Recomposing the Reinforcement Learning Building-Blocks with Hypernetworks: Supplementary Materials

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. $|\nabla_a Q^\pi(s,a_1) - \nabla_a Q^\pi(s,a_2)| \le \kappa_q \|a_1 - a_2\|$. Define the average gradient operator $\overline{\nabla}_\phi f = \mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{\varepsilon\sim p_\varepsilon}[\nabla_\phi \mu_\phi(\varepsilon|s)\cdot f(s,\mu_\phi(\varepsilon|s))]\right]$. If there exists a gradient estimation $g(s,a)$ and $0 < \alpha < 1$ s.t.*

$$\|\overline{\nabla}_\phi \cdot g - \overline{\nabla}_\phi \cdot \nabla_a Q^\pi\| \le \alpha \|\overline{\nabla}_\phi \cdot \nabla_a Q^\pi\| \tag{1}$$

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

Proof. First, recall the objective to be optimized:

$$\begin{aligned} J(\phi) &= \mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{\varepsilon\sim p_\varepsilon}\left[Q^\pi(s,\mu_\phi(\varepsilon;s))\right]\right] \\ \nabla_\phi J(\phi) &= \mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{\varepsilon\sim p_\varepsilon}\left[\nabla_\phi \mu_\phi(\varepsilon;s)\cdot\nabla_a Q^\pi(s,\mu_\phi(\varepsilon;s))\right]\right] = \overline{\nabla}_\phi \cdot \nabla_a Q^\pi \end{aligned} \tag{2}$$

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 $\|\nabla_a Q^\pi\| \le \sigma_q$ and $\|\nabla_\phi \mu_\pi\| \le \sigma_\mu$.

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

Proof.

$$\begin{aligned} \|c(x_1) - c(x_2)\| &= \|A(x_1)\cdot b(x_1) - A(x_2)\cdot b(x_2)\| \\ &= \|A(x_1)\cdot b(x_1) - A(x_1)\cdot b(x_2) + A(x_1)\cdot b(x_2) - A(x_2)\cdot b(x_2)\| \\ &\le \|A(x_1)\cdot(b(x_1) - b(x_2))\| + \|(A(x_1) - A(x_2))\cdot b(x_2)\| \\ &\le \|A(x_1)\|\|b(x_1) - b(x_2)\| + \|A(x_1) - A(x_2)\|\|b(x_2)\| \\ &\le (\beta M_a + \alpha M_b)\|x_1 - x_2\| \end{aligned}$$

□

The Lipschitz constant of the objective gradient is bounded by

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

Applying Lemma 1, we obtain

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

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

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

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

$$J(\phi') \geq J(\phi) + \eta \left(\overline{\nabla}_\phi \cdot g\right) \cdot \left(\overline{\nabla}_\phi \cdot Q^\pi\right) - \frac{\eta^2 (\kappa_q \sigma_\mu + \kappa_\mu \sigma_q)^2}{2} \|\overline{\nabla}_\phi \cdot g\|^2. \tag{3}$$

Taking the second term on the right-hand side,

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

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. (3) 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)

$$cs(Q^{\pi}_{\theta}) = \mathbb{E}_{s\sim\mathcal{D}}\left[\frac{\nabla_a Q^{\pi}_{\theta}(s,a_\mu)\cdot\nabla_a Q^{\pi}(s,a_\mu)}{\|\nabla_a Q^{\pi}_{\theta}(s,a_\mu)\|\,\|\nabla_a Q^{\pi}(s,a_\mu)\|}\right], \tag{4}$$

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

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

, therefore

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

for a_1, a_2 at the vicinity of a_μ.

To find the best fit $g\simeq\nabla_a Q^{\pi}_{\theta}(s,a_\mu)$ 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. 1 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. 2 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.

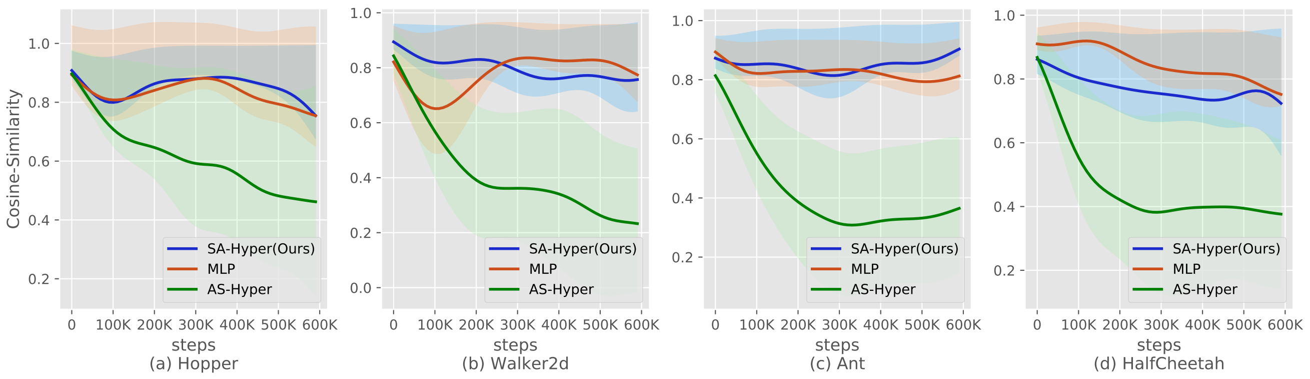


Figure 1. The Cosine-Similarity between the LMSE estimator of the Q-net outputs and the parametric gradient averaged with a window size of $W = 20$.

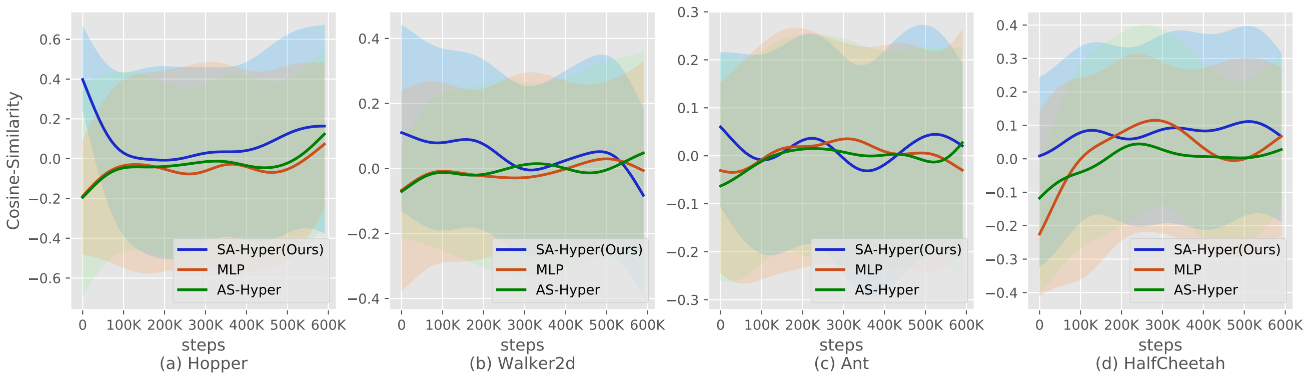


Figure 2. The Cosine-Similarity between the LMSE estimator of the empirical sum of rewards and the parametric gradient averaged with a window size of $W = 20$.

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. 3). As is shown in Fig. 4, 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. 3 and Fig. 4 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. 3 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).

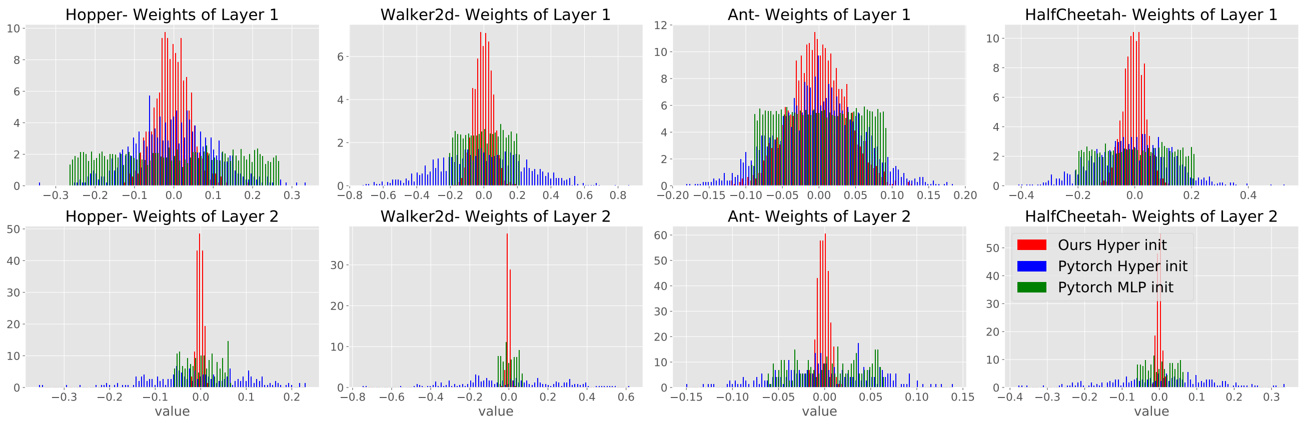


Figure 3. Dynamic network weight distribution of different initialization schemes **at the beginning of the training**. The “Hyper init” refers to a primary network initialized with our suggested initialization scheme. ’Pytorch Hyper init’ refers to the Pytorch default initialization of the primary network and “Pytorch MLP init” refers to the Pytorch default initialization of the MLP-Small model (same architecture as the dynamic network).

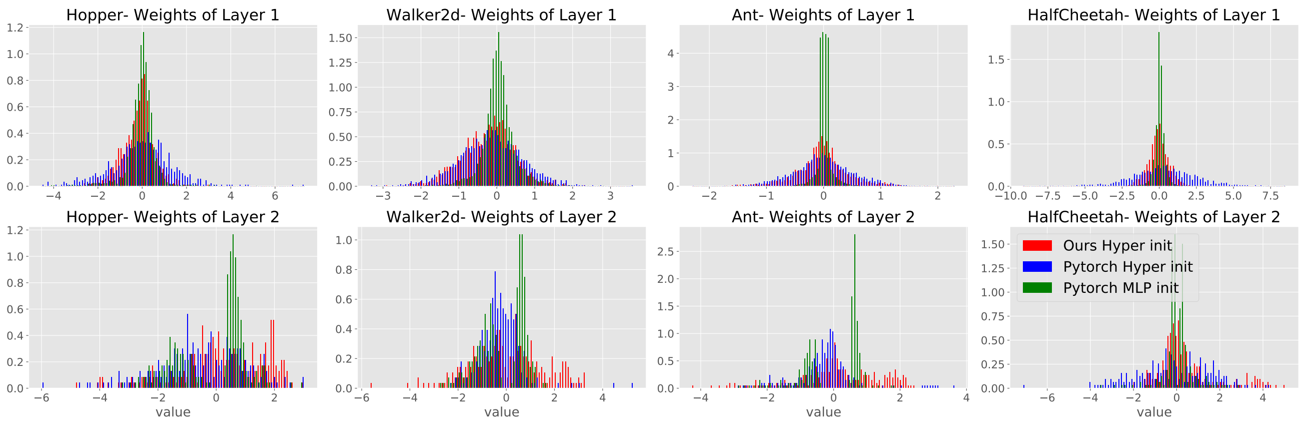


Figure 4. Dynamic network weight distribution of different initialization schemes **after 100K training steps**. The “Hyper init” refers to a primary network initialized with our suggested initialization scheme. ’Pytorch Hyper init’ refers to the Pytorch default initialization of the primary network 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.

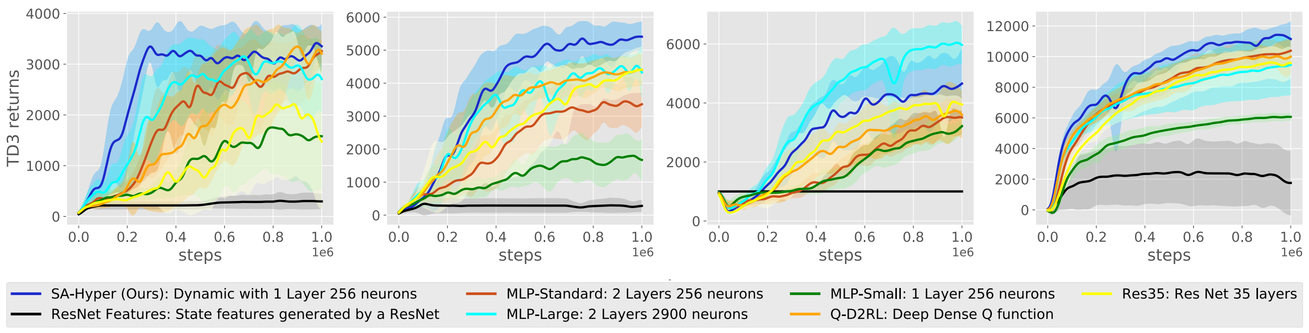
Table 3. Comparing the algorithms' average running time between Hyper and MLP models: Single iteration training time for the MAML algorithm and 5K steps training time for all other algorithms. Note that each agent was trained using a single NVIDIA® GeForce® RTX 2080 Ti GPU with a 11019 MiB memory.

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

E. Experiments

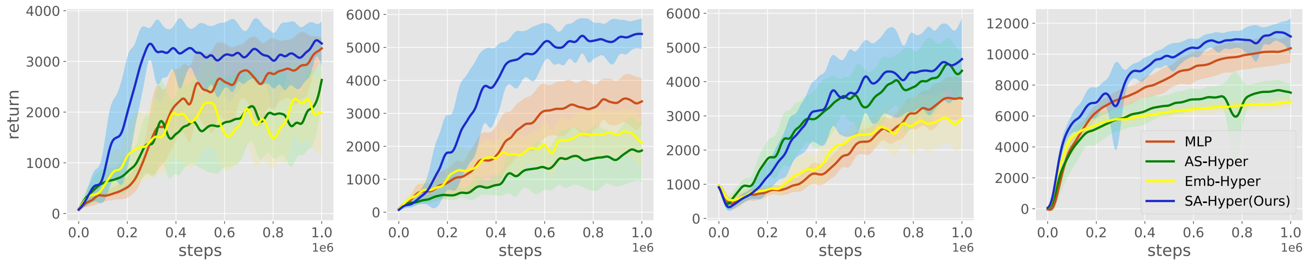
In this section, we report the training results of all tested algorithms as well as the hyperparameters used in these experiments. For each algorithm, we plot the mean reward and standard deviation over five different seeds. The evaluation procedure of single task RL algorithms was done every $5K$ training 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



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

Figure 5. 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.



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

Figure 6. 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	**6042 ± 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)	**3418 ± 318**	**5412 ± 445**	4660 ± 1194	**11423 ± 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

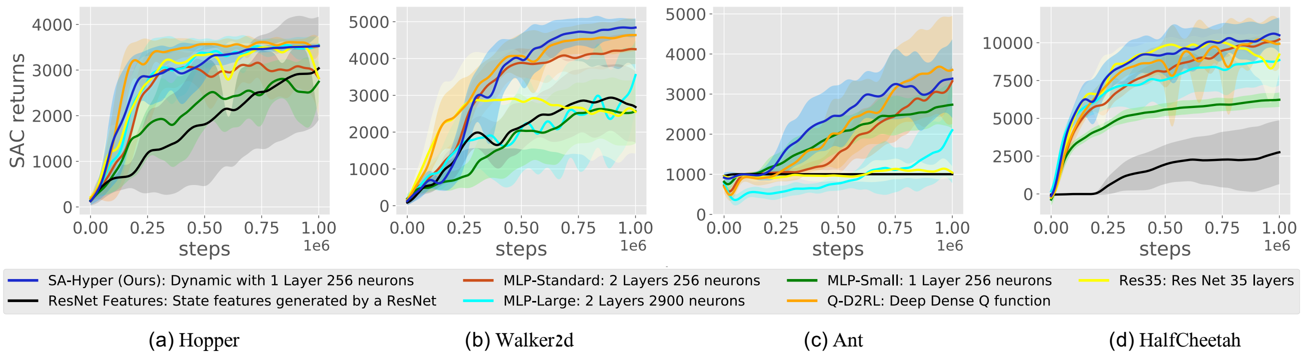


Figure 7. SAC Performance of different critic models.

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	**3612** $\pm$ **51**	4638 ± 441	**3684** $\pm$ **1207**	10224 ± 1090
SA-Hyper (Ours)	3527 ± 40	**4844** $\pm$ **254**	3385 ± 983	**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

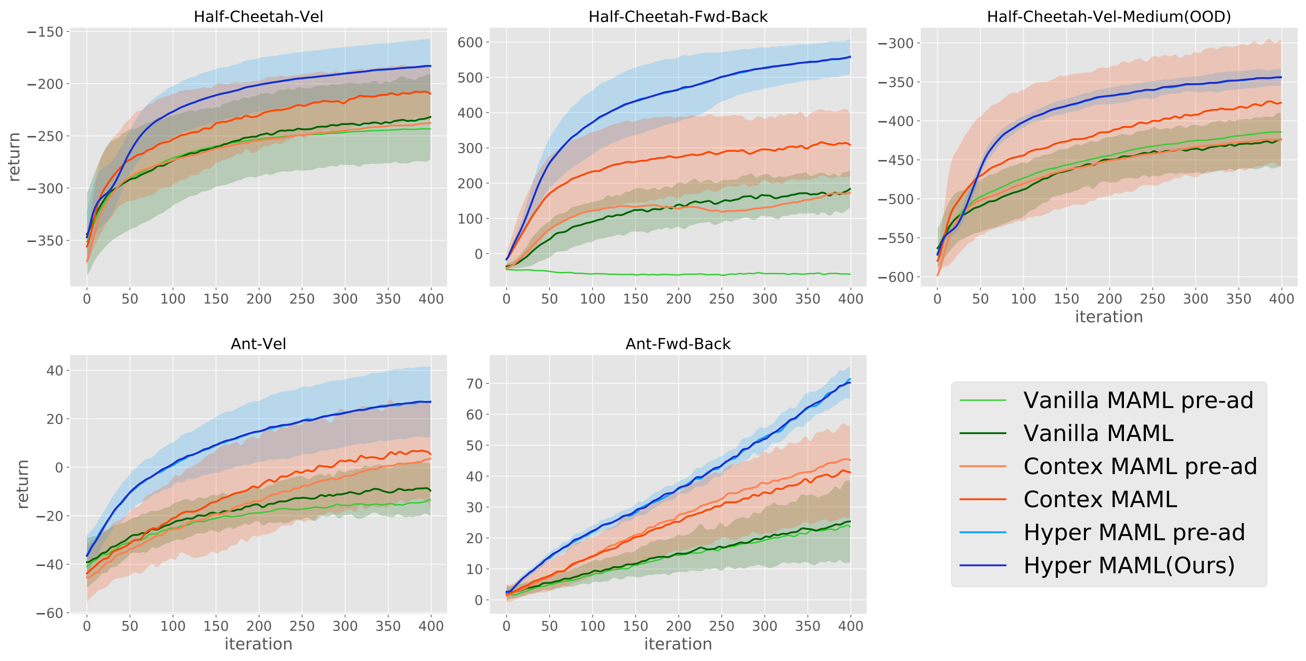


Figure 8. MAML Performance over **test tasks** with a Hypernetwork policy compared to MLP policy with and without a given context of the tasks by an *oracle*. The oracle-context improves the MAML performance but Hyper-MAML outperforms Context-MAML and, importantly, it does not require an adaptation step.

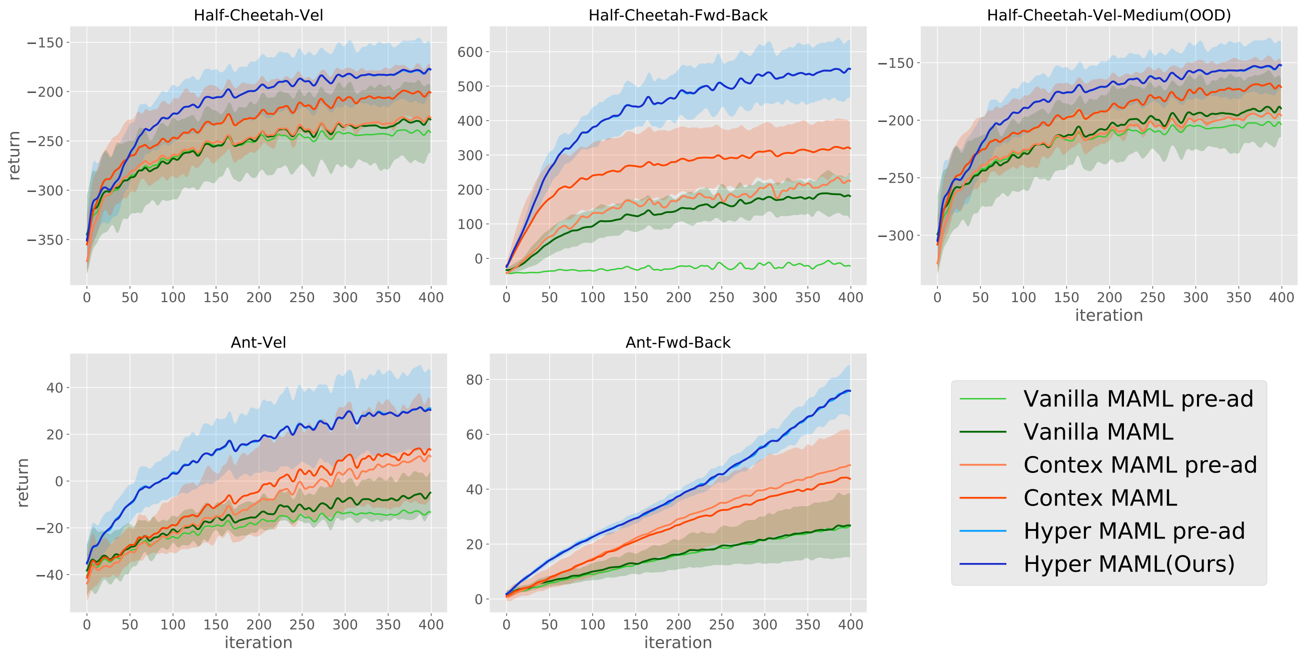


Figure 9. MAML Performance over **training tasks** with a Hypernetwork policy compared to MLP policy with and without a given context of the tasks by an *oracle*. The oracle-context improves the MAML performance but Hyper-MAML outperforms Context-MAML and, importantly, it does not require an adaptation 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. 10 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.

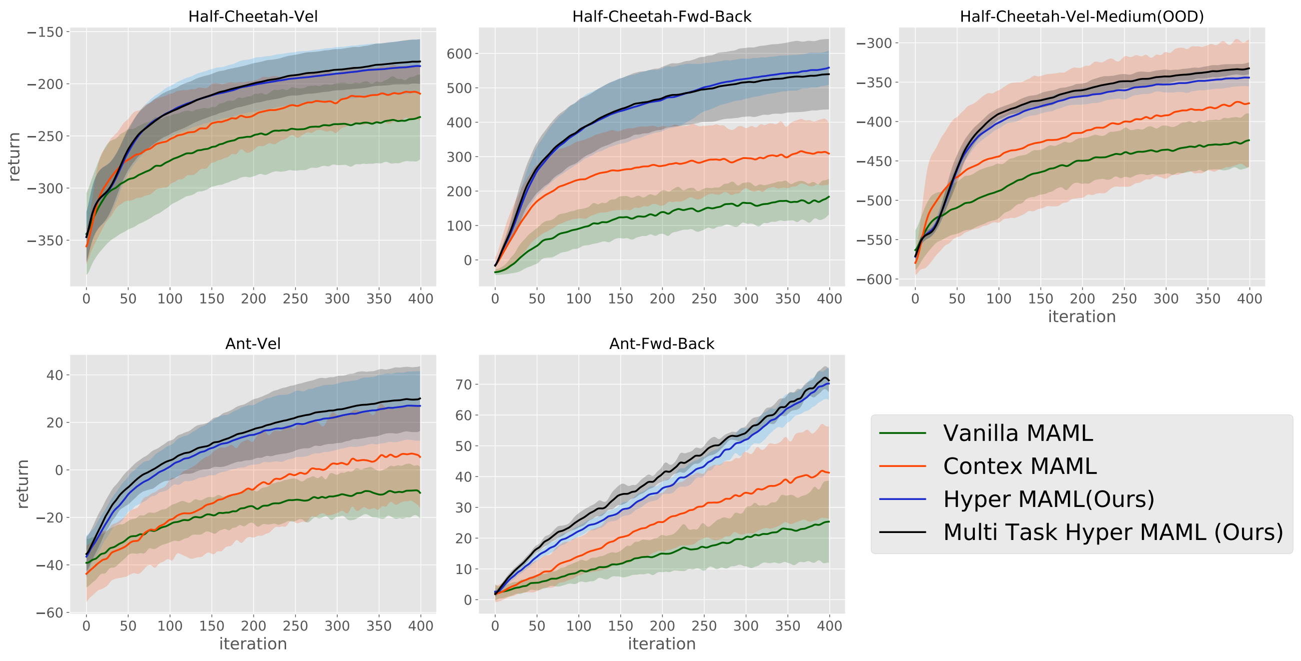


Figure 10. Multi-Task Hyper-MAML performance over the **test tasks** with a Hypernetwork policy and a multi-task objective. Using a multi-task objective matches or outperforms the MAML objective without the need for the adaption step training.

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

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

E.4. PEARL

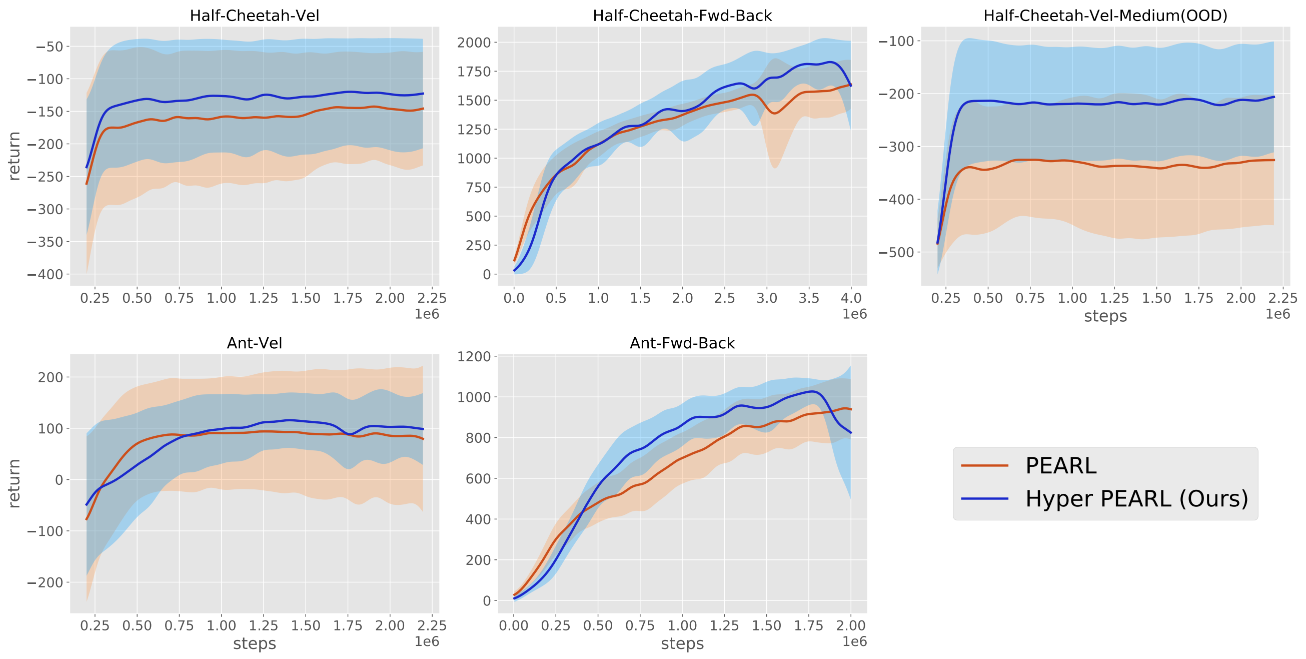


Figure 11. PEARL Performance over the **test tasks** with policy and critic Hypernetworks compared to MLP policy and critic. Hypernetwork outperforms or matches MLP in all environments.

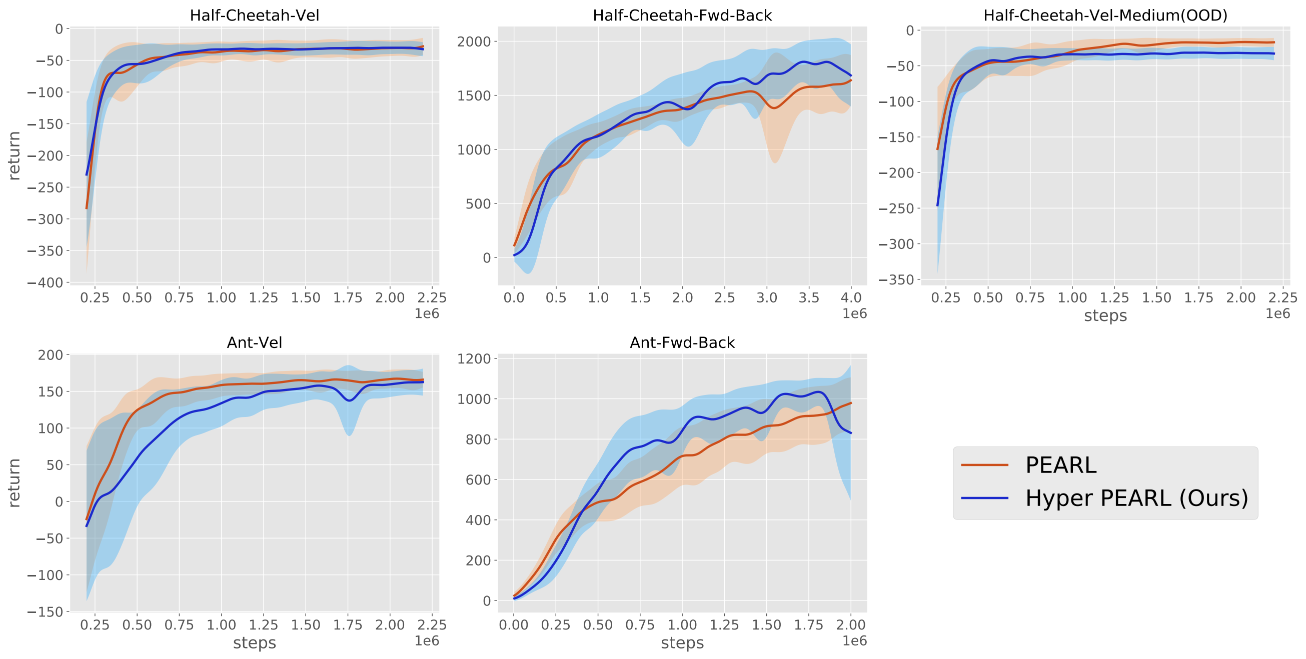


Figure 12. PEARL Performance over **training tasks** with policy and critic Hypernetworks compared to MLP policy and critic.

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)	$-\mathbf{119} \pm \mathbf{82}$	$\mathbf{1828} \pm \mathbf{203}$	$-\mathbf{206} \pm \mathbf{104}$	$\mathbf{115} \pm \mathbf{54}$	$\mathbf{1026} \pm \mathbf{62}$

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

References

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

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

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

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.

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.

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

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

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.

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

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