A Survey on Meta-RL

Jiaqi Xi

im.sisixi@pku.edu.cn

December 21, 2020

Outline

Meta-RL

Why is deep RL slow?[1]

Sample efficiency: the amount of data required for a learning system to attain any chosen target level of performance.

1. Incremental parameter adjustment.
   To maximize generalization and avoid overwriting earlier learning, step-sizes in learning are usually small.
2. Weak inductive bias.
   Bias-variance trade-off: A stronger bias requires less data, while a weaker bias(sample-inefficient) can matter a wider range of patterns.

[1] M. Botvinick, S. Ritter, J. X. Wang, *et al.*, "Reinforcement Learning, Fast and Slow",, vol. 23, no. 5, 2019. DOI: 10.1016/j.tics.2019.02.006. [Online]. Available: http://creativecommons.org/licenses/by/4.0/.

Episodic Memory

- Train: Keep **an explicit record of past events**, and use this record directly as a point of reference in making new decisions.
- Test: Compare an internal representation of the current situation with stored representations of past situations. The action chosen is the one associated with the highest value.

MetaRL

- Meta: An ‘outer loop’ that uses its experiences over many task contexts to gradually adjust parameters that govern the operation of an ‘inner loop’, so that the inner loop can adjust rapidly to new tasks.
- MetaRL: Both the inner and outer loop implement RL algorithms, learning from reward outcomes and optimizing toward behaviors that yield maximal reward.

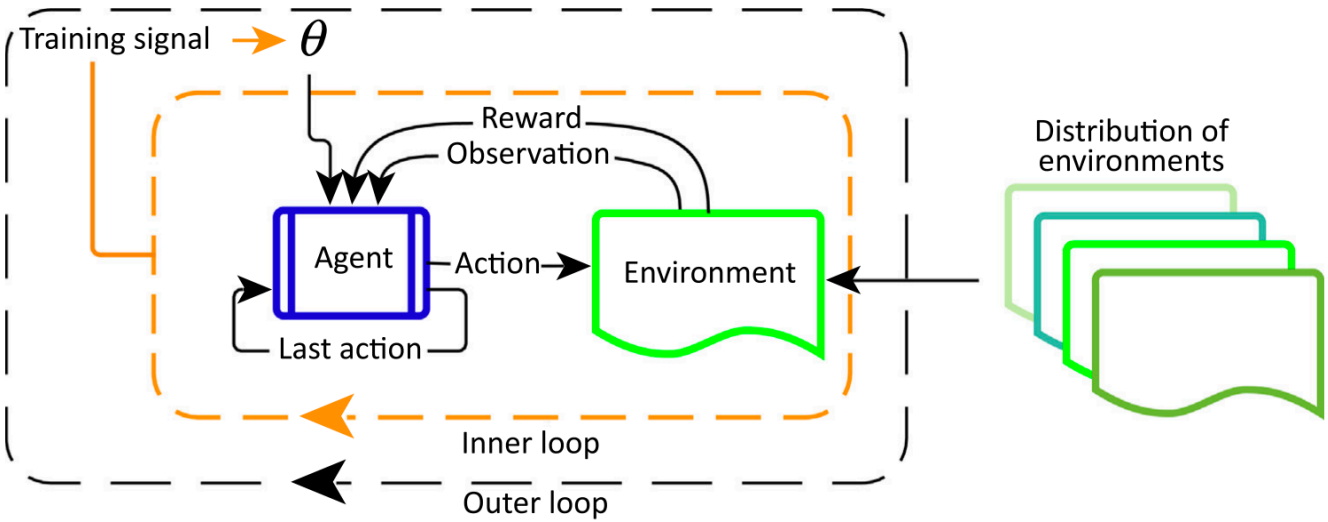


Figure 1: The outer loop aims at learning strong inductive biases that enable the inner loop to rapidly solve novel tasks.

Fast Learning is enabled by slow learning.

Slow learning is

- In Episodic Memory, the gradual learning of useful state representations to compute state similarity.
- In MetaRL, the dynamics of inner recurrent network slowly updated across tasks.

Gaps between human and AI systems

- Rich task environments.
- Some inductive biases of human are acquired through evolution and genetically, and others are acquired through learning.
- How do AI systems explore and seek information?
- Does AI techniques like gradient descent learning respond to some human mechanisms?

Early idea[2]

- In RL: $\pi_\theta(s_t) \rightarrow$ a distribution over actions
- In meta-RL: $\pi_\theta(a_{t-1}, r_{t-1}, s_t) \rightarrow$ a distribution over actions

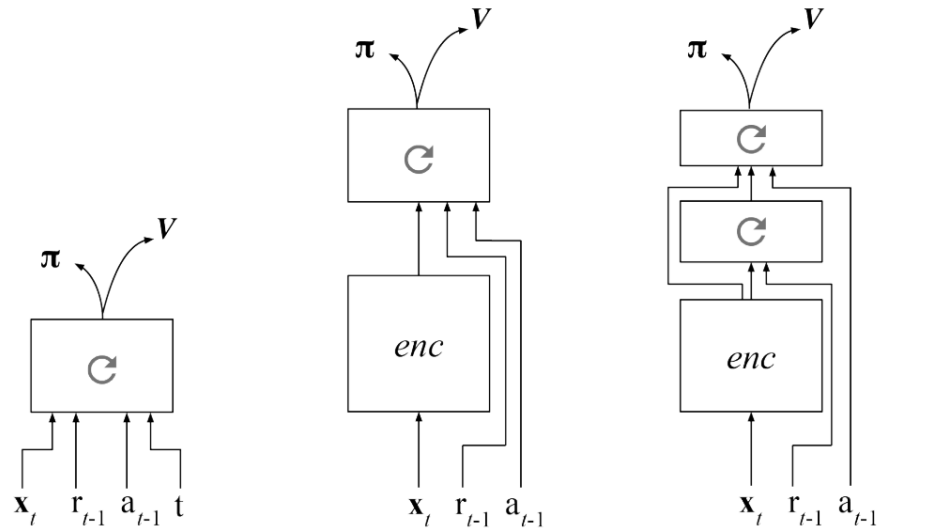


Figure 2: Learn the trajectory by LSTM. The observation is either a one-hot vector or an embedding vector. π = policy, V = value function.

[2] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, *et al.*, "Learning to reinforcement learn",, 2016. arXiv: 1611.05763. [Online]. Available: http://arxiv.org/abs/1611.05763.

Early model: RL2[3]

- Key idea: Take past trajectories as input and cast learning an RL algorithm as a reinforcement learning problem.
- Optimization goal: Maximize the expected total discounted reward accumulated across a fixed MDP.
  How? Preserve the hidden state to the next episode, but not t.he next trial.
- Use Gated Recurrent Units (GRUs) to avoid vanishing and exploding gradients.

[3] Y. Duan, J. Schulman, X. Chen, *et al.*, "FAST REINFORCEMENT LEARNING VIA SLOW REINFORCEMENT LEARNING", Tech. Rep., 2016. arXiv: 1611.02779v2.

Early model: RL2

- Input: Embedding of $< s, a, r, d >$, d is the determination flag.
- Output: A distribution over actions.

$$h_{t+1}, < s_{t+1}, a_t, r_t, d_t > \rightarrow h_{t+2}, a_{t+1}$$

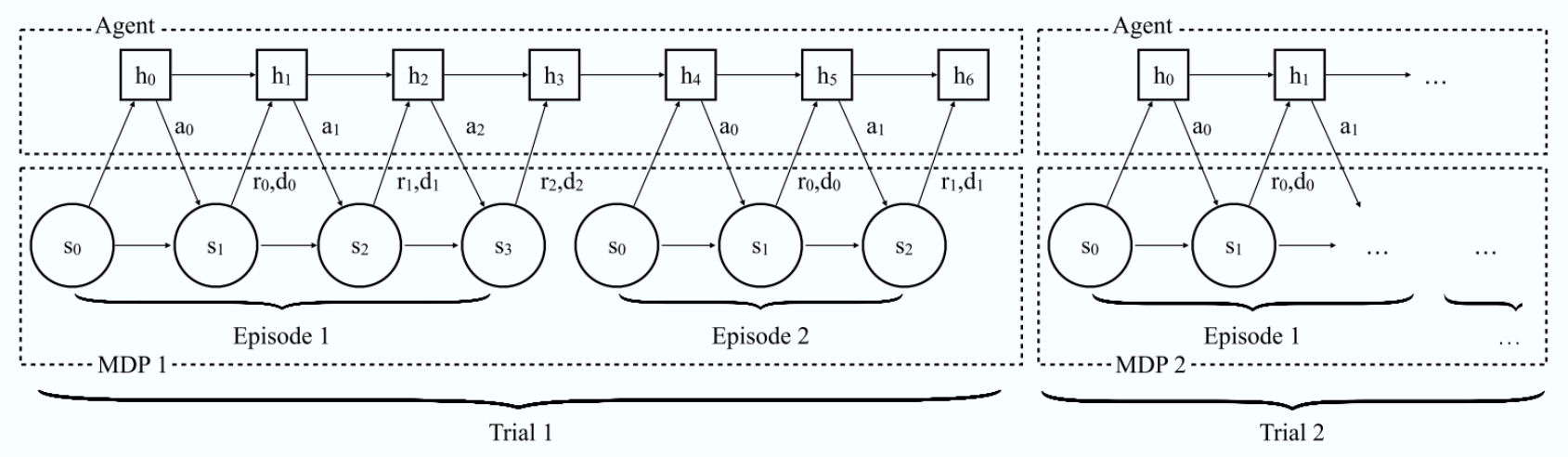


Figure 3: A trial consists of a series of episodes of interaction with a fixed MDP.

Meta-RL

MetaRL Obejective

Consider a family of MDPs $\mathcal{M} = \{M_i\}_{i=1}^{N}$ which comprise a distribution of tasks.
The goal of meta RL is to find a policy $\pi\theta$ and paired update method U such that, when $M_i \sim M$ is sampled, $\pi_{U(\theta)}$ solves M_i quickly.

$$\min_{\theta} \sum_{M_i} \mathbb{E}_{\pi_{U(\theta)}} \left[\mathcal{L}_{M_i}\right]$$

Wavenet + soft attention[4]

RNNs only take last timestep as input, what about the whole sequence?

Wavenet (Temporal convolutions)

- Pros: **High bandwidth** access to past timesteps rather than keeping a linearly dependent hidden state: $p(\mathbf{x}) = \prod_{t=1}^{T} p\left(x_t \mid x_1, \ldots, x_{t-1}\right)$.
- Cons: Coarser access to inputs that are further back in time.

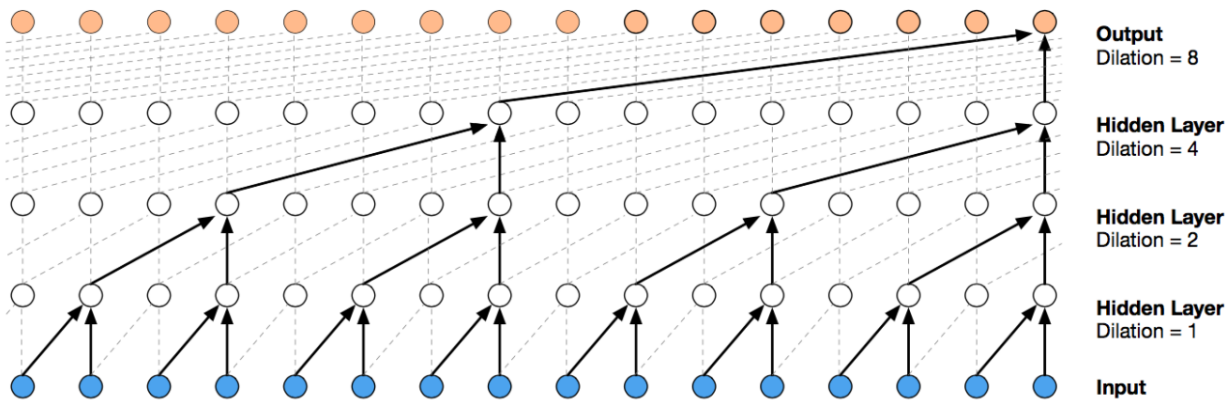


Figure 4: Dilated causal convolutions.

[4] N. Mishra, M. Rohaninejad, X. Chen, *et al.*, "A Simple Neural Attentive Meta-Learner", *arXiv*, vol. abs/1707.0, 14 pp.–14 pp. 2017. arXiv: 1707.03141. [Online]. Available: http://arxiv.org/abs/1707.03141.

Wavenet + soft attention

How to help Wavenet focus on earlier timesteps?

Soft attention

- Pros: **Pinpoint** a specific piece of information from a potentially infinitely-large context, which is viewed as an unordered key-value store.
- Cons: Lack of positional dependence, fail to take the sequential advantages of RL.

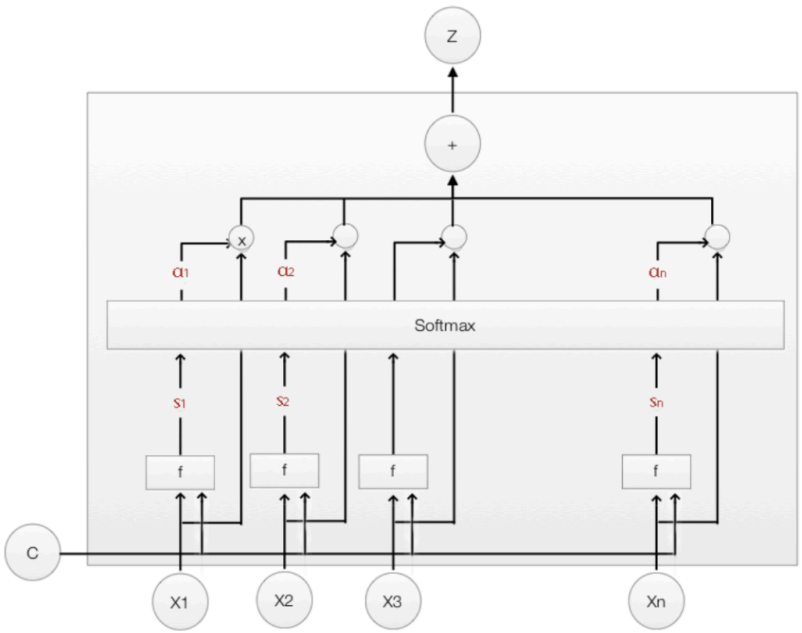


Figure 5: Take weighted addition of x as the input of LSTM.

Wavenet + soft attention

Why and how to combine Wavenet and soft attention?
Use temporal convolutions to produce the context over which we use a causal attention operation.

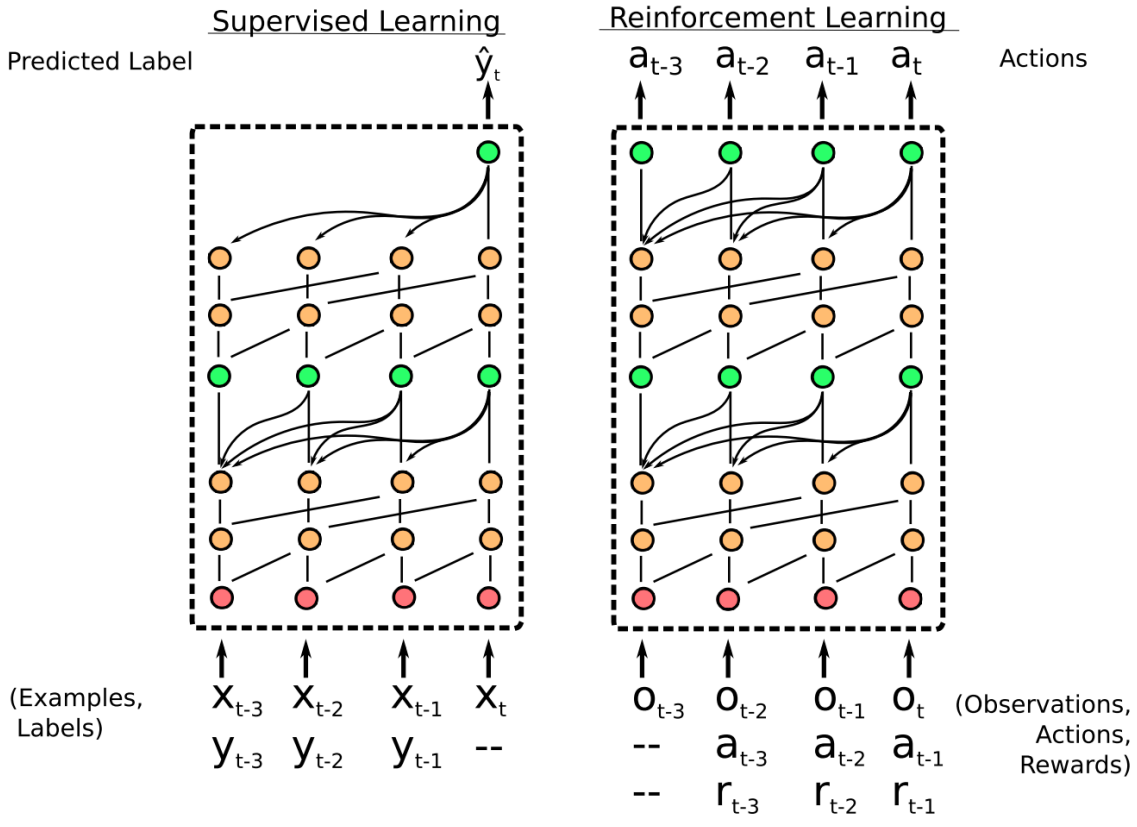


Figure 6: Orange: TC layers; Green: Causal attention layers.

Meta-loss: MAML

- Key idea: Some internal representations are more transferable than others. Try to encourage such general-purpose representations.
- How? Maximizing the **sensitivity of the loss functions** of new tasks with respect to the parameters. Namely, small changes in the parameters will produce large improvements on the loss function of any task.
- While optimizing θ, minimize the loss of updated parameter θ' so that the velocity towards this task is maximized.

$$\min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}\left(f_{\theta_i'}\right) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}\left(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)}\right)$$

Meta-loss: MAML

Equation 4: $\mathcal{L}_{\mathcal{T}_i}(f_\phi) = -\mathbb{E}_{\mathbf{x}_t, \mathbf{a}_t \sim f_\phi, q_{\tau_i}} \left[\sum_{t=1}^{H} R_i(\mathbf{x}_t, \mathbf{a}_t) \right]$

Algorithm 3 MAML for Reinforcement Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Sample K trajectories $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using f_θ in $\mathcal{T}_i$
6: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 4
7: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
8: Sample trajectories $\mathcal{D}_i' = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using $f_{\theta_i'}$ in $\mathcal{T}_i$
9: **end for**
10: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$ using each $\mathcal{D}_i'$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 4
11: **end while**

Figure 7: Suitable for all models that update with gradient descent.

Meta-loss: Evolved policy gradients[5]

- Inner loop: RL trains on the loss function given by the outer loop.

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\tau \sim \mathcal{M}, \pi_{\boldsymbol{\theta}}} \left[L_{\boldsymbol{\phi}} \left(\pi_{\boldsymbol{\theta}}, \tau \right) \right]$$

- Outer loop: ES produce a learned loss that can train agents faster with higher reward.

$$\phi^* = \arg\max_{\phi} \mathbb{E}_{\mathcal{M} \sim \boldsymbol{p}(\mathcal{M})} \mathbb{E}_{\tau \sim \mathcal{M}, \pi_\theta} \left[R_\tau \right]$$

- Key idea: Encode history trajectories in the design of loss function

[5] R. Houthooft, R. Y. Chen, P. Isola, *et al.*, "Evolved policy gradients", *Advances in Neural Information Processing Systems*, vol. 2018-Decem, no. 21, pp. 5400–5409, 2018, ISSN: 10495258. arXiv: 1802.04821.

Meta-loss: Evolved policy gradients

Outer loop update (ES):

- Why ES? We do not have an analytical expression of the reward of inner loop.
- Sample returns of V MDPs to avoid consistently choosing MDPs that always generate higher returns.

$$\phi \leftarrow \phi + \delta_{\text{out}} \cdot \frac{1}{V\sigma} \sum_{v=1}^{V} F\left(\phi + \sigma \epsilon_v\right) \epsilon_v$$

$$F\left(\phi + \sigma \epsilon_v\right) = \frac{R_{(v-1)*W/V+1} + \cdots + R_{v*W/V}}{W/V}$$

Algorithm 1: Evolved Policy Gradients (EPG)

[Outer Loop] for epoch $e = 1, \ldots, E$ **do**
  Sample $\boldsymbol{\epsilon}_v \sim \mathcal{N}(\mathbf{0}, \mathrm{I})$ and calculate the loss parameter $\boldsymbol{\phi} + \sigma \boldsymbol{\epsilon}_v$ for $v = 1, \ldots, V$
  Each worker $w = 1, \ldots, W$ gets assigned noise vector $\lceil wV/W \rceil$ as $\boldsymbol{\epsilon}_w$
  for each worker $w = 1, \ldots, W$ **do**
    Sample MDP $\mathcal{M}_w \sim p(\mathcal{M})$
    Initialize buffer with N zero tuples
    Initialize policy parameter $\boldsymbol{\theta}$ randomly
    [Inner Loop] for step $t = 1, \ldots, U$ **do**
      Sample initial state $s_t \sim p_0$ if $\mathcal{M}_w$ needs to be reset
      Sample action $a_t \sim \pi_{\boldsymbol{\theta}}(\cdot|s_t)$
      Take action a_t in $\mathcal{M}_w$ and receive r_t, s_{t+1}, and termination flag d_t
      Add tuple (s_t, a_t, r_t, d_t) to buffer
      if $t \bmod M = 0$ **then**
        With loss parameter $\boldsymbol{\phi} + \sigma \boldsymbol{\epsilon}_w$, calculate losses L_i for steps $i = t - M, \ldots, t$ using buffer tuples $i - N, \ldots, i$
        Sample minibatches mb from last M steps shuffled, compute $L_{\text{mb}} = \sum_{j \in \text{mb}} L_j$, and update the policy parameter $\boldsymbol{\theta}$ and memory parameter (Eq. (6))
    In $\mathcal{M}_w$, using the trained policy $\pi_{\boldsymbol{\theta}}$, sample several trajectories and compute the mean return R_w
  Update the loss parameter $\boldsymbol{\phi}$ (Eq. (7))
Output: Loss $L_{\boldsymbol{\phi}}$ that trains π from scratch according to the inner loop scheme, on MDPs from $p(\mathcal{M})$

Figure 8: Inner: gradient descent

Meta-reward: Inverse RL[6]

- Key idea: Learn a prior to infer possible reward functions from demonstrations of new tasks.
- Motivation: Few-shot IRL needs prior indicating relevant features between tasks. How to automatically learn the prior?
- Maximum Entropy Inverse RL

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{T}}(\boldsymbol{\theta}) = \frac{\partial r_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} \left[\mathbb{E}_{\tau} \left[\boldsymbol{\mu}_{\tau} \right] - \boldsymbol{\mu}_{\mathcal{D}\mathcal{T}} \right]$$

State Visitations: The expected number of times an agent will visit each state.

[6] K. Xu, E. Ratner, A. Dragan, *et al.*, "Learning a Prior over Intent via Meta-Inverse Reinforcement Learning", *36th International Conference on Machine Learning, ICML 2019*, vol. 2019-June, pp. 12 036–12 050, 2018. arXiv: 1805.12573. [Online]. Available: http://arxiv.org/abs/1805.12573.

Meta-reward: Inverse RL

Algorithm 1 Meta Reward and Intention Learning (MandRIL)

Input: Set of meta-training tasks $\{\mathcal{T}\}^{\text{meta-train}}$
Input: hyperparameters α, β
function MAXENTIRL-GRAD($r_{\boldsymbol{\theta}}, \mathcal{T}, \mathcal{D}$)
  # Compute state visitations of demos
  $\boldsymbol{\mu}_{\mathcal{D}} =$ STATE-VISITATIONS-TRAJ($\mathcal{T}, \mathcal{D}$)
  # Compute Max-Ent state visitations
  $\mathbb{E}_{\tau}[\boldsymbol{\mu}_{\tau}] =$ STATE-VISITATIONS-POLICY($r_{\boldsymbol{\theta}}, \mathcal{T}$)
  # MaxEntIRL gradient (Ziebart et al., 2008)
  $\frac{\partial \mathcal{L}}{\partial r_{\boldsymbol{\theta}}} = \mathbb{E}_{\tau}[\boldsymbol{\mu}_{\tau}] - \boldsymbol{\mu}_{\mathcal{D}}$
  Return $\frac{\partial \mathcal{L}}{\partial r_{\boldsymbol{\theta}}}$
end function

Figure 9: MaxEntIRL-Grad

Randomly initialize $\boldsymbol{\theta}$
while not done **do**
  Sample batch of tasks $\mathcal{T}_i \sim \{\mathcal{T}\}^{\text{meta-train}}$
  for all $\mathcal{T}_i$ **do**
    Sample demos $\mathcal{D}^{\text{tr}} = \{\tau_1, \ldots, \tau_K\} \sim \mathcal{T}_i$
    # Inner loss computation
    $\frac{\partial \mathcal{L}^{\text{tr}}_{\mathcal{T}_i}(\boldsymbol{\theta})}{\partial r_{\boldsymbol{\theta}}} =$ MAXENTIRL-GRAD($r_{\boldsymbol{\theta}}, \mathcal{T}_i, \mathcal{D}^{\text{tr}}$)
    Compute $\nabla_{\boldsymbol{\theta}} \mathcal{L}^{\text{tr}}_{\mathcal{T}_i}(\boldsymbol{\theta})$ from $\frac{\partial \mathcal{L}^{\text{tr}}_{\mathcal{T}_i}(\boldsymbol{\theta})}{\partial r_{\boldsymbol{\theta}}}$
    Compute $\boldsymbol{\phi}_{\mathcal{T}_i} = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\text{tr}}_{\mathcal{T}_i}(\boldsymbol{\theta})$
    Sample demos $\mathcal{D}^{\text{test}} = \{\tau'_1, \ldots, \tau'_{K'}\} \sim \mathcal{T}_i$
    # Outer loss computation
    $\frac{\partial \mathcal{L}^{\text{test}}_{\mathcal{T}_i}}{\partial r_{\boldsymbol{\theta}}} =$ MAXENTIRL-GRAD($r_{\boldsymbol{\phi}_{\mathcal{T}_i}}, \mathcal{T}_i, \mathcal{D}^{\text{test}}$))
    # Compute meta-gradient
    Compute $\nabla_{\boldsymbol{\theta}} \mathcal{L}^{\text{test}}_{\mathcal{T}_i}$ from $\frac{\partial \mathcal{L}^{\text{test}}_{\mathcal{T}_i}}{\partial r_{\boldsymbol{\theta}}}$ via chain rule
  end for
  Compute update to $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \beta \sum_i \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\text{test}}_{\mathcal{T}_i}$
end while

Figure 10: MAML using MaxEntIRL-Grad instead of gradient descent.

Meta-reward: Parameterized return[7]

- Key idea: Learn the return function by treating it as a parametric function with tunable meta-parameters η, including the discount factor γ, and the bootstrapping parameter λ.

$$g_\eta(\tau_t) = R_{t+1} + \gamma(1-\lambda)v_\theta(S_{t+1}) + \gamma\lambda g_\eta(\tau_{t+1})$$

- Optimization goal: Minimize MSE between value function and $g_\eta(\tau)$:

$$J(\tau, \theta, \eta) = (g_\eta(\tau) - v_\theta(S))^2$$

- 1. maximize return; 2. approximate value function; 3. policy regularization:

$$-\frac{\partial J(\tau,\theta,\eta)}{\partial\theta} = (g_\eta(\tau) - v_\theta(S))\frac{\partial \log \pi_\theta(A \mid S)}{\partial\theta} + b\,(g_\eta(\tau) - v_\theta(S))\frac{\partial v_\theta(S)}{\partial\theta} + c\frac{\partial H(\pi_\theta(\cdot \mid S))}{\partial\theta}$$

[7] Z. Xu, H. van Hasselt, and D. Silver, "Meta-Gradient Reinforcement Learning", *Advances in Neural Information Processing Systems*, vol. 2018-Decem, pp. 2396–2407, 2018. arXiv: 1805.09801. [Online]. Available: http://arxiv.org/abs/1805.09801.

Meta-reward: Parameterized return

1. Update the parameters θ to θ' on a sample of experience τ by $\frac{\partial J(\tau,\theta,\eta)}{\partial \theta}$.
2. Update meta-parameters η according to the gradient of the meta-objective:

$$\Delta\eta = -\beta \frac{\partial J'\left(\tau', \theta', \eta'\right)}{\partial \theta'} z'$$

$$z' = \mu z + \frac{\partial f(\tau, \theta, \eta)}{\partial \eta} \approx \frac{d\theta}{d\eta}$$

$$\frac{\partial f(\tau, \theta, \eta)}{\partial \eta} = \alpha \frac{\partial g_\eta(\tau)}{\partial \eta} \left[\frac{\partial \log \pi_\theta(A \mid S)}{\partial \theta} + c \frac{\partial v_\theta(S)}{\partial \theta} \right]$$

$$\frac{\partial J'\left(\tau', \theta', \eta'\right)}{\partial \theta'} = \left(g_{\eta'}\left(\tau'\right) - v_{\theta'}\left(S'\right)\right) \frac{\partial \log \pi_{\theta'}\left(A' \mid S'\right)}{\partial \theta'}$$

Meta-RL

Memory Table[8]

Keep a table $Q^{EC}(s, a)$ that contains the highest return ever obtained by taking action a from state s.

- Update at the end of each episode:

$$Q^{\mathrm{EC}}(s_t, a_t) \leftarrow \begin{cases} R_t & \text{if } (s_t, a_t) \notin Q^{\mathrm{EC}} \\ \max\left\{Q^{\mathrm{EC}}(s_t, a_t), R_t\right\} & \text{otherwise} \end{cases}$$

- When we need to use the table,

$$\widehat{Q^{\mathrm{EC}}}(s, a) = \begin{cases} \frac{1}{k}\sum_{i=1}^{k} Q^{\mathrm{EC}}\left(s^{(i)}, a\right) & \text{if } (s, a) \notin Q^{\mathrm{EC}} \\ Q^{\mathrm{EC}}(s, a) & \text{otherwise} \end{cases}$$

- Use VAE to extract salient features of the state space.
- Use LRU to fix the memory size.

[8] C. Blundell, B. Uria, A. Pritzel, *et al.*, "Model-Free Episodic Control",, 2016. arXiv: 1606.04460. [Online]. Available: http://arxiv.org/abs/1606.04460.

Memory Table

Algorithm 1 Model-Free Episodic Control.

for each episode **do**
  for $t = 1, 2, 3, \ldots, T$ **do**
    Receive observation o_t from environment.
    Let $s_t = \phi(o_t)$.
    Estimate return for each action a via (2)
    Let $a_t = \arg\max_a \widehat{Q^{\mathrm{EC}}}(s_t, a)$
    Take action a_t, receive reward r_{t+1}
  end for
  for $t = T, T-1, \ldots, 1$ **do**
    Update $Q^{\mathrm{EC}}(s_t, a_t)$ using R_t according to (1).
  end for
end for

Figure 11: Take action according to $\widehat{Q^{\mathrm{EC}}}$ and update it after a whole episode.

Episodic MetaRL[9][10]

- Meta Learning: Capitalizing on shared structure to learn faster with each new task.
- Episodic Memory: Avoid exploration when a task reoccurs.
- LSTM design: Adding a reinstatement gate to balance the weight of memory table DND.
- DND design: Take context embedding as the key and the final cell state as the value.

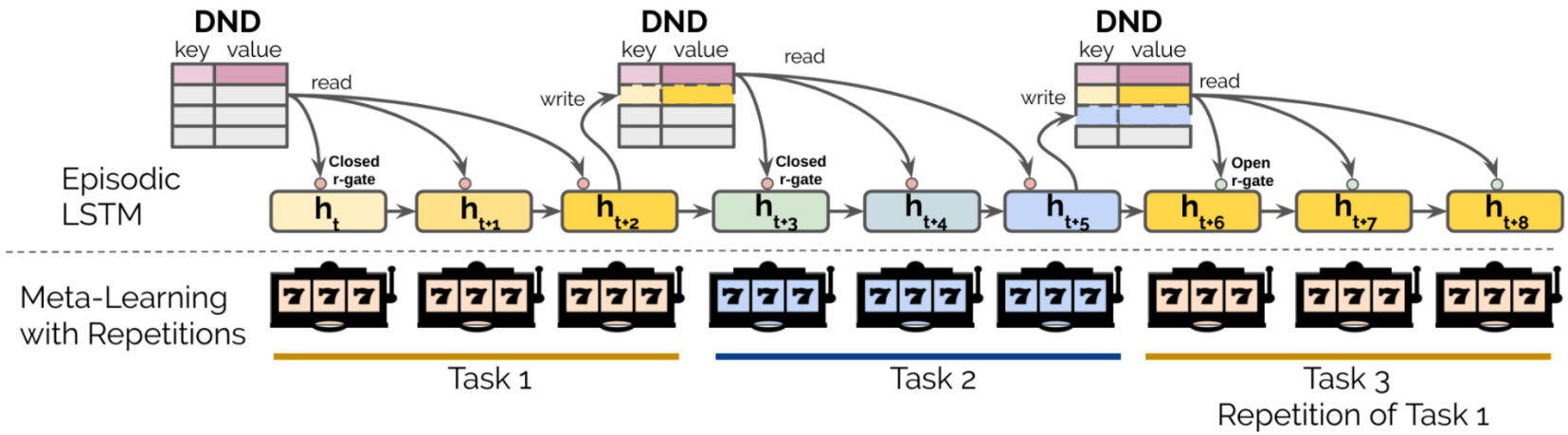


Figure 12: Episodic LSTM.

[9] S Ritter, J. X. Wang, Z Kurth-Nelson, *et al.*, "Episodic Control as Meta-Reinforcement Learning",, DOI: 10.1101/360537. [Online]. Available: https://doi.org/10.1101/360537.

[10] S. Ritter, J. X. Wang, Z. Kurth-Nelson, *et al.*, "Been There, Done That: Meta-Learning with Episodic Recall",

Episodic MetaRL

$$\mathbf{c}_t = \mathbf{i}_t \circ \mathbf{c}_{in} + \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{r}_t \circ \mathbf{c}_{ep}$$

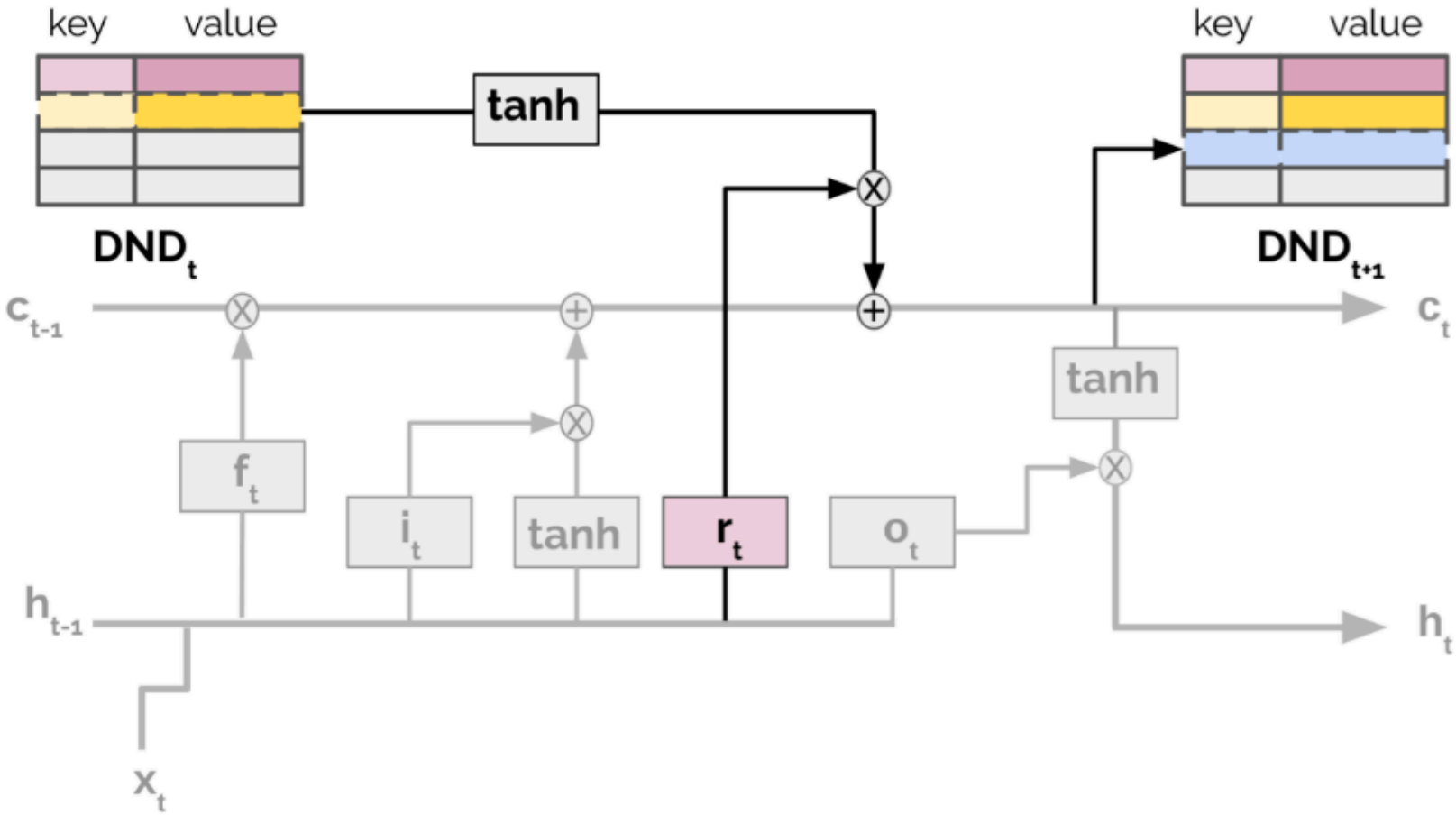


Figure 13: Adding a reinstatement gate to LSTM for episodic memory.

Meta-RL

Structured Exploration Strategies[11]

- Motivation: Temporally coherent randomness is dismissed when adding noise independently at each time step.
- Key idea: Condition the policy on per-episode random variables drawn from a learned latent distribution.

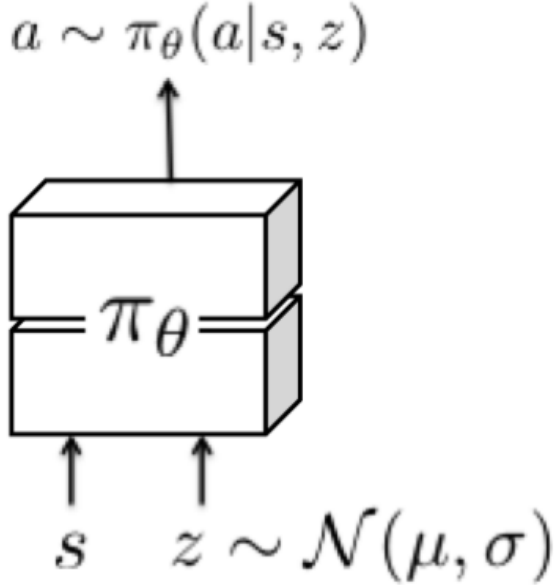


Figure 14: Structured stochasticity: latent variable $z \sim N(\mu, \sigma)$ is sampled once per episode. μ, σ are learnable parameters.

[11] A. Gupta, R. Mendonca, Y. Liu, *et al.*, "Meta-Reinforcement Learning of Structured Exploration Strategies", Tech. Rep., 2018. arXiv: 1802.07245v1.

Structured Exploration Strategies

Algorithm 1 MAESN meta-RL algorithm

1: Initialize variational parameters μ_i, σ_i for each training task τ_i
2: **for** iteration $k \in \{1, \ldots, K\}$ **do**
3: Sample a batch of N training tasks from $p(\tau)$
4: **for** task $\tau_i \in \{1, \ldots, N\}$ **do**
5: Gather data using the latent conditioned policy θ, (μ_i, σ_i)
6: Compute inner policy gradient on variational parameters via Equation (4) and (5) (optionally (6))
7: **end for**
8: Compute meta update on both latents and policy parameters by optimizing (3) with TRPO
9: **end for**

Figure 15: Each task has per-task variational parameters $\theta'_i, \mu'_i, \sigma'_i$. Meta-training involves optimizing: 1. initial policy parameters that are shared by all tasks; 2. per-task (μ_i, σ_i).

$$\max_{\theta, \mu_i, \sigma_i} \sum_{i \in tasks} E_{\substack{a_t \sim \pi(a_t|s_t;\theta'_i,z'_i) \\ z'_i \sim \mathcal{N}(\mu'_i,\sigma'_i)}} [\sum_t R_i(s_t)] - \tag{3}$$

$$\sum_{i \in tasks} D_{KL}(\mathcal{N}(\mu_i, \sigma_i) \| \mathcal{N}(0, I)) \tag{4}$$

$$\mu'_i = \mu_i + \alpha_\mu \circ \nabla_{\mu_i} E_{\substack{a_t \sim \pi(a_t|s_t;\theta,z_i) \\ z_i \sim \mathcal{N}(\mu_i,\sigma_i)}} [\sum_t R_i(s_t)] \tag{5}$$

$$\sigma'_i = \sigma_i + \alpha_\sigma \circ \nabla_{\sigma_i} E_{\substack{a_t \sim \pi(a_t|s_t;\theta,z_i) \\ z_i \sim \mathcal{N}(\mu_i,\sigma_i)}} [\sum_t R_i(s_t)] \tag{6}$$

$$\theta'_i = \theta + \alpha_\theta \circ \nabla_\theta E_{\substack{a_t \sim \pi(a_t|s_t;\theta,z_i) \\ z_i \sim \mathcal{N}(\mu_i,\sigma_i)}} [\sum_t R_i(s_t)].$$

Figure 16: Meta-training optimizes for post-update rewards, so MAESN = MAML + structured noise. KL divergence aims to push the Gaussian distribution to the latent variable prior, which is simply a unit Gaussian.

E-MAML[12]

- Key idea: The best exploration would be a policy that generates the **most informative samples** for identifying the task.
- Samples $\bar{\tau}$ drawn under π_θ will impact the final returns $R(\tau)$ by influencing the initial update $U(\theta, \bar{\tau})$. So we modify the expectation as:

$$\iint R(\tau)\pi_{U(\theta,\bar{\tau})}(\tau)\pi_\theta(\bar{\tau})\mathrm{d}\bar{\tau}\mathrm{d}\tau$$

Thus, the initial samples $\bar{\tau}$ are encouraged to cover the state space enough to ensure that the update $U(\theta, \bar{\tau})$ is maximally effective.

$$\frac{1}{T}\sum_{i=1}^{T} R\left(\tau^i\right)\frac{\partial}{\partial\theta}\log\pi_\theta\left(\bar{\tau}^i\right)\Bigg|_{\bar{\tau}^i\sim\pi_\theta,\tau^i\sim\pi_{U(\theta,\bar{\tau})}}$$

[12] B. C. Stadie, G. Yang, R. Houthooft, *et al.*, "Some Considerations on Learning to Explore via Meta-Reinforcement Learning", *Advances in Neural Information Processing Systems*, vol. 2018-Decem, pp. 9280–9290, 2018. arXiv: 1803.01118. [Online]. Available: http://arxiv.org/abs/1803.01118.

E-MAML

Algorithm 1 E-MAML

Require: Task distribution: $P(\mathcal{T})$
Require: α, β learning step size hyperparameters
Require: $n_{\text{inner}}, n_{\text{meta}}$ number of gradient updates for per-task and meta learning

function $U^k(\varphi, \mathcal{L}, \tau_{[0,\cdots,k-1]})$ ▷ Inner Update Function
  for i in [0, $\cdots$, k-1] **do**
    $\varphi \leftarrow \varphi - \alpha \nabla_{\varphi} \mathcal{L}(\varphi, \tau_i)$
  return φ
procedure E-MAML(θ, ϕ)
  randomly initialize θ and ϕ
  while not done **do**
    Sample batch of tasks $\mathcal{T}_i$ from $p(\mathcal{T})$
    for $\mathcal{T}_i \in \mathcal{T}$ **do**
      Sample rollouts $[\tau]_{[0:n_{\text{inner}}-1]} \sim \pi_\theta$;
      $\theta^n \leftarrow U^{n_{\text{inner}}}(\theta, \mathcal{L}, \tau_{[0:n_{\text{inner}}-1]})$ ▷ High-order update
      with $\pi_{\theta^{n_{\text{inner}}}}$, sample $\tau_i^{\text{meta}} = \{s, a, r\}$ from task $\mathcal{T}_i$;
    for i in n_{meta} **do** ▷ Meta-update the model prior θ
      $\theta \leftarrow \theta - \beta \sum_{\tau_i^{\text{meta}}} \nabla_\theta \mathcal{L}_{\text{meta}}[\pi_{U^n_{(\theta,\mathcal{L})}}, \tau_i^{\text{meta}}]$ ▷ no resample

Figure 17: U = stochastic gradient descent. E-MAML changes the deterministic update of θ in MAML into a stochastic method.

E-RL2

- Key idea: Consider the impact of its initial sampling distribution on its final returns.
- Call the rollouts that help account for the impact of this initial sampling distribution Explore-rollouts. Call rollouts that do not account for this dependence Exploit-rollouts.
- Graident computation will only get rewards provided by Exploit-Rollouts.
- During Explore-rollouts the policy will take actions which may not lead to immediate rewards but rather to the RNN hidden weights that perform better system. identification.

Meta-reward for exploration policy[13]

- Motivation: Adding noise is restricted to local region exploration. How to explore globally?
- Meta reward is used to judge the improvement of exploration policy compared with the original policy.

$$\hat{\mathcal{R}}\left(D_0\right) = \hat{R}_{\pi'} - \hat{R}_{\pi}$$

- Teacher-student framework: Teacher is a meta-learner that offers an exploration policy π_e. Student is a DDPG(Deep Deterministic Policy Gradient) network learning new policy π' and help update teacher using meta reward:

$$\max_{\theta^{\pi}} \left\{ J\left(\theta^{\pi}\right) := \mathbb{E}_{s\sim B}\left[Q_{\theta}\left(s, \mu\left(s, \theta^{\pi}\right)\right)\right]\right\}$$

[13] T. Xu, Q. Liu, L. Zhao, *et al.*, "Learning to Explore with Meta-Policy Gradient",, 2018. arXiv: 1803.05044. [Online]. Available: http://arxiv.org/abs/1803.05044.

Meta-reward for exploration policy

Algorithm 1 Teacher: Learn to Explore

1: Initialize π_e and π.
2: Draw D_1 from π to estimate the reward $\hat{R}_\pi$ of π.
3: Initialize the Replay Buffer $B = D_1$.
4: **for** iteration t **do**
5: Generate D_0 by executing teacher's policy π_e.
6: Update actor policy π to π' using DDPG based on D_0: $\pi' \leftarrow \text{DDPG}(\pi, D_0)$.
7: Generate D_1 from π' and estimate the reward of π'. Calculate the meta reward: $\hat{\mathcal{R}}(D_0) = \hat{R}_{\pi'} - \hat{R}_\pi$.
8: Update Teacher's Policy π_e with meta policy gradient

$$\theta^{\pi_e} \leftarrow \theta^{\pi_e} + \eta \nabla_{\theta^{\pi_e}} \log \mathcal{P}(D_0|\pi_e)\hat{\mathcal{R}}(D_0)$$

9: Add both D_0 and D_1 into the Replay Buffer $B \leftarrow B \bigcup D_0 \bigcup D_1$.
10: Update π using DDPG based on Replay Buffer, that is, $\pi \leftarrow \text{DDPG}(\pi,\ B)$. Compute the new $\hat{R}_\pi$.
11: **end for**

Figure 18: Teacher and student learn from each other and update at each iteration. So does the replay buffer $B = \{s_j, a_j, r_j, s_{j+1}\}$.

Meta-RL

Learn diversified skills[14]

- Motivation: Learning skills without reward, especially for cases where reward is sparse or requires human feedback.
- Optimization goal: Train the skills so that they maximize coverage over the set of possible behaviors. Use discriminability between skills as an objective.
- Key idea:
  1. Use state instead of actions to distinguish skills.
  2. Encourage randomness in actions so that skills with high entropy explore a part of the state space far away from other skills.

[14] B. Eysenbach, A. Gupta, J. I. Google, *et al.*, "DIVERSITY IS ALL YOU NEED: LEARNING SKILLS WITHOUT A REWARD FUNCTION", Tech. Rep., 2018. arXiv: 1802.06070v6. [Online]. Available: https://sites.google.com/view/diayn/.

Learn diversified skills

Objective function

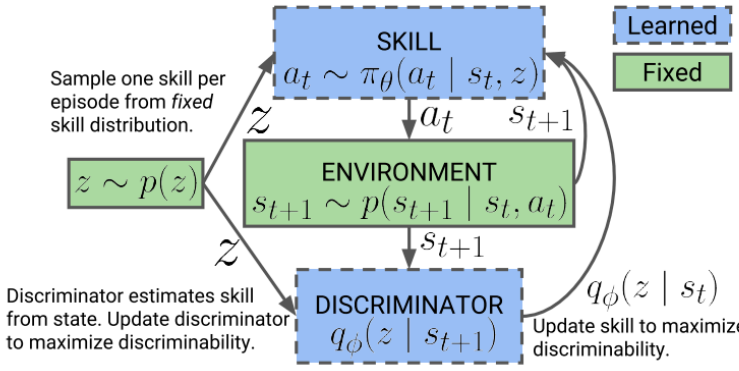
$$\begin{aligned}\mathcal{F}(\theta) &\triangleq I(S;Z) + \mathcal{H}[A \mid S] - I(A;Z \mid S) \\ &= (\mathcal{H}[Z] - \mathcal{H}[Z \mid S]) + \mathcal{H}[A \mid S] - (\mathcal{H}[A \mid S] - \mathcal{H}[A \mid S, Z]) \\ &= \mathcal{H}[Z] - \mathcal{H}[Z \mid S] + \mathcal{H}[A \mid S, Z]\end{aligned}$$

1. Prior distribution over skills should have high entropy.
2. It should be easy to infer the skill z from the current state.
3. Each skill should act as randomly as possible,

$$\begin{aligned}\mathcal{F}(\theta) &= \mathcal{H}[A \mid S, Z] - \mathcal{H}[Z \mid S] + \mathcal{H}[Z] \\ &= \mathcal{H}[A \mid S, Z] + \mathbb{E}_{z\sim p(z), s\sim\pi(z)}[\log p(z \mid s)] - \mathbb{E}_{z\sim p(z)}[\log p(z)] \\ &\geq \mathcal{H}[A \mid S, Z] + \mathbb{E}_{z\sim p(z), s\sim\pi(z)}\left[\log q_\phi(z \mid s) - \log p(z)\right] \triangleq \mathcal{G}(\theta, \phi)\end{aligned}$$

Approximate this posterior $p(z \mid s)$ with a learned discriminator $q_\phi(z \mid s)$

Learn diversified skills



Algorithm 1: DIAYN

while *not converged* **do**
  Sample skill $z \sim p(z)$ and initial state $s_0 \sim p_0(s)$
  for $t \leftarrow 1$ **to** *steps_per_episode* **do**
    Sample action $a_t \sim \pi_\theta(a_t \mid s_t, z)$ from skill.
    Step environment: $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$.
    Compute $q_\phi(z \mid s_{t+1})$ with discriminator.
    Set skill reward $r_t = \log q_\phi(z \mid s_{t+1}) - \log p(z)$
    Update policy (θ) to maximize r_t with SAC.
    Update discriminator (ϕ) with SGD.

Figure 19: Discriminator wants to better infer the skill z from states visited. Skill learner wants to visit states that are easy to discriminate.

Mutual information reward[15]

- Goal: Produce an environment-specific learning algorithm f that can quickly learn an optimal policy $\pi_r^*(a \mid s)$ for any reward function r.
- DIAYN optimizes mutual information by training a discriminator network $D_\phi(z|s)$ that predicts which skill z was used to generate the given states. Define the reward function: $r_z(s) = \log D_\phi(z|s)$
- Apply the reward function $r_z(s)$ to a meta-learner, e.g. MAML.

Algorithm 1 Unsupervised Meta-RL Pseudocode

Input: $\mathcal{M} \setminus R$, an MDP without a reward function
$D_\phi \leftarrow$ DIAYN() or $D_\phi \leftarrow random$
while not converged **do**
  Sample latent task variables $z \sim p(z)$
  Define task reward $r_z(s)$ using $D_\phi(z|s)$
  Update f using MAML with reward $r_z(s)$
end while
Return: a learning algorithm $f : D_\phi \rightarrow \pi$

Figure 20: A task z is sampled at each iteration.

[15] A. Gupta, B. Eysenbach, C. Finn, *et al.*, "Unsupervised Meta-Learning for Reinforcement Learning",, 2018. arXiv: 1806.04640. [Online]. Available: http://arxiv.org/abs/1806.04640.

Environment generation[16]

- Key idea: Maintain a list of active environment-agent pairs EA_List that begins with a single starting pair (a simple environment, a randomly initialized weight vector).
- Generate new environment: Randomly perturb the encoding (parameter vector) of an active environment.
- Optimize the paired agents: Evolution Strategies are applied to the optimization of θ^m of each active environment E^m.
- Transfer attempts: To improve local optimum problem, transfer parameters between active environments.

Algorithm 4 EVALUATE_CANDIDATES

1: **Input:** candidate agents denoted by their policy parameter vectors θ^1, θ^2, ..., θ^M, target environment $E(\cdot)$, learning rate α, noise standard deviation σ
2: **Initialize:** Set list C empty
3: **for** $m = 1$ **to** M **do**
4: Add θ^m to C
5: Add $(\theta^m + \text{ES\_STEP}(\theta^m, E(\cdot), \alpha, \sigma))$ to C
6: **end for**
7: **Return:** $\text{argmax}_{\theta \in C} E(\theta)$

Figure 21: Transfer attempts.

[16] R. Wang, J. Lehman, J. Clune, *et al.*, "Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions", Tech. Rep., 2019. arXiv: 1901.01753v3.

Environment generation

Algorithm 2 POET Main Loop

Input: initial environment $E^{\text{init}}(\cdot)$, its paired agent denoted by policy parameter vector θ^{init}, learning rate α, noise standard deviation σ, iterations T, mutation interval N^{mutate}, transfer interval N^{transfer}
Initialize: Set `EA_list` empty
Add $(E^{\text{init}}(\cdot), \theta^{\text{init}})$ to `EA_list`
for $t = 0$ **to** $T - 1$ **do**
  if $t > 0$ **and** $t \mod N^{\text{mutate}} = 0$ **then**
    `EA_list` = MUTATE_ENVS(`EA_list`) **# new environments created by mutation**
  end if
  M = len(`EA_list`)
  for $m = 1$ **to** M **do**
    $E^m(\cdot), \theta_t^m$ = `EA_list`[m]
    $\theta_{t+1}^m = \theta_t^m +$ ES_STEP$(\theta_t^m, E^m(\cdot), \alpha, \sigma)$ **# each agent independently optimized**
  end for
  for $m = 1$ **to** M **do**
    if $M > 1$ **and** $t \mod N^{\text{transfer}} = 0$ **then**
      $\theta^{\text{top}} =$ EVALUATE_CANDIDATES$(\theta_{t+1}^1, \ldots, \theta_{t+1}^{m-1}, \theta_{t+1}^{m+1}, \ldots, \theta_{t+1}^M, E^m(\cdot), \alpha, \sigma)$
      if $E^m(\theta^{\text{top}}) > E^m(\theta_{t+1}^m)$ **then**
        $\theta_{t+1}^m = \theta^{\text{top}}$ **# transfer attempts**
      end if
    end if
    `EA_list`$[m] = (E^m(\cdot), \theta_{t+1}^m)$
  end for
end for

Figure 22: Three steps 1.generation; 2.optimization; 3.transferring

Model-based meta-adapt[17]

Motivation: Considers each timestep to potentially be a new task, e.g. accidental disturbance. Meta-adapt:

$$\min_{\boldsymbol{\theta},\psi} \mathbb{E}_{\tau_{\mathcal{E}}(t-M,t+K)\sim\mathcal{D}} \left[\mathcal{L}\left(\tau_{\mathcal{E}}(t,t+K),\boldsymbol{\theta}'_{\mathcal{E}}\right)\right]$$

$$\text{s.t.:} \quad \boldsymbol{\theta}'_{\mathcal{E}} = u_{\psi}\left(\tau_{\mathcal{E}}(t-M,t-1),\boldsymbol{\theta}\right)$$

- Shorten the time range from a whole trajectory $\mathcal{D}$ to K timesteps.
- $\mathcal{L}\left(\tau_{\mathcal{E}}(t,t+K),\boldsymbol{\theta}'_{\mathcal{E}}\right) \triangleq -\frac{1}{K}\sum_{k=t}^{t+K} \log \hat{p}_{\boldsymbol{\theta}'_{\mathcal{E}}}\left(\mathbf{s}_{k+1} \mid \mathbf{s}_k, \mathbf{a}_k\right)$

[17] A. Nagabandi, I. Clavera, S. Liu, *et al.*, "Learning to Adapt in Dynamic, Real-World Environments Through Meta-Reinforcement Learning", *arXiv preprint*, 2018. arXiv: 1803.11347. [Online]. Available: http://arxiv.org/abs/1803.11347.

Model-based meta-adapt

Algorithm 1 Model-Based Meta-Reinforcement Learning (train time)

Require: Distribution $\rho_{\mathcal{E}}$ over tasks
Require: Learning rate $\beta \in \mathbb{R}^+$
Require: Number of sampled tasks N, dataset $\mathcal{D}$
Require: Task sampling frequency $n_S \in \mathbb{Z}^+$

Randomly initialize $\boldsymbol{\theta}$
for $i = 1, \ldots$ **do**
  if $i \bmod n_S = 0$ **then**
    Sample $\mathcal{E} \sim \rho(\mathcal{E})$
    Collect $\tau_{\mathcal{E}}$ using Alg. 2
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{\tau_{\mathcal{E}}\}$
  end if
  for $j = 1 \ldots N$ **do**
    $\tau_{\mathcal{E}}(t-M, t-1), \tau_{\mathcal{E}}(t, t+K) \sim \mathcal{D}$
    $\boldsymbol{\theta}'_{\mathcal{E}} \leftarrow u_{\boldsymbol{\psi}}(\tau_{\mathcal{E}}(t-M, t-1), \boldsymbol{\theta})$
    $\mathcal{L}_j \leftarrow \mathcal{L}(\tau_{\mathcal{E}}(t, t+K), \boldsymbol{\theta}'_{\mathcal{E}})$
  end for
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \beta \nabla_{\boldsymbol{\theta}} \frac{1}{N} \sum_{j=1}^{N} \mathcal{L}_j$
  $\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} - \eta \nabla_{\boldsymbol{\psi}} \frac{1}{N} \sum_{j=1}^{N} \mathcal{L}_j$
end for
Return $(\boldsymbol{\theta}, \boldsymbol{\psi})$ as $(\boldsymbol{\theta}_*, \boldsymbol{\psi}_*)$

Algorithm 2 Online Model Adaptation (test time)

Require: Meta-learned parameters $\boldsymbol{\theta}_*, \boldsymbol{\psi}_*$
Require: controller(), H, r, n_A

$\mathcal{D} \leftarrow \emptyset$
for each timestep t **do**
  $\boldsymbol{\theta}'_* \leftarrow u_{\boldsymbol{\psi}_*}(\mathcal{D}(t-M, t-1), \boldsymbol{\theta}_*)$
  $\mathbf{a} \leftarrow \text{controller}(\boldsymbol{\theta}'_*, r, H, n_A)$
  Execute $\mathbf{a}$, add result to $\mathcal{D}$
end for
Return rollout $\mathcal{D}$

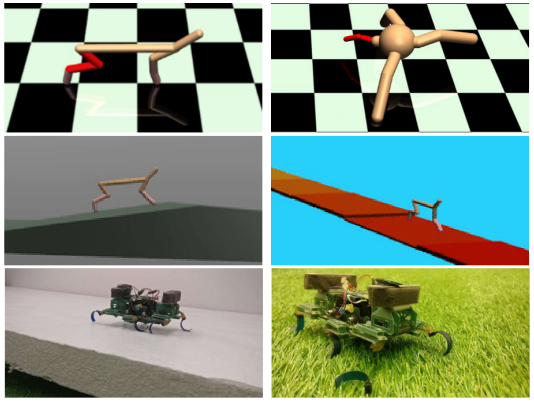


Figure 2: Two real-world and four simulated environments on which our method is evaluated and adaptation is crucial for success (e.g., adapting to different slopes and leg failures)

Figure 23: Adaptation is realized with MAML gradient descent.

Meta-RL

Tracking: Meta Model Predictor[18]

- Siamese: Learn a feature embedding, and find the image region most similar to the target template.
  Cons: Ignore background appearance information.
- Differential online classifier: Distinguish the target object from the background.
  Cons: Online but not end-to-end.
- Common disadvantage: Rely on pre-trained features for image classification or similarity measure, no update potential while meeting unfamiliar object tracking tasks.

Key idea: Discriminate between the target and background after few iterations. We need to reduce optimization recursions.

[18] G. Bhat, M. Danelljan, L. Van Gool, *et al.*, "Learning discriminative model prediction for tracking", *arXiv*, 2019.

Tracking: Meta Model Predictor

- Initializer: conv + ROI pooling (Input: target appearance)
- Optimizer
  - Discriminative learning loss design
  - Steepest descent with variant learning rate
  - Automatically learn all free parameters in the loss

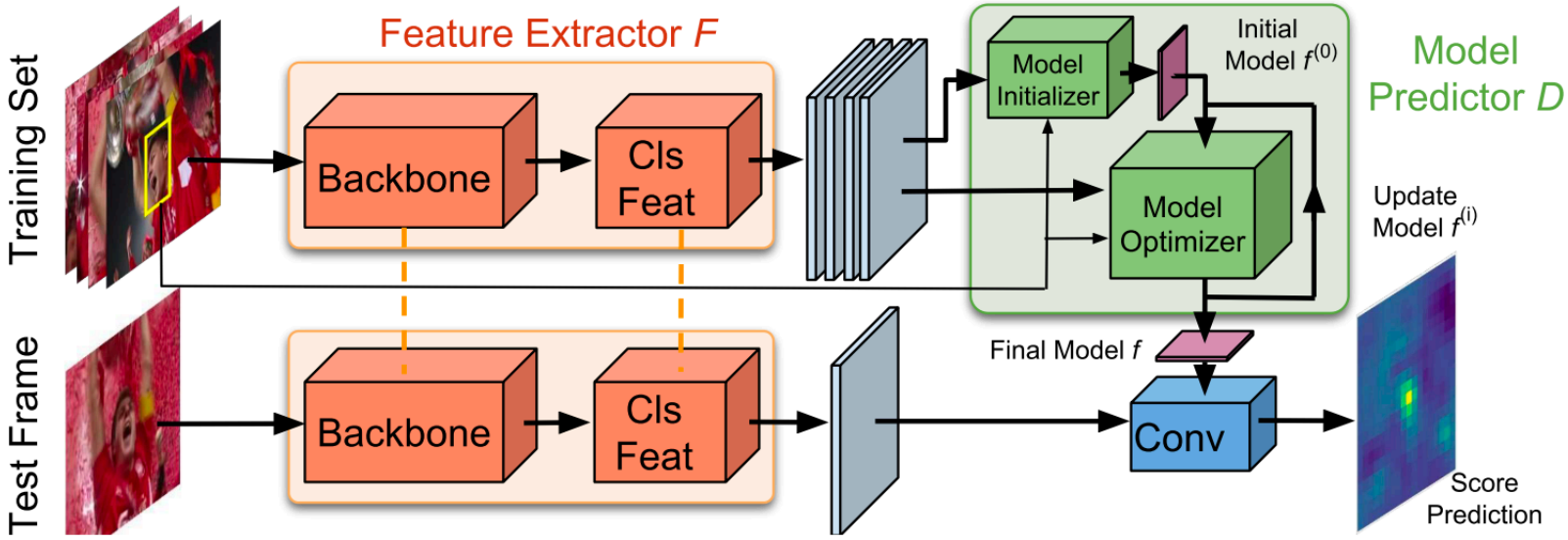


Figure 24: Target classification branch learns the weights f of a convolutional layer that predicts score distribution given a feature map. Bounding box branch is omitted.

Discriminative Learning Loss

Input: $S_{\text{train}} = \{(x_j, c_j)\}_{j=1}^{n}$ contains feature maps and center coordinates of the target.

$$L(f) = \frac{1}{|S_{\text{train}}|} \sum_{(x,c)\in S_{\text{train}}} \|r(x * f, c)\|^2 + \|\lambda f\|^2$$

Here, residual $r(s, c) = s - y_c$, where y_c is the desired target score at each location.
Note the **data imbalance** between target and background, we define $s = max(0, x * f)$ in the background region. Thus, s will not become large negative values that have too much impact on the loss. Target region remains the same due to imbalance.

$$r(s, c) = v_c \cdot (m_c s + (1 - m_c) \max(0, s) - y_c)$$

To separate background and target region, and to alleviate data imbalance, we introduce space weight vector v_c and mask m_c:

Steepest descent

To speed up gradient descent, methods like MAML would take the gradient direction which minimizes the loss of next classifier $f^{(i+1)} = f^{(i)} - \alpha \nabla L\left(f^{(i)}\right)$.
We approximate the loss of next classifier as:

$$L(f) \approx \tilde{L}(f) = \frac{1}{2}\left(f - f^{(i)}\right)^{\mathrm{T}} Q^{(i)}\left(f - f^{(i)}\right) + \left(f - f^{(i)}\right)^{\mathrm{T}} \nabla L\left(f^{(i)}\right) + L\left(f^{(i)}\right)$$

$$\alpha = \frac{\nabla L\left(f^{(i)}\right)^{\mathrm{T}} \nabla L\left(f^{(i)}\right)}{\nabla L\left(f^{(i)}\right)^{\mathrm{T}} Q^{(i)} \nabla L\left(f^{(i)}\right)}$$

When $Q^{(i)} = \frac{1}{\beta} I$, we have a constant learning rate $\alpha = \beta$.

Steepest descent

Let $J^{(i)}$ be the Jacobian of residuals at $f^{(i)}$. Here, only first-order derivatives are involved. Let $Q^{(i)} = (J^{(i)})^\top J^{(i)}$.

Algorithm 1 Target model predictor D.

Input: Samples $S_{\text{train}} = \{(x_j, c_j)\}_{j=1}^{n}$, iterations N_{iter}

1: $f^{(0)} \leftarrow \texttt{ModelInit}(S_{\text{train}})$ # Initialize filter (sec 3.3)
2: **for** $i = 0, \ldots, N_{\text{iter}} - 1$ **do** # Optimizer module loop
3: $\nabla L(f^{(i)}) \leftarrow \texttt{FiltGrad}(f^{(i)}, S_{\text{train}})$ # Using (1)-(2)
4: $h \leftarrow J^{(i)} \nabla L(f^{(i)})$ # Apply Jacobian of (2)
5: $\alpha \leftarrow \|\nabla L(f^{(i)})\|^2 / \|h\|^2$ # Compute step length (5)
6: $f^{(i+1)} \leftarrow f^{(i)} - \alpha \nabla L(f^{(i)})$ # Update filter
7: **end for**

Figure 25: We have now built the whole target classifier branch. It transforms a frame into feature maps and outputs the score distribution regarding the target.

Auto-learning parameters

We have three free parameters in loss design: the label confidence scores y_c, the spatial weight function v_C, and the target mask m_c. Suppose current position is t:

$$y_c(t) = \sum_{k=0}^{N-1} \phi_k^y \rho_k(\|t - c\|)$$

We parameterize them similarly with coefficients ϕ_k. $d - k\Delta$ represents locations of different distance from the center.

$$\rho_k(d) = \begin{cases} \max\left(0, 1 - \frac{|d-k\Delta|}{\Delta}\right), & k < N-1 \\ \max\left(0, \min\left(1, 1 + \frac{d-k\Delta}{\Delta}\right)\right), & k = N-1 \end{cases}$$

Machine Theory of Mind[19]

- Motivation from cognitive psychology: We understand another agent based on models representing the mental states of others.
- Agents: $\mathcal{A}_i = (\Omega_i, \omega_i, R_i, \gamma_i, \pi_i)$ whose policies are not necessarily optimal.
- Observer (to be learned): $\tau_{ij}^{(obs)} = \left\{ \left(x_t^{(obs)}, a_t^{(obs)} \right) \right\}_{t=0}^{T}$
- From meta-learning perspective:
  - General case: Multiple trajectories for one reward, train on tasks of different rewards, test on adaptability to various tasks.
  - ToM case: Multiple agents for one species, train on different species, test on adaptability to various species.

[19] N. C. Rabinowitz, F. Perbet, H. F. Song, *et al.*, "Machine Theory of mind", *35th International Conference on Machine Learning, ICML 2018*, vol. 10, pp. 6723–6738, 2018. arXiv: 1802.07740.

Machine Theory of Mind

Goal: Predictions about future behaviour

- Action prediction: next-step action probabilities
- Goal prediction: probabilities of whether certain objects will be consumed
- State sequence prediction: successor representations(the expected discounted state occupancy)

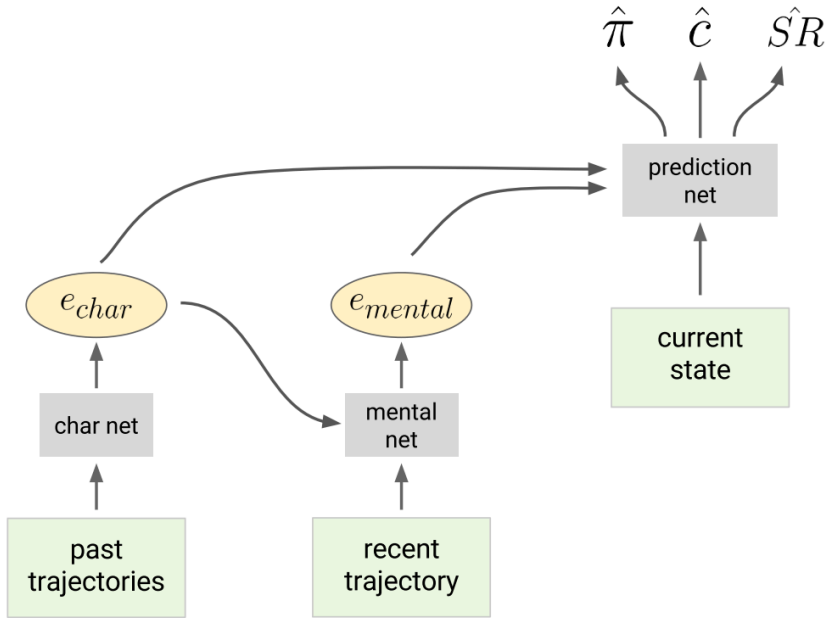
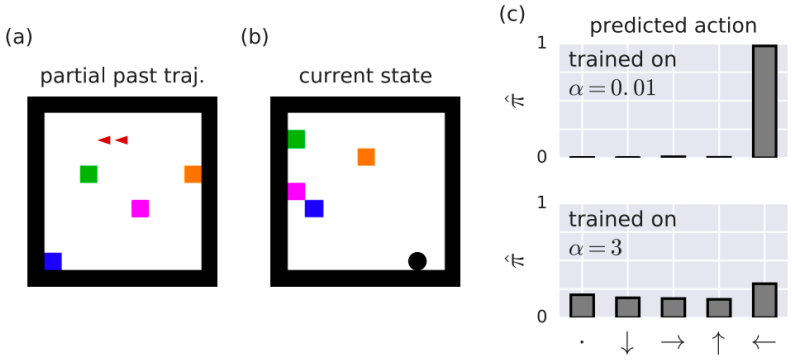


Figure 26: Here recent trajectory usually means the current episode. Past trajectory is highly adjustable.

Random policy agent



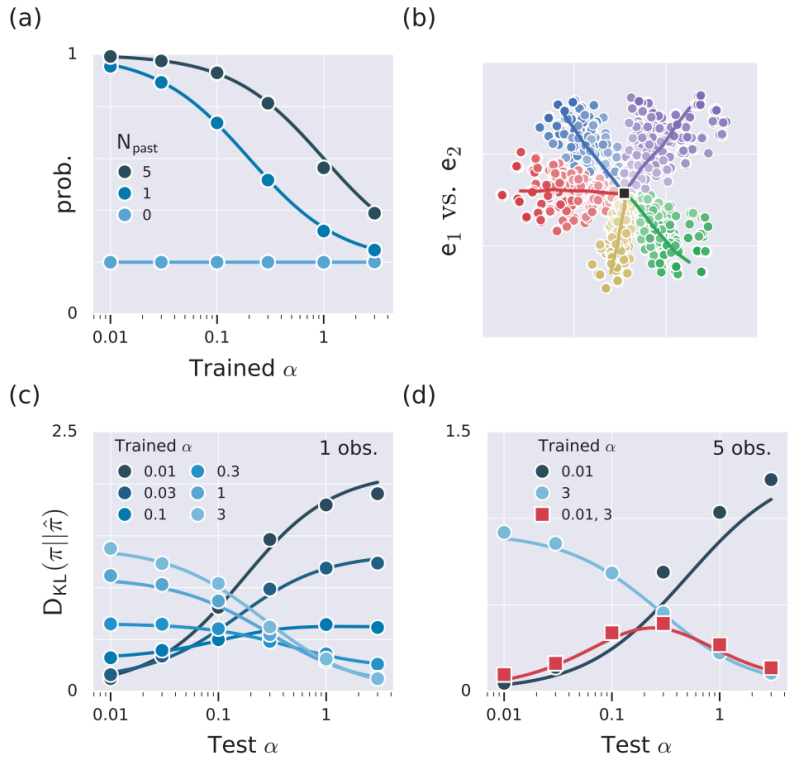


Figure 2. **Example gridworld in which a random agent acts. (a)** Example past episode. Coloured squares indicate objects. Red arrows indicate the positions and actions taken by the agent. **(b)** Example query: a state from a new MDP. Black dot indicates agent position. **(c)** Predictions for the next action taken by the agent shown in (a) in query state (b). Top: prediction from ToMnet trained on agents with near-deterministic policies. Bottom: prediction from ToMnet trained on agents with more stochastic policies.

Figure 27: When given a mixture of species, ToMnet implicitly learns to perform hierarchical inference.

Goal-driven policy agent

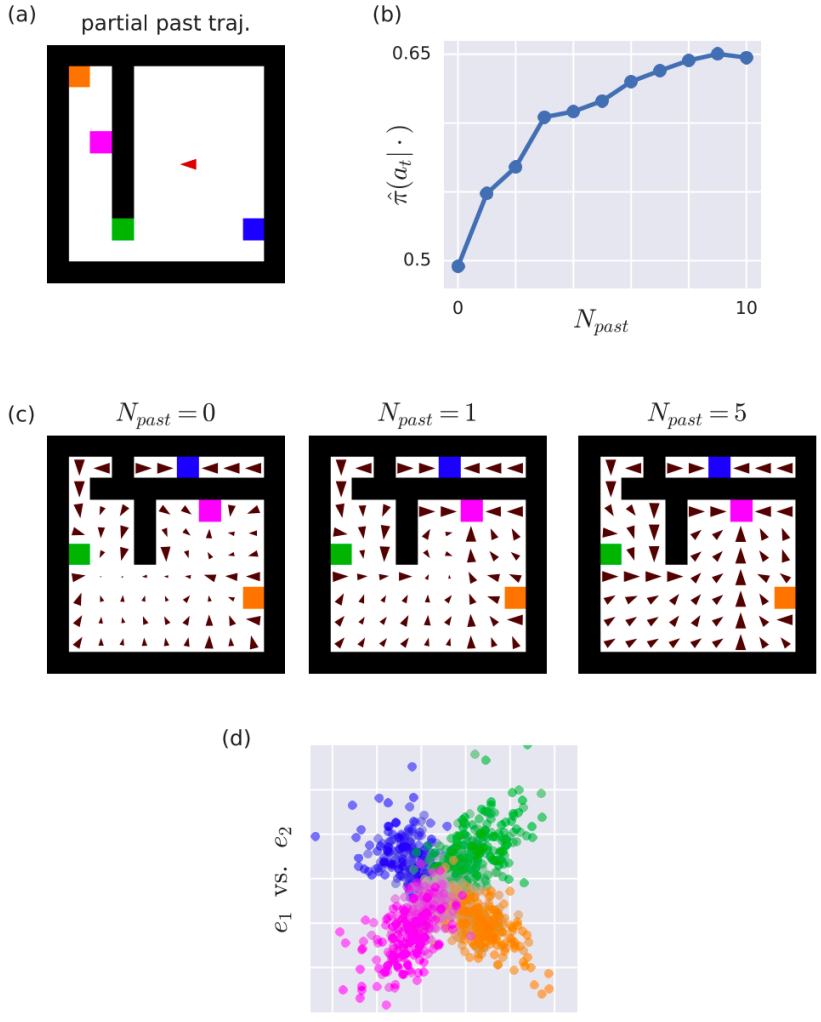


Figure 28: When $N_{\text{past}} = 0$, ToMnet learns shared policies of some states. When increasing N_{past}, preferences for the goal is learned.

Sally-Anne test

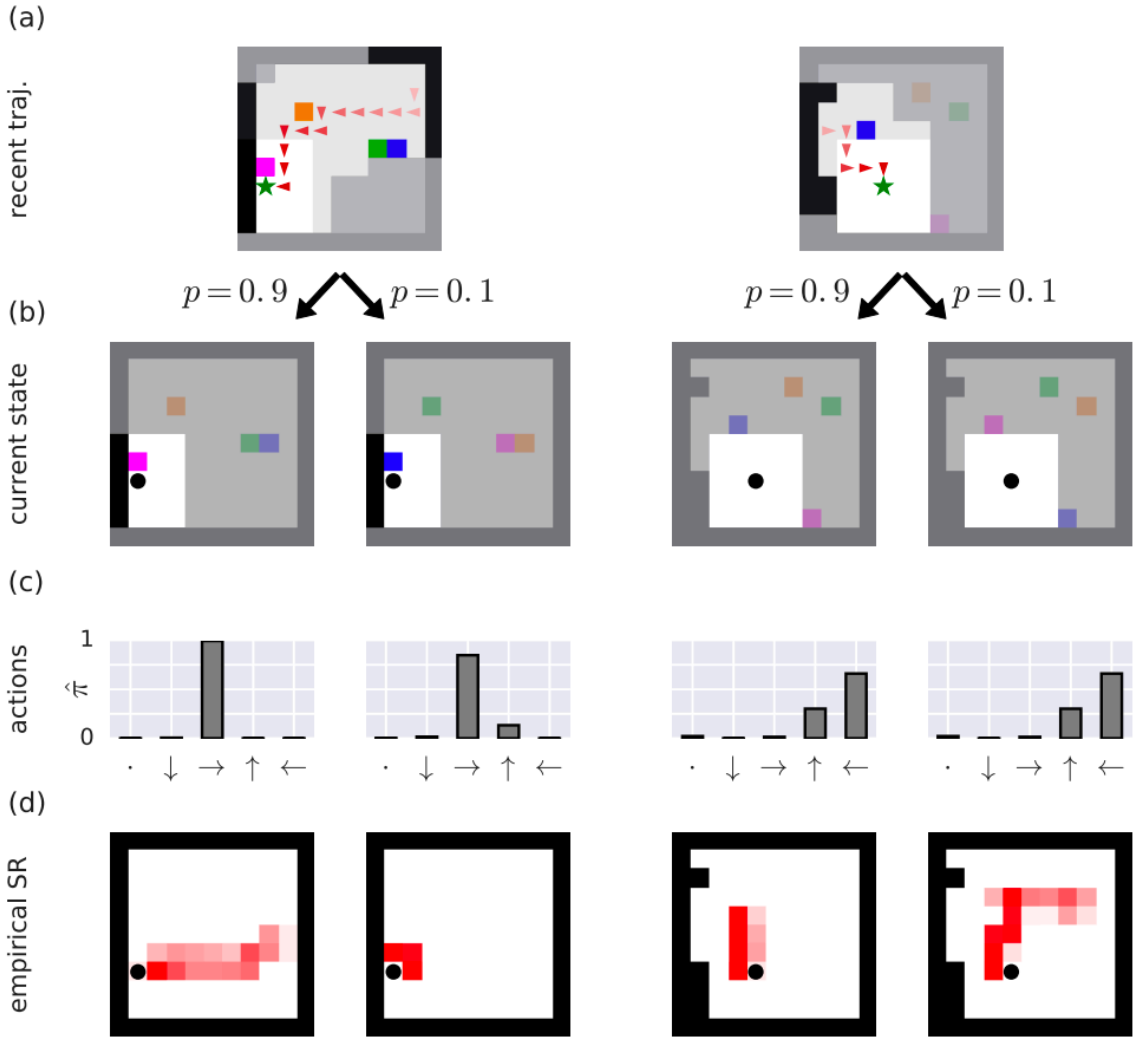


Figure 29: ToMnet is able to detect the belief change of the agent.

Reference I

[1] M. Botvinick, S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell, and D. Hassabis, "Reinforcement Learning, Fast and Slow",, vol. 23, no. 5, 2019. DOI: 10.1016/j.tics.2019.02.006. [Online]. Available: http://creativecommons.org/licenses/by/4.0/.

[2] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn",, 2016. arXiv: 1611.05763. [Online]. Available: http://arxiv.org/abs/1611.05763.

[3] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "FAST REINFORCEMENT LEARNING VIA SLOW REINFORCEMENT LEARNING", Tech. Rep., 2016. arXiv: 1611.02779v2.

Reference II

[4] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A Simple Neural Attentive Meta-Learner", *arXiv*, vol. abs/1707.0, 14 pp.–14 pp. 2017. arXiv: 1707.03141. [Online]. Available: http://arxiv.org/abs/1707.03141.

[5] R. Houthooft, R. Y. Chen, P. Isola, B. C. Stadie, F. Wolski, J. Ho, and P. Abbeel, "Evolved policy gradients", *Advances in Neural Information Processing Systems*, vol. 2018-Decem, no. 21, pp. 5400–5409, 2018, ISSN: 10495258. arXiv: 1802.04821.

[6] K. Xu, E. Ratner, A. Dragan, S. Levine, and C. Finn, "Learning a Prior over Intent via Meta-Inverse Reinforcement Learning", *36th International Conference on Machine Learning, ICML 2019*, vol. 2019-June, pp. 12 036–12 050, 2018. arXiv: 1805.12573. [Online]. Available: http://arxiv.org/abs/1805.12573.

Reference III

[7] Z. Xu, H. van Hasselt, and D. Silver, "Meta-Gradient Reinforcement Learning", *Advances in Neural Information Processing Systems*, vol. 2018-Decem, pp. 2396–2407, 2018. arXiv: 1805.09801. [Online]. Available: http://arxiv.org/abs/1805.09801.

[8] C. Blundell, B. Uria, A. Pritzel, Y. Li, A. Ruderman, J. Z. Leibo, J. Rae, D. Wierstra, and D. Hassabis, "Model-Free Episodic Control",, 2016. arXiv: 1606.04460. [Online]. Available: http://arxiv.org/abs/1606.04460.

[9] S Ritter, J. X. Wang, Z Kurth-Nelson, and M Botvinick, "Episodic Control as Meta-Reinforcement Learning",, DOI: 10.1101/360537. [Online]. Available: https://doi.org/10.1101/360537.

Reference IV

[10] S. Ritter, J. X. Wang, Z. Kurth-Nelson, S. M. Jayakumar, C. Blundell, R. Pascanu, and M. Botvinick, "Been There, Done That: Meta-Learning with Episodic Recall", Tech. Rep., 2018. arXiv: 1805.09692v2.

[11] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-Reinforcement Learning of Structured Exploration Strategies", Tech. Rep., 2018. arXiv: 1802.07245v1.

[12] B. C. Stadie, G. Yang, R. Houthooft, X. Chen, Y. Duan, Y. Wu, P. Abbeel, and I. Sutskever, "Some Considerations on Learning to Explore via Meta-Reinforcement Learning", *Advances in Neural Information Processing Systems*, vol. 2018-Decem, pp. 9280–9290, 2018. arXiv: 1803.01118. [Online]. Available: http://arxiv.org/abs/1803.01118.

[13] T. Xu, Q. Liu, L. Zhao, and J. Peng, "Learning to Explore with Meta-Policy Gradient",, 2018. arXiv: 1803.05044. [Online]. Available: http://arxiv.org/abs/1803.05044.

Reference V

[14] B. Eysenbach, A. Gupta, J. I. Google, and B. S. Levine, "DIVERSITY IS ALL YOU NEED: LEARNING SKILLS WITHOUT A REWARD FUNCTION", Tech. Rep., 2018. arXiv: 1802.06070v6. [Online]. Available: https://sites.google.com/view/diayn/.

[15] A. Gupta, B. Eysenbach, C. Finn, and S. Levine, "Unsupervised Meta-Learning for Reinforcement Learning",, 2018. arXiv: 1806.04640. [Online]. Available: http://arxiv.org/abs/1806.04640.

[16] R. Wang, J. Lehman, J. Clune, and K. O. Stanley, "Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions", Tech. Rep., 2019. arXiv: 1901.01753v3.

Reference VI

[17] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to Adapt in Dynamic, Real-World Environments Through Meta-Reinforcement Learning", *arXiv preprint*, 2018. arXiv: 1803.11347. [Online]. Available: http://arxiv.org/abs/1803.11347.

[18] G. Bhat, M. Danelljan, L. Van Gool, and R. Timofte, "Learning discriminative model prediction for tracking", *arXiv*, 2019.

[19] N. C. Rabinowitz, F. Perbet, H. F. Song, C. Zhang, and M. Botvinick, "Machine Theory of mind", *35th International Conference on Machine Learning, ICML 2018*, vol. 10, pp. 6723–6738, 2018. arXiv: 1802.07740.