldsymbol{k}(\mathbf{x}, \mathbf{x}') = \exp(-\frac{1}{2}\|\phi_{\mathbf{x}} - \phi_{\mathbf{x}'}\|^2)$. Instead, our method use $\boldsymbol{k}(\mathbf{x}, \mathbf{x}') = \exp(-\frac{1}{2}\|\phi^c_{\mathbf{x}} - \phi^c_{\mathbf{x}'}\|^2)$
- `DKL[35]`. Gaussian Process with the learnable kernel. The kernel function is parameterized by neural networks and learned end-to-end.
- `Non-Bayesian`. We re-weight the observed data points based on data similarity [1]
- `No Adaptation`. Agents perform tasks solely relying on $p(y \mid \mathbf{x})$ without adaptation.

### 4.1 SINUSOID REGRESSION

We mainly compare the basic properties of GPs, NPs, and EMAML on this toy dataset. The sinusoids have amplitude and phase uniformly sampled from range $[0.1, 5.0]$ and $[0, \pi]$. The input space is uniform on $[-5.0, 5.0]$. We further add Gaussian noise with zero mean and a standard deviation of $0.3$ to the target labels, consistent with MAML [3]. In each episode, we allow models to observe 20 data points and receive a single test data point at a time (up to 9). Notice that, in [3], test data points are presented at a time rather than in a sequence. To evaluate (and utilize) the uncertainty estimation, we let models pick the most uncertain data point to observe at a time.

In Fig.3, the 1st column shows $y \mid \mathbf{x}$. We can see that GPs and EMAML can model the sine-like function, while NPs can not. Column 2$\sim$4 show $y \mid \mathbf{x}, \hat{X}, \hat{\mathbf{y}}$. GPs and NPs can capture the uncertainty, while EMAML can not. Furthermore, GPs are more accurate and sensitive in terms of uncertainty estimation. It predicts low uncertainty around observed data points and high uncertainty over regions far away from observations. As observing more data points, the uncertainty converges quickly. On the other hand, NPs almost predict the same amount of uncertainty over the entire region. In addition, both EMAML and NPs under-fit observed data points. We summarize the findings in Table.1.

[1] $\boldsymbol{\mu}^{\tau}_{\mathbf{x}} = \mathbb{E}_{w_{\hat{y}}}[\hat{y}]$, $\boldsymbol{\Sigma}^{\tau}_{\mathbf{x}} = \mathbb{VAR}_{w_{\hat{y}}}[\hat{y}]$, where $w_{\hat{y}_i} = \exp\left(\boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\hat{X}_i)\right) / \sum_j \exp\left(\boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\hat{X}_j)\right)$

Table 1: Basic properties according to Fig.3

| | $y \mid \mathbf{x}$ | Uncertainty | | Observed data |
|---|---|---|---|---|
| | | Yes/No | Sensitivity | |
| EMAML NPs | ✓ | ✓ | | |
| GPs | ✓ | ✓ | ✓ | ✓ |

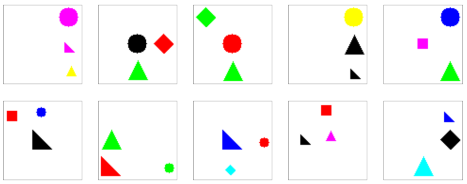


Figure 4: Concept learning. A concept defined by two constraints $c_0$: *"Triangle is on the bottom of Circle"* and $c_1$: *"Circle is large"*. Top row: positive samples and bottom row: negative samples. Clearly, if a model only captures $c_0$, it can correctly distinguish positive and negative samples. Similarly, models that capture $c_1$ or $c_0 \& c_1$ can do so as well.


(a) Observed data points (b) Training tasks (c) Model distribution

Figure 5: Results on Concept learning. Our method is sample efficient in terms of (a) the number of data points observed in each task and (b) the number of meta-training tasks. (c) Model distribution converges to the true distribution, *i.e.*, $p(c_0) \to 0.5$ and $p(c_1) \to 0.5$.

### 4.2 Concept Learning

In each episode, we take ten images as inputs, among which five images correctly represent the specified concept. Each image has nine cells with three objects located in three cells, and each object has three properties: color, shape, and size, sampled from the predefined vocabulary.

As shown in Fig.4, we use two constraints (relative location, color, or size) to define a concept. Positive examples should satisfy both of them, while negative examples satisfy neither of them. Thus, there are multiple plausible models, capturing $c_0$, $c_1$ or $(c_0 \& c_1)$, that can produce correct predictions.

Similar to Sec. 4.1, we start with two images and annotate the most uncertain image at a time. As shown in Table.2, our method outperforms all baselines. Fig.5a shows that our method can converge faster by leveraging fewer examples and reaches higher final accuracy. Fig.5b shows that our method is data-efficient in terms of the number of meta-training tasks. In Fig.5c, model distribution converges quickly, *i.e.*, $p(c_0) \to 0.5$ and $p(c_1) \to 0.5$, as observing more data points (see A.2).

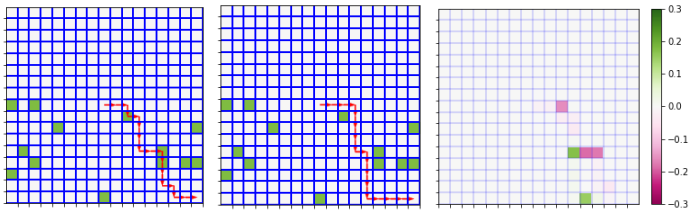
### 4.3 Reinforcement Learning: Navigation Tasks

#### 4.3.1 Random Traps

As explained in Fig.1, we randomly placed traps in each episode. The unknown, fixed goal is to move to the right corner of the grid. Therefore, agents have to guess where the traps are after a few attempts and infer the shortest path. We allow agents to take up to 30 steps in the first attempt and measure their performance in the 2ed attempt. The state is the agent's location, the action is moving "up", "down", "left" or "right", and the reward is +1 if reaching the goal, 0 otherwise.

We learn Q-function according to Eqn.(9) and use stochastic policy in Eqn.(10) to execute task. Experiments show that agents can obtain -0.02 reward at the 2ed attempt, conditioned on the 1st attempt, and perform significantly better than picking the best path over 20 attempts, -0.618 reward, (*i.e.*, completing the task using stochastic policy 20 times, each is a single attempt).

Fig.6 illustrates a qualitative example. After a single attempt, the agents can walk around traps and become more certain about which actions to take when revisiting the state.




(a) 1st attempt (b) 2ed attempt (c) $\Delta$ Entropy

Figure 6: Qualitative examples for random traps. Green cells are traps. $\Delta$ entropy means the change in $H(\mathbf{a} \mid \mathbf{s})$. "red" means $\Delta < 0$, where models become more confident about what action to take at the current state and "green" implies the opposite.

Table 2: Our vs. baselines. We report classification accuracy for concept learning. Move: the number of moves of perfect "prey", # Pos - # Neg: the difference in the numbers of positive and negative objects, and Obsessive hit: # hit -1, where # hit means how many times of repeating hitting a negative object.

| | Concept learning | "Predator"vs "Perfect" | | Hidden treasure | |
|---|---|---|---|---|---|
| | | Reward | Move (↑) | # Pos - # Neg | # Obsessive hit (↓) |
| EMAML | 0.58 | -0.41 | 3 | 3.56 | 0.73 |
| NPs | 0.79 | -0.39 | 16 | 0.26 | 1.38 |
| DKL | 0.67 | -0.69 | 8 | 0.1 | 1.60 |
| GPs | 0.5 | -0.35 | 24 | 0.0 | 1.68 |
| Non-Bayesian | 0.82 | -0.50 | 12 | 0.2 | 1.51 |
| No Adaptation | 0.5 | -1.0 | 2 | 0.1 | 1.6 |
| **Our** | **0.86** | **-0.34** | **25** | **5.5** | **0.55** |

### 4.3.2 HIDDEN TREASURES

As explained in Fig.1, in each episode, we randomly sample two categories from mini-imagenet dataset [24; 30], each category with 20 images. Then, we randomly place those images in the grid. The state is the agent's location, action is moving "up", "down", "left" or "right", and reward is the number of positive objects minus the number of negative objects. We follow the experimental setup in MAML [3] and pre-train function $I(\mathbf{s})$ in Eqn.(11) where the true target value is +1 if the object is a treasure, 0 otherwise. Then, we replace "Actor" in A2C [21] with Eqn.(12) and learn "Actor" and "Critic" jointly via the standard A2C training protocol, while keeping $I(\mathbf{s})$ fixed. Please check out video clips[2].

As shown in Table.2, our method beats all baselines. By averaging over 500 tasks, our method obtains +5.5 rewards. In addition, to evaluate whether agents can escape bad situations, *i.e.*, moving away from negative objects, we delete (observed) negative objects until agents hit the next negative object. We observe that our method has the lowest obsessive hit, which indicates that agents can adjust behavior on the fly and correct mistakes.

Fig.7 shows the change in function $I(\mathbf{s})$ over time. We can see that agents can quickly infer positive objects and walk towards them while avoiding hitting negative objects.

### 4.3.3 "PREDATOR" VS. "PREY"

The goal of "Predator" is to chase "Prey" while "Prey" attempts to run away from it. As demonstrated in Fig.1, "Predator" and "Prey" are invisible to each other. The state is the agent's location, the reward is inversely proportional to their distance, and the action is moving "up", "down", "left", or "right". Moreover, we restrict a limited short-term memory (say, 3 time steps) to store their recent states, actions, and rewards.

We learn Q-function according to Eqn.(9) and use stochastic policy in Eqn.(10) to execute task. We evaluate our method in the "Predator" vs. "Perfect" setting where only "Predator" is end-to-end learnable while "Prey" is manually pre-designed and always takes correct moves. For example, we let "Predator" move 100 steps and "Prey" only move when "Predator" is a single step away. In Table.2, we report averaged rewards and number of moves of perfect "Prey". We can see that our method beats all baselines with $> 15\%$ relative improvement. In addition, there is $56\%$ more chance (w.r.t. NPs) that 'Predator" is successfully a single step away from "Prey". We further upgrade the game

[2] HT.clip0.mov, HT.clip1.mov

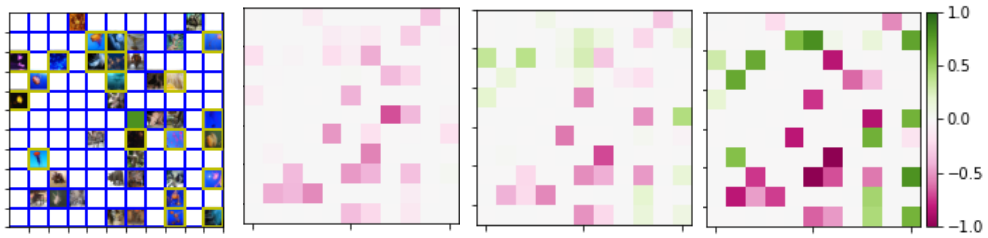



Figure 7: $I(\mathbf{s})$. The first column is the environment with two categories, each has 20 images. The positive objects are highlighted in yellow edges. Column 2∼4 show how map $I(\mathcal{S})$ changes as agents gather more information. From step 0 to step 2, agents collect two positive objects at $(6, 6)$, $(5, 3)$. Based on the gathered information, $I(\mathbf{s})$ is updated accordingly. With the updated $I(\mathbf{s})$, agents collect two more positive objects at $(2, 4)$, $(2, 5)$ without hitting negative objects.

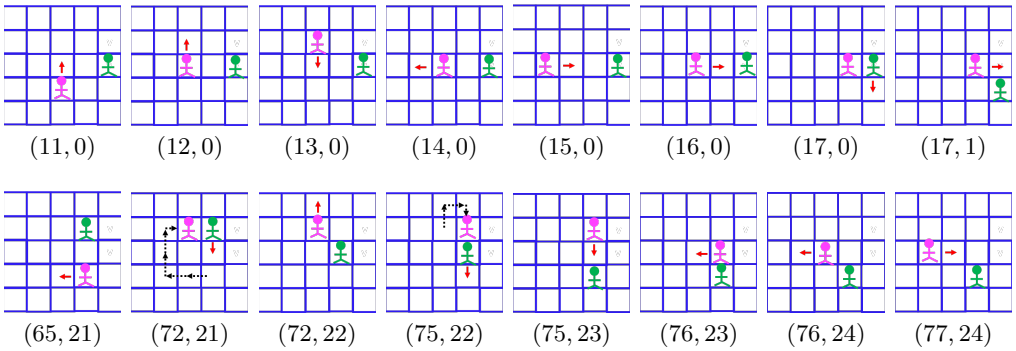



Figure 8: Qualitative examples for "Predator" vs. "Prey". In this setting, both agents are learned end-to-end. We sample 2 sequences of consecutive moves $(11, 0) \rightarrow (17, 1)$ and $(65, 21) \rightarrow (77, 24)$ where $(x, y)$ indicates the number of moves of "Predator" (green) and "Prey" (red). To save space, we skip several moves, *e.g.*, $(31, 7) \rightarrow (36, 7)$, but plot the last three moves with a lighter color to display the trajectory. In addition, we zoom in on the grid map and only plot $5 \times 5$ cells. Note that, in this setting, we also restrict "Prey" to move only when "Predator" is a single step away.

to "Predator" vs. "Prey" where both agents are learned end-to-end. Similar to what we observed in "Predator" vs. "Perfect", "Predator" gets a $-0.34$ reward on average and has a $99\%$ chance to make correct moves. "Prey" takes 30 moves in total and $76\%$ is correct. Please check out video clips[3]

Fig.8 illustrates two sequences of consecutive moves in "Predator" vs. "Prey" setting. (1) $(11, 0) \rightarrow (17, 1)$, "Predator" first moves $up$ until $(13, 0)$ where the reward drops and then moves back. At the moment, "Predator" is uncertain about which side ($right$ or $left$) "Prey" is on. It simply attempts to turn $left$. After realizing the reward drops, it moves back and turns $right$. (2) $(65, 21) \rightarrow (72, 21)$, "Predator" receives higher rewards by moving $up$ and $right$. Then, it continues to move $up$ after "Prey" moved $down$. However, with only a single wrong move, "Predator" immediately turns $right$ and moves $down$. Clearly, our agents are able to reduce ambiguity via exploration (Eqn.10) and quickly react to the dynamic environment.

## 5 Conclusion

We proposed a plug-and-play regression module to handle dynamic, partially observed environments, flexible, data-efficient, and applicable to large-scale, real problems. Our model can capture uncertainty, absorb knowledge and react to changing environments quickly. Further research can leverage a massive amount of off-policy data and extend our module to "high-level" abstracted spaces for continuous, high-dimensional reinforcement learning tasks. Furthermore, We can improve our models by learning to utilize uncertainty estimation, *e.g.*, learnable exploration in RL tasks.

[3] Prey.clip0.mov, Prey.clip1.mov, Prey.clip2.mov

## REFERENCES


Norris I Bruce, BPS Murthi, and Ram C Rao. A dynamic model for digital advertising: The effects of creative format, message content, and targeting on engagement. *Journal of marketing research*, 54(2):202–218, 2017.

Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl 2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1126–1135. JMLR. org, 2017.

Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*, pp. 9516–9527, 2018.

Marta Garnelo, Dan Rosenbaum, Chris J Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo J Rezende, and SM Eslami. Conditional neural processes. *arXiv preprint arXiv:1807.01613*, 2018a.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018b.

Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361. IEEE, 2012.

Jonathan Gordon, Wessel P Bruinsma, Andrew YK Foong, James Requeima, Yann Dubois, and Richard E Turner. Convolutional conditional neural processes. *arXiv preprint arXiv:1910.13556*, 2019.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, pp. 1352–1361. PMLR, 2017.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. *arXiv preprint arXiv:1901.05761*, 2019.

Taesup Kim, Jaesik Yoon, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. Bayesian model-agnostic meta-learning. *arXiv preprint arXiv:1806.03836*, 2018.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017.

Malte Kuss and Carl E Rasmussen. Gaussian processes in reinforcement learning. In *Advances in neural information processing systems*, pp. 751–758, 2004.

Jimmy Lei Ba, Kevin Swersky, Sanja Fidler, et al. Predicting deep zero-shot convolutional neural networks using textual descriptions. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4247–4255, 2015.

Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. In *Advances in neural information processing systems*, pp. 2378–2386, 2016.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PMLR, 2016.

Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pp. 5331–5340. PMLR, 2019.

Sachin Ravi and Alex Beatson. Amortized bayesian meta-learning. 2018.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Bokui Shen, Fei Xia, Chengshu Li, Roberto Martın-Martın, Linxi Fan, Guanzhi Wang, Shyamal Buch, Claudia D'Arpino, Sanjana Srivastava, Lyne P Tchapmi, Kent Vainio, Li Fei-Fei, and Silvio Savarese. igibson, a simulation environment for interactive tasks in large realistic scenes. *arXiv preprint*, 2020.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pp. 4077–4087, 2017.

Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1199–1208, 2018.

Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Utku Evci, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, et al. Meta-dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pp. 3630–3638, 2016.

Risto Vuorio, Shao-Hua Sun, Hexiang Hu, and Joseph J Lim. Multimodal model-agnostic meta-learning via task-aware modulation. *arXiv preprint arXiv:1910.13616*, 2019.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial intelligence and statistics*, pp. 370–378, 2016.

Huaxiu Yao, Ying Wei, Junzhou Huang, and Zhenhui Li. Hierarchically structured meta-learning. In *International Conference on Machine Learning*, pp. 7045–7054. PMLR, 2019.

# A Appendix

## A.1 Application: multi-class Logistic regression

We reformulate it as a linear regression problem.

$$\text{Minimize } \big(r(\mathbf{s},c) - S(\mathbf{s},c)\big)^2, \text{where } S(\mathbf{x},c) = \boldsymbol{\phi}(\mathbf{x})^T\mathbf{w} + z \tag{13}$$

where, $r(\mathbf{x},c) = 1.0$ if $c$ is the true category, otherwise 0. At test time, we pick the category with the highest probability

$$y = \text{argmax}_c \frac{\exp\big(S(\mathbf{x},c)\big)}{\sum_{c'}\exp\big(S(\mathbf{x},c')\big)}, \text{ where } S(\mathbf{x},c) \sim y \mid \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(M_x), M_y \tag{14}$$

## A.2 Concept learning

### A.2.1 Evaluation setup

At test time, we maintain two sets: labeled images $S_o$ and un-labeled images $S_u$. We start with two images and annotate the most uncertain image in $S_u$ at each time, according to $\text{argmax}_{\mathbf{x}\in S_u} \big|p(S(\mathbf{x},c) < 0.5) - p(S(\mathbf{x},c) > 0.5)\big|$, where $S(\mathbf{x},c)$ is the posterior score at $c = 1$ for un-labeled image $\mathbf{x}$, as defined in Eqn.(14). We evaluate 200 tasks that are never seen during meta-training and report averaged classification accuracy.

### A.2.2 Model distribution.

In Concept learning, we use two constraints to define a concept: $\mathbf{c}_0\&\mathbf{c}_1$ as positive, $\neg\mathbf{c}_0\&\neg\mathbf{c}_1$ as negative. Thus, either $\mathbf{c}_0$ or $\mathbf{c}_1$ can explain the data. We define the model distribution as

$$P(\mathbf{c}_j) = s(\mathbf{c}_j)/\sum_j s(\mathbf{c}_j),\ s(\mathbf{c}_j) = \sum_{i=\mathbf{c}_j} P(_i) \tag{15}$$

$P(_i)$ is the probability of image $_i$ being positive (i.e., softmax output) and $_i = \mathbf{c}_j$ means image $_i$ satisfies constraint $\mathbf{c}_j$. We generate 150 images: 50 for $\mathbf{c}_0\&\neg\mathbf{c}_1$, 50 for $\neg\mathbf{c}_0\&\mathbf{c}_1$ and 50 for $\neg\mathbf{c}_0\&\neg\mathbf{c}_1$. We report averaged results over 200 tasks. The real distribution is $P(\mathbf{c}_0)$ = 0.5, $P(\mathbf{c}_1)$ = 0.5, else 0, as shown in Fig.5c