Recomposing the Reinforcement Learning Building Blocks with Hypernetworks

Elad Sarafian* [1]   Shai Keynan* [1]   Sarit Kraus [1]

Abstract

The Reinforcement Learning (RL) building blocks, i.e. Q-functions and policy networks, usually take elements from the cartesian product of two domains as input. In particular, the input of the Q-function is both the state and the action, and in multi-task problems (Meta-RL) the policy can take a state and a context. Standard architectures tend to ignore these variables' underlying interpretations and simply concatenate their features into a single vector. In this work, we argue that this choice may lead to poor gradient estimation in actor-critic algorithms and high variance learning steps in Meta-RL algorithms. To consider the interaction between the input variables, we suggest using a Hypernetwork architecture where a primary network determines the weights of a conditional dynamic network. We show that this approach improves the gradient approximation and reduces the learning step variance, which both accelerates learning and improves the final performance. We demonstrate a consistent improvement across different locomotion tasks and different algorithms both in RL (TD3 and SAC) and in Meta-RL (MAML and PEARL).

1. Introduction

The rapid development of deep neural-networks as general-purpose function approximators has propelled the recent Reinforcement Learning (RL) renaissance (Zai and Brown, 2020). RL algorithms have progressed in robustness, e.g. from (Lillicrap et al., 2016) to (Fujimoto et al., 2018); exploration (Haarnoja et al., 2018); gradient sampling (Schulman et al., 2017; 2015a); and off-policy learning (Fujimoto et al.,
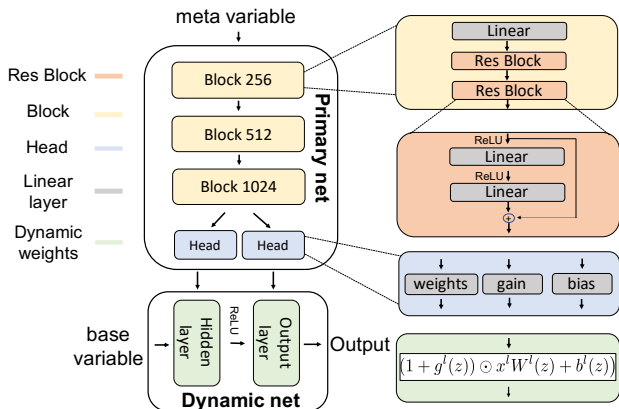

Figure 1. The Hypernetwork architecture

2019; Kumar et al., 2019). Many actor-critic algorithms have focused on improving the critic learning routines by modifying the target value (Hasselt et al., 2016), which enables more accurate and robust Q-function approximations. While this greatly improves the policy optimization efficiency, the performance is still bound by the networks' ability to represent Q-functions and policies. Such a constraint calls for studying and designing neural models suited for the representation of these RL building blocks.

A critical insight in designing neural models for RL is the reciprocity between the state and the action, which both serve as the input for the Q-function. At the start, each input can be processed individually according to its source domain. For example, when s is a vector of images, it is common to employ CNN models (Kaiser et al., 2019), and when s or a are natural language words, each input can be processed separately with embedding vectors (He et al., 2016). The common practice in incorporating the state and action learnable features into a single network is to concatenate the two vectors and follow with MLP to yield the Q-value (Schulman et al., 2017). In this work, we argue that for actor-critic RL algorithms (Grondman et al., 2012), such an off-the-shelf method could be significantly improved with Hypernetworks.

In actor-critic methods, for each state, sampled from the

---

*Equal contribution: authors' order was randomly selected [1]Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel. Correspondence to: Elad Sarafian, Shai Keynan <elad.sarafian@gmail.com, shai.keynan@gmail.com>.

dataset distribution, the actor's task is to solve an optimization problem over the action distribution, i.e. the policy. This motivates an architecture where the Q-function is explicitly modeled as the value function of a contextual bandit (Lattimore and Szepesvári, 2020) $Q^\pi(s,a) = Q_s^\pi(a)$ where s is the context. While standard architectures are not designed to model such a relationship, Hypernetworks were explicitly constructed for that purpose (Ha et al., 2016). Hypernetworks, also called meta-networks, can represent hierarchies by transforming a *meta* variable into a context-dependent function that maps a *base* variable to the required output space. This emphasizes the underlying dynamic between the meta and base variables and has found success in a variety of domains such as Bayesian neural-networks (Lior Deutsch, 2019), continual learning (von Oswald et al., 2019), generative models (Ratzlaff and Li, 2019) and adversarial defense (Sun et al., 2017). The practical success has sparked interest in the theoretical properties of Hypernetworks. For example, it has recently been shown that they enjoy better parameter complexity than classical models which concatenate the base and meta-variables together (Galanti and Wolf, 2020a;b).

When analyzing the critic's ability to represent the Q-function, it is important to notice that in order to optimize the policy, modern off-policy actor-critic algorithms (Fujimoto et al., 2018; Haarnoja et al., 2018) utilize only the parametric neural gradient of the critic with respect to the action input, i.e., $\nabla_a Q_\theta^\pi(s,a)$.[1] Recently, (Ilyas et al., 2019) examined the accuracy of the policy gradient in on-policy algorithms. They demonstrated that standard RL implementations achieve gradient estimation with a near-zero cosine similarity when compared to the "true" gradient. Therefore, recovering better gradient approximations has the potential to substantially improve the RL learning process. Motivated by the need to obtain high-quality gradient approximations, we set out to investigate the gradient accuracy of Hypernetworks with respect to standard models. In Sec. 3 we analyze three critic models and find that the Hypernetwork model with a state as a meta-variable enjoys better gradient accuracy which translates into a faster learning rate.

Much like the induced hierarchy in the critic, meta-policies that optimize multi-task RL problems have a similar structure as they combine a task-dependent context and a state input. While some algorithms like MAML (Finn et al., 2017) and LEO (Rusu et al., 2019) do not utilize an explicit context, other works, e.g. PEARL (Rakelly et al., 2019) or MQL (Fakoor et al., 2019), have demonstrated that a context improves the generalization abilities. Recently, (Jayakumar et al., 2019) have shown that Multiplicative Interactions (MI) are an excellent design choice when combining states

and contexts. MI operations can be viewed as shallow Hypernetwork architectures. In Sec. 4, we further explore this approach and study context-based meta-policies with *deep* Hypernetworks. We find that with Hypernetworks, the task and state-dependent gradients are disentangled s.t. the state-dependent gradients are marginalized out, which leads to an empirically lower learning step variance. This is specifically important in on-policy methods such as MAML, where there are fewer optimization steps during training.

The contributions of this paper are three-fold. First, in Sec. 3 we provide a theoretical link between the Q-function gradient approximation quality and the allowable learning rate for monotonic policy improvement. Next, we show empirically that Hypernetworks achieve better gradient approximations which translates into a faster learning rate and improves the final performance. Finally, in Sec. 4 we show that Hypernetworks significantly reduce the learning step variance in Meta-RL. We summarize our empirical results in Sec. 5, which demonstrates the gain of Hypernetworks both in single-task RL and Meta-RL. Importantly, we find empirically that Hypernetwork policies eliminate the need for the MAML adaptation step and improve the Out-Of-Distribution generalization in PEARL.

2. Hypernetworks

A Hypernetwork (Ha et al., 2016) is a neural-network architecture designed to process a tuple $(z, x) \in Z \times X$ and output a value $y \in Y$. It is comprised of two networks, a *primary* network $w_\theta : Z \to \mathbb{R}^{n_w}$ which produces weights $w_\theta(z)$ for a *dynamic* network $f_{w_\theta(z)} : X \to Y$. Both networks are trained together, and the gradient flows through f to the primary networks' weights θ. During test time or inference, the primary weights are fixed while the z input determines the dynamic network's weights.

The idea of learnable context-dependent weights can be traced back to (McClelland, 1985; Schmidhuber, 1992). However, only in recent years have Hypernetworks gained popularity when they have been applied successfully with many dynamic network models, e.g. recurrent networks (Ha et al., 2016), MLP networks for 3D point clouds (Littwin and Wolf, 2019), spatial transformation (Potapov et al., 2018), convolutional networks for video frame prediction (Jia et al., 2016) and few-shot learning (Brock et al., 2018). In the context of RL, Hypernetworks were also applied, e.g., in QMIX (Rashid et al., 2018) to solve Multi-agent RL tasks and for continual model-based RL (Huang et al., 2020).

Fig. 1 illustrates our Hypernetwork model. The primary network $w_\theta(z)$ contains residual blocks (Srivastava et al., 2015) which transform the meta-variable into a 1024 sized latent representation. This stage is followed by a series of parallel linear transformations, termed "heads", which

---

[1]This is in contrast to the REINFORCE approach (Williams, 1992) based on the policy gradient theorem (Sutton et al., 2000) which does not require a differentiable Q-function estimation.

output the sets of dynamic weights. The dynamic network $f_{w_\theta(z)}(x)$ contains only a single hidden layer of 256 which is smaller than the standard MLP architecture used in many RL papers (Fujimoto et al., 2018; Haarnoja et al., 2018) of 2 hidden layers, each with 256 neurons. The computational model of each dynamic layer is

$$x^{l+1} = \sigma_{ReLU}\left((1 + g^l(z)) \odot x^l W^l(z) + b^l(z)\right) \quad (1)$$

where the non-linearity is applied only over the hidden layer and g^l is an additional gain parameter that is required in Hypernetwork architectures (Littwin and Wolf, 2019). We defer the discussion of these design choices to Sec. 5.

3. Recomposing the Actor-Critic's Q-Function

3.1. Background

Reinforcement Learning concerns finding optimal policies in Markov Decision Processes (MDPs). An MDP (Dean and Givan, 1997) is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R)$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P}$ is a set of probabilities to switch from a state s to s' given an action a, and $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a scalar reward function. The objective is to maximize the expected discounted sum of rewards with a discount factor $\gamma > 0$

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| a_t \sim \pi(\cdot|s_t)\right]. \quad (2)$$

$J(\pi)$ can also be written, up to a constant factor $1 - \gamma$, as an expectation over the Q-function

$$J(\pi) = \mathbb{E}_{s \sim d^\pi}\left[\mathbb{E}_{a \sim \pi(\cdot|s)}\left[Q^\pi(s, a)\right]\right], \quad (3)$$

where the Q-function is the expected discounted sum of rewards following visitation at state s and execution of action a (Sutton and Barto, 2018), and d^π is the state distribution induced by policy π.

Actor-critic methods maximize $J(\pi)$ over the space of parameterized policies. Stochastic policies are constructed as a state dependent transformation of an independent random variable

$$\pi_\phi(a|s) = \mu_\phi(\varepsilon|s)\text{s.t.}\varepsilon \sim p_\varepsilon, \quad (4)$$

where p_ε is a predefined multivariate distribution over $\mathbb{R}^{n_a}$ and n_a is the number of actions.[2] To maximize $J(\pi_\phi)$ over the ϕ parameters, actor-critic methods operate with an iterative three-phase algorithm. First, they collect into a replay buffer $\mathcal{D}$ the experience tuples (s, a, r, s') generated with the parametric π_ϕ and some additive exploration noise policy (Zhang and Sutton, 2017). Then they fit a critic which is

---

[2]Deterministic policies, on the other hand, are commonly defined as a deterministic transformation of the state's feature vector.

a parametric model Q_θ^π for the Q-function. For that purpose, they apply TD-learning (Sutton and Barto, 2018) with the loss function

$$\mathcal{L}_{critic}(\theta) =$$
$$\mathbb{E}_{s,a,r,s' \sim \mathcal{D}}\left[\left|Q_\theta^\pi(s,a) - r - \gamma \mathbb{E}_{a' \sim \pi_\phi(\cdot|s')}[Q_{\bar{\theta}}^\pi(s',a')]\right|^2\right],$$

where $\bar{\theta}$ is a lagging set of parameters (Lillicrap et al., 2016). Finally, they apply gradient descent updates in the direction of an off-policy surrogate of $J(\pi_\phi)$

$$\phi \leftarrow \phi + \eta \nabla_\phi J_{actor}(\phi)$$
$$\nabla_\phi J_{actor}(\phi) = \mathbb{E}_{\left\{\substack{s \sim \mathcal{D} \\ \varepsilon \sim p_\varepsilon}\right\}}\left[\nabla_\phi \mu_\phi(\varepsilon|s) \nabla_a Q_\theta^\pi(s, \mu_\phi(\varepsilon|s))\right].$$
$$(5)$$

Here, $\nabla_\phi \mu_\phi(\varepsilon|s)$ is a matrix of size $n_\phi \times n_a$ where n_ϕ is the number of policy parameters to be optimized.

Two well-known off-policy algorithms are TD3 (Fujimoto et al., 2018) and SAC (Haarnoja et al., 2018). TD3 optimizes deterministic policies with additive normal exploration noise and double Q-learning to improve the robustness of the critic part (Hasselt et al., 2016). On the other hand, SAC adopts stochastic, normally distributed policies but it modifies the reward function to include a high entropy bonus $\tilde{R}(s,a) = R(s,a) + \alpha H(\pi(\cdot|s))$ which eliminates the need for exploration noise.

3.2. Our Approach

The gradient of the off-policy surrogate $\nabla_\phi J_{actor}(\phi)$ differs from the true gradient $\nabla_\phi J(\pi)$ in two elements: First, the distribution of states is the empirical distribution in the dataset and not the policy distribution d^π; and second, the Q-function gradient is estimated with the critic's parametric neural gradient $\nabla_a Q_\theta^\pi \simeq \nabla_a Q^\pi$. Avoiding a distribution mismatch is the motivation of many constrained policy improvement methods such as TRPO and PPO (Schulman et al., 2015a; 2017). However, it requires very small and impractical steps. Thus, many off-policy algorithms ignore the distribution mismatch and seek to maximize only the empirical advantage

$$A(\phi', \phi) = \mathbb{E}_{s \sim \mathcal{D}}\left[\mathbb{E}_{a \sim \pi'}\left[Q^\pi(s,a)\right] - \mathbb{E}_{a \sim \pi}\left[Q^\pi(s,a)\right]\right].$$

In practice, a positive empirical advantage is associated with better policies and is required by monotonic policy improvement methods such as TRPO (Kakade and Langford, 2002; Schulman et al., 2015a). Yet, finding positive empirical advantage policies requires a good approximation of the gradient $\nabla_a Q^\pi$. The next proposition suggests that with a sufficiently accurate approximation, applying the gradient step as formulated in the actor update in Eq. (5) yields positive empirical advantage policies.

Proposition 1. *Let* $\pi(a|s) = \mu_\phi(\varepsilon|s)$ *be a stochastic parametric policy with* $\varepsilon \sim p_\varepsilon$, *and* $\mu_\phi(\cdot|s)$ *a*

transformation with a Lipschitz continuous gradient and a Lipschitz constant κ_μ. Assume that its Q-function $Q^\pi(s,a)$ has a Lipschitz continuous gradient in a, i.e. $|\nabla_a Q^\pi(s,a_1) - \nabla_a Q^\pi(s,a_2)| \leq \kappa_q \|a_1 - a_2\|$. Define the average gradient operator $\overline{\nabla}_\phi \cdot f = \mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{\varepsilon\sim p_\varepsilon}[\nabla_\phi \mu_\phi(\varepsilon|s) \cdot f(s,\mu_\phi(\varepsilon|s))]\right]$. If there exists a gradient estimation $g(s,a)$ and $0 < \alpha < 1$ s.t.

$$\|\overline{\nabla}_\phi \cdot g - \overline{\nabla}_\phi \cdot \nabla_a Q^\pi\| \leq \alpha \|\overline{\nabla}_\phi \cdot \nabla_a Q^\pi\| \qquad (6)$$

then the ascent step $\phi' \leftarrow \phi + \eta \overline{\nabla}_\phi \cdot g$ with $\eta \leq \frac{1}{\tilde{k}}\frac{1-\alpha}{(1+\alpha)^2}$ yields a positive empirical advantage policy.

We define $\tilde{k}$ and provide the proof in the appendix. It follows that a positive empirical advantage can be guaranteed when the gradient of the Q-function is sufficiently accurate, and with better gradient models, i.e. smaller α, one may apply larger ascent steps. However, instead of fitting the gradient, actor-critic algorithms favor modeling the Q-function and estimate the gradient with the parametric gradient of the model $\nabla_a Q_\theta^\pi$. It is not obvious whether better models for the Q-functions, with lower Mean-Squared-Error (MSE), provide better gradient estimation. A more direct approach could be to explicitly learn the gradient of the Q-function (Sarafian et al., 2020; Saremi, 2019); however, in this work, we choose to explore which architecture recovers more accurate gradient approximation based on the parametric gradient of the Q-function model.

We consider three alternative models:

1. MLP network, where state features $\xi(s)$ (possibly learnable) are concatenated into a single input of a multi-layer linear network.

2. Action-State Hypernetwork (AS-Hyper) where the actions are the *meta* variable, input of the primary network w, and the state features are the *base* variable, input for the dynamic network f.

3. State-Action Hypernetwork (SA-Hyper), which reverses the order of AS-Hyper.

To develop some intuition, let us first consider the simplest case where the dynamic network has a single linear layer and the MLP model is replaced with a plain linear model. Starting with the linear model, the Q-function and its gradient take the following parametric model:

$$\begin{aligned} Q_\theta^\pi(s,a) &= [w_s, w_a] \cdot [\xi(s), a] \\ \nabla_a Q_\theta^\pi(s,a) &= w_a \end{aligned} \qquad (7)$$

where $\theta = [w_s, w_a]$. Clearly, in this case, the gradient is not a function of the state, therefore it is impossible to exploit this model for actor-critic algorithms. For the AS-Hyper we obtain the following model

$$\begin{aligned} Q_\theta^\pi(s,a) &= w(a) \cdot \xi(s) \\ \nabla_a Q_\theta^\pi(s,a) &= \nabla_a w(a)\xi(s) \end{aligned} \qquad (8)$$
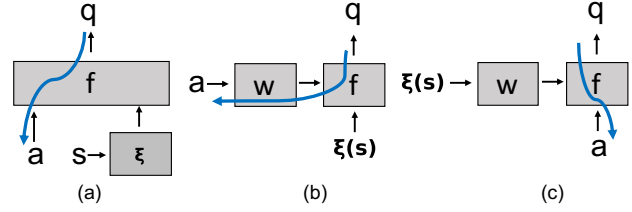

Figure 2. Illustrating three alternatives for combining states and actions: (a) MLP; (b) AS-Hyper; and (c) SA-Hyper. The blue arrows represent the backpropagation calculation of the actions gradient. Notice that in the SA-Hyper, the gradient flows only through the dynamic network, which enables more efficient implementation as the dynamic network is much smaller than the primary network.

Usually, the state feature vector $\xi(s)$ has a much larger dimension than the action dimension n_a. Thus, the matrix $\nabla_a w(a)$ has a large null-space which can potentially hamper the training as it may yield zero or near-zero gradients even when the true gradient exists.

On the other hand, the SA-Hyper formulation is

$$\begin{aligned} Q_\theta^\pi(s,a) &= w(s) \cdot a \\ \nabla_a Q_\theta^\pi(s,a) &= w(s) \end{aligned} \qquad (9)$$

which is a state-dependent constant model of the gradient in a. While it is a relatively naive model, it is sufficient for localized policies with low variance as it approximates the tangent hyperplane around the policy mean value.

Moving forward to a multi-layer architecture, let us first consider the AS-Hyper architecture. In this case the gradient is $\nabla_a Q_\theta^\pi(s,a) = \nabla_a w(a)\nabla_w f_w(s)$. We see that the problem of the single layer is exacerbated since $\nabla_a w(a)$ is now a $n_a \times n_w$ matrix where $n_w \gg n_a$ is the number of dynamic network weights.

Next, the MLP and SA-Hyper models can be jointly analyzed. First, we calculate the input's gradient of each layer

$$x^{l+1} = f^l(x^l) = \sigma\left(x^l W^l + b^l\right) \qquad (10)$$

$$\nabla_a x^{l+1} = (\nabla_a x^l)\nabla_{x^l} f^l(x^l) = (\nabla_a x^l)W^l \Lambda^l(x^l) \quad (11)$$

$$\Lambda^l(x^l) = \mathrm{diag}\left(\sigma'\left(x^l W^l + b^l\right)\right), \qquad (12)$$

where σ is the activation function and W^l and b^l are the weights and biases of the l-th layer, respectively. By the chain rule, the input's gradient of an L-layers network is the product of these expressions. For the MLP model we obtain

$$\nabla_a Q_\theta^\pi(s,a) = W^a \Lambda^1(s,a)\left(\prod_{l=2}^{L-1} W^l \Lambda^l(s,a)\right)W^L. \qquad (13)$$
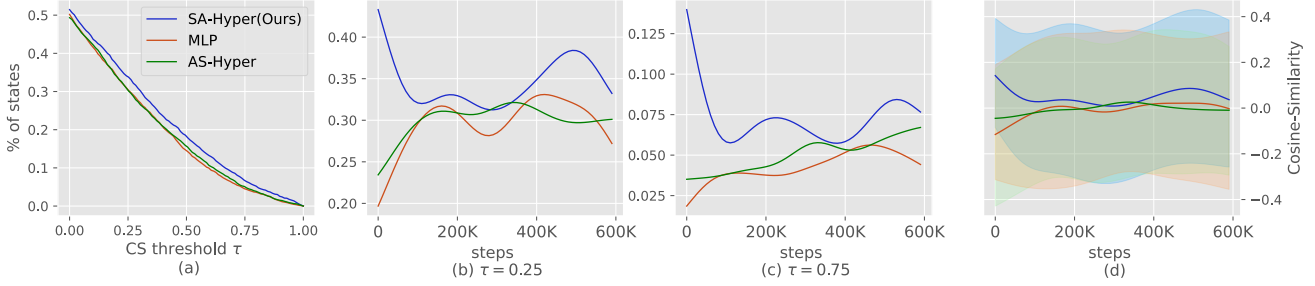
Figure 3. Comparing the Cosine-Similarity of different critic models: (a) The percentage of states with CS better than a τ threshold. (b-c) The percentage of states with CS better than $\tau = 0.25$ and $\tau = 0.75$ with respect to the learning step. (d) The mean CS over time averaged over all seeds and environments. The shaded area is the interquartile range $Q3 - Q1$. In all cases, the CS was evaluated every $10K$ steps with $N_s = 15$ states and $N_r = 15$ independent trajectories for each state.

On the other hand, in SA-Hyper the weights are the outputs of the primary network, thus we have

$$\nabla_a Q_\theta^\pi(s, a) = $$
$$W^1(s)\Lambda^1(s, a)\left(\prod_{l=2}^{L-1} W^l(s)\Lambda^l(s, a)\right)W^L(s). \quad (14)$$

Importantly, while the SA-Hyper's gradient configuration is controlled via the state-dependent matrices $W^l(s)$, in the MLP model, it is a function of the state only via the diagonal elements in $\Lambda^l(s, a)$. These local derivatives of the non-linear activation functions are usually piecewise constant when the activations take the form of ReLU-like functions. Also, they are required to be bounded and smaller than one in order to avoid exploding gradients during training (Philipp et al., 2017). These restrictions significantly reduce the expressiveness of the parametric gradient and its ability to model the true Q-function gradient. For example, with ReLU, for two different pairs (s_1, a_1) and (s_2, a_2) the estimated gradient is equal if they have same active neurons map (i.e. the same ReLUs are in the active mode). Following this line of reasoning, we postulate that the SA-Hyper configuration should have better gradient approximations.

Empirical analysis To test our hypothesis, we trained TD3 agents with different network models and evaluated their parametric gradient $\nabla_a Q_\theta(s, a)$. To empirically analyze the gradient accuracy, we opted to estimate the true Q-function gradient with a non-parametric local estimator at the policy mean value, i.e. at $a_\mu = \mathbb{E}_{\varepsilon \sim p_\varepsilon}[\mu_\phi(\varepsilon|s)]$. For that purpose, we generated N_r independent trajectories with actions sampled around the mean value, i.e. $a = a_\mu + \Delta_a$, and fit with a Least-Mean-Square (LMS) estimator a linear model for the empirical return of the sampled trajectories. The "true" gradient is therefore the linear model's gradient. Additional technical details of this estimator are found in the appendix.

As our Q-function estimator is based on Temporal-Difference (TD) learning, it bears bias. Hence, in practice we cannot hope to reconstruct the true Q-function scale.

Thus, instead of evaluating the gradient's MSE, we take the Cosine Similarity (CS) as a surrogate for measuring the gradient accuracy.

$$cs(Q_\theta^\pi) = \mathbb{E}_{s \sim \mathcal{D}}\left[\frac{\nabla_a Q_\theta^\pi(s, a_\mu) \cdot \nabla_a Q^\pi(s, a_\mu)}{\|\nabla_a Q_\theta^\pi(s, a_\mu)\| \, \|\nabla_a Q^\pi(s, a_\mu)\|}\right],$$

Fig. 3 summarizes our CS evaluations with the three model alternatives averaged over 4 Mujoco (Todorov et al., 2012) environments. Fig. 3d presents the mean CS over states during the training process. Generally, the CS is very low, which indicates that the RL training is far from optimal. While this finding is somewhat surprising, it corroborates the results in (Ilyas et al., 2019) which found near-zero CS in policy gradient algorithms. Nevertheless, note that the impact of the CS accuracy is cumulative as in each gradient ascent step the policy accumulates small improvements. This lets even near-zero gradient models improve over time. Overall, we find that the SA-Hyper CS is higher, and unlike other models, it is larger than zero during the entire training process. The SA-Hyper advantage is specifically significant at the first $100K$ learning steps, which indicates that SA-Hyper learns faster in the early learning stages.

Assessing the gradient accuracy by the average CS can be somewhat confounded by states that have reached a local equilibrium during the training process. In these states the true gradient has zero magnitude s.t. the CS is ill-defined. For that purpose, in Fig. 3a-c we measure the percentage of states with a CS higher than a threshold τ. This indicates how many states are *learnable* where more learnable states are attributed to a better gradient estimation. Fig. 3a shows that for all thresholds $\tau \in [0, 1]$ SA-Hyper has more learnable states, and Fig. 3b-c present the change in learnable states for different τ during the training process. Here we also find that the SA-Hyper advantage is significant particularly at the first stage of training. Finally, Fig. 4 demonstrates how gradient accuracy translates to better learning curves. As expected, we find that SA-Hyper outperforms both the MLP architecture and the AS-Hyper configuration which is also generally inferior to MLP.
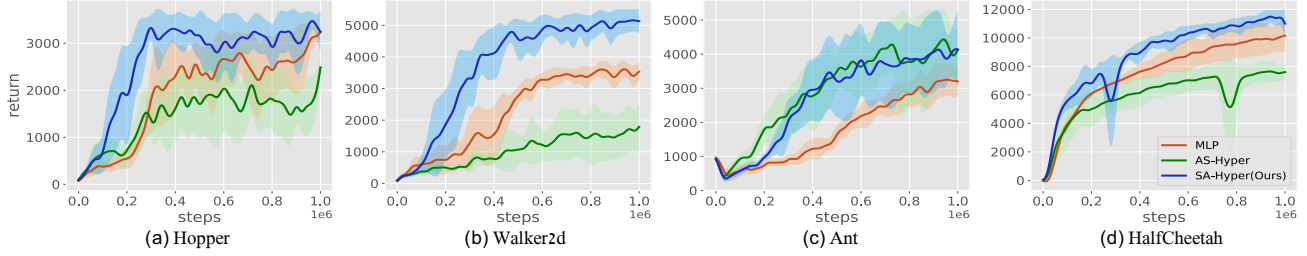
Figure 4. Learning curves of the TD3 algorithm with different critic models. SA-Hyper refers to $Q_\theta^\pi = f_{w_\theta(s)}(a)$, AS-Hyper refers to $Q_\theta^\pi = f_{w_\theta(a)}(s)$ and MLP refers to $Q_\theta^\pi = f_\theta(s, a)$, which concatenates both inputs.

In the next section, we discuss the application of Hypernetworks in Meta-RL for modeling context conditional policies. When such a context exists, it also serves as an input variable to the Q-function. In that case, when modeling the critic with a Hypernetwork, one may choose to use the context as a meta-variable or alternatively as a base variable. Importantly, when the context is the dynamic's input, the dynamic weights are fixed for each state, regardless of the task. In our PEARL experiments in Sec. 5 we always used the context as a base variable of the critic. We opted for this configuration since: (1) we found empirically that it is important for the generalization to have a constant set of weights for each state; and (2) As the PEARL context is learnable, we found that when the context gradient backpropagates through three networks (primary, dynamic and the context network), it hampers the training. Instead, as a base variable, the context's gradient backpropagates only via two networks as in the original PEARL implementation.

4. Recomposing the Policy in Meta-RL

4.1. Background

Meta-RL is the generalization of Meta-Learning (Mishra et al., 2018; Sohn et al., 2019) to the RL domain. It aims at learning meta-policies that solve a distribution of different tasks $p(\mathcal{T})$. Instead of learning different policies for each task, the meta-policy shares weights between all tasks and thus can generalize from one task to the other (Sung et al., 2017). A popular Meta-RL algorithm is MAML (Finn et al., 2017), which learns a set of weights that can quickly adapt to a new task with a few gradient ascent steps. To do so, for each task, it estimates the policy gradient (Sutton et al., 2000) at the adaptation point. The total gradient is the sum of policy gradients over the task distribution $p(\mathcal{T})$:

$$\nabla_\phi J_{maml}(\phi) = \mathbb{E}_{\left\{\substack{\mathcal{T}_i \sim p(\mathcal{T}) \\ \pi_{\phi_i}}\right\}} \left[\sum_{t=0}^{\infty} \hat{A}_{i,t} \nabla_\phi \log \pi_{\phi_i}(a_t|s_t) \right]$$

$$\phi_i = \phi + \eta \mathbb{E}_{\pi_\phi} \left[\sum_{t=0}^{\infty} \hat{A}_{i,t} \nabla_\phi \log \pi_{\phi_i}(a_t|s_t) \right], \quad (15)$$

where $\hat{A}_{i,t}$ is the empirical advantage estimation at the t-th step in task i (Schulman et al., 2015b). On-policy algorithms tend to suffer from high sample complexity as each update step requires many new trajectories sampled from the most recent policy in order to adequately evaluate the gradient direction.

Off-policy methods are designed to improve the sample complexity by reusing experience from old policies (Thomas and Brunskill, 2016). Although not necessarily related, in Meta-RL, many off-policy algorithms also avoid the MAML approach of weight adaptation. Instead, they opt to condition the policy and the Q-function on a context which distinguishes between different tasks (Ren et al., 2019; Sung et al., 2017). A notable off-policy Meta-RL method is PEARL (Rakelly et al., 2019). It builds on top of the SAC algorithm and learns a Q-function $Q_\theta^\pi(s, a, z)$, a policy $\pi_\phi(s, z)$ and a context $z \sim q_\nu(z|c^{\mathcal{T}_i})$. The context, which is a latent representation of task $\mathcal{T}_i$, is generated by a probabilistic model that processes a trajectory $c^{\mathcal{T}_i}$ of (s, a, r) transitions sampled from task $\mathcal{T}_i$. To learn the critic alongside the context, PEARL modifies the SAC critic loss to

$$\mathcal{L}_{pearl}^{critic}(\theta, \nu) =$$
$$\mathbb{E}_{\mathcal{T}} \left[\mathbb{E}_{q_\nu(z|c^{\mathcal{T}_i})} \left[\mathcal{L}_{sac}^{critic}(\theta, \nu) + D_{KL} \left(q_\nu(z|c^{\mathcal{T}_i}) \| p(z) \right) \right] \right],$$

where $p(z)$ is a prior probability over the latent distribution of the context. While PEARL's context is a probabilistic model, other works (Fakoor et al., 2019) have suggested that a deterministic learnable context can provide similar results.

In this work, we consider both a learnable context and also the simpler approach of an *oracle-context* $c^{\mathcal{T}_i}$ which is a unique, predefined identifier for task i (Jayakumar et al., 2019). It can be an index when there is a countable number of tasks or a continuous number when the tasks are sampled from a continuous distribution. In practice, the oracle identifier is often known to the agent. Moreover, sometimes, e.g., in goal-oriented tasks, the context cannot be recovered directly from the transition tuples without prior knowledge, since there are no rewards unless the goal is reached, which rarely happens without policy adaptation.
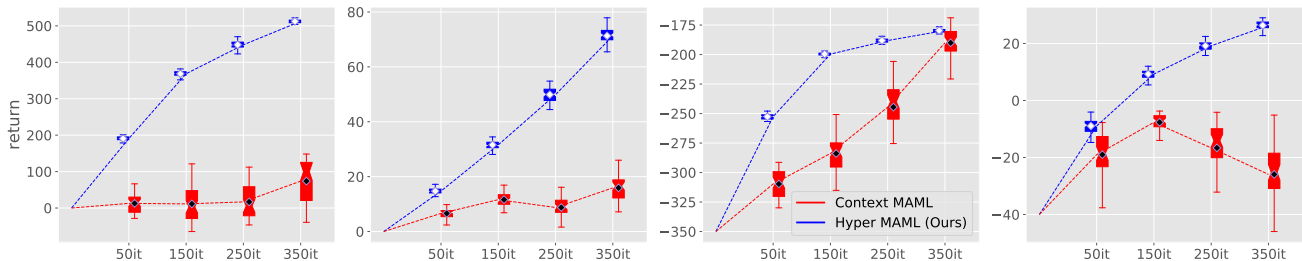
Figure 5. Visualizing gradient noise in MAML: The statistical population of the performance after 50 uncorrelated update steps is plotted for 4 different time steps. Hyper-MAML refers to Hypernetwork where the oracle-context is the meta variable and the state features are the base variable. Context-MAML refers to MLP policy where the oracle-context is concatenated with the state features.

4.2. Our Approach

Hypernetworks naturally fit into the meta-learning formulation where the context is an input to the primary network (von Oswald et al., 2019; Zhao et al., 2020). Therefore, we suggest modeling meta-policies s.t. the context is the meta variable and the state is the dynamic's input

$$\pi_\phi(a|s, c) = \mu_{w(c)}(\varepsilon|s) \text{ s.t. } \varepsilon \sim p_\varepsilon. \quad (16)$$

Interestingly, this modeling disentangles the state dependent gradient and the task dependent gradient of the meta-policy. To see that, let us take for example the on-policy objective of MAML and plug in a context dependent policy $\pi_\phi(a|s, c) = \mu_\phi(\varepsilon|s, c)$. Then, the objective in Eq. (15) becomes

$$J(\phi) = \sum_{\mathcal{T}_i} \sum_{s_j \in \mathcal{T}_i} \hat{A}_{i,j} \frac{\nabla_\phi \mu_{\phi_i}(\varepsilon_j|s_j, c_i)}{\mu_{\phi_i}(\varepsilon_j|s_j, c_i)}. \quad (17)$$

Applying the Hypernetwork modeling of the meta-policy in Eq. (16), this objective can be written as

$$J(\phi) = \sum_{\mathcal{T}_i} \nabla_\phi w(c_i) \cdot \sum_{s_j \in \mathcal{T}_i} \hat{A}_{i,j} \frac{\nabla_w \mu_{w(c_i)}(\varepsilon_j|s_j)}{\mu_{w(c_i)}(\varepsilon_j|s_j)} \quad (18)$$

In this form, the state-dependent gradients of the dynamic weights $\nabla_w \mu_{w(c_i)}(\varepsilon_j, s_j)$ are averaged independently for each task, and the task-dependent gradients of the primary weights $\nabla_\phi w(c_i)$ are averaged only over the task distribution and not over the joint task-state distribution as in Eq. (17). We postulate that such disentanglement reduces the gradient noise for the same number of samples. This should translate to more accurate learning steps and thus a more efficient learning process.

To test our hypothesis, we trained two different meta-policy models based on the MAML algorithm: (1) an MLP model where a state and an oracle-context are joined together; and (2) a Hypernetwork model, as described, with an oracle-context as a meta-variable. Importantly, note that, other than the neural architecture, both algorithms are *identical*. For four different timestamps during the learning process,

we constructed 50 different uncorrelated gradients from different episodes and evaluating the updated policy's performance. We take the performance statistics of the updated policies as a surrogate for the gradient noise. In Fig. 5, we plot the performance statistics of the updated meta-policies. We find that the variance of the Hypernetwork model is significantly lower than the MLP model across all tasks and environments. This indicates more efficient improvement and therefore we also observe that the mean value is consistently higher.

5. Experiments

5.1. Experimental Setup

We conducted our experiments in the MuJoCo simulator (Todorov et al., 2012) and tested the algorithms on the benchmark environments available in OpenAI Gym (Brockman et al., 2016). For single task RL, we evaluated our method on the: (1) Hooper-v2; (2) Walker2D-v2; (3) Ant-v2[3]; and (4) Half-Cheetah-v2 environments. For meta-RL, we evaluated our method on the: (1) Half-Cheetah-Fwd-Back and (2) Ant-Fwd-Back, and on velocity tasks: (3) Half-Cheetah-Vel and (4) Ant-Vel as is done in (Rakelly et al., 2019). We also added the Half-Cheetah-Vel-Medium environment as presented in (Fakoor et al., 2019), which tests out-of-distribution generalization abilities. For Context-MAML and Hyper-MAML we adopted the *oracle-context* as discussed in Sec. 4. For the forward-backward tasks, we provided a binary indicator, and for the velocity tasks, we adopted a continuous context in the range $[0, 3]$ that maps to the velocities in the training distribution.

In the RL experiments, we compared our model to SAC and TD3, and in Meta-RL, we compared to MAML and PEARL. We used the authors' official implementations (or open-source PyTorch (Ketkar, 2017) implementation when the official one was not available) and the original baselines' hyperparameters, as well as strictly following each algorithm evaluation procedure. The Hypernetwork training was executed with the baseline loss s.t. we changed only the networks model and adjusted the learning rate to fit the

different architecture. All experiments were averaged over 5 seeds. Further technical details are in the appendix.

5.2. The Hypernetwork Architecture

Our Hypernetwork model is illustrated in Fig. 1 and in Sec. 2. When designing the Hypernetwork model, we did not search for the best performance model, rather we sought a proper comparison to the standard MLP architecture used in RL (denoted here as MLP-Standard). To that end, we used a smaller dynamic network than the MLP model (single hidden layer instead of two layers and the same number of neurons (256) in a layer). With this approach, we wish to show the gain of using dynamic weights with respect to a fixed set of weights in the MLP model. To emphasize the gain of the dynamic weights, we added an MLP-Small baseline with equal configuration to the dynamic model (one hidden layer with 256 neurons).

Unlike the dynamic network, the role of the primary network is missing from the MLP architecture. Therefore, for the primary network, we used a high-performance ResNet model (Srivastava et al., 2015) which we found apt for generating the set of dynamic weights (Glorot and Bengio, 2010). To make sure that the performance gain is not due to the expressiveness of the ResNet model or the additional number of learnable weights, we added three more baselines: (1) ResNet Features: the same primary and dynamic architecture, but the output of the primary is a state feature vector which is concatenated to the action as the input for an MLP-Standard network; (2) MLP-Large: two hidden layers, each with 2900 neurons which sum up to $9M$ weights as in the Hypernetwork architecture; and (3) Res35: ResNet with 35 blocks to yield the Q-value, which sum up to $4.5M$ weights. In addition, we added a comparison to the Q-D2RL model: a deep dense architecture for the Q-function which was recently suggested in (Sinha et al., 2020).

One important issue with Hypernetworks is their numerical stability. We found that they are specifically sensitive to weight initialization as bad primary initialization may amplify into catastrophic dynamic weights (Chang et al., 2019). We solved this problem by initializing the primary s.t. the average initial distribution dynamic weights resembles the Kaiming-uniform initialization (He et al., 2015). Further details can be found in the appendix.

5.3. Results

The results and the comparison to the baselines are summarized in Fig. 6. In all four experiments, our Hypernetwork model achieves an average of 10% - 70% gain over the MLP-Standard baseline in the final performance and reaches the

---

[3]We reduced the control cost as is done in PEARL (Rakelly et al., 2019) to avoid numerical instability problems.
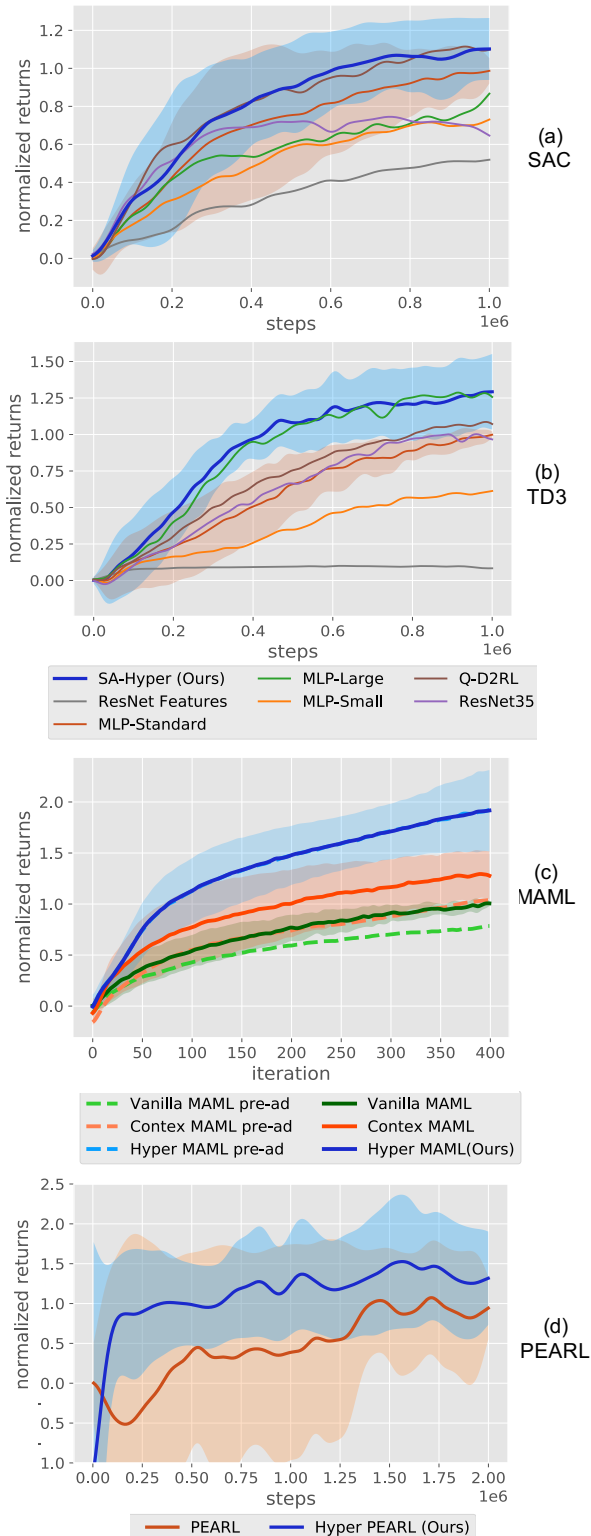

Figure 6. The mean normalized score with respect to different baseline models: (a) SAC; (b) TD3; (c) MAML; and (d) PEARL. The Hypernetwork consistently improves all baselines in all algorithms.

baseline's score, with only 20%-70% of the total training steps. As described in Sec. 5.2, for the RL experiments, in addition to the MLP-Standard model, we tested five more baselines: (1) MLP-Large; (2) MLP-Small; (3) ResNet Features; (4) ResNet35; and (5) Q-D2RL. Both on TD3 and SAC, we find a consistent improvement over all baselines and SA-Hyper outperforms in all environments with two exceptions: where MLP-Large or Q-D2RL achieve a better score than SA-Hyper in the Ant-v2 environment (the learning curves for each environment are found in the appendix). While it may seem like the Hypernetwork improvement is due to its large parametric dimension or the ResNet design of the primary model, our results provide strong evidence that this assumption is not true. The SA-Hyper model outperforms other models with the same number of parameters (MLP-Large and ResNet Features[4]) and also models that employ ResNet architectures (ResNet Features and Res35). In addition, it is as good (SAC) or better (TD3) than Q-D2RL, which was recently suggested as an architecture tailored for the RL problem (Sinha et al., 2020). Please note that as discussed in Sec. 5.2 and unlike D2RL, we do not optimize the number of layers in the dynamic model.[5]

In Fig. 6c we compared different models for MAML: (1) Vanilla-MAML; (2) Context-MAML, i.e. a context-based version of MAML with an oracle-context; and (3) Hyper-MAML, similar to context-MAML but with a Hypernetwork model. For all models, we evaluated both the pre-adaptation (pre-ad) as well as the post-adaptation scores. First, we verify the claim in (Fakoor et al., 2019) that context benefits Meta-RL algorithms just as Context-MAML outperforms Vanilla-MAML. However, we find that Hyper-MAML outperforms Context-MAML by roughly 50%. Moreover, unlike the standard MLP models, we find that Hyper-MAML does not require any adaptation step (no observable difference between the pre- and post-adaptation scores). We assume that this result is due to the better generalization capabilities of the Hypernetwork architecture as can also be seen from the next PEARL experiments.

In Fig. 6d we evaluated the Hypernetwork model with the PEARL algorithm. The context is learned with a probabilistic encoder as presented in (Rakelly et al., 2019) s.t. the only difference with the original PEARL is the policy and critic neural models. The empirical results show that

Hyper-PEARL outperforms the MLP baseline both in the final performance (15%) and in sample efficiency (70% fewer steps to reach the final baseline score). Most importantly, we find that Hyper-PEARL generalizes better to the unseen test tasks. This applies both to test tasks sampled from the training distribution (as the higher score and lower variance of Hyper-PEARL indicate) and also to Out-Of-Distribution (OOD) tasks, as can be observed in Fig. 7.
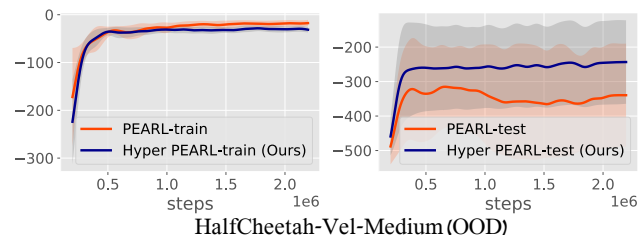

Figure 7. PEARL results in an **Out Of Distribution** environment, HalfCheetah-Vel-Medium, where the training tasks' target is [0,2.5] and the test tasks' target is [2.5,3]. The Hypernetwork achieved slightly lower returns over the training tasks, yet it generalizes better over the OOD test tasks.

6. Conclusions

In this work, we set out to study neural models for the RL building blocks: Q-functions and meta-policies. Arguing that the unique nature of the RL setting requires unconventional models, we suggested the Hypernetwork model and showed empirically several significant advantages over MLP models. First, Hypernetworks are better able to estimate the parametric gradient signal of the Q-function required to train actor-critic algorithms. Second, they reduce the gradient variance in training meta-policies in Meta-RL. Finally, they improve OOD generalization and they do not require any adaptation step in Meta-RL training, which significantly facilitates the training process.

7. Code

Our Hypernetwork PyTorch implementation is found at https://github.com/keynans/HypeRL.

---

[4]Interestingly, The Resnet Features baseline achieved very low scores even as compared to the MLP-Standard baseline. Indeed, this result is not surprising as the action gradient model of Resnet Features is identical to the action gradient model of MLP-Small (single hidden layer with 256 neurons). While ResNet generated state features may improve the Q-function estimation, they do not necessarily improve the gradient estimation $\nabla_a Q^\pi$ as the network is not explicitly trained to model the gradient.

[5]We do not compare to the full D2RL model which also modifies the policy architecture as our SA-Hyper model only changes the Q-net model.

References

Brock, A., Lim, T., Ritchie, J., and Weston, N. (2018). Smash: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *CoRR*, abs/1606.01540.

Chang, O., Flokas, L., and Lipson, H. (2019). Principled weight initialization for hypernetworks. In *International Conference on Learning Representations*.

Dean, T. and Givan, R. (1997). Model minimization in markov decision processes. In *AAAI/IAAI*, pages 106–111.

Fakoor, R., Chaudhari, P., Soatto, S., and Smola, A. J. (2019). Meta-q-learning. In *International Conference on Learning Representations*.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*.

Fujimoto, S., Meger, D., and Precup, D. (2019). Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062.

Fujimoto, S., Van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*.

Galanti, T. and Wolf, L. (2020a). Comparing the parameter complexity of hypernetworks and the embedding-based alternative. *arXiv preprint arXiv:2002.10006*.

Galanti, T. and Wolf, L. (2020b). On the modularity of hypernetworks. *Advances in Neural Information Processing Systems*, 33.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

Grondman, I., Busoniu, L., Lopes, G. A., and Babuska, R. (2012). A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307.

Ha, D., Dai, A., and Le, Q. V. (2016). Hypernetworks. *arXiv*, pages arXiv–1609.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*.

Hasselt, H. v., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 2094–2100.

He, J., Chen, J., He, X., Gao, J., Li, L., Deng, L., and Ostendorf, M. (2016). Deep reinforcement learning with a natural language action space. In *ACL (1)*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

Huang, Y., Xie, K., Bharadhwaj, H., and Shkurti, F. (2020). Continual model-based reinforcement learning with hypernetworks. *arXiv preprint arXiv:2009.11997*.

Ilyas, A., Engstrom, L., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., and Madry, A. (2019). A closer look at deep policy gradients. In *International Conference on Learning Representations*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR.

Jayakumar, S. M., Czarnecki, W. M., Menick, J., Schwarz, J., Rae, J., Osindero, S., Teh, Y. W., Harley, T., and Pascanu, R. (2019). Multiplicative interactions and where to find them. In *International Conference on Learning Representations*.

Jia, X., De Brabandere, B., Tuytelaars, T., and Gool, L. V. (2016). Dynamic filter networks. In *Advances in neural information processing systems*, pages 667–675.

Kaiser, Ł., Babaeizadeh, M., Mił]]os, P., Osiński, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., et al. (2019). Model based reinforcement learning for atari. In *International Conference on Learning Representations*.

Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *In Proc. 19th International Conference on Machine Learning*. Citeseer.

Ketkar, N. (2017). Introduction to pytorch. In *Deep learning with python*, pages 195–208. Springer.

Kumar, A., Fu, J., Soh, M., Tucker, G., and Levine, S. (2019). Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pages 11784–11794.

Lattimore, T. and Szepesvári, C. (2020). *Bandit algorithms*. Cambridge University Press.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *ICLR (Poster)*.

Lior Deutsch, Erik Nijkamp, Y. Y. (2019). A generative model for sampling high-performance and diverse weights for neural networks. *CoRR*.

Littwin, G. and Wolf, L. (2019). Deep meta functionals for shape representation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1824–1833.

McClelland, J. L. (1985). Putting knowledge in its place: A scheme for programming parallel processing structures on the fly. *Cognitive Science*, 9(1):113–146.

Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. (2018). A simple neural attentive meta-learner. In *International Conference on Learning Representations*.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. (2018). Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*.

Philipp, G., Song, D., and Carbonell, J. G. (2017). The exploding gradient problem demystified-definition, prevalence, impact, origin, tradeoffs, and solutions. *arXiv preprint arXiv:1712.05577*.

Potapov, A., Shcherbakov, O., Zhdanov, I., Rodionov, S., and Skorobogatko, N. (2018). Hypernets and their application to learning spatial transformations. In *International Conference on Artificial Neural Networks*, pages 476–486. Springer.

Rakelly, K., Zhou, A., Finn, C., Levine, S., and Quillen, D. (2019). Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pages 5331–5340. PMLR.

Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., and Whiteson, S. (2018). Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304.

Ratzlaff, N. and Li, F. (2019). Hypergan: A generative model for diverse, performant neural networks. *CoRR*, abs/1901.11058.

Ren, H., Garg, A., and Anandkumar, A. (2019). Context-based meta-reinforcement learning with structured latent space. *Skills Workshop NeurIPS 2019*.

Rusu, A. A., Rao, D., Sygnowski, J., Vinyals, O., Pascanu, R., Osindero, S., and Hadsell, R. (2019). Meta-learning with latent embedding optimization. In *International Conference on Learning Representations*.

Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv preprint arXiv:1602.07868*.

Sarafian, E., Sinay, M., Louzoun, Y., Agmon, N., and Kraus, S. (2020). Explicit gradient learning for black-box optimization. In *International Conference on Machine Learning*, pages 8480–8490. PMLR.

Saremi, S. (2019). On approximating $\setminus \nabla f$ with neural networks. *arXiv preprint arXiv:1910.12744*.

Schmidhuber, J. (1992). Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015a). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sinha, S., Bharadhwaj, H., Srinivas, A., and Garg, A. (2020). D2rl: Deep dense architectures in reinforcement learning. *arXiv preprint arXiv:2010.09163*.

Sohn, S., Woo, H., Choi, J., and Lee, H. (2019). Meta reinforcement learning with autonomous inference of subtask dependencies. In *International Conference on Learning Representations*.

Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Training very deep networks. In *NIPS*.

Sun, Z., Ozay, M., and Okatani, T. (2017). Hypernetworks with statistical filtering for defending adversarial examples. *arXiv preprint arXiv:1711.01791*.

Sung, F., Zhang, L., Xiang, T., Hospedales, T., and Yang, Y. (2017). Learning to learn: Meta-critic networks for sample efficient learning. *arXiv preprint arXiv:1706.09529*.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

Thomas, P. and Brunskill, E. (2016). Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 2139–2148. PMLR.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE.

von Oswald, J., Henning, C., Sacramento, J., and Grewe, B. F. (2019). Continual learning with hypernetworks. In *International Conference on Learning Representations*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Zai, A. and Brown, B. (2020). *Deep reinforcement learning in action*. Manning Publications.

Zhang, S. and Sutton, R. S. (2017). A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*.

Zhao, D., von Oswald, J., Kobayashi, S., Sacramento, J., and Grewe, B. F. (2020). Meta-learning via hypernetworks. *4th Workshop on Meta-Learning at NeurIPS 2020*.

A. Proof of Proposition 1

Proposition 1. *Let* $\pi(a|s) = \mu_\phi(\varepsilon|s)$ *be a stochastic parametric policy with* $\varepsilon \sim p_\varepsilon$ *and* $\mu_\phi(\cdot|s)$ *a transformation with a Lipschitz continuous gradient and a Lipschitz constant* κ_μ. *Assume that its Q-function* $Q^\pi(s, a)$ *has a Lipschitz continuous gradient in* a, *i.e.* $|\nabla_a Q^\pi(s, a_1) - \nabla_a Q^\pi(s, a_2)| \leq \kappa_q \|a_1 - a_2\|$. *Define the average gradient operator* $\overline{\nabla}_\phi f = \mathbb{E}_{s \sim \mathcal{D}} [\mathbb{E}_{\varepsilon \sim p_\varepsilon} [\nabla_\phi \mu_\phi(\varepsilon|s) \cdot f(s, \mu_\phi(\varepsilon|s))]]$. *If there exists a gradient estimation* $g(s, a)$ *and* $0 < \alpha < 1$ *s.t.*

$$\|\overline{\nabla}_\phi \cdot g - \overline{\nabla}_\phi \cdot \nabla_a Q^\pi\| \leq \alpha \|\overline{\nabla}_\phi \cdot \nabla_a Q^\pi\| \tag{19}$$

then the ascent step $\phi' \leftarrow \phi + \eta \overline{\nabla}_\phi \cdot g$ *with* $\eta \leq \frac{1}{k} \frac{1-\alpha}{(1+\alpha)^2}$ *yields a positive empirical advantage policy.*

Proof. First, recall the objective to be optimized:

$$J(\phi) = \mathbb{E}_{s \sim \mathcal{D}} [\mathbb{E}_{\varepsilon \sim p_\varepsilon} [Q^\pi(s, \mu_\phi(\varepsilon; s))]]$$
$$\nabla_\phi J(\phi) = \mathbb{E}_{s \sim \mathcal{D}} [\mathbb{E}_{\varepsilon \sim p_\varepsilon} [\nabla_\phi \mu_\phi(\varepsilon; s) \cdot \nabla_a Q^\pi(s, \mu_\phi(\varepsilon; s))]] = \overline{\nabla}_\phi \cdot \nabla_a Q^\pi \tag{20}$$

Notice that as Q^π is bounded by the maximal reward and its gradient is Lipschitz continuous, the gradient $\nabla_a Q^\pi$ is therefore bounded. Similarly, since the action space is bounded, and the terministic transformation μ_π has a Lipschitz continuous gradient, it follows that $\nabla_\phi \mu_\pi$ is also bounded. Define $\|\nabla_a Q^\pi\| \leq \sigma_q$ and $\|\nabla_\phi \mu_\pi\| \leq \sigma_\mu$.

Lemma 1. *Let* $A(x) : \mathbb{R}^n \to M^{k \times l}$ *s.t.* $\|A(x)\| \leq M_a$ *and* $\|A(x_1) - A(x_2)\| \leq \alpha \|x_1 - x_2\|$ *and* $\|\cdot\|$ *is the induced vector norm. And let* $b(x) : \mathbb{R}^n \to \mathbb{R}^l$ *s.t.* $\|b(x)\| \leq M_b$ *and* $\|b(x_1) - b(x_2)\| \leq \beta \|x_1 - x_2\|$. *The operator* $c(x) = A(x) \cdot b(x) : \mathbb{R}^n \to \mathbb{R}^k$ *is Lipschitz with constant* $\kappa_c \leq \alpha M_a + \beta M_b$.

Proof.

$$\|c(x_1) - c(x_2)\| = \|A(x_1) \cdot b(x_1) - A(x_2) \cdot b(x_2)\|$$
$$= \|A(x_1) \cdot b(x_1) - A(x_1) \cdot b(x_2) + A(x_1) \cdot b(x_2) - A(x_2) \cdot b(x_2)\|$$
$$\leq \|A(x_1) \cdot (b(x_1) - b(x_2))\| + \|(A(x_1) - A(x_2)) \cdot b(x_2)\|$$
$$\leq \|A(x_1)\| \|b(x_1) - b(x_2)\| + \|A(x_1) - A(x_2)\| \|b(x_2)\|$$
$$\leq (\beta M_a + \alpha M_b) \|x_1 - x_2\|$$

$\square$

The Lipschitz constant of the objective gradient is bounded by

$$\|\nabla_\phi J(\phi_1) - \nabla_\phi J(\phi_2)\| = \|\mathbb{E} [\nabla_\phi Q^\pi(s, \mu_{\phi_1}(\varepsilon; s)) - \nabla_\phi Q^\pi(s, \mu_{\phi_2}(\varepsilon; s))]\| \leq$$
$$\mathbb{E} [\|\nabla_\phi \mu_{\phi_1}(\varepsilon; s) \cdot \nabla_a Q^\pi(s, \mu_{\phi_1}(\varepsilon; s)) - \nabla_\phi \mu_{\phi_2}(\varepsilon; s) \cdot \nabla_a Q^\pi(s, \mu_{\phi_2}(\varepsilon; s))\|]$$

Applying Lemma 1, we obtain

$$\|\nabla_\phi J(\phi_1) - \nabla_\phi J(\phi_2)\| \leq (\kappa_q \sigma_\mu + \kappa_\mu \sigma_q) \|\phi_1 - \phi_2\|.$$

Therefore, $J(\phi_1)$ is also Lipschitz. Hence, applying Taylor's expansion around ϕ, we have that

$$J(\phi') \geq J(\phi) + (\phi' - \phi) \cdot \nabla_\phi J(\phi) - \frac{\kappa_J^2}{2} \|\phi' - \phi\|^2 \geq J(\phi) + (\phi' - \phi) \cdot \nabla_\phi J(\phi) - \frac{(\kappa_q \sigma_\mu + \kappa_\mu \sigma_q)^2}{2} \|\phi' - \phi\|^2.$$

Plugging in the iteration $\phi' \leftarrow \phi + \eta \overline{\nabla}_\phi \cdot g$ we obtain

$$J(\phi') \geq J(\phi) + \eta (\overline{\nabla}_\phi \cdot g) \cdot (\overline{\nabla}_\phi \cdot Q^\pi) - \frac{\eta^2 (\kappa_q \sigma_\mu + \kappa_\mu \sigma_q)^2}{2} \|\overline{\nabla}_\phi \cdot g\|^2. \tag{21}$$

Taking the second term on the right-hand side,

$$(\overline{\nabla}_\phi \cdot g) \cdot (\overline{\nabla}_\phi \cdot Q^\pi) = (\overline{\nabla}_\phi \cdot Q^\pi - (\overline{\nabla}_\phi \cdot g - \overline{\nabla}_\phi \cdot Q^\pi)) \cdot (\overline{\nabla}_\phi \cdot Q^\pi)$$
$$\geq \|\overline{\nabla}_\phi \cdot Q^\pi\|^2 - \|(\overline{\nabla}_\phi \cdot g - \overline{\nabla}_\phi \cdot Q^\pi) \cdot (\overline{\nabla}_\phi \cdot Q^\pi)\|$$
$$\geq \|\overline{\nabla}_\phi \cdot Q^\pi\|^2 - \|\overline{\nabla}_\phi \cdot g - \overline{\nabla}_\phi \cdot Q^\pi\| \cdot \|\overline{\nabla}_\phi \cdot Q^\pi\|$$
$$\geq (1 - \alpha) \|\overline{\nabla}_\phi \cdot Q^\pi\|^2.$$

For the last term we have

$$
\begin{aligned}
\left\|\overline{\nabla}_\phi \cdot g\right\|^2 &= \left\|\overline{\nabla}_\phi \cdot Q^\pi - (\overline{\nabla}_\phi \cdot g - \overline{\nabla}_\phi \cdot Q^\pi)\right\|^2 \\
&= \left\|\overline{\nabla}_\phi \cdot Q^\pi\right\|^2 - 2\left(\overline{\nabla}_\phi \cdot Q^\pi\right) \cdot (\overline{\nabla}_\phi \cdot g - \overline{\nabla}_\phi \cdot Q^\pi) + \left\|\overline{\nabla}_\phi \cdot g - \overline{\nabla}_\phi \cdot Q^\pi\right\|^2 \\
&\leq \left\|\overline{\nabla}_\phi \cdot Q^\pi\right\|^2 + 2\left\|\overline{\nabla}_\phi \cdot Q^\pi\right\| \cdot \left\|\overline{\nabla}_\phi \cdot g - \overline{\nabla}_\phi \cdot Q^\pi\right\| + \alpha^2 \left\|\overline{\nabla}_\phi \cdot Q^\pi\right\|^2 \\
&\leq (1 + 2\alpha + \alpha^2) \left\|\overline{\nabla}_\phi \cdot Q^\pi\right\|^2.
\end{aligned}
$$

Plugging both terms together into Eq. (21) we get

$$
J(\phi') \geq J(\phi) + \left\|\overline{\nabla}_\phi \cdot Q^\pi\right\|^2 \left(\eta(1-\alpha) - \frac{1}{2}\eta^2(\kappa_q \sigma_\mu + \kappa_\mu \sigma_q)^2(1+\alpha)^2\right).
$$

To obtain a positive empirical advantage we need

$$
\eta(1-\alpha) - \frac{1}{2}\eta^2(\kappa_q \sigma_\mu + \kappa_\mu \sigma_q)^2(1+\alpha)^2 \geq 0
$$

Thus the sufficient requirement for the learning rate is

$$
\eta \leq \frac{1}{\tilde{k}} \frac{1-\alpha}{(1+\alpha)^2}.
$$

where $\tilde{k} = \frac{1}{2}(\kappa_q \sigma_\mu + \kappa_\mu \sigma_q)$.

$\square$

B. Cosine Similarity Estimation

To evaluate the averaged Cosine Similarity (CS)

$$cs(Q_\theta^\pi) = \mathbb{E}_{s \sim \mathcal{D}} \left[\frac{\nabla_a Q_\theta^\pi(s, a_\mu) \cdot \nabla_a Q^\pi(s, a_\mu)}{\|\nabla_a Q_\theta^\pi(s, a_\mu)\| \, \|\nabla_a Q^\pi(s, a_\mu)\|} \right], \tag{22}$$

we need to estimate the local CS for each state. To that end, we estimate the "true" Q-function $Q^\pi(s, a)$ at the vicinity of $a = a_\mu$ with a non-parametric local linear model

$$Q^\pi(s, a) \simeq f_{s, a_\mu}(a) = a \cdot g$$

where $g \in \mathbb{R}^{N_a}$ s.t. the Q-function gradient is constant $\nabla_a Q^\pi(s, a) \simeq g$. To fit the linear model, we sample N_r unbiased samples of the Q-function around a_μ, i.e. $q_i = \hat{Q}^\pi(s, a_i)$. These samples are the empirical discounted sum of rewards following execution of action $a_i = a_\mu + \Delta_i$ at state s and then applying policy π.

To fit the linear model we directly fit the constant model g for the gradient. Recall that applying the Taylor's expansion around a_μ gives

$$Q^\pi(s, a) = Q^\pi(s, a_\mu) + (a - a_\mu) \cdot \nabla_a Q_\theta^\pi(s, a_\mu) + O\left(\|a - a_\mu\|^2\right)$$

, therefore

$$Q^\pi(s, a_2) - Q^\pi(s, a_1) - (a_2 - a_1) \cdot \nabla_a Q_\theta^\pi(s, a_\mu) = O\left(\|a_2 - a_1\|^2\right)$$

for a_1, a_2 at the vicinity of a_μ.

To find the best fit $g \simeq \nabla_a Q_\theta^\pi(s, a_\mu)$ we minimize averaged the quadratic error term over all pairs of sampled trajectories

$$g^* = \arg\min_g \sum_i^{N_r} \sum_j^{N_r} |(a_j - a_i) \cdot g - q_j + q_i|^2.$$

This problem can be expressed in a matrix notation as

$$g^* = \arg\min_g \left\| \tilde{X} g - \delta \right\|^2,$$

where $\tilde{X} \in R^{N_r^2 \times N_a}$ is a matrix with N_r^2 rows of all the vectors $a_j - a_i$ and δ is a N_r^2 element vector of all the differences $q_j - q_i$. Its minimization is the Least-Mean-Squared Estimator (LMSE)

$$g^* = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T \delta.$$

In our experiments we evaluated the CS every $K = 10^4$ learning steps and used $N_s = 15$, $N_r = 15$ and $\Delta_i \sim \mathcal{N}(0, 0.3)$ for each evaluation. This choice trades off somewhat less accurate local estimators with more samples during training. To test our gradient estimator, we first applied it to the outputs of the Q-function network (instead of the true returns) and calculated the CS between a linear model based on the network outputs and the network parametric gradient. The results in Fig. 8 show that our g^* estimator obtains a high CS between the Q-net outputs of the SA-Hyper and MLP models and their respective parametric gradients. This indicates that these networks are locally ($\Delta \propto 0.3$) linear. On the other hand, the CS between the linear model based on the AS-Hyper outputs and its parametric gradient is lower, which indicates that the network is not necessarily close to linear with $\Delta \propto 0.3$. We assume that this may be because the action in the AS-Hyper configuration plays the meta-variable role which increases the non-linearity of the model with respect to the action input. Importantly, note that this does not indicate that the true Q-function of the AS-Hyper model is more non-linear than other models.

In Fig. 9 we plot the CS for 4 different environments averaged with a window size of $W = 20$. The results show that on average the SA-Hyper configuration obtains a higher CS, which indicates that the policy optimization step is more accurate s.t. the RL training process is more efficient.
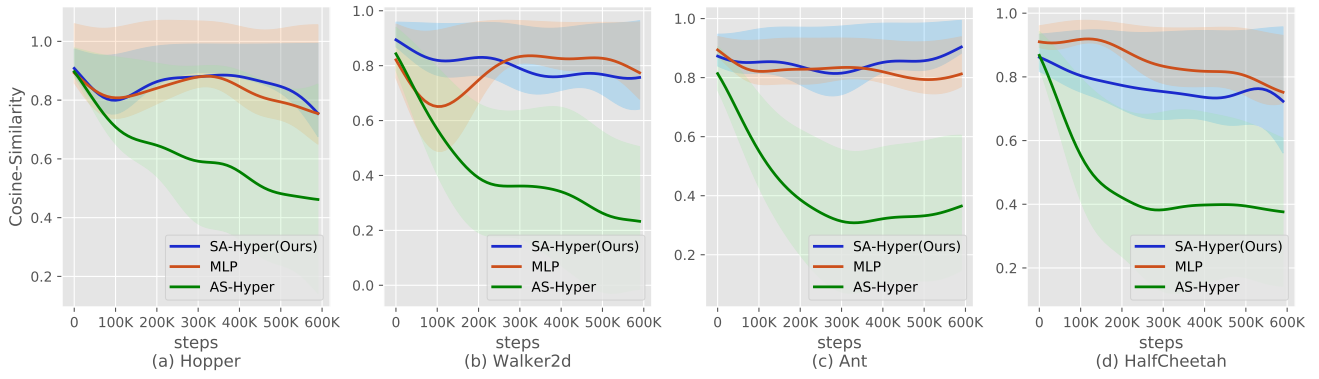
Figure 8. The Cosine-Similarity between the LMSE estimator of the Q-net outputs and the parametric gradient averaged with a window size of $W = 20$.
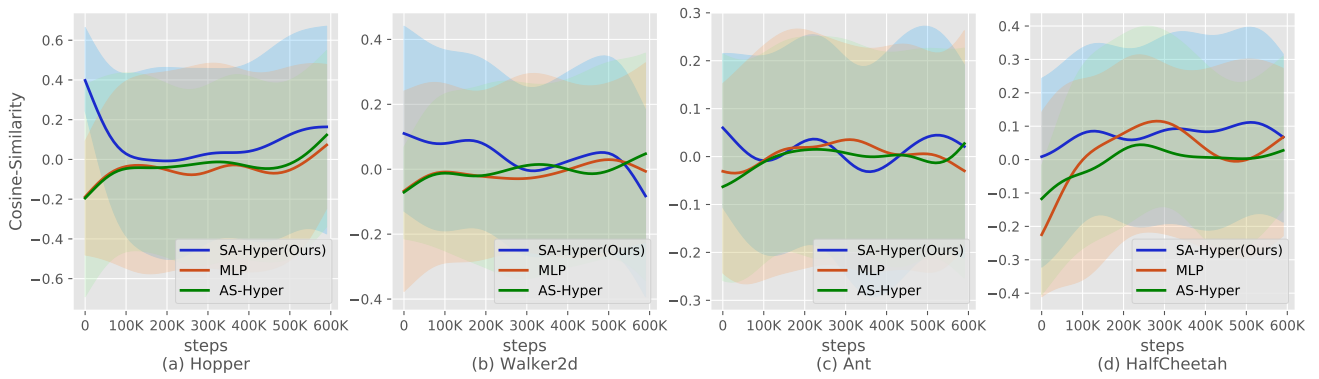

Figure 9. The Cosine-Similarity between the LMSE estimator of the empirical sum of rewards and the parametric gradient averaged with a window size of $W = 20$.

C. Gradient Step Noise Statistics in MAML

Hypernetworks disentangle the state-dependent gradient and the task-dependent gradient. As explained in the paper, we hypothesized that these characteristics reduce the gradient ascent step noise during policy updates

$$\phi \leftarrow \phi - \eta \widehat{\nabla}_\phi J$$

where $\widehat{\nabla}_\phi J$ is the gradient step estimation and η is the learning rate. It is not obvious how to define the gradient noise properly as any norm-based measure depends on the network's structure and size. Therefore, we take an alternative approach and define the gradient noise as the performance statistics after applying a set of independent gradient steps. In simple words, this definition essentially corresponds to how noisy the learning process is.

To estimate the performance statistics, we take $N = 50$ different independent policy gradients based on independent trajectories at 4 different time steps during the training process. For each gradient step, we sampled 20 trajectories with a maximal length of 200 steps (identical to a single policy update during the training process) out of 40 tasks. After each gradient step, we evaluated the performance and restored the policy's weights s.t. the gradient steps are independent.

We compared two different network architectures, both with access to an oracle context: (1) Hyper-MAML; and (2) Context-MAML. We did not evaluate Vanilla-MAML as it has no context and the gradient noise, in this case, might also be due to higher adaptation noise as the context must be recovered from the trajectories' rewards. In the paper, we presented the performance statistics after $N = 50$ different updates. In Table 1 we present the variance of those statistics.

Table 1. The gradient coefficient of variation $\frac{\sigma}{|\mu|}$ and the variance (in brackets) in MAML. Hyper-MAML refers to Hypernetwork policy where the oracle-context is the meta-variable and the state features are the base-variable. Context-MAML refers to the MLP model policy where the oracle-context is concatenated with the state features. To compare between different policies with different reward scales, we report both the coefficient of variation and the variance in brackets.

Envrionment	50 iter	150 iter	300 iter	450 iter
HalfCheetah-Fwd-Back				
Context-MAML	1.184 (774)	4.492 (2595)	2.590 (1891)	0.822 (3689)
Hyper MAML (Ours)	**0.027** (26)	**0.017** (43)	**0.021** (96)	**0.014** (53)
HalfCheetah-Vel				
Context-MAML	0.035 (122)	0.050 (208)	0.093 (520)	0.066 (161)
Hyper MAML (Ours)	**0.009** (5)	**0.005** (1)	**0.008** (2)	**0.009** (2)
Ant-Fwd-Back				
Context-MAML	0.274 (3)	0.199 (5)	0.400 (12)	0.285 (20)
Hyper MAML (Ours)	**0.073** (1)	**0.047** (2)	**0.050** (6)	**0.047** (11)
Ant-Vel				
Context-MAML	0.379 (52)	0.377 (8)	0.628 (109)	0.418 (117)
Hyper MAML (Ours)	**0.252** (5)	**0.159** (2)	**0.080** (2)	**0.057** (2)

D. Models Design

D.1. Hypernetwork Architecture

The Hypernetwork's primary part is composed of three main blocks followed by a set of heads. Each block contains an up-scaling linear layer followed by two pre-activation residual linear blocks (ReLU-linear-ReLU-linear). The first block up-scales from the state's dimension to 256 and the second and third blocks grow to 512 and 1024 neurons respectively. The total number of learnable parameters in the three blocks is $\sim 6.5M$. The last block is followed by the heads which are a set of linear transformations that generate the $\sim 2K$ dynamic parameters (including weights, biases and gains). The heads have $\sim 2.5M$ learnable parameters s.t. the total number of parameters in the primary part is $\sim 9M$.

D.2. Primary Model Design: Negative Results

In our search for a primary network that can learn to model the weights of a state-dependent dynamic Q-function, we experimented with several different architectures. Here we outline a list of negative results, i.e. models that failed to learn good primary networks.

1. we tried three network architecture: (1) MLP; (2) Dense Blocks (Huang et al., 2017); and (3) ResNet Blocks (He et al., 2016). The MLP did not converge and the dense blocks were sensitive to the initialization with spikes in the policy's gradient which led to an unstable learning process.

2. We found that the head size (the last layer that outputs all the dynamic network weights) should not be smaller than 512 and the depth should be at least 5 blocks. Upsampling from the low state dimension can either be done gradually or at the first layer.

3. We tried different normalization schemes: (1) weight normalization (Salimans and Kingma, 2016); (2) spectral normalization (Miyato et al., 2018); and (3) batch normalization (Ioffe and Szegedy, 2015). All of them did not help and slowed or stopped the learning.

4. For the non-linear activation functions, we tried RELU and ELU which we found to have similar performances.

D.3. Hypernetwork Initialization

A proper initialization for the Hypernetwork is crucial for the network's numerical stability and its ability to learn. Common initialization methods are not necessarily suited for Hypernetworks (Chang et al., 2019) since they fail to generate the dynamic weights in the correct scale. We found that some RL algorithms are more affected than others by the initialization scheme, e.g, SAC is more sensitive than TD3. However, we leave this question of why some RL algorithms are more sensitive than others to the weight initialization for future research.

To improve the Hypernetwork weight initialization, we followed (Lior Deutsch, 2019) and initialized the primary weights with smaller than usual values s.t. the initial dynamic weights were also relatively small compared to standard initialization (Fig. 10). As is shown in Fig. 11, this enables the dynamic weights to converge during the training process to a relatively similar distribution of a normal MLP network.

The residual blocks in the primary part were initialized with a fan-in Kaiming uniform initialization (He et al., 2015) with a gain of $\frac{1}{\sqrt{12}}$ (instead of the normal gain of $\sqrt{2}$ for the ReLU activation). We used fixed uniform distributions to initialize the weights in the heads: $U(-0.05, 0.05)$ for the first dynamic layer, $U(-0.008, 0.008)$ for the second dynamic layer and for the standard deviation output layer in the PEARL meta-policy we used the $U(-0.001, 0.001)$ distribution.

In Fig. 10 and Fig. 11 we plot the histogram of the TD3 critic dynamic network weights with different primary initializations: (1) our custom primary initialization; and (2) The default Pytorch initialization of the primary network. We compare the dynamic weights to the weights of a standard MLP-Small network (the same size as the dynamic network). We take two snapshots of the weight distribution: (1) in Fig. 10 before the start of the training process; and (2) after $100K$ training steps. In Table 2 we also report the total-variation distance between each initialization and the MLP-Small weight distribution. Interestingly, the results show that while the dynamic weight distribution with the Pytorch primary initialization is closer to the MLP-Small distribution at the beginning of the training process, after 100K training steps our primary initialized weights produce closer dynamic weight distribution to the MLP-Small network (also trained for $100K$ steps).
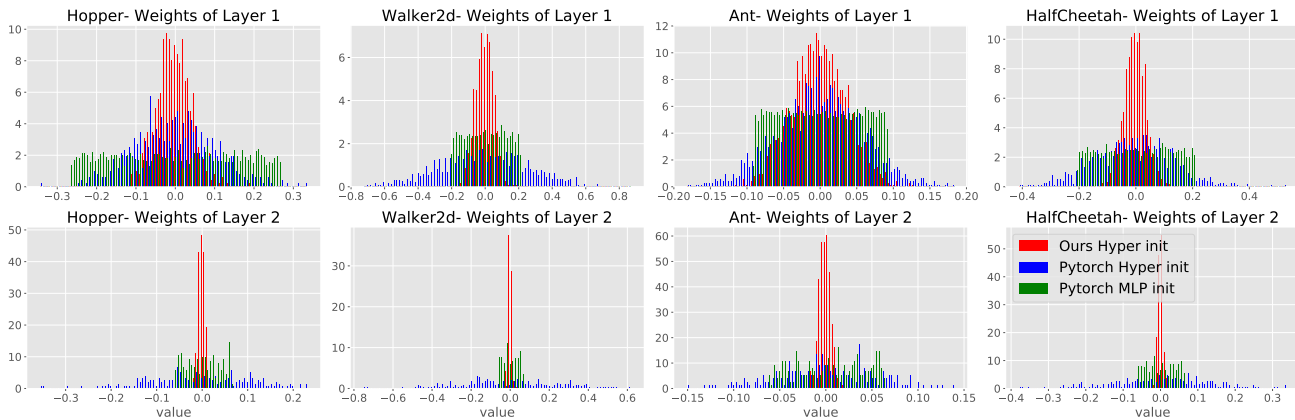
Figure 10. Dynamic network weight distribution of different initialization schemes **at the beginning of the training**. The "Hyper init" refers to a primary network initialized with our suggested initialization scheme. 'Pytorch Hyper init' refers to the Pytorch default initialization of the primary network and "Pytorch MLP init" refers to the Pytorch default initialization of the MLP-Small model (same architecture as the dynamic network).
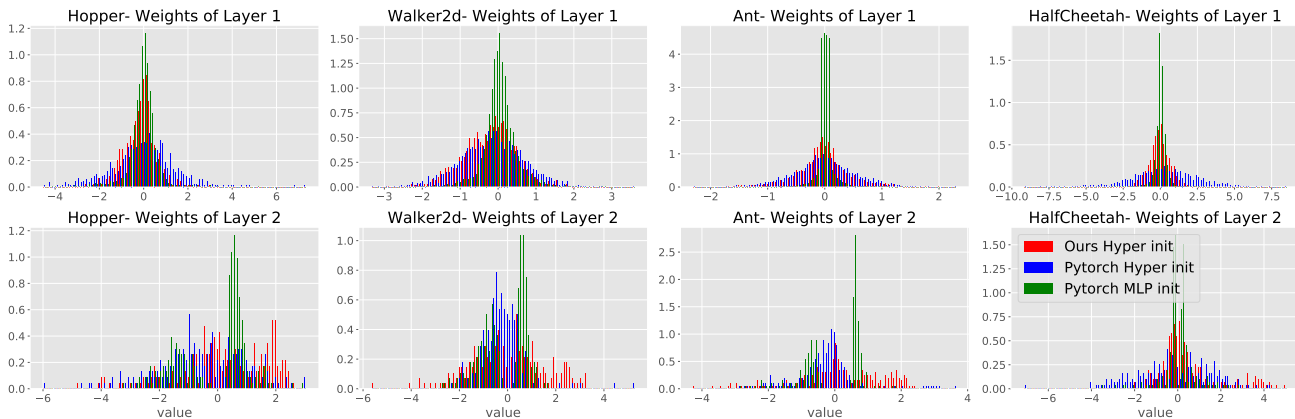

Figure 11. Dynamic network weight distribution of different initialization schemes **after 100K training steps**. The "Hyper init" refers to a primary network initialized with our suggested initialization scheme. 'Pytorch Hyper init' refers to the Pytorch default initialization of the primary network and 'Pytorch MLP init' weight distribution of the MLP-Small model (same architecture as the dynamic network).

D.4. Baseline Models for the SAC and TD3 algorithms

In our TD3 and SAC experiments, we tested the Hypernetwork architecture with respect to 7 different baseline models.

D.4.1. MLP-STANDARD

A standard MLP architecture, which is used in many RL papers (e.g. SAC and TD3) with 2 hidden layers of 256 neurons each with ReLU activation function.

D.4.2. MLP-SMALL

The MLP-Small model helps in understanding the gain of using context-dependent dynamic weights. It is an MLP network with the same architecture as our dynamic network model, i.e. 1 hidden layer with 256 neurons followed by a ReLU activation function. As expected, although the MLP-Small and MLP-Standard configurations are relatively similar with only a different number of hidden layers (1 and 2 respectively), the MLP-Small achieved close to half the return of the MLP-Standard. However, our experiments show that when using even a shallow MLP network with context-dependent weights (i.e. our SA-Hyper model), it can significantly outperform both shallow and deeper standard MLP models.

Table 2. Total-variation distance between the dynamic weight distribution and the "Pytorch MLP init" weight distribution: at the beginning of the training process we find that 'Pytorch Hyper init' is closer to the 'Pytorch MLP init' weight distribution while after $100K$ training steps we find that our initialization is closer to the 'Pytorch MLP init' weights (also trained for $100K$ steps).

Primary Initialization Scheme	Hopper	Walker2d	Ant	HalfCheetah
First Layer				
Ours Hyper init	31.4	23.9	13.6	29.4
Pytorch Hyprer init	**16.3**	**20.5**	**9.2**	**8.8**
Second Layer				
Ours Hyper init	34.8	**30.77**	37.7	36.9
Pytorch Hyprer init	**24.7**	39.6	**11.2**	**29.3**
First Layer After 100K Steps				
Ours Hyper init	**14.4**	**19.6**	**29.9**	**16.4**
Pytorch Hyprer init	24.9	22.6	34.0	22.4
Second Layer After 100K Steps				
Ours Hyper init	**31.2**	28.5	**30.6**	**21.1**
Pytorch Hyprer init	32.11	**20.8**	30.7	31.1

D.4.3. MLP-LARGE

To make sure that the performance gain is not due to the large number of weights in the primary network, we evaluated MLP-Large, an MLP network with 2 hidden layers as the MLP-Standard but with 2,900 neurons in each layer. This yields a total number of $\sim 9M$ learnable parameters, as in our entire primary model. While this large network usually outperformed other baselines, in almost all environments it still did not reach the Hypernetwork performance with one exception in the Ant-v2 environment in the TD3 algorithm. This provides another empirical argument that Hypernetworks are more suited for the RL problem and their performance gain is not only due to their larger parametric space.

D.4.4. RESNET FEATURES

To test whether the performance gain is due to the expressiveness of the ResNet model, we evaluated ResNet-Features: an MLP-Small model but instead of plugging in the raw state features, we use the primary model configuration (with ResNet blocks) to generate 10 learnable features of the state. Note that the feature extractor part of ResNet-Features has a similar parameter space as the Hypernetwork's primary model except for the head units. The ResNet-Features was unable to learn on most environments in both algorithms, even though we tried several different initialization schemes. This shows that the primary model is not suitable for a state's features extraction, and while it may be possible to find other models with ResNet that outperform this ResNet model, it is yet further evidence that the success of the Hypernetwork architecture is not attributed solely to the ResNet expressiveness power in the primary network.

D.4.5. AS-HYPER

This is the reverse configuration of our SA-Hyper model. In this configuration, the action is the meta-variable and the state serves as the base-variable. Its lower performance provides another empirical argument (alongside the lower CS, see Sec. B) that the "correct" Hypernetwork composition is when the state plays the context role and the action is the base-variable.

D.4.6. EMB-HYPER

In this configuration, we replace the input of the primary network with a learnable embedding of size 5 (equal to the PEARL context size) and the dynamic part gets both the state and the action as its input variables. This produces a learnable set of weights that is constant for all states and actions. However, unlike MLP-Small, the weights are generated via the primary model and are not independent as in normal neural network training. Note that we did not include this experiment in the

main paper but we have added it to the results in the appendix. This is another configuration that aims to validate that the Hypernetwork gain is not due to the over-parameterization of the primary model and that the disentanglement of the state and action is an important ingredient of the Hypernetwork performance.

D.4.7. RESNET 35

To validate that the performance gain is not due to a large number of weights in the primary network combined with the expressiveness of the residual blocks, we evaluated a full ResNet architecture: The state and actions are concatenated and followed by 35 ResNet blocks. Each block contains two linear layers of 256 size (and an identity path). This yields a a total number of $\sim 4.5M$ learnable parameters, which is half of the $9M$ parameters in the Hypernetwork model. In almost all environments it underperformed both with respect to SA-Hyper and also with respect to the MLP-Standard baseline.

D.4.8. Q-D2RL

The Deep Dense architecture (D2RL) (Sinha et al., 2020) suggests to add skip connections from the input to each hidden layer. In the original paper this applies both to the Q-net model, where states and actions are concatenated and added to each hidden layer, and to policies where only states are added to each hidden layer. According to the paper, the best performing model contains 4 hidden layers. Here, we compared to Q-D2RL which only modifies the Q-net as our SA-Hyper model but does not alter the policy network. Q-D2RL shows an inconsistent performance between SAC and TD3. In the SAC algorithm, it performs close to the SA-Hyper in all environments. On the other hand, in the TD3 algorithm, Q-D2RL was unable to reach the SA-Hyper performance in any environment.

D.5. Complexity and Run Time Considerations

Modern deep learning packages such as Pytorch and Tensorflow currently do not have optimized implementation of Hypernetworks as opposed to conventional neural architectures such as CNN or MLP. Therefore, it is not surprising that the training of Hypernetwork can take a longer time than MLP models. However, remarkably, in MAML we were able to reduce the training time as the primary weights and gradients are calculated only once for each task and the dynamic network is smaller than the Vanilla-MAML MLP network. Therefore, within each task, both data collection and gradient calculation with the dynamic model requires less time than the Vanilla-MAML network. In Table D.5 we summarize the average training time of each algorithm and compare the Hyper and MLP configurations.

Table 3. Comparing the algorithms' average running time between Hyper and MLP models: Single iteration training time for the MAML algorithm and 5K steps training time for all other algorithms. Note that each agent was trained using a single NVIDIA® GeForce® RTX 2080 Ti GPU with a 11019 MiB memory.

Algorithm	MLP	Hyper
SAC	$120s$	$200s$
TD3	$40s$	$140s$
PEARL	$450s$	$700s$
MAML	$150s$	$145s$
Multi-Task MAML	-	$120s$

E. Experiments

In this section, we report the training results of all tested algorithms as well as the hyperparameters used in these experiments. For each algorithm, we plot the mean reward and standard deviation over five different seeds. The evaluation procedure of single task RL algorithms was done every $5K$ training steps, with a mean calculated over ten independent trajectory roll-outs, without exploration, as described in (Fujimoto et al., 2018). The evaluation procedure of the Meta-RL algorithms was done after every algorithm's iteration, with a mean calculated over all test tasks' roll-outs, as was done in (Rakelly et al., 2019). In 'Velocity' tasks in Meta-RL, we sample training and test tasks from $[0, 3]$ except for the HalfCheetah-Vel-Medium(OOD) environment which the training tasks sample from $[0, 2.5]$ and the test tasks sample from $[2.5, 3]$. We used 100 training tasks and 30 tests tasks for both algorithms (PEARL and MAML) on "Velocity" tasks and 2 tasks for the "Direction" tasks, forward and backward.
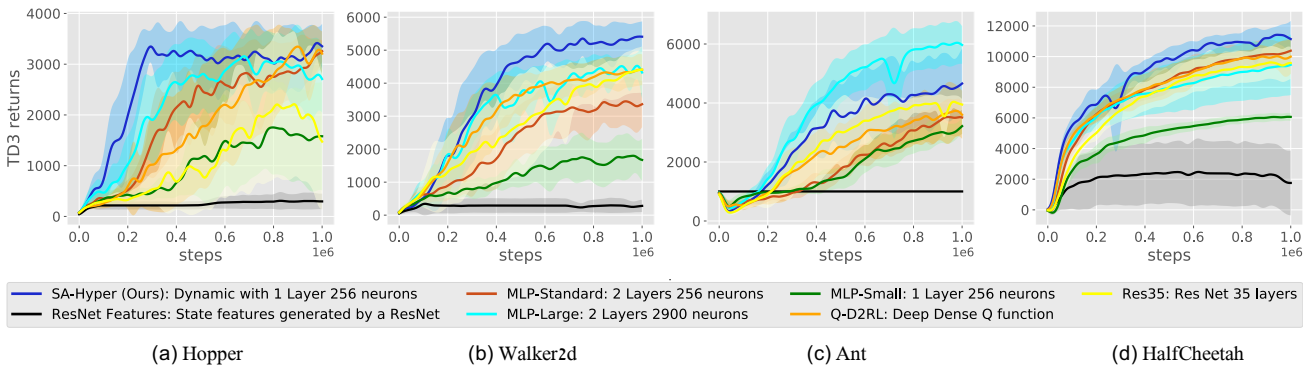
E.1. TD3

Figure 12. TD3 performance of different MLP architectures compared to the SA-Hyper. SA-Hyper shows consistent high performance in all environments and outperforms all other architectures except for the Ant environment.
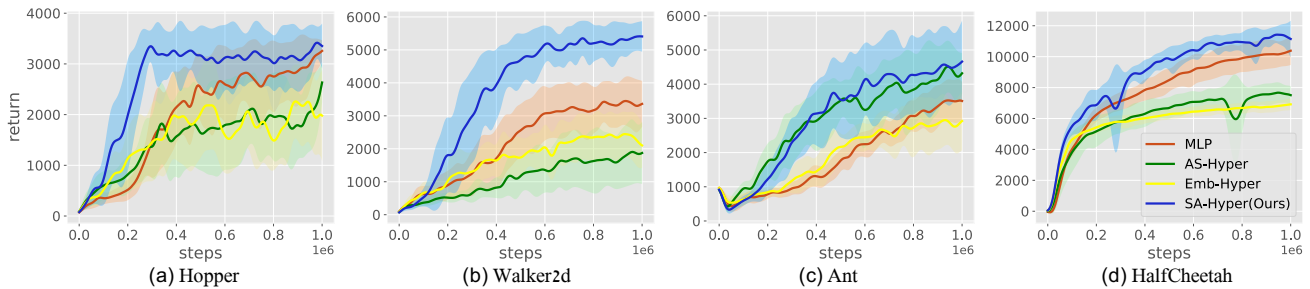

Figure 13. TD3 Performance with Hypernetwork critic compared to MLP critic over different 'Mujoco' environments. In all environments, Hypernetwork outperforms all the baselines.

Table 4. TD3 highest rewards.

Network	Hopper	Walker2D	Ant	HalfCheetah
MLP-Standard	3256 ± 211	3449 ± 730	3524 ± 617	10384 ± 923
MLP-Large	3156 ± 368	4527 ± 397	$\mathbf{6042 \pm 731}$	9467 ± 1978
MLP-Small	1756 ± 926	$1799 \pm 538m$	3215 ± 267	6071 ± 256
ResNet-Features	307 ± 173	343 ± 349	1001 ± 1	2474 ± 2184
ResNet35	2213 ± 1431	4411 ± 703	4042 ± 215	9621 ± 1072
Q-D2RL	3347 ± 270	4408 ± 473	3736 ± 881	10023 ± 867
AS-Hyper	2633 ± 391	1905 ± 985	4513 ± 759	7669 ± 667
Emb-Hyper	2261 ± 728	2446 ± 676	2949 ± 741	6915 ± 374
SA-Hyper (Ours)	$\mathbf{3418 \pm 318}$	$\mathbf{5412 \pm 445}$	4660 ± 1194	$\mathbf{11423 \pm 560}$

Table 5. TD3 Hyper Parameters

Hyper-parameter	TD3	Hyper TD3 (Ours)
Actor Learning Rate	$3e^{-4}$	$3e^{-4}$
Critic Learning Rate	$3e^{-4}$	$5e^{-5}$
Optimizer	Adam	Adam
Batch Size	100	100
Policy update frequency	2	2
Discount Factor	0.99	0.99
Target critic update	0.005	0.005
Target policy update	0.005	0.005
Reward Scaling	1	1
Exploration Policy	$N(0, 0.1)$	$N(0, 0.1)$

E.2. SAC

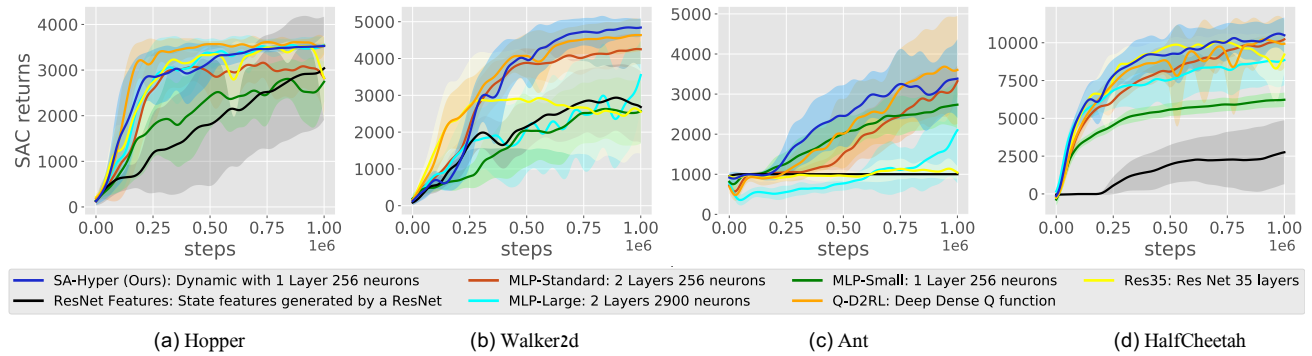
(a) Hopper   (b) Walker2d   (c) Ant   (d) HalfCheetah

Figure 14. SAC Performance of different critic models.

Table 6. SAC highest rewards.

Network	Hopper	Walker2D	Ant	HalfCheetah
MLP-Standard	3160 ± 327	4258 ± 413	3323 ± 389	10225 ± 324
MLP-Large	3549 ± 160	3550 ± 936	2100 ± 1322	8853 ± 1663
MLP-Small	2806 ± 425	2629 ± 804	2735 ± 589	6229 ± 475
ResNet-Features	3038 ± 1129	2936 ± 896	1002 ± 1	2755 ± 2114
ResNet35	3525 ± 40	2923 ± 1369	1138 ± 252	10096 ± 468
Q-D2RL	$\mathbf{3612 \pm 51}$	4638 ± 441	$\mathbf{3684 \pm 1207}$	10224 ± 1090
SA-Hyper (Ours)	3527 ± 40	$\mathbf{4844 \pm 254}$	3385 ± 983	$\mathbf{10600 \pm 950}$

Table 7. SAC Hyper Parameters

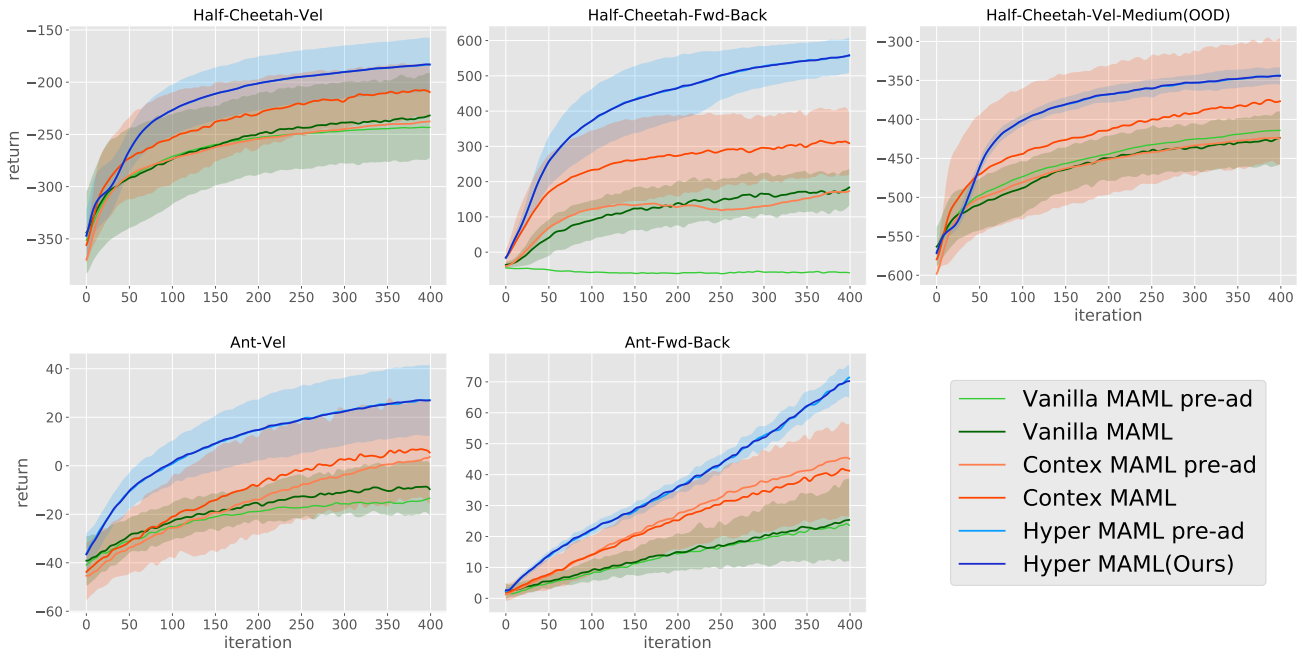| Hyper-parameter	SAC	Hyper SAC (Ours)
Actor Learning Rate	$3e^{-4}$	$2e^{-5}$, $1e^{-4}$ for 'HalfCheetah'
Critic Learning Rate	$3e^{-4}$	$5e^{-5}$
Optimizer	Adam	Adam
Batch Size	256	256
Discount Factor	0.99	0.99
Target critic update	0.005	0.005
Reward Scaling	5	5

E.3. MAML

Figure 15. MAML Performance over **test tasks** with a Hypernetwork policy compared to MLP policy with and without a given context of the tasks by an *oracle*. The oracle-context improves the MAML performance but Hyper-MAML outperforms Context-MAML and, importantly, it does not require an adaptation step.
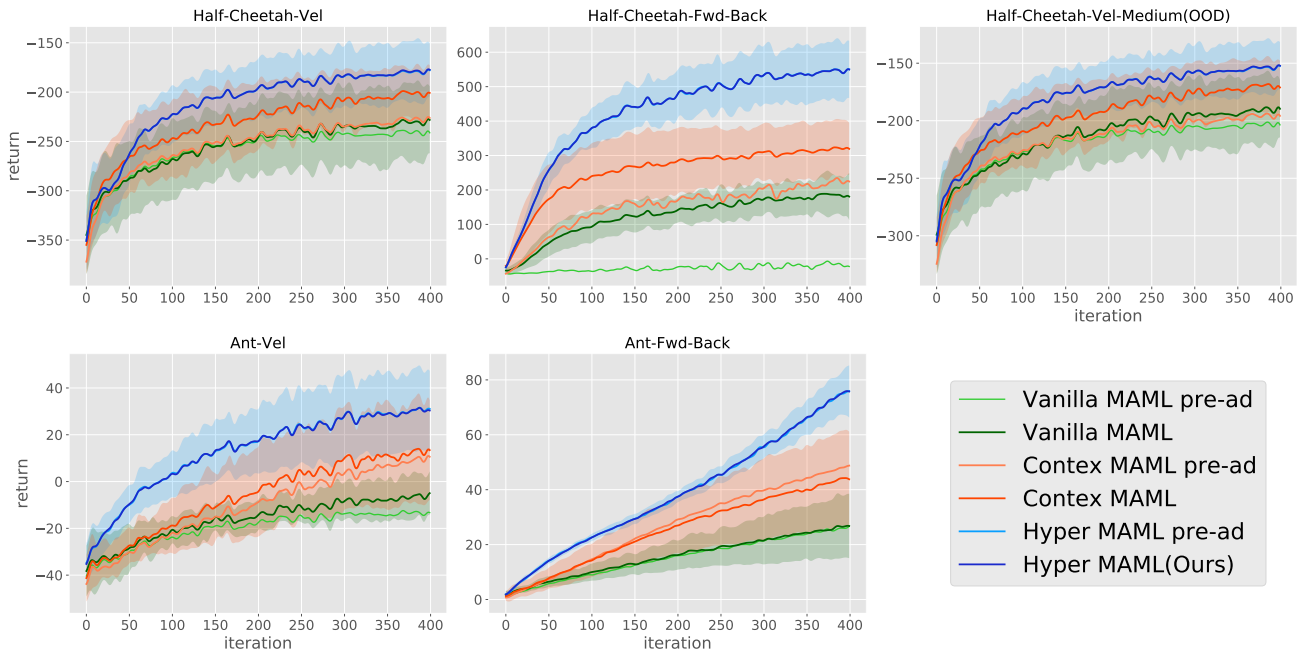

Figure 16. MAML Performance over **training tasks** with a Hypernetwork policy compared to MLP policy with and without a given context of the tasks by an *oracle*. The oracle-context improves the MAML performance but Hyper-MAML outperforms Context-MAML and, importantly, it does not require an adaptation step.

E.3.1. ELIMINATING THE ADAPTATION STEP

Our experiments show that taking the MAML adaptation step is unnecessary when using the Hyper-MAML model with an oracle-context (as opposed to Context-MAML which uses an oracle-context but still benefits from the adaptation step). We further investigate whether we can also eliminate the adaptation step during training s.t. the gradient of each task is calculated with the policy current weights as opposed to MAML which calculates the gradient at the policy's adapted weights. We term this method as Multi-Task Hyper-MAML (following (Fakoor et al., 2019) which termed the Meta-RL objective without adaptation as a multi-task objective). In Fig. 17 we find that Multi-Task Hyper-MAML outperforms the Hyper-MAML with adaptation. Moreover, Table D.5 shows that it also requires less training time as it removes the unnecessary complexity of the MAML adaptation training.
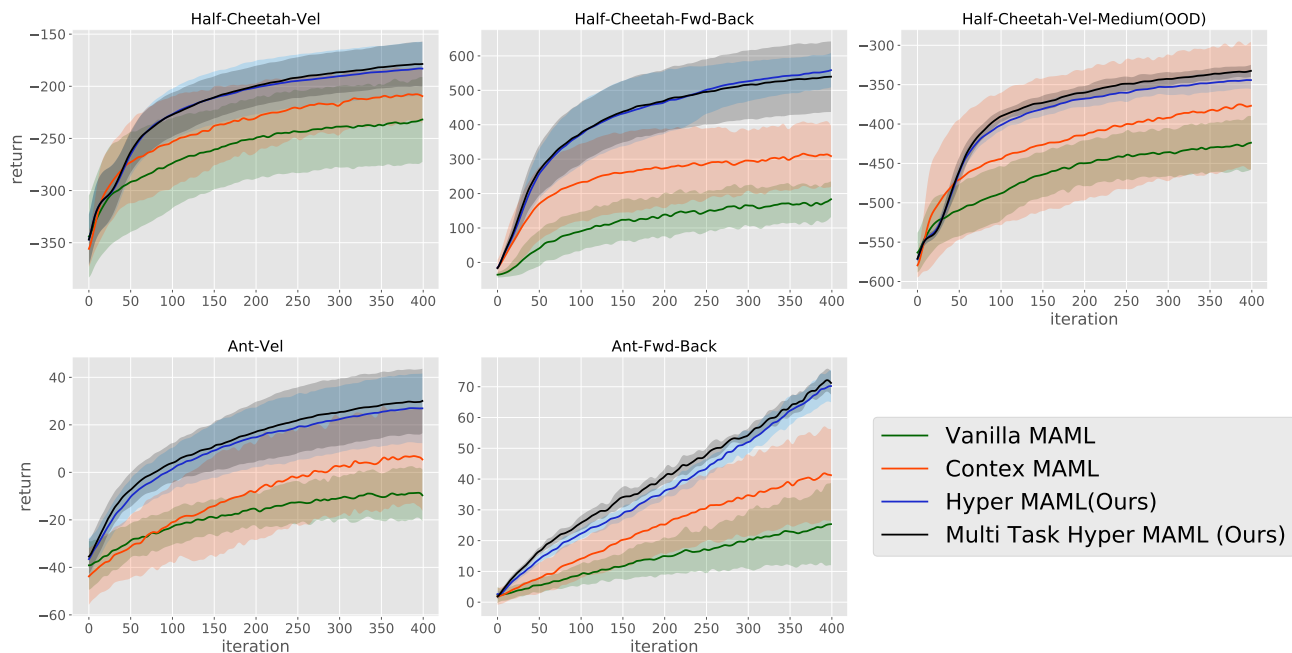

Figure 17. Multi-Task Hyper-MAML performance over the **test tasks** with a Hypernetwork policy and a multi-task objective. Using a multi-task objective matches or outperforms the MAML objective without the need for the adaption step training.

Table 8. MAML highest rewards.

Algorithm	Cheetah-Vel	Cheetah-Fwd-Back	Cheetah-Vel-Med	Ant-Vel	Ant-Fwd-Back
MAML	-231 ± 40	183 ± 51	-423 ± 33	-8 ± 10	25 ± 13
Context MAML	-207 ± 25	315 ± 93	-374 ± 79	6 ± 19	41 ± 15
Hyper Multi-Task (Ours)	$\mathbf{-178 \pm 21}$	539 ± 102	$\mathbf{-332 \pm 7}$	$\mathbf{30 \pm 13}$	$\mathbf{72 \pm 3}$
Hyper MAML (Ours)	-182 ± 25	$\mathbf{558 \pm 49}$	-344 ± 10	27 ± 14	70 ± 5

E.4. PEARL

Table 9. MAML Hyperparameters

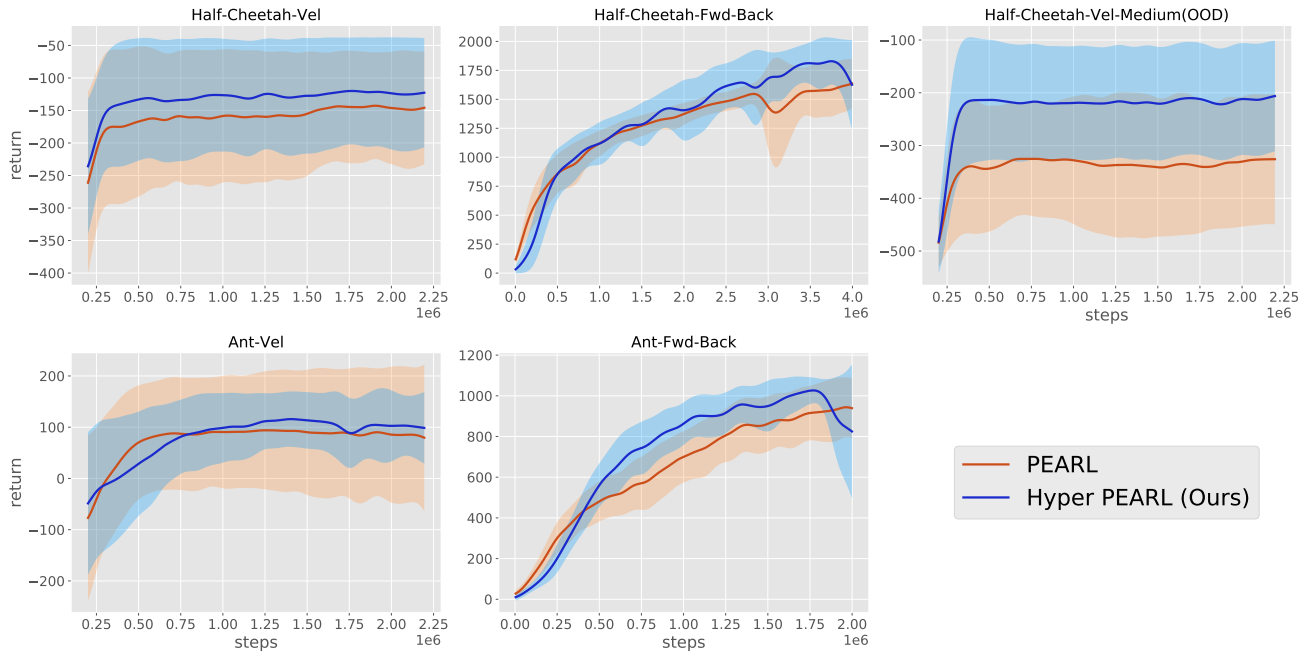| Hyperparameter	MAML	Hyper MAML (Ours)
Batch Size	20	20
Meta batch Size	40	40
Discount Factor	0.95	0.95
Num of Iterations	400	400
Max KL	$1e^{-2}$	$1e^{-2}$
LS Max Steps	20	20
Episode Max Steps	200	200

Figure 18. PEARL Performance over the **test tasks** with policy and critic Hypernetworks compared to MLP policy and critic. Hypernetwork outperforms or matches MLP in all environments.

Table 10. PEARL highest rewards.

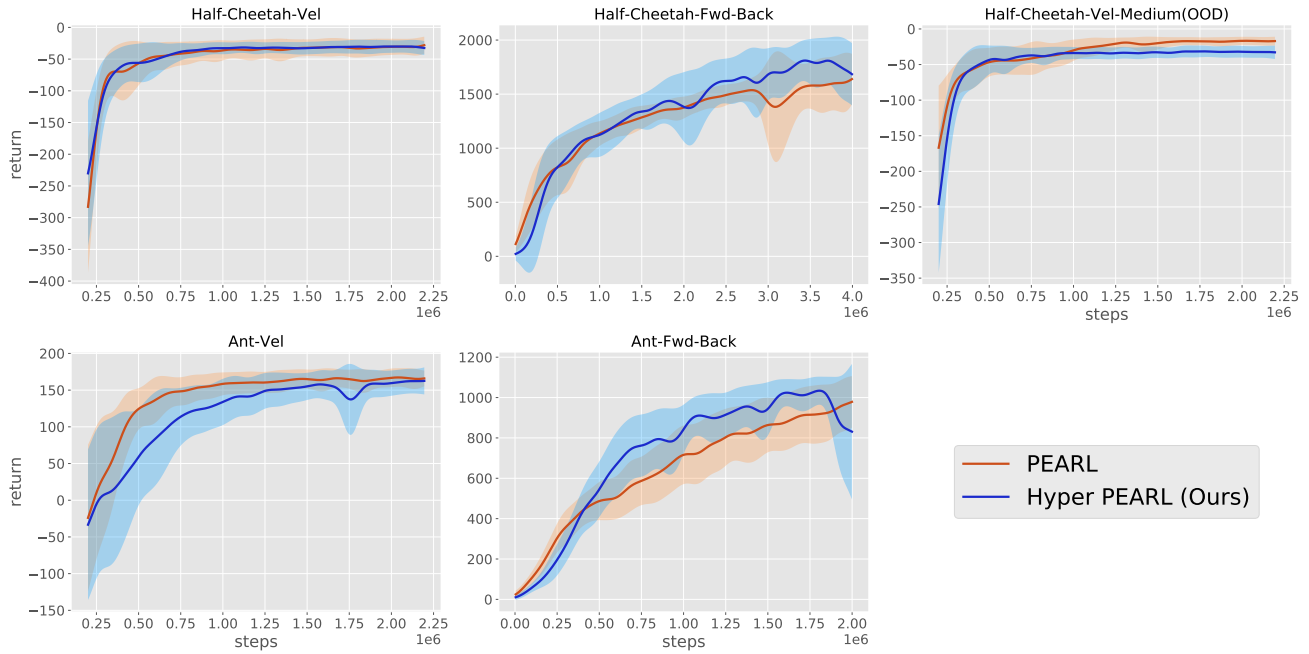| Algorithm	Cheetah-Vel	Ant-Vel	Cheetah-Vel-Med	Cheetah-Fwd-Back	Ant-Fwd-Back
PEARL	-142 ± 82	1636 ± 210	-325 ± 109	93 ± 115	943 ± 146
Hyper PEARL (Ours)	$\mathbf{-119 \pm 82}$	$\mathbf{1828 \pm 203}$	$\mathbf{-206 \pm 104}$	$\mathbf{115 \pm 54}$	$\mathbf{1026 \pm 62}$

Figure 19. PEARL Performance over **training tasks** with policy and critic Hypernetworks compared to MLP policy and critic.

Table 11. PEARL Hyperparameters

Hyperparameter	PEARL	Hyper PEARL
Actor Learning Rate	$3e^{-4}$	$1e^{-4}$
Critic Learning Rate	$3e^{-4}$	$5e^{-5}$
Context Learning Rate	$3e^{-4}$	$3e^{-4}$
Value Learning Rate	$3e^{-4}$	$5e^{-5}$
Optimizer	Adam	Adam
Batch Size	256	256
'Dir' Tasks Meta batch Size	4	4
'Vel' Tasks Meta batch Size	16	16
Target critic update	0.005	0.005
Discount Factor	0.99	0.99
Num of Iterations	400	400
Reward Scaling	5	5
Episode Max Steps	200	200