Teacher-apprentices RL (TARL)

leveraging complex policy distribution through generative adversarial hypernetwork in reinforcement learning

Tang, Shi Yuan; Irissappane, Athirai A.; Oliehoek, Frans A.; Zhang, Jie

DOI
10.1007/s10458-023-09606-9

Publication date
2023

Document Version
Final published version

Published in
Autonomous Agents and Multi-Agent Systems

Autonomous Agents and Multi-Agent Systems (2023) 37:25
https://doi.org/10.1007/s10458-023-09606-9

Teacher-apprentices RL (TARL): leveraging complex policy distribution through generative adversarial hypernetwork in reinforcement learning

Shi Yuan Tang[1,2] · Athirai A. Irissappane[3] · Frans A. Oliehoek[4] · Jie Zhang[1]

Accepted: 20 March 2023

Abstract
Typically, a Reinforcement Learning (RL) algorithm focuses in learning a single deployable policy as the end product. Depending on the initialization methods and seed randomization, learning a single policy could possibly leads to convergence to different local optima across different runs, especially when the algorithm is sensitive to hyper-parameter tuning. Motivated by the capability of Generative Adversarial Networks (GANs) in learning complex data manifold, the adversarial training procedure could be utilized to learn a population of good-performing policies instead. We extend the teacher-student methodology observed in the Knowledge Distillation field in typical deep neural network prediction tasks to RL paradigm. Instead of learning a single compressed student network, an adversarially-trained generative model (hypernetwork) is learned to output network weights of a population of good-performing policy networks, representing a school of apprentices. Our proposed framework, named Teacher-Apprentices RL (TARL), is modular and could be used in conjunction with many existing RL algorithms. We illustrate the performance gain and improved robustness by combining TARL with various types of RL algorithms, including direct policy search Cross-Entropy Method, Q-learning, Actor-Critic, and policy gradient-based methods.

Keywords Reinforcement learning · Hypernetwork · Generative model · Teacher-apprentices

1 Introduction

With recent advancement of Generative Adversarial Networks (GANs), the use of generative models and adversarial training approaches are pervading in many areas involving deep neural networks. GAN [1] began to surface for image enhancement and synthesis tasks to produce better illustrations [2, 3]. Over the years, advanced image-based applications for GANs have been explored, spanning from image segmentation, image restoration [4], image noise reduction [4, 5], to generating 3D images from 2D images [6]. In

✉ Shi Yuan Tang
shiyuan002@e.ntu.edu.sg

Extended author information available on the last page of the article

Published online: 28 April 2023

addition, other domains such as speech or music synthesis [7, 8], Natural Language Processing (NLP) [9, 10] and even cross-domain applications (generating images from text) [11, 12] have seen the increased adoption of GANs. Notably, the continuous adoption of GANs within various Reinforcement Learning (RL) algorithms [13–15] shows the growing consensus that GAN itself is versatile and modular to be incorporated to existing algorithms with minimal changes, while the robustness of the original algorithm is improved in return. Particularly, the underlying adversarial learning process of GAN in approximating the complex data manifold tends to apply a data augmentation effect especially when the amount of labeled data is scarce or limited [16]. The performance gain by incorporating GAN is not surprising as it is widely used in semi-supervised or weakly supervised learning scenarios, which is similar to RL problems where limited experience is available in the early learning cycles.

In a general RL problem setting, the algorithm learns a single deployable policy as the end product. In many cases, the performance of different policies trained with the same algorithm varies depending on the initialization methods and seed randomization. The high hyper-parameter sensitivity of an algorithm could lead to convergence to different local optima across different runs. Instead of learning a single policy, the robustness of the policy could be improved by the collective decision of a population of good-performing policies. This could be achieved through a highly effective ensemble model during deployment, in which ensembles have been widely shown to provide stabilizing effect and reliable performance gain [17–19].

In this paper, we introduce a Teacher-Apprentice Reinforcement Learning (TARL) framework to represent and learn a population of policies effectively. To represent the population of policies, we use an adversarially-trained hypernetwork [20] to represent the policy distribution, bearing analogy to sampling a policy from a distribution, which we refer to as the policy distribution.[1] This adversarially-trained hypernetwork is derived from the hybrid of GAN together with hypernetwork, which the generator outputs the weights of a policy network. In a recent work [21], we showed that direct policy search RL algorithms like Cross-Entropy Method (CEM), which employ a multivariate Gaussian distribution to represent the policy distribution, could benefit from using hypernetwork as the representation. In the prior work, the CEM originally adopts multiple policies during the policy search process and CEM-AH is used to enhance the policy distribution representation. Different from the motivation of this prior work, we recognize that single policy algorithms could observe the benefits of GAN and multi-policy ensemble methods with the teacher-guiding mechanism. In this work, we generalise this framework and introduce the architecture to extend algorithms from single policy to multiple policies, by transforming the goal from learning a policy into learning the distribution of policies.

Figure 1 illustrates the overall workflow, highlighting the difference between a typical single policy approach and our proposed TARL multiple policies approach. Since adversarial learning is involved to train the hypernetwork, a guiding teacher network $\pi_{teacher}$ co-learns with the hypernetwork using a base RL algorithm. The teacher policy network provides intermediate guiding labels to facilitate hypernetwork's training process and the knowledge is transferred through adversarial learning.

[1] Not to be confused with action distribution, which typically refers to the policy itself.

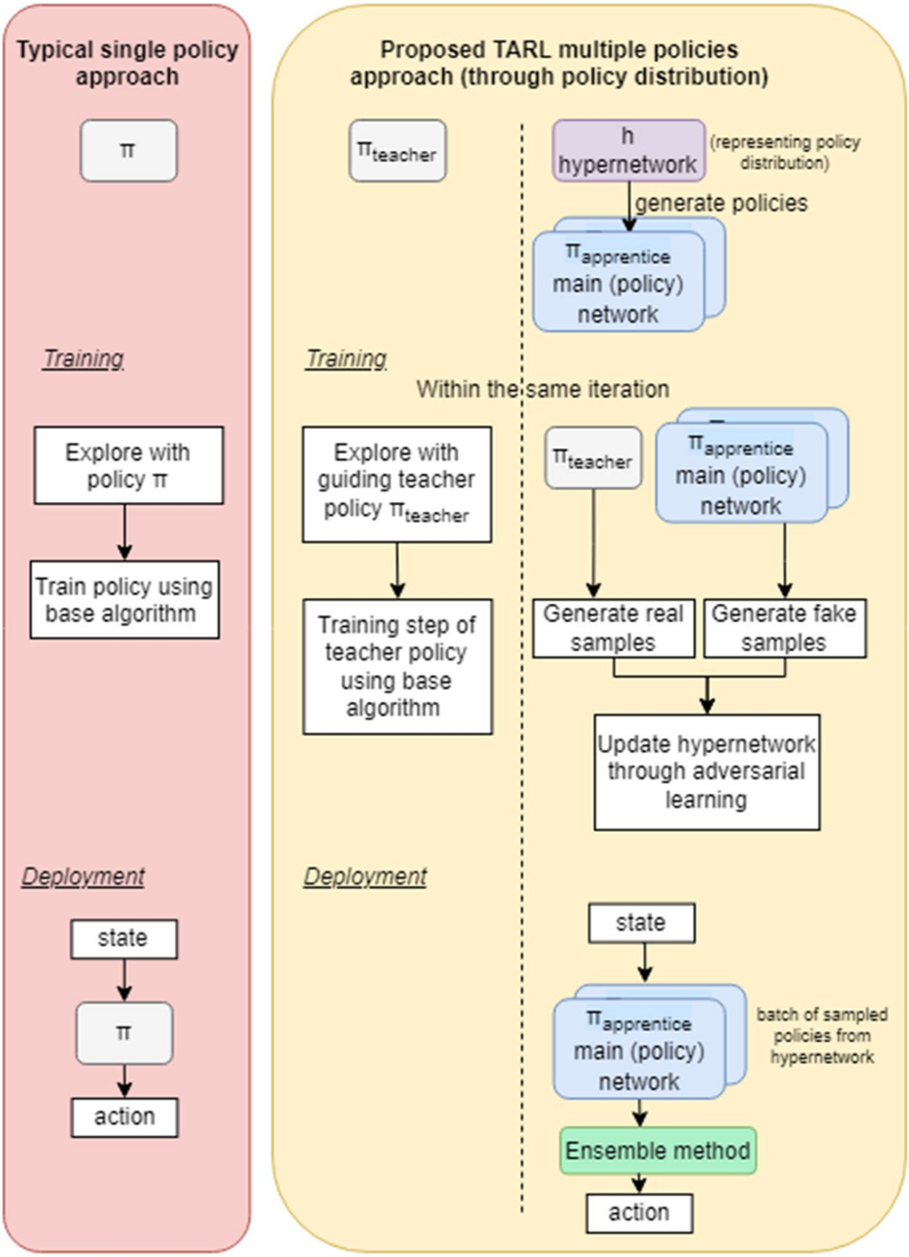

Typical single policy approach
π
Training
Explore with policy π
Train policy using base algorithm
Deployment
state
π
action
Proposed TARL multiple policies approach (through policy distribution)
π_teacher
h hypernetwork
(representing policy distribution)
generate policies
π_apprentice main (policy) network
Training
Within the same iteration
Explore with guiding teacher policy π_teacher
Training step of teacher policy using base algorithm
π_teacher
π_apprentice main (policy) network
Generate real samples
Generate fake samples
Update hypernetwork through adversarial learning
Deployment
state
π_apprentice main (policy) network
batch of sampled policies from hypernetwork
Ensemble method
action

Fig. 1 Overall Workflow of Typical Single Policy Approach vs Proposed TARL Multiple Policies Approach

This is resemblance to the teacher-student framework in Knowledge Distillation (KD) in supervised learning domain, but with several key differences. First, the motivation of KD is focusing on network compression, producing a single student network which is smaller in size compared to teacher network, while our motivation is focusing on improving the robustness and overall performance by learning a population of policy networks, in the context of deep RL. Second, in stark contrast with supervised learning regime in KD which the teacher is pre-trained before training the student network, the teacher in RL is acting as soft label guidance for the adversarial hypernetwork and learning in parallel with the apprentices (students). This hybrid combination of teacher-guided adversarial hypernetwork allows an effective way to learn the complex policy distribution (via hypernetwork). During deployment, as policy distribution is learned through the TARL framework, a batch of good-performing policies $\pi_{apprentice}$ could be sampled. Subsequently, ensemble methods could be employed to boost the performance and improve robustness over the base algorithm, as illustrated at the bottom of Fig. 1.

Our approach can achieve better performance with limited tuning, showing reduced hyper-parameter sensitivity, as well as consistently producing better exploration during early stages of training or in scenarios which experience is limited. Our main contributions are:

- We introduce an innovative framework to effectively learn a population of good performing policies in an end-to-end procedure. Instead of learning a single policy network, the distribution of the policy network weights is learned to resemble the policies (which we refer it as the policy distribution in this paper). Naturally, a method to represent the potentially complex and multi-modal policy weight distributions is desired, which we employed an adversarially-trained hypernetwork [22–24] to achieve this. By introducing better policy distribution representation capability along with uncertainty in the policy network weights [25] during the adversarial training process, our approach exhibits improved robustness, less sensitive to hyper-parameters and increased overall performance compared to the guiding teacher network.
- Our approach is modular and compatible with existing RL algorithms regardless of the optimization method. This is because only the behaviours of the teacher network is required, which acts as the intermediate real sample labels for the adversarial training. This is possible through the use of a replay buffer to store *only* the elite policy behaviours from the guiding teacher and use them to train the hypernetwork. The elite policy behaviours are also carefully assessed before being added to the replay buffer to prevent premature sub-optimal convergence. It also improves sample efficiency, especially compared to direct policy search method [26] since not all samples are discard after every iteration.
- Our proposed method borrows and extends the methodology of teacher-student framework in supervised learning domain to RL domain. At the same time, our proposal differs where a separate procedure to pre-train the teacher network is not required. Instead, the teacher is co-trained with the apprentice adversarial hypernetwork incrementally.
- We conduct experiments on discrete and continuous action problems. Results shows that: (1) our approach enables richer policy representation, outperforming the guiding teacher policy network by up to 16.5% in rewards; (2) our approach is less sensitive to hyper-parameters tuning and sampling a population of apprentice policies followed by an ensemble output only results in minimal impact to inference time.

2 Related works

2.1 Sampling policies in direct policy search methods

A policy could be defined and optimized using different approaches, while following the generalized objective in RL of maximizing the expected return. To be specific, direct policy search (gradient-free method) or policy gradient method (such as REINFORCE) are the two major distinct families. In direct policy search algorithms, the policy parameters are sampled from a distribution. Cross-Entropy Method (CEM) and Evolutionary Algorithms are examples of direct policy search methods. In CEM [27], the policy parameters $\boldsymbol{\phi}$ are generally sampled from a multivariate Gaussian distribution, $\boldsymbol{\phi} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu, \Sigma)$, where μ and Σ represent the mean and covariance. The core idea of our proposal involves learning the distribution of good-performing policy network weights, and representing the policy distribution using a multi-modal complex distribution, instead of using a restrictive uni-modal Gaussian distribution.

The adversarial hypernetwork (generator) in our approach is analogous to the Gaussian distribution in direct policy search method which the parameters are sampled from. However, the similarity ends here. The optimization method for learning these distribution are fundamentally different. Direct policy search methods are non-gradient based method which updates the policy distribution directly towards the elite parameters, whereas our approach optimizes through gradient descent of loss functions.

In recent years, there are several hybrid approaches being introduced, which combine CEM with deep-RL techniques to improve training stability. CEM-RL [26] combines CEM and TD3 to reduce hyper-parameter sensitivity. Qt-Opt [28] is a CEM guided Q-learning method for continuous actions. In Qt-Opt, there is no policy being learned directly and CEM is only used for action selection based on an estimated Q-value from the Q-network. Qt-Opt+DDPG [29] further extends Qt-Opt by learning a separate policy to approximate the CEM action selection process, which lowers the inference time. All the methods above use multivariate Gaussian to represent the CEM policy distribution. Our method shares the idea of using a base algorithm (teacher) to guide the training process, which the above methods utilize CEM as the teacher to benefit from its training stability. Our method could be utilized to improve a base algorithm, but not limited to assigning CEM as the teacher. The teacher could be substituted by any RL algorithm. In addition, as explained earlier, when CEM is used as the base algorithm, our method could enhance the CEM method by using a more complex policy distribution representation by adopting a hypernetwork, instead of a restrictive multivariate Gaussian.

2.2 Hypernetwork

Hypernetwork refers to network which is designed to generate the weights of a primary network. Usually, the primary network is the network which is responsible to perform the actual task [20]. This formulation could be viewed as an extension to standard neural networks, similar to how attention layers and batch normalisation layers were implemented on top of standard neural network architectures [30]. Since hypernetworks promotes knowledge sharing when learning the weight generation sub-task, it is well-suited to be incorporated with meta-learning, neural architecture search [31, 32] and hyper-parameter selection [33]. Recent work from Galanti et al. [30] also shows that the total number of trainable parameters in a hypernetwork is much smaller than that of a corresponding standard neural network, further showing hypernetwork's improved capacity efficiency compared to standard neural networks.

Naturally, adversarial learning is used to enhance weight generation capabilities of the hypernetwork to match the primary network's task. Adversarially-trained hypernetworks were seen in image classification tasks [34], where the hypernetwork generates weights for multiple classification models. It is also seen in decoder architectures for continuous image generation [35]. In our work, we use an adversarially-trained hypernetwork to generate weights for policy network. However, unlike the supervised image problems, there is no ground-truth available for our RL case where continuous exploration is needed. Instead, we rely on a carefully evaluated replay buffer to store *only* the elite policy behaviours from the guiding teacher. These elite past experiences are used as soft-labels to train the hypernetwork.

2.3 Knowledge distillation

Knowledge Distillation (KD) originates from the model compression topic [36] and the topic was introduced by the investigation of how supervision of soft-targets provides superior regularization in the process [37]. The idea has been expanded from distilling the function of a powerful model or teacher into a single neural network [36, 37], to distilling powerful yet easy-to-train larger networks in to smaller but harder-to-train student networks that could outperform the teacher [38]. The first introduction of teacher-student framework was popularised by Hinton et al. [37]. A lower-capacity student network is trained to imitate the outputs produced by a pre-trained high-capacity teacher network together with the one-hot ground-truth labels. Since the teacher-student framework was introduced, many KD variants have been proposed [38–44] that utilizes feature mapping, attention mapping and deep supervision from the hidden layers of teacher network to that of the student network as extra hints for KD supervision. All these methods were built on top of the same teacher-student framework, which the teacher network is pre-trained before the student network is trained. More recently, instead of the one-way knowledge transfer in KD, a two-way Deep Mutual Learning [45] is introduced so that both the teacher and student networks are incrementally trained to mutually improve the performances. In this paper, we maintain our focus in the one-way knowledge transfer as we first extend the methodology of teacher-student framework in supervised learning domain to RL domain. Our proposal also differs from the regular teacher-student framework where a separate procedure to pre-train the teacher network is not required. At the same time, although it is a one-way knowledge transfer, the teacher is co-trained with the apprentice adversarial hypernetwork in parallel and incrementally.

2.4 Generative adversarial imitation learning (GAIL)

At the motivation level, our approach shares the core idea with other approaches that incorporate multi-modality in behaviors. For instance, in the imitation learning domain, GAIL [46] aims to produce a diverse set of actions that aims to be consistent with the expert's behaviour. GAIL's extension using InfoGAN such as InfoGAIL [47] and Triple-GAIL [48] aim to improve the quality of the generator's behavior by maximizing the mutual information between the observed state-action pairs and latent variables. Similarly, our approach intends to exploit GAN's ability to generate diverse policy behaviors. However, there are several differences between our approach and GAIL-based methods.

The application context and problem settings of GAIL-based methods are different. In imitation learning, the goal is to learn a policy that mimics the behavior of an expert based on observed demonstrations. These methods are catered towards applications that are costly to deviate significantly from human expert's behaviours and exploration is secondary. Consequently, the generator's similarity with the expert's behaviour is prioritized, whereas exploration is secondary and achieved through regularizations.

Different from the context of imitation learning, our approach operates in the context of learning from scratch without ground truths. The GAN relies on the behaviours from a guiding teacher policy learning in parallel, therefore a varied behaviour from the non-expert teacher policy is welcomed to encourage exploration. The above coupled with an elite behaviour replay buffer ensures continuous improvement in the generator. GAIL prefers learning behaviours which are close to the shown expert behaviours. On the other hand, our approach prefers a more diverse set of behaviours for exploration purposes.

Due to the reasons above, the architectures to model diversity in behaviours between GAIL and our approach are different; and specifically, the difference lies in the generator (policy). In GAIL, the generator network takes an input noise and state and generates the actions; whereas the generator in our approach does not generate the actions directly. It comprises two components, a hypernetwork and the main policy network. The input noise is passed to the hypernetwork to generate network weights of the main policy network, which subsequently takes an input state and outputs the actions. The use of adversarial hypernetwork adds further flexibility and capacity for modelling diversity. We defer further details to Sect. 4.3.

3 Preliminaries

Generally, an RL problem consists of interactions of agent(s) with an environment, and is often described by a standard Markov Decision Process (MDP), modeled by the tuple $(\mathcal{S}, \mathcal{A}, r, T, \gamma)$, where $\mathcal{S}$ is a set of states $s \in \mathcal{S}$, $\mathcal{A}$ is a set of actions $a \in \mathcal{A}$, T defines transition distribution in the form $T\left(\mathrm{s}_{t+1} \mid \mathrm{s}_t, \mathrm{a}_t\right)$ which describes the dynamics of the system, $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ defines the state- and action-dependent reward function, and $\gamma \in (0, 1]$ is a scalar discount factor. The total returns of an episode or trajectory is defined as $G_t = \sum_{t=0}^{\infty} r\left(s_t, a_t\right)$. Discounted returns are often incorporated to accommodate infinite horizon, $t \rightarrow \infty$. The aim of RL is to maximize the expectation of the sum of discounted returns following a policy π [49], written as:

$$\begin{aligned} J(\pi) &= \mathop{\mathbb{E}}_{t \rightarrow \infty} \left[G_t\right] \\ &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r\left(s_t, a_t\right)\right] \mid a_t \sim \pi\left(\cdot \mid s_t\right), s_{t+1} \sim T\left(\cdot \mid s_t, a_t\right) \end{aligned} \tag{1}$$

This is regarded as the standard maximum expected reward objective. A policy could be defined differently and optimized following the generalized objective above. For approaches where only action distribution is learned as the policy, a direct policy search (gradient-free method) or policy gradient method (such as REINFORCE) could be used.

Since our method is suitable to improve the base algorithm regardless whether it belongs to direct policy search method or policy gradient method, we give a brief overview to highlight the distinction of core ideas of both methods. It is noteworthy that the policy used in both direct policy search method and policy gradient method could be defined similarly using the same policy parameters. The main difference between the two methods are in their optimization procedures.

3.1 Direct policy search

Cross-Entropy Method (CEM) and Evolutionary Algorithms are examples of direct policy search methods. In CEM [27], the policy parameters $\boldsymbol{\phi}$ are generally sampled from a multivariate Gaussian distribution, $\boldsymbol{\phi} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu, \Sigma)$, where μ and Σ represent the mean and covariance. Each policy $\pi_{\boldsymbol{\phi}}$ is evaluated based on the rewards obtained from episodes sampled following the policy. After ranking the policies based on the episodic rewards, the elite policies $\pi_{\boldsymbol{\phi}[\text{elite}]}$ are identified by choosing the top ρ percentile of the sorted list of policies. The multivariate Gaussian policy distribution $\mathcal{N}(\mu, \Sigma)$ is then updated towards the distribution of the rare elite policies. This optimization family considers sampling the policy parameters as the core optimization procedure.

3.2 Policy gradients

The most straightforward gradient-based method to optimize the RL objective in Eq. 1 is to estimate its gradient based on returns:

$$\nabla_{\phi} J(\pi_{\phi}) = \mathbb{E}_{\pi_{\phi}} \left[\nabla_{\phi} \log \pi_{\phi}(a_t \mid s_t)\, G_t \right] \tag{2}$$

Equation 2 is regarded as REINFORCE [50], which is the vanilla policy gradient method. The returns G_t are simply estimated with Monte Carlo samples, where the rewards are summed up over time steps of a sampled trajectory. Subsequently, other methods are developed to replace G_t in Eq. 2 with other return estimators. For example, instead of expected return G_t, an estimate of the expected return Q_t (from a Q-function) or the an advantage value could be used. Equation 3 shows the definition of a typical advantage function, outputting an advantage value which represents how much better an action is with respect to an “average action” at a given state. The “average action” corresponds to a baseline value $b(s_t)$, which could be either estimated using the average reward over the sampled trajectory, or by definition, the state value V.

$$\begin{aligned} A\left(s_t, a_t\right) &= Q\left(s_t, a_t\right) - b(s_t) \\ &= Q\left(s_t, a_t\right) - V\left(s_t\right) \\ &= r_{t+1}(s_{t+1}, a_{t+1}) + \gamma V(s_{t+1}) - V(s_t) \end{aligned} \tag{3}$$

4 From single policy to batch policy learning

Generally, in a typical RL problem setting, a single policy is learned and deployed as the end product. The caveat of learning a single policy is reflected in the inconsistency of reproducing similar results across different training runs. In other words, the performance of the same algorithm could be impacted drastically through different initialization and seed randomization, leading to different local optima across different runs. Instead of learning a single policy, we propose a framework to learn a batch or population of good-performing policies using a complex distribution representation (for the policy distribution) with the following motivations. First, the hyper-parameter sensitivity and the effect of initialization methods could be reduced since the batch policy could be viewed as multiple initialization instances. Second, the robustness and the overall performance could be improved during inference (or testing), utilizing the collective decision of the batch policies using ensemble models. Third, in addition to enable the policy population is diverse, leveraging a complex policy distribution could directly improve the performance when sampling is used to optimize the policy in the base algorithm, as in direct policy search methods.

To help with further elaboration on the third argument, we first give a brief overview on the CEM algorithm since it adopts a basic an easy to understand policy parameters sampling and updating procedure. Next we illustrate the how the distribution representation capability directly impacts the optimization process through a complex objective function in Sect. 4.2.1.

4.1 Cross-entropy method (CEM)

A detailed CEM training process is shown in Algorithm 1. CEM tries to optimize the policy's parameters towards those with higher episodic rewards. Each policy parameter is sampled from its own distribution, and usually, a uni-modal Gaussian is used to represent these distributions. We collectively call the set of distributions as policy distribution. Formally, the policy parameters (represented using $\boldsymbol{\phi} \subset \mathbb{R}^d$) is a vector of dimension $d = |s| \times |a|$, where $|s|$ and $|a|$ denote the dimensions of state and action spaces. The parameters are sampled from a multivariate Gaussian distribution, $\boldsymbol{\phi} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu, \Sigma)$, where μ and Σ represent the mean and covariance matrices. We assume Σ to be a diagonal matrix, signifying independence between each uni-modal Gaussian. During the evaluation phase, each CEM policy $\pi^{\boldsymbol{\phi}}_{\mathsf{CEM}}$ is evaluated based on the episodic rewards obtained using the sampled policy. The elite policies $\boldsymbol{\pi}_{\mathsf{CEM[elite]}}$ of each iteration are identified by choosing the top ρ percentile (or the elite fraction) of the sorted list of policies, based on which the multivariate Gaussian distribution mean and covariance matrices are updated.

At the start of the CEM policy learning, a d-dimensional Gaussian distribution $\mathcal{N}(\mu, \Sigma)$ with zero mean and unit covariance is initialised. During each iteration i (Line 3), parameters $\boldsymbol{\phi}_i$ are sampled from the policy distribution and added to the population (Line 4). For each sample policy $\pi^{\boldsymbol{\phi}_i}_{\mathsf{CEM}}$, defined by the parameters $\boldsymbol{\phi}_i$, we sample actions and obtain rewards until the episode terminates (Line 5–9), and the total episodic reward R_i for the policy $\pi^{\boldsymbol{\phi}_i}_{\mathsf{CEM}}$ is stored for ranking the policies subsequently. A total of N episodes are sampled, each corresponds to a different policy. Naturally, the average of multiple episodes could also be used to evaluate a policy more thoroughly. At the end of each iteration, policies are sorted based on $R \in \mathcal{R}$ and the ρ top-percentile are filtered as the elite policies (Lines 11–12). The policy distribution is then updated based on the mean and covariance of the elite policies (Lines 13–14).

Algorithm 1 Cross-Entropy Method (CEM)

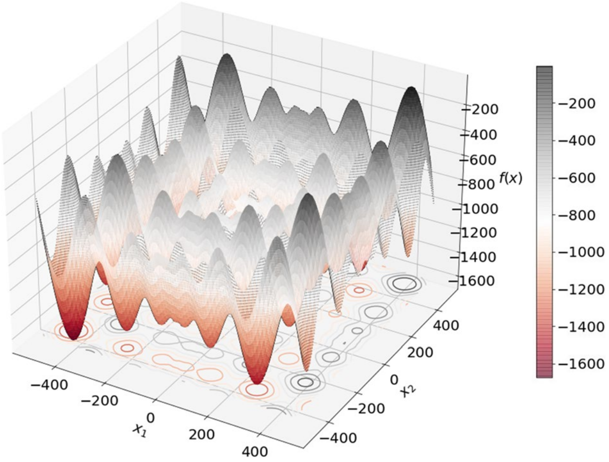
```
Require: $d$-dimensional Gaussian distribution $\mathcal{N}(\mu, \Sigma)$
    Inputs:      population size $N$, elite fraction $\rho$, CEM policy $\pi^{\phi}_{\mathsf{CEM}}(a \mid s)$
    Initialize:  $\boldsymbol{\mu} \leftarrow 0^d, \boldsymbol{\Sigma} \leftarrow diag(1)^{d \times d}$

 1: function TRAIN CEM($N$, $\rho$, $\pi^{\phi}_{\mathsf{CEM}}(a \mid s)$)
 2:     Population $\Phi = \{\}$
 3:     for $i = 1 \rightarrow N$ do
            ▷Sample policy parameters $\boldsymbol{\phi}_i$
 4:         $\boldsymbol{\phi}_i \sim \mathcal{N}(\mu_i, \Sigma_i)$, Append $\boldsymbol{\phi}_i$ to $\Phi$
            ▷Sample episodes following policy $\pi^{\phi_i}_{\mathsf{CEM}}$
 5:         while not terminal state do
 6:             $a \leftarrow \pi^{\phi_i}_{\mathsf{CEM}}(s)$
 7:             Observe $s'$ and obtain reward $r$
 8:             $R_i + = r$
 9:         end while
10:     end for
        ▷Filter the elite policy parameters
11:     $I \leftarrow \{\text{sort}(\mathcal{R})_i : i \in [1, \ldots, k]\}$, $k = \rho$ top-percentile
12:     $\boldsymbol{\pi}_{\mathsf{CEM[elite]}} = \{\pi^{\phi_i}_{\mathsf{CEM}}, i \in I\}$
        ▷Update CEM policy using elite policy parameters
13:     $\boldsymbol{\mu} \leftarrow \frac{1}{k}\sum_{i \in I} \boldsymbol{\phi_i}$
14:     $\boldsymbol{\Sigma} \leftarrow \text{diag}(\text{var}_{i \in I}(\boldsymbol{\phi_i}))$
15:     return
16: end function
```

4.2 Limitation of restrictive distribution representation

The major limitation of using multivariate Gaussian distribution as the policy distribution representation is it imposes overly severe constraints on the $\boldsymbol{\phi}_i$ values that could be sampled, especially after the first few updates. The uni-modal restriction limits the ability to explore the search space and properly guide the search. This constrain of multivariate Gaussian, or any similar parametric distributions, is more pronounce in the case of policy networks, whose parameters (weights) are continuous. When such restrictive distribution representation is used in conjunction with the overly greedy rare-event probability maximization nature of CEM updates, the search space quickly shrinks to a much smaller neighbourhood, causing subsequent distribution updates difficult to escape from a local optimum and limiting further explorations. This amplifies the likelihood of converging to different local optima across different runs. Even with greedy updates, this behaviour could be alleviated if there is capability in handling multiple modes [51].

Fig. 2 Schwefel Objective Function



4.2.1 Example: non-convex, multi-modal optimization surface

We illustrate the above limitations through a maximization problem using a negative of Schwefel function [52], optimized through multivariate Gaussian CEM. The negative Schwefel function[2] is defined by

$$f(x) = -418.982889 \cdot n - \sum_{i=1}^{n} x_i \cdot \sin \sqrt{|x_i|}. \tag{4}$$

The function is non-convex and complex, with multiple local optima. These optima are deliberately made to be geometrically distant. The function is great to evaluate whether an optimization algorithm is susceptible to being trapped in local optima, and how well the algorithm could navigate to the global optimum. For illustration purposes, the dimension of parameter space $i = 2$ is chosen, with the search range defined as $x_i \in [-500, 500], \forall x_i$. The global maximum in this range is $f(x^*) = 0$ with a solution of $x^* = (420.968746, 420.968746)$. Figure 2 illustrates the configuration described above.

Two parameters x_1 and x_2 (two-dimensional Schwefel) are used. The color gradient of red to black represents low to high function values. The black regions thus represent maxima and the red regions represent minima. Transferring the 3D plot Fig. 2 into a 2D plan view, we show the detailed contour plots of all iterations in Fig. 3. The contour plots show the convergence process of CEM across a sequence of iterations. The global optimum is marked with a yellow star and the blue dots represents the sampled values from the distribution. Using a 2-layer neural network architecture (1 input, 1 output), We ran the CEM optimization using the following parameters: for CEM, N=100, ρ=90, batch size=100 and learning rate=10^{-4}.

The (uni-modal) multivariate Gaussian distribution quickly collapses into a narrow distribution along the x_2-axis during the 10th iteration. This quickly eliminates any further exploration along x_2, and subsequently converges to a point mass distribution at a local optimum within 15 iterations. Further, multivariate Gaussian assumes i.i.d (independent

[2] We included a negative in the function to change from the original minimization to maximization problem.

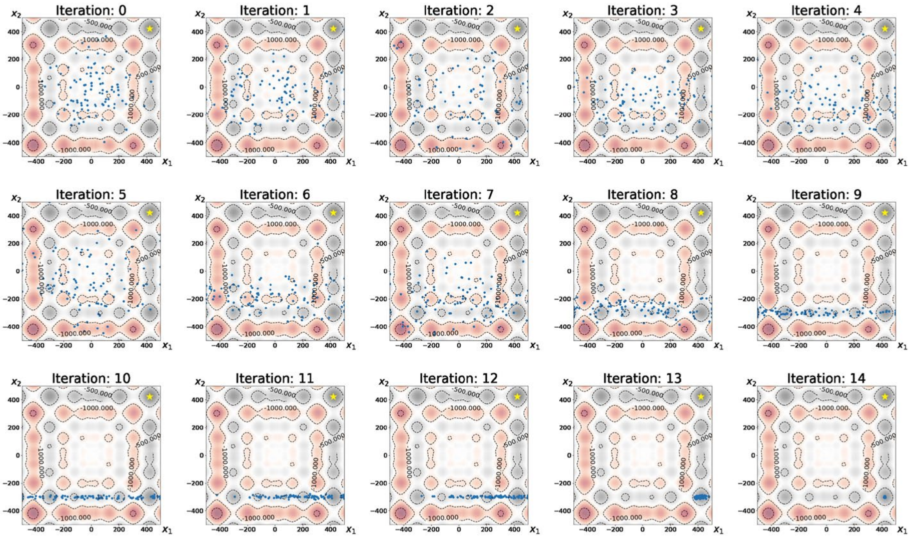


Fig. 3 Convergence of Uni-modal Multivariate Gaussian CEM

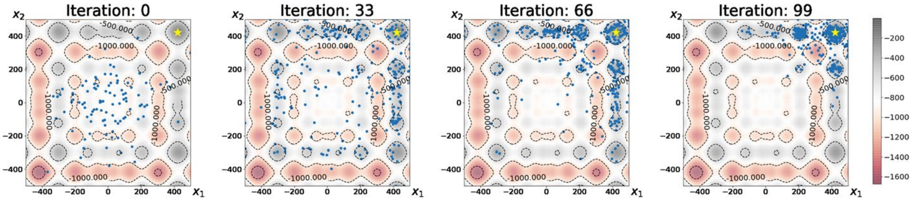


Fig. 4 Convergence of multi-modal hypernetwork

and identically distributed) parameters, implying that the convergence favours a parameter value which leads to high $f(x)$ score independent of other parameters. In Fig. 3, this means that the relatively high $f(x)$ scores when $x_1 \in [375, 425]$ and $x_2 \in [-200, 200]$ will be eliminated early from candidate solutions and will not be explored. The low $f(x)$ scores from $x_1 \in [-425, -375]$ decreases the favourability of selecting $x_2 \in [-200, 200]$.

To incorporate multi-modality, Gaussian mixture models can be used. However, it requires user-defined assumptions on the number of modes [53]. Also, with the intervention of heuristic parameter, there is no guarantee that it performs better than the uni-modal counterpart at all times [54]. It is also noteworthy that the Gaussian mixture extension still belongs to the parametric distribution family.

4.2.2 Leveraging adversarial hypernetwork (AH) for distribution representation

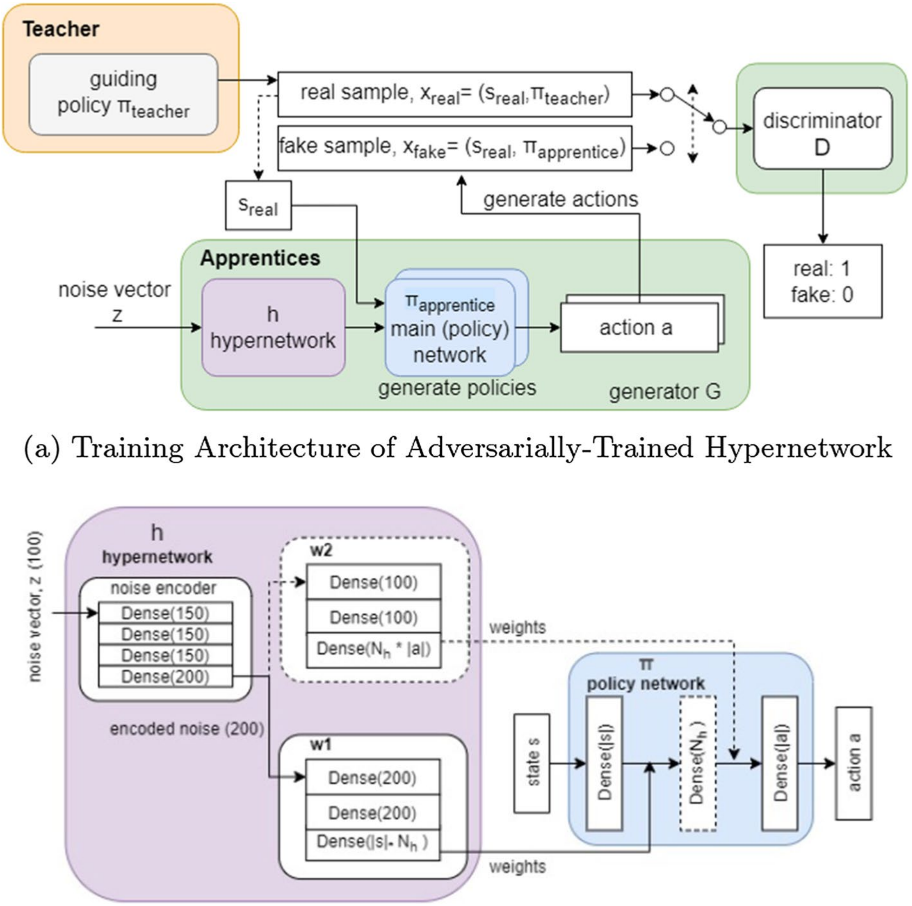
As part of the proposed TARL framework, we utilize an adversarially-trained hypernetwork as the generator of policy weights. The adversarial hypernetwork could thus be interpreted as the policy distribution representation. Figure 4 shows the contour plots Ah-enhanced CEM algorithm (CEM-AH) instead of using multivariate Gaussian. Our approach does not require assumptions on the number of modes or the underlying probability distribution. We notice that multiple modes can be maintained across iterations. In addition, the sampled values are spread out in the search space while concentrated near different local maxima. As a side effect, the algorithm needs to be trained longer with lower sample efficiency while the distribution space moves slowly towards the global maximum (yellow star) by iteration 99.

4.3 Teacher-apprentices reinforcement learning (TARL)

To reiterate, the motivation is to enable batch policy learning through the use of complex policy distribution representation, bearing analogy to sampling a policy from a distribution. As hypernetworks can learn high-dimensional distributions mapped from a lower-dimensional weight space [55, 56] and can be used to model complex multi-modal weight posteriors [22–24], such flexibility would allow us to maintain multiple modes in the policy distribution across update. Our proposed framework uses adversarial hypernetwork to model multi-modal distribution for its superior weight generation capability. However, a challenge exists in applying adversarial learning in reinforcement learning (RL). Ground-truths are typically present in adversarial learning for supervised tasks. For example, real samples of images are essential and used for training the discriminator for GAN in the image domain. On the other hand, since there is no knowledge of optimal actions in RL settings, the ground-truths are absent. To overcome this, a teacher policy is trained in parallel to guide the adversarial learning process. The teacher policy (or the base algorithm) is modular and could be derived from any RL algorithm. We name our proposed framework Teacher-Apprentices Reinforcement Learning (TARL) as the population of apprentices (policies) are guided through the soft-label behaviours of the teacher policy. This hybrid combination of teacher-guided adversarial hypernetwork allows an effective way to learn the complex policy distribution.

The architecture for our proposed TARL framework is shown in Fig. 5a. The adversarial hypernetwork (AH) is implemented similar to a GAN, consisting of generator G and discriminator D components. The generator is made up of two neural networks, the hypernetwork and the main policy network, which collectively correspond to the apprentice component in TARL. The hypernetwork, denoted by $h(\theta \mid z;\alpha)$ is conditioned by Gaussian input noise z along with network parameters α, and represents the policy distribution. It generates the network weights θ for the (main) policy network, denoted by $\pi(a \mid s;\theta)$. The policy network in turn predicts the action probabilities for a given state s. The teacher component is a separate policy network optimized through any RL algorithms. In Sect. 5, we conducted experiments by comparing of several RL methods with AH-enhanced version trained through the TARL framework, including CEM, REINFORCE, DQN, DDPG and PPO.

Figure 5b shows the weights θ generation by hypernetwork $h(\theta \mid z;\alpha)$ when a policy network π is sampled. We also adopted the following hypernetwork architecture in our experiments in Sect. 5. When a policy π is sampled, a Gaussian noise vector z of size 100 is first sampled and input to a noise encoder (with 4 dense layers). The output of the noise encoder is then passed through two weight-generators ($w1$, $w2$) which generate the weights for the policy network π. For this specific implementation, the policy network π consists of an input layer (with state space dimension $|s|$ nodes), a hidden dense layer with $N_h = 200$ nodes and an output layer (with action space dimension $|a|$ nodes). $w1$ generates weights for the connections between the input layer and the hidden layer and $w2$ generates weights for the connections between the hidden layer and the



(a) Training Architecture of Adversarially-Trained Hypernetwork

(b) Weights Generation by Hypernetwork

Fig. 5 Architecture of Teacher-Apprentices Reinforcement Learning. The guiding policy network's behaviours (actions) act as real labels for the adversarial learning process

output layer. Depending on the depth of the desired main policy network, the number of weight-generators could be increased as required. All layers except the last layer are followed by RELU activation. A tanh activation is used in the last layer to ensure compatibility with continuous action problems, which the actions could be normalised between (− 1, 1). The policy network π is then used to infer action probabilities for a given state s. A different policy is sampled by feeding a different noise vector z.

The training data for AH is a set of tuples of the form $x = \langle s, a \rangle$, which are the behaviours of a policy. As part of the adversarial learning process, the discriminator $D(x;\beta)$, with parameters β is tasked to categorise a tuple x to real or fake categories. A teacher policy π_{Teacher} is used to guide the training of AH. Real tuples are obtained through the guiding teacher policy, i.e., $x_{real} = \langle s, \pi_{\text{Teacher}}(s) \rangle$. The fake tuples are generated using randomly sampled policies from the hypernetwork $X_{fake} = \langle s, \pi(s;h(z)) \rangle$ with the same set of states used to obtain the real tuples. To improve sample efficiency, a same state s is used to generate both x_{real} and x_{fake} during training.

Algorithm 2 Teacher-Apprentices Reinforcement Learning

```
Require: hypernetwork $h(\theta \mid z;\ \alpha)$, policy network $\pi(a \mid s;\ \theta)$,
         discriminator $D(x;\ \beta)$, standard Gaussian $\mathcal{N}_z(0,1)$
    Inputs:     NTrain, $M$, $\eta$, G-MSE, D-Adv, G-Adv
    Initialize:    $\alpha$, $\beta$

    ▷Main Function to train the hypernetwork
    function Train TARL( )
        $\Phi_{buf} = \{\}$
        for $t = 1 \rightarrow$ NTrain epochs do
            ▷Train the teacher
            Update $\pi_{\mathsf{Teacher}}$
            ▷Add high reward teacher behaviours as real tuples to replay buffer
            if $\mathcal{R}_i >= \mathcal{R}_{\Phi_{buf}}$ then
                $\Phi_{buf} = \{\Phi_{buf} \cup \langle s_{real}, \pi_{\mathsf{CEM}}(s_{real})\rangle\}$[:capacity]
            end if
            ▷Pre-train Generator (Apprentices)
            for G-MSE epochs do
                Sample a set of $S_{real}$ states from $\Phi_{buf}$
                $X_{real} \leftarrow \{\langle s_{real}, \pi_{\mathsf{Teacher}}(s_{real})\rangle, \forall s_{real} \in S_{real}\}$
                $X_{fake} \leftarrow$ GenerateFake($S_{real}, M$)
                $\nabla L_{pre} \leftarrow \mathrm{MSE}\left(X_{real}, X_{fake} \mid s_{real} \in S_{real}\right)$
                $\alpha \leftarrow \alpha - \eta \nabla L_{pre}$
            end for
            ▷Adversarial Training for Discriminator
            for D-Adv epochs do
                Sample a set of $S_{real}$ states from $\Phi_{buf}$
                $X_{real} \leftarrow \{\langle s_{real}, \pi_{\mathsf{Teacher}}(s_{real})\rangle, \forall s_{real} \in S_{real}\}$
                $X_{fake} \leftarrow$ GenerateFake($S_{real}, M$)
                $\nabla L_D \leftarrow \mathbb{E}_{real} \nabla_\beta [\log D_\beta(X_{real})] + \mathbb{E}_{fake} \nabla_\beta [\log(1 - D_\beta(X_{fake})]$
                $\beta \leftarrow \beta - \eta \nabla L_D$
            end for
            ▷Adversarial Training for Generator (Apprentices)
            for G-Adv epochs do
                Use $X_{fake}$ generated during discriminator training
                $\nabla L_G \leftarrow \mathbb{E}_{fake} \nabla_\alpha [-\log(D_\beta(X_{fake})]$
                $\alpha \leftarrow \alpha - \eta \nabla L_G$
            end for
        end for
        return
    end function
    ▷Function to generate fake tuples using hypernetwork
    function GenerateFake($S_{real}$, $M$)
        Split $S_{real} = \{S^i_{real}, i = 1, \ldots, M\}$  $X_{fake} = \{\}$
        for $i = 1 \rightarrow N$ do
            $i = 1, \ldots, M$
            $z_i \leftarrow$ sample noise, $z_i \sim \mathcal{N}_z(0,1)$
            $X_{fake} \leftarrow X_{fake} \cup \langle s_{real}, \pi_i(s_{real};\ h(z_i))\rangle, \forall s_{real} \in S^i_{real}$
        end for
        return $X_{fake}$
    end function
```

Algorithm 2 shows the training routine of TARL framework. The guiding teacher policy and its AH-enhanced counterpart are trained sequentially. At each training iteration, the teacher policy π_{Teacher} is updated by its own algorithm procedure after gathering the initial samples by exploring the environment following the teacher policy, with or without replay buffer depending on the algorithm (Line 4). The TARL framework contains a separate replay buffer Φ_{buf} which stores the real tuples used for the adversarial training. After the teacher policy update in each iteration, we generate several real tuples $\langle s_{real}, \pi_{\mathsf{CEM}}(s_{real})\rangle$ using the states s_{real} observed and actions output from the $\pi_{\mathsf{Teacher}}(s_{real})$. Only the real tuples from the high episodic rewards (top ρ percentile of policies) are added to Φ_{buf} to ensure it captures the behaviours of elite policies in each iteration (Line 5–6). To be more specific, for a given train iteration t, the real tuples are only added to the replay buffer when the mean reward of the elite policies $\boldsymbol{\pi}_{\mathsf{Teacher[elite]}}$ used to sample these real tuples is greater than or equal to mean reward of the policies for the tuples already present in Φ_{buf}. Using replay buffer also improves sample efficiency as the same set of state s is used to generate the fake samples x_{fake} (Line 11). The Φ_{buf} follows first-in-first-out (FIFO) principle of removing old tuples when its maximum capacity is reached.

Before the adversarial training, the generator is pre-trained (Lines 8-14) for an advantage against the discriminator. For this, a set of real states S_{real} from Φ_{buf} (Line 9) to create a new set of real tuples X_{real} using the updated teacher policy π_{Teacher} (Line 10). Also, a set of fake tuples X_{fake} are generated using the same set of real states S_{real} (Line 11). The procedures to generate fake tuples are defined in function GenerateFake (Lines 30-38). The states S_{real} are divided into M sets and M different policy networks π_i, $i = 1, \dots, M$ are sampled by feeding M different noise vectors z_i to the hypernetwork $h(z_i)$. Each policy network π_i then generates fake tuples $\langle s_{real}, \pi_i(s_{real};h(z_i))\rangle$ for the divided sets of S^i_{real} states. Mean Squared Error (MSE) loss between the real and corresponding fake tuples, i.e., the error between $\pi_{\mathsf{CEM}}(s_{real})$ and $\pi(s_{real};h(z))$ is used to update the generator parameters with learning rate η (Lines 12–13). Pre-training the generator every time there is a change in the replay buffer Φ_{buf} (updating the guiding π_{Teacher}) stabilizes the adversarial training process, as demonstrated in ablation experimental results in Fig. 10.

During adversarial training, the discriminator D and generator G are trained in sequence. The discriminator, parameterized with β is updated using the cross-entropy loss (Eq. 5) which maximizes the log probability for categorising the corresponding real tuples X_{real} and fake tuples X_{fake} (Lines 15–21). To improve convergence stability, the discriminator is updated more often than the generator, i.e., D-Adv > G-Adv during initial training. A decaying noise is also added to the discriminator input to improve its generalization capability [57]. The generator is updated using the loss L_G (Eq.. 6), which is based on how well the discriminator identifies real and fake tuples (Lines 22–27).

$$\nabla L_D \leftarrow \mathbb{E}_{real} \nabla_\beta [\log D_\beta(X_{real})] + \mathbb{E}_{fake} \nabla_\beta [\log(1 - D_\beta(X_{fake})] \tag{5}$$

$$\nabla L_G \leftarrow \mathbb{E}_{fake} \nabla_\alpha [-\log(D_\beta(X_{fake})] \tag{6}$$

The replay buffer Φ_{buf} is crucial to increase the likelihood of apprentices outperforming the teacher as it is only filled with global elite samples from $\boldsymbol{\pi}_{\mathsf{Teacher[elite]}}$ encountered throughout the training process when the distribution is being updated. To reiterate, the generative hypernetwork $h(\theta \mid z;\alpha)$ learns a mapping from a sampled noise vector $z_i \sim \mathcal{N}_z(0,1)$ to a target policy distribution such that the sampled apprentice policies produce optimal actions (similar to those suggested by $\boldsymbol{\pi}_{\mathsf{Teacher[elite]}}$). This allows multiple actions which lead to high reward to be captured. From a different view, AH learns multiple ways of accomplishing a task. This aligns with the motivation of learning a the distribution of policies instead of a single policy.

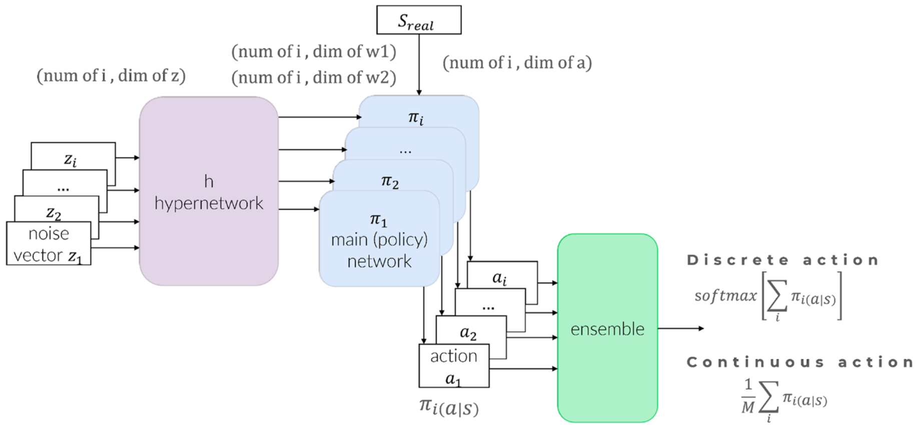


Fig. 6 Inference: sampling a batch of apprentice policies from adversarial hypernetwork (AH)

4.4 Leveraging ensemble techniques with AH

Learning a population of high reward policies using AH-enhanced TARL allows more robust policy behaviour by using bagging ensemble method during deployment, thereby boosting overall performance. Ensembles are known to provide stabilizing effect on the final output and are able to achieve reliable performance gain [17–19]. A collection of randomly sampled policies from the policy distribution could be used to obtain a reliable action.

During the deployment (inference) or any evaluation phase of the AH policy, only the generator component (apprentices) from Fig. 5a is used. A batch of M apprentice policies is randomly sampled $\Pi_i = \{\pi_i(a \mid s;\theta_i);\theta_i = h(z_i),\ i = 1,\dots,M\}$ by feeding M random noise vectors z_i, $i = 1,\dots,M$ to the hypernetwork h of the generator, generating the respective policy weights and thus the corresponding policies. An ensemble of the M policies, $\pi_{\mathrm{AH}} = ensemble(\Pi_i)$ is implemented after the outputs of the apprentice policies to obtain the final policy, as illustrated in Fig. 6.

In discrete action settings, ensemble techniques including majority voting, rank voting, Boltzmann multiplication and Boltzmann addition are generally used to combine the actions derived from each policy [58]. However, insufficient work serves to investigate the performance of these ensemble methods for continuous action problems. In addition, to the best of our knowledge, many ensemble-related works mainly operates in the Actor-Critic framework and focus on combining different Q-value algorithms to attain a more accurate estimator (critic), with the actor remains a single policy. In contrast for TARL, as the batch of policies are sampled from a learned policy distribution, the ensemble technique is applied directly on the policies.

Investigating the effect of different ensemble methods is not within the scope for this paper. We opted to demonstrate the complementary benefit of learning a distribution of policies using the Boltzmann addition ensemble method. This is because the Boltzmann distribution function is identical to *softmax* used in stochastic policy to transform the policy network outputs into probabilities, which we employ in the discrete experiment problem as:

$$ensemble(\Pi_i) = softmax\left[\sum_i \pi_i(a \mid s)\right] \tag{7}$$

At the same time, implementation consistency could be maintained by retaining the same ensemble layers in the network architecture for continuous action problems, and only required to replace the *softmax* layer with a *pooling* layer, which essentially is an averaging (mean) formulation:

$$ensemble(\Pi_i) = \frac{1}{M}\sum_i \pi_i(a \mid s)] \tag{8}$$

Intuitively, we note that there may be circumstances where averaging may yield undesirable results in continuous action problems, which will be discussed in Sect. 6.1 along with suggestions of possible remedies.

From a practical point of view, this opens up to several highly scalable and computationally effective strategies. First, the policy sampling procedure is relatively cheap. Different noise vectors z_i could be sampled in parallel using tensors, which effectively produces a batch of apprentice policies in a single pass, thereby do not impact the inference time during deployment compared to single policy implementation. Second, sampled policies are independent to each other before passing to the ensemble, this could be deployed in distributed systems if larger batch is needed or more thorough policy evaluations are needed during the training phase. Naturally, efficiency strategies used in distributed training systems like shard buffers and multiple teacher policies could be explored in future works.

5 Experiments

Experiments were conducted on both discrete and continuous action problems using RL algorithms including CEM, REINFORCE, DQN, DDPG and PPO together with their TARL AH-enhanced counterparts. The output of the policies could be interpreted differently in these different problems. In continuous control tasks, the output acts as control values of each action, whereas in discrete actions, the policy could be interpreted as a stochastic policy and the output is interpreted as the probability distribution over actions.

For discrete actions, a fully observable maze [59][3] environment with sparse rewards is used. The maze is a 7×7 grid with state space dimension of 243 and action space dimension of 3. The agent is required to reach the goal, with shorter path translates to higher episodic rewards in the end. The state vector encodes information about the presence of a wall, empty space, presence of the agent/goal, color and agent's orientation on different channels of the image grid. The environment is configured so that the agent's actions include moving forward, turning left and right, each having a small negative reward of −0.1. Reaching the goal gives a large reward of 1000, with a 10, 000 maximum time-steps per episode.

For continuous actions, several Pybullet-Gym [60][4] dense reward environments were used, including reacher, ant and walker. The reacher environment consists of a robotic arm

[3] Code could be found at https://github.com/maximecb/gym-minigrid.

[4] This is an open source implementation of Roboschool and MuJoCo environments.

with 2 rotary joints, centered in the environment. The reacher's state space has a dimension of 8, consisting of x, y coordinates of end-effector's initial and target locations. The action space dimension is 2, representing a continuous angle value for the 2 joints. The reacher is required to move the end-effector to reach a target. The reacher environment is configured with a 100 maximum time-steps per episode and the reward is calculated using the inverse distance between end-effector and the target. Pybullet-Gym's default settings with 1000 maximum of time-steps per episode are used in ant and walker environments. Both require controlling an agent to travel as fast as possible. The ant's state space and action space have dimensions of 28 and 8 respectively. For the walker, the dimensions are 22 and 6 respectively.

For hyper-parameters tuning and selection, learning rate, batch size and number of neural network layers are the major ones being considered. Specifically, tuning was performed with respect to learning rate $\in \{0.001, 0.005, 0.0001\}$ (using Adam optimizer), batch size $\in \{50, 100, 1000\}$, and number of neural network layers $\in \{2, 3\}$. The 3-layer policy network architecture is the same as shown in Fig. 5. For a 2-layer policy network, the weight generator $w2$ in the hypernetwork is also removed along with the corresponding hidden layer in the policy network. The result of the best performing combination of architecture and hyper-parameters for each algorithm is shown in Fig. 5 and Table 1.

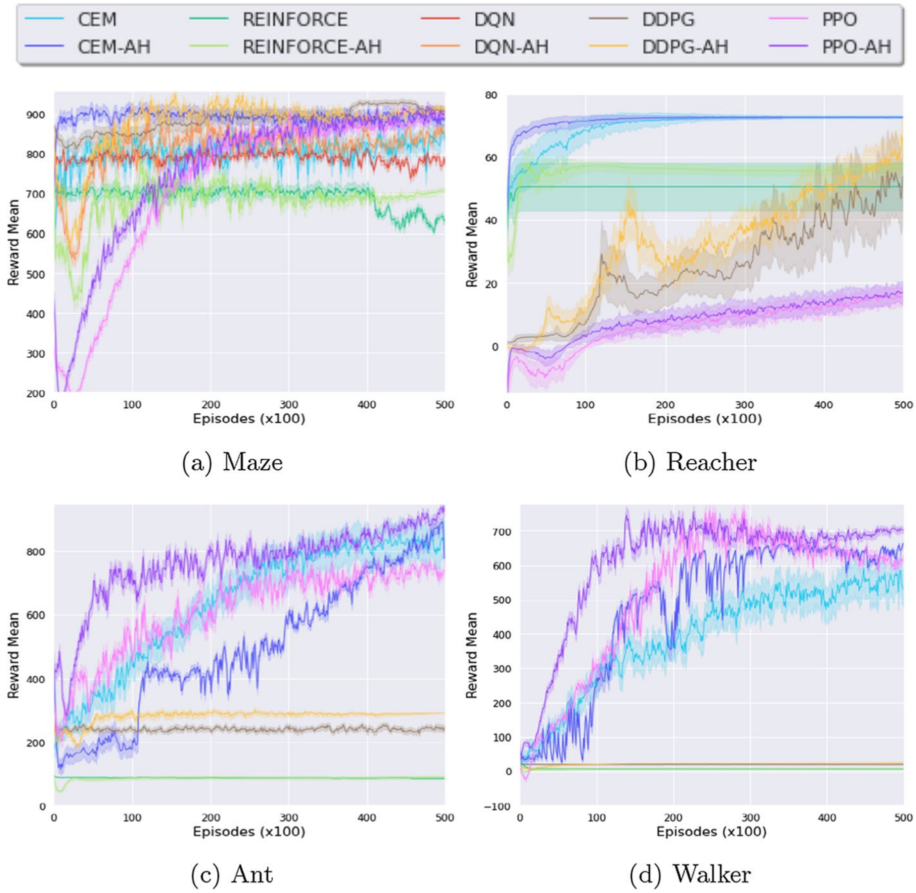
5.1 Evaluation and results

For evaluation, we measure the mean episodic rewards during training as well as the testing[5] performance (averaged over 10 episodes) of the learned policy after every training iteration. The purpose is to compare the actual quality of the learned policy and the rate of convergence, since most algorithms have varied exploration and exploitation strategies. For example, CEM and PPO may use a stochastic policy implementation during training, while decreased stochasticity during testing; REINFORCE and DQN use epsilon-greedy strategy for exploration during training, while they use absolute greedy policy during testing; DDPG use a Q-network with output noise for exploration during training while a separate actor network is used during testing.

Figure 7 shows the mean episodic rewards during training and testing with the best performing set of hyper-parameters for each method, along with the standard deviation (shaded region) across 3–7 runs. A final numerical result is also presented in Table 1,[6] along with the training and testing time per episode. For the maze problem, all AH counterparts have better test performance than the base algorithm except for PPO, which the difference with PPO-AH is relatively small. More importantly, we notice significantly improved performance and consistency in the early training stages across all AH algorithms compared to their corresponding base algorithms. This shows that the richer policy representation successfully captured the behaviours of elite policies and the ensemble model has further boosted the test performance. We notice that the performance of REINFORCE drops considerably after 40k episodes, which REINFORCE-AH successfully prevented attributed to the elite behaviour buffer and multi-modality of the policy distribution. Similar observation is present in the case of DDPG, which achieves a higher reward of 931.31 after 38k episodes, but later dropped to 854.44. For

[5] The term is also used interchangeably with deployment or inference.

[6] The results align with the findings of Pybullet-Gym benchmark [61].



(a) Maze

(b) Reacher

(c) Ant

(d) Walker

Fig. 7 Performance comparison on: **a** Maze; **b** Reacher; **c** Ant; **d** Walker2D in Pybullet-Gym environment

algorithms involving a Q-network such as DQN and DDPG, we concatenate the action probabilities instead of continuous action values along with the state to the input of Q-network, as it is a discrete problem. This stochastic nature for discrete actions makes Q-network training difficult, resulting to relatively noisy training process.

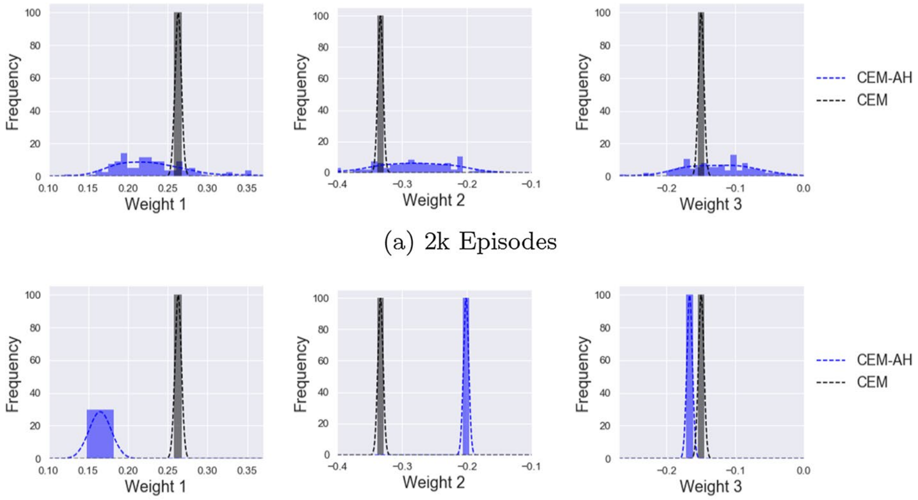
Compared to maze environment, the performance in the continuous environments (Fig. 7b–d) have noticeable larger standard deviations, especially for DDPG. Similarly, we observe improved test performance in AH algorithms during early iterations. As the policy search space is relatively low-dimensional and less complex, similar performances between the base and AH algorithms are observed at convergence. For higher-dimensional problems like ant and walker, REINFORCE and DDPG learns much slower and REINFORCE struggles to learn effectively in both problems. Both ant and walker involve controlling the joints of an agent for walking movements and are more unstable. Consequently, the advantage of AH algorithms compared to the base algorithms is more pronounce. Larger performance gains are observed in AH algorithms attributed to bagging actions through ensemble inference.

Table 1 Performance comparison of various base algorithms with AH-enhanced TARL algorithms

Environment	Algorithm	Training	Testing	Train Time (s)	Test Time (s)
Maze	CEM	820.43	821.44	0.84 ± 0.13	0.84 ± 0.30
	CEM-AH	888.66	894.65	4.03 ± 1.60	2.79 ± 0.48
	REINFORCE	622.30	620.23	3.85 ± 0.37	3.64 ± 0.20
	REINFORCE-AH	702.34	711.23	5.63 ± 2.63	3.77 ± 0.62
	DQN[1]	805.47	798.38	2.18 ± 0.14	1.95 ± 0.79
	DQN-AH[1]	804.67	873.93	4.34 ± 1.68	2.09 ± 0.44
	DDPG	909.90	854.44	2.67 ± 0.72	2.55 ± 0.51
	DDPG-AH	**911.45**	**912.45**	4.67 ± 2.72	2.65 ± 0.54
	PPO	905.72	908.35	2.52 ± 0.66	2.66 ± 0.14
	PPO-AH	905.43	905.89	4.77 ± 1.99	2.86 ± 0.34
Reacher	**CEM**	**73.71**	**73.32**	0.64 ± 0.63	0.54 ± 0.71
	CEM-AH	73.63	72.61	4.32 ± 1.60	2.47 ± 0.18
	REINFORCE	51.56	51.11	1.85 ± 0.57	1.93 ± 0.31
	REINFORCE-AH	55.45	57.25	3.85 ± 2.34	2.91 ± 0.21
	DDPG	59.47	60.79	1.62 ± 0.32	1.67 ± 0.51
	DDPG-AH	66.23	67.32	3.42 ± 1.98	2.87 ± 0.42
	PPO	18.76	18.33	1.77 ± 0.55	1.79 ± 0.33
	PPO-AH	18.53	18.34	3.68 ± 2.23	2.63 ± 0.59
Ant	CEM	740.23	739.59	2.52 ± 1.33	2.32 ± 0.31
	CEM-AH	741.33	845.51	6.70 ± 1.60	2.55 ± 0.48
	REINFORCE	85.34	85.34	3.85 ± 0.37	3.64 ± 0.20
	REINFORCE-AH	88.35	88.15	7.21 ± 2.10	4.02 ± 0.47
	DDPG	242.31	242.31	1.64 ± 0.26	1.55 ± 0.41
	DDPG-AH	288.34	289.57	3.34 ± 2.04	2.02 ± 0.38
	PPO	775.28	754.34	1.87 ± 0.85	1.95 ± 0.21
	PPO-AH	**895.35**	**893.87**	3.89 ± 2.51	2.14 ± 0.33
Walker	CEM	550.78	547.68	3.84 ± 2.13	3.24 ± 0.26
	CEM-AH	**640.54**	643.44	11.82 ± 1.60	4.07 ± 0.49
	REINFORCE	7.01	7.10	4.75 ± 0.68	4.72 ± 0.56
	REINFORCE-AH	7.14	7.46	10.71 ± 2.42	5.12 ± 0.37
	DDPG	19.52	18.46	2.32 ± 0.62	2.45 ± 0.41
	DDPG-AH	20.54	25.56	7.31 ± 2.89	2.62 ± 0.19
	PPO	612.64	603.46	2.61 ± 0.89	2.32 ± 0.31
	PPO-AH	601.35	**708.34**	8.55 ± 1.93	3.02 ± 0.59

Scores are obtained after $50k$ training episodes. The algorithms which achieve the highest testing score across each environment are bolded. The highest training scores in each environment are also bolded. Overall, the testing score of all AH-enhanced TARL algorithms perform consistently better during testing than during training using ensemble of a sampled batch policies. In contrast, the testing performance of base algorithms do not always outperform the training scores. Additionally, AH-enhanced algorithms almost always outperform the base algorithms with the exception of the case in reacher environment

[1] DQN and DQN-AH are discrete action RL algorithms. They were only used in Maze, which is a discrete action environment



(a) 2k Episodes

(b) 50k Episodes

Fig. 8 Weight distribution of CEM-AH vs CEM

In addition, the testing score of all AH-enhanced TARL algorithms perform consistently better during testing than during training using ensemble of a sampled batch policies. In contrast, the testing performance of base algorithms do not always outperform the training scores. Additionally, AH-enhanced algorithms almost always outperform the base algorithms, with minimal difference in the exception cases. Overall, we notice significantly improved performance, lesser hyper-parameter sensitivity (refer to Sect. 5.3) and higher consistency in the early training stages across all AH algorithms compared to their corresponding base algorithms. This is especially true when the training of the base algorithm is relatively stagnant and slow as we notice in ant and walker.

5.2 Effect of hypernetwork on the distribution of policy network weights

The hypernetwork component allows more flexible and richer representation of the policy distribution to be learned. This is important in enabling the motivation to leverage the benefits of multiple policies, which may concentrate to different modes during the learning process, as illustrated in Fig. 4. To further illustrate the idea of the effect of hypernetwork on the policy network weights compared to using a uni-modal multivariate Gaussian, we visualize the histogram of weights value within the policy networks. Comparison is done between CEM (multivariate Gaussian) and CEM-AH (hypernetwork) as CEM is the only base algorithm that adopts multiple policies during its direct policy search process; other base algorithms are single policy in nature.

A total of 100 samples were collected from each CEM and CEM-AH distribution on the maze problem and the histogram for 3 randomly selected policy weights were shown in Fig. 8. The sampled weights are discretized through binning and the approximate distribution shape were fitted through kernel density estimation and shown with dashed lines

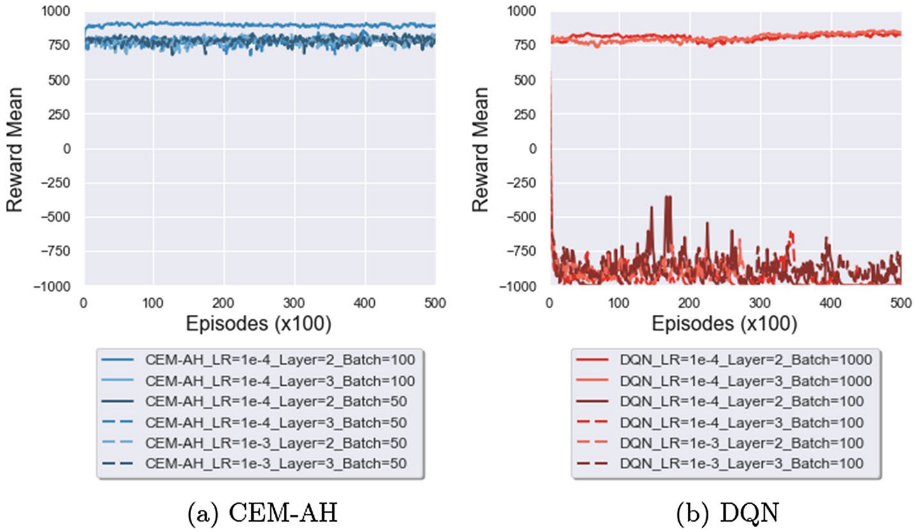


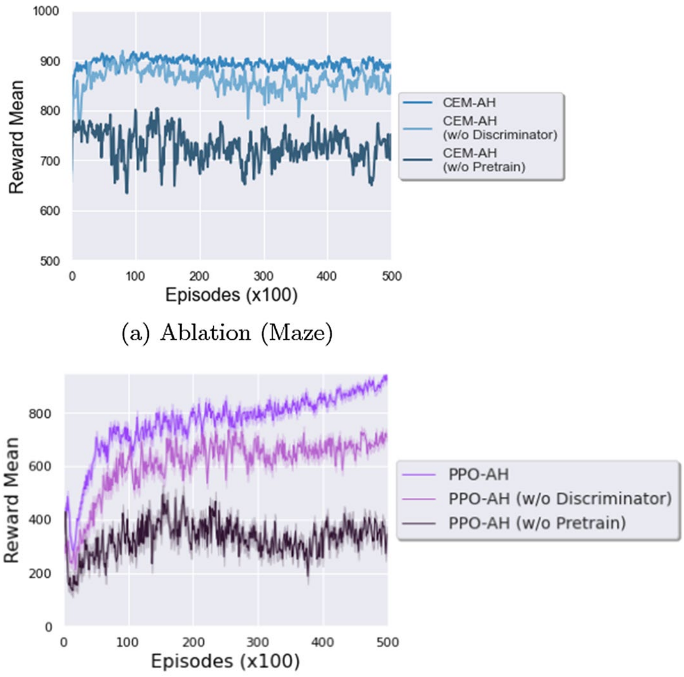
Fig. 9 Hyper-parameter sensitivity: **a** CEM-AH; **b** DQN

for better visibility. Within short training of $2k$ episodes, we observe that CEM quickly converges to a narrow distribution in Fig. 8a, whereas with the generative hypernetwork, CEM-AH maintains a relatively well-spread distribution that could exhibit multi-modal characteristics; despite the base algorithm CEM is using a uni-modal Gaussian distribution. This is because the knowledge transfer between the teacher and apprentices is indirect, at the same time, the selective process for adding past experience in the replay buffer further decouples the performance restriction from the teacher base algorithm, allowing the apprentices to outperform the teacher.

Only after longer training episode of $50k$ episodes, majority of sampled weights from CEM-AH converge to a narrow distribution, as shown in Fig. 8b. However, we can also observe a mixture of Dirac deltas for Weight 1 even after the performance converged at $50k$ episodes. The well-spread and multi-modal characteristics of hypernetwork contributed in retaining better exploration, especially during the early stages of training.

5.3 Hyper-parameter sensitivity and ablation study

As mentioned in Sect. 5, we perform hyper-parameter tuning with respect to learning rate $\in \{0.001, 0.005, 0.0001\}$, batch size $\in \{50, 100, 1000\}$, and number of neural network layers $\in \{2, 3\}$. An illustration is available in Fig. 9, the AH-enhanced algorithms are less sensitive to hyper-parameters, in contrast with algorithm like DQN. During the hyper-parameter tuning process, we notice that DQN failed abruptly in several runs due to the difficult sparse rewards scenario, while this is not observed in DQN-AH. Also, DDPG learns only with a 3-layer network, however, its convergence speed varies significantly with respect to different configurations. While the guiding base algorithms do affect the performance of its AH-enhanced counterparts, all AH algorithms do exhibit significantly improved



(a) Ablation (Maze)

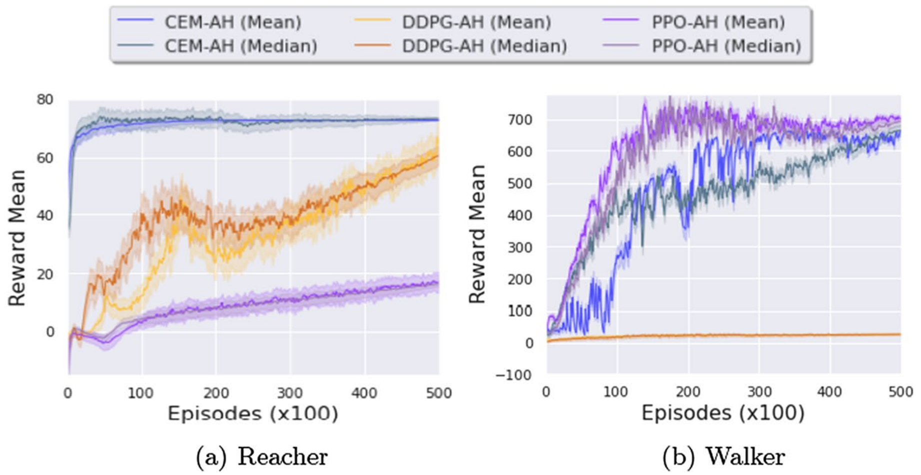
(b) Ablation (Ant)

Fig. 10 Ablation Study: **a** CEM-AH; **b** PPO-AH

performance and stability in the early training stages regardless to the selection of apprentice-related hyper-parameters.

We would like to clarify that the choice of teacher algorithm will affect and limit the performance of apprentices. The abrupt cases which certain hyper-parameter combinations in teacher that results in complete failure of the teacher learning would also limit the performance of apprentices; due to the reliance in teacher for exploration. However, the performance of apprentices is less sensitive to the choice of hyper-parameters within itself, and performance boost is generally observed regardless of the choice of batch size, learning rate and network layers for the hypernetwork training, as shown in Fig. 9a. In parallel, the selective process for adding past experience in the replay buffer also contributed to the result of apprentices outperforming the teacher. Consequently, when the performance difference due to the choice of hyper-parameter within the teacher base algorithm is not large, TARL further minimizes this hyper-parameter sensitivity that originally could be observed within the teacher.

Ablation study is performed on the CEM-AH algorithm for discrete action problem (maze) to demonstrate the significance of adversarial learning and its pre-training procedure, as shown in Fig. 10a. Similarly, ablation is performed on PPO-AH for continuous action problem (ant) and shown in Fig. 10b. Without pre-training and the added prior advantage for the generator, AH performs significantly worse and suffers from convergence issues during adversarial training. On the other hand, the performance drops slightly when the hypernetwork is trained without the adversarial component (without the discriminator).



(a) Reacher

(b) Walker

Fig. 11 Performance Comparison Between Mean and Median Ensemble Approaches on: **a** Reacher; **b** Walker

6 Limitations

6.1 Choice of ensemble method may affect performance

Although mean (or averaging) was used as the ensemble method for continuous action problems in this paper, it is merely a demonstration of a cheap and straightforward approach that is architecturally compatible with the Boltzmann addition ensemble method we used for discrete action problem; which the *softmax* layer is replace with pooling layer. As noted in Sect. 4.4, there may be circumstances where averaging may yield undesirable results.

For example, imagine driving a vehicle and the continuous action represents the degree of turning the steering wheel. An obstacle at the center would require the control of steering left or right. However, if averaging were to be used and an equal number of policies were to suggest turning in both directions, a "centered" result may be produced and resulting in a slower turn or no turning in the worst case. As a suggested remedy for this scenario, median ensemble could be used with an odd number of policies as:

$$ensemble(\Pi_k) = \pi_k(a \mid s),\ \ k = 2i + 1 \text{ and } i \in \mathbb{Z} \tag{9}$$

The reasoning is twofold; first, median could effectively reduce the effect of outliers, and second, the median ensemble method would not modify the policy outputs (with odd number of policies), which could address the "centering" scenario. A comparison between mean and median ensemble approach is shown in Fig. 11. We found that there is only noticeable difference in the early stages of training and both methods converge to identical performance towards the later stages of training. At early stages, we observed that compared to mean ensemble method, the median ensemble method would improve the performance in the reacher problem, but equal or negligible performance drop for the walker problem. This may suggest that mean ensemble would work better for path or position control tasks like reacher. Although, more throughout investigations should be conducted to draw a better conclusion.

We would like to reiterate that investigating the best ensemble approach is not within the scope of this paper. The intention is to show the complementary benefit of learning a distribution of policies, which could be sampled cheaply and increasing the overall performance through ensemble techniques.

6.2 Increased computational resources and time during training process

Training and testing time per episode are included in Table 1, showing the computation cost differences between the single policy base algorithm and AH batch policy approaches. A dual core Intel Xeon Gold 6148 CPU @ 2.40GHz with a Tesla V100 GPU workstation is used. We observe longer training time in AH algorithms due to co-training of teacher and apprentices. Although, we note that the training time can be improved by parallelizing the policy and episode sampling processes in AH.

The performance boosting effect is significant during early stages of training compared to single policy method, suggesting that the method is suitable when there are limited samples or exploration remains needed, while performance is at the same time of great importance, especially in sparse rewards scenarios. At the same time, although an overall higher performance could potentially be obtained at convergence, significant computational time and resources during the training stage are required. Due to the above, the method would not be recommended when cost-effectiveness is the main consideration of the application.

On the other hand, as explained in Sect. 4.4, the testing time of AH algorithms remains comparable to the corresponding base algorithms as the sampling procedure is lightweight and executed in parallel. The increase in computational resources is negligible during deployment when convergence is achieved, suggesting that the method could be useful for scenarios where environment is relatively static and less frequent training is required, provided that performance boosting is desired.

7 Concluding remarks

We present Teacher-Apprentices RL (TARL), a framework which is modular and utilizes an adversarially-trained hypernetwork as a complex policy distribution to learn a population of good-performing policies effectively. We demonstrated how adopting a richer policy distribution representation (using a hypernetwork) would maintain a well-spread distribution and exhibits capability of capturing multiple modes. We overcome the challenges of adversarial learning in RL with the lack of ground truth by employing a guiding teacher policy which provides intermediate soft-labels. Since the knowledge transfer from the teacher to apprentices is through adversarial learning, it could be employed with any RL algorithm as the teacher. We have shown that performance boost could be achieved by combining the outputs of apprentices using ensemble techniques. Experiments demonstrate that AH algorithms exhibits improved robustness, lowered hyper-parameter sensitivity, and improved overall performance especially in the early training stages across all AH algorithms compared to their corresponding base algorithms, without significant impact to the inference time.

Even though the performance of apprentices is less sensitive to the choice of hyperparameters within itself, we noted that the choice of teacher algorithm will affect and limit the performance of apprentices due to the reliance in teacher for exploration. Nevertheless, the performance of apprentices (AH algorithms) is less sensitive to the choice of hyperparameters within itself, and performance boost is generally observed using ensemble. It is notable that the indirect knowledge transfer from teacher to apprentices through adversarial learning, coupled with the selective process for adding past experience in the replay buffer contributed in allowing apprentices to outperform the teacher. Although it is beyond the scope of this paper, we noted that the choice of ensemble method may affect the AH performance in the early stages of training in certain continuous action problems and suggested that median could be used to alleviate the issue.

As such, future research could investigate the effect of the choice of ensemble in greater detail, including majority voting, rank voting, Boltzmann multiplication, Boltzmann addition and more sophisticated technique like ensemble networks. On the other hand, although the teacher co-learns in parallel with the apprentices, the knowledge transfer is one-way from teacher to the apprentices, which they do not have influence over the teacher's learning process. Another line of research could thus explore a two-way mutual learning process between the teacher and apprentices policy networks.

Acknowledgements Shi Yuan Tang acknowledges support from the Alibaba Group and the Alibaba-NTU Singapore Joint Research Institute.

References

1. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems 27 (NIPS)*.
2. Jin, Y., Zhang, J., Li, M., Tian, Y., & Zhu, H. (2017). Towards the high-quality anime characters generation with generative adversarial networks. In *Proceedings of the machine learning for creativity and design workshop at NIPS*.
3. Chen, Y., Shi, F., Christodoulou, A.G., Xie, Y., Zhou, Z., & Li, D. (2018). Efficient and accurate MRI super-resolution using a generative adversarial network and 3D multi-level densely connected network. In *International conference on medical image computing and computer-assisted intervention* (pp. 91–99). Springer.
4. Zhou, H., Cai, R., Quan, T., Liu, S., Li, S., Huang, Q., Ertürk, A., & Zeng, S. (2020). 3d high resolution generative deep-learning network for fluorescence microscopy imaging. *Optics Letters, 45*(7), 1695–1698.
5. Zhang, S., Wang, L., Chang, C., Liu, C., Zhang, L., & Cui, H. (2020). An image denoising method based on BM4D and GAN in 3D shearlet domain. *Mathematical Problems in Engineering, 2020*, 1–11.
6. Li, C., & Wand, M. (2016). Precomputed real-time texture synthesis with Markovian generative adversarial networks. In *European conference on computer vision* (pp 702–716). Springer.
7. Kumar, K., Kumar, R., de Boissiere, T., Gestin, L., Teoh, W. Z., Sotelo, J., de Brébisson, A., Bengio, Y., & Courville, A. C. (2019). Melgan: Generative adversarial networks for conditional waveform synthesis. In *Advances in neural information processing systems 32*.
8. Latifi, S., & Torres-Reyes, N. (2019). Audio enhancement and synthesis using generative adversarial networks: A survey. *International Journal of Computer Applications, 182*(35), 27.
9. Croce, D., Castellucci, G., & Basili, R. (2020). Gan-bert: Generative adversarial learning for robust text classification with a bunch of labeled examples. In *Proceedings of the 58th annual meeting of the association for computational linguistics* (pp. 2114–2119).
10. Hu, Z., Luo, F., Tan, Y., Zeng, W., & Sui, Z. (2019). WSD-GAN: Word sense disambiguation using generative adversarial networks. In *Proceedings of the AAAI conference on artificial intelligence* (vol. 33, pp. 9943–9944).
11. Mokhayeri, F., Kamali, K., & Granger, E. (2020). Cross-domain face synthesis using a controllable GAN. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision* (pp. 252–260).

12. Spick, R., Demediuk, S., & Alfred Walker, J. (2020). Naive mesh-to-mesh coloured model generation using 3D GANs. In *Proceedings of the Australasian computer science week multiconference* (pp. 1–6).
13. Gao, R., Xia, H., Li, J., Liu, D., Chen, S., & Chun, G. (2019) DRCGR: Deep reinforcement learning framework incorporating CNN and GAN-based for interactive recommendation. In *2019 IEEE international conference on data mining (ICDM)* (pp. 1048–1053). IEEE.
14. Tian, Y., Wang, Q., Huang, Z., Li, W., Dai, D., Yang, M., Wang, J., & Fink, O. (2020). Off-policy reinforcement learning for efficient and effective GAN architecture search. In *European conference on computer vision* (pp. 175–192). Springer.
15. Wang, Q., Ji, Y., Hao, Y., & Cao, J. (2020). GRL: Knowledge graph completion with GAN-based reinforcement learning. *Knowledge-Based Systems, 209*, 106421.
16. Sandfort, V., Yan, K., Pickhardt, P. J., & Summers, R. M. (2019). Data augmentation using generative adversarial networks (cycleGAN) to improve generalizability in CT segmentation tasks. *Scientific Reports, 9*(1), 1–9.
17. Hans, A., & Udluft, S. (2011). Ensemble usage for more reliable policy identification in reinforcement learning. In *ESANN*.
18. Duell, S., & Udluft, S. (2013). Ensembles for continuous actions in reinforcement learning. In *ESANN*.
19. Elliott, D., Santosh, K., & Anderson, C. (2020). Gradient boosting in crowd ensembles for Q-learning using weight sharing. *International Journal of Machine Learning and Cybernetics, 11*, 2275–2287.
20. Ha, D., Dai, A. M., & Le, Q. V. (2017). Hypernetworks. In *International conference on learning representations (ICLR)*.
21. Tang, S. Y., Irissappane, A. A., Oliehoek, F. A., & Zhang, J. (2021). Learning complex policy distribution with CEM guided adversarial hypernetwork. In *AAMAS* (pp. 1308–1316).
22. von Oswald, J., Henning, C., Sacramento, J., & Grewe, B. F. (2020). Continual learning with hypernetworks. In *International conference on learning representations (ICLR)*.
23. Louizos, C., & Welling, M. (2017). Multiplicative normalizing flows for variational bayesian neural networks. In *International conference on machine learning (ICML),* (pp. 2218–2227).
24. Pawlowski, N., Rajchl, M., & Glocker, B. (2017). Implicit weight uncertainty in neural networks. In *Bayesian deep learning workshop, advances in neural information processing systems (NIPS)*.
25. Blundell, C., Cornebise, J., Kavukcuoglu, K., & Wierstra, D. (2015). Weight uncertainty in neural network. In *International conference on machine learning (ICML)* (pp. 1613–1622).
26. Pourchot, A., & Sigaud, O. (2018). Cem-rl: Combining evolutionary and gradient-based methods for policy search. arXiv preprint arXiv:1810.01222.
27. Mannor, S., Rubinstein, R. Y., & Gat, Y. (2003). The cross entropy method for fast policy search. In *International conference on machine learning (ICML)* (pp. 512–519).
28. Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V, & Levine S. (2018). Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on robot learning (CoRL)* (pp. 651–673).
29. Simmons-Edler, R., Eisner, B., Mitchell, E., Seung, S., & Lee, D. (2019). Q-learning for continuous actions with cross-entropy guided policies. In *RL4RealLife workshop, international conference on machine learning (ICML)*.
30. Galanti, T., & Wolf, L. (2020). On the modularity of hypernetworks. *Advances in Neural Information Processing Systems, 33*, 10409–10419.
31. Zhang, C., Ren, M., & Urtasun, R. (2018). Graph hypernetworks for neural architecture search. In *International Conference on Learning Representations*.
32. Brock, A., Lim, T., Ritchie, J., & Weston, N. (2018). Smash: One-shot model architecture search through hypernetworks. In *International conference on learning representations*.
33. Navon, A., Shamsian, A., Fetaya, E., & Chechik, G. (2020). Learning the pareto front with hypernetworks. In *International conference on learning representations*.
34. Henning, C., von Oswald, J., Sacramento, J., Surace, S. C., Pfister, J. -P., & Grewe, B. F. (2018). Approximating the predictive distribution via adversarially-trained hypernetworks. In *Bayesian deep learning workshop, advances in neural information processing systems (NeurIPS)*.
35. Skorokhodov, I., Ignatyev, S., & Elhoseiny, M. (2021). Adversarial generation of continuous images. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 10753–10764).
36. Buciluǎ, C., Caruana, R., & Niculescu-Mizil, A. (2006). Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 535–541).
37. Hinton, G., Vinyals, O., & Dean, J. et al. (2015). Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531
38. Adriana, R., Nicolas, B., Ebrahimi, K. S., Antoine, C., Carlo, G., & Yoshua, B. (2015). Fitnets: Hints for thin deep nets. In *Proc. ICLR* (pp. 1–13).

39. Yim, J., Joo, D., Bae, J., & Kim, J. (2017). A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4133–4141).
40. Lee, S. H., Kim, D. H., & Song, B. C. (2018). Self-supervised knowledge distillation using singular value decomposition. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 335–350).
41. Komodakis, N., & Zagoruyko, S. (2017). Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *ICLR*.
42. Kim, J., Park, S., & Kwak, N. (2018). Paraphrasing complex network: Network compression via factor transfer. In *Advances in neural information processing systems 31*.
43. Sun, D., Yao, A., Zhou, A., & Zhao, H. (2019). Deeply-supervised knowledge synergy. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 6997–7006).
44. Tian, Y., Krishnan, D., & Isola, P. (2019). Contrastive representation distillation. In *International conference on learning representations*.
45. Zhang, Y., Xiang, T., Hospedales, T. M., & Lu, H. (2018). Deep mutual learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4320–4328).
46. Ho, J., & Ermon, S. (2016). Generative adversarial imitation learning. In *Advances in neural information processing systems 29*.
47. Li, Y., Song, J., & Ermon, S. (2017). Infogail: Interpretable imitation learning from visual demonstrations. In *Advances in neural information processing systems 30*.
48. Fei, C., Wang, B., Zhuang, Y., Zhang, Z., Hao, J., Zhang, H., Ji, X., & Liu, W. (2020). Triple-gail: A multi-modal imitation learning framework with generative adversarial nets. In: Bessiere, C. (ed.) Proceedings of the twenty-ninth international joint conference on artificial intelligence, IJCAI-20, pp. 2929–2935. International Joint Conferences on Artificial Intelligence Organization. https://doi.org/10.24963/ijcai.2020/405. Main track.
49. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: an introduction*. MIT press.
50. Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning, 8*(3–4), 229–256.
51. Faury, L., Calauzenes, C., Fercoq, O., & Krichen, S. (2019). Improving evolutionary strategies with generative neural networks. arXiv preprint arXiv:1901.11271.
52. Schwefel, H.-P. (1981). *Numerical optimization of computer models*. John Wiley & Sons Inc.
53. Kurtz, N., & Song, J. (2013). Cross-entropy-based adaptive importance sampling using Gaussian mixture. *Structural Safety, 42*, 35–44.
54. Geyer, S., Papaioannou, I., & Straub, D. (2019). Cross entropy-based importance sampling using Gaussian densities revisited. *Structural Safety, 76*, 15–27.
55. Deutsch, L. (2018). Generating neural networks with neural networks. arXiv preprint arXiv:1801.01952.
56. Ukai, K., Matsubara, T., & Uehara, K. (2018). Hypernetwork-based implicit posterior estimation and model averaging of CNN. In *Asian conference on machine learning* (pp. 176–191).
57. Roth, K., Lucchi, A., Nowozin, S., & Hofmann, T. (2017). Stabilizing training of generative adversarial networks through regularization. In *Advances in neural information processing systems (NIPS)* (pp. 2018–2028).
58. Wiering, M. A., & Van Hasselt, H. (2008). Ensemble algorithms in reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 38*(4), 930–936.
59. Chevalier-Boisvert, M., Willems, L., & Pal, S. (2018). Minimalistic Gridworld Environment for OpenAI Gym. GitHub.
60. Ellenberger, B. (2018). Pybullet Gymperium, Open-source implementations of OpenAI Gym MuJoCo environments. GitHub.
61. Sung, J.-c. (2018) Benchmark results for TD3 and DDPG using the PyBullet reinforcement learning environments. GitHub.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Shi Yuan Tang[1,2] · Athirai A. Irissappane[3] · Frans A. Oliehoek[4] · Jie Zhang[1]

Athirai A. Irissappane
athirai@uw.edu

Frans A. Oliehoek
f.a.oliehoek@tudelft.nl

Jie Zhang
zhangj@ntu.edu.sg

[1] School of Computer Science and Engineering, Nanyang Technological University, Singapore, Singapore

[2] Alibaba-NTU Singapore Joint Research Institute, Singapore, Singapore

[3] Computer Science Department, University of Washington, Seattle, USA

[4] Department of Intelligent Systems, Delft University of Technology, Delft, Netherlands