Supplemental Material for Policy Optimization as Wasserstein Gradient Flows

A. More Details on Preliminaries

We provide more details on some parts of the preliminaries section in the main text.

A.1. Gradient Flows on the Euclidean Space

For a smooth (convex) function[§] $F : \mathbb{R}^n \to \mathbb{R}$, a starting point $\mathbf{x}_0 \in \mathbb{R}^n$. The gradient flow of $F(\mathbf{x})$ is defined as the solution of the following function:

$$\begin{cases} \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = -\nabla F(\mathbf{x}(t)), & \text{for } t > 0 \\ \mathbf{x}(0) = \mathbf{x}_0 \end{cases} \tag{16}$$

This is a standard Cauchy problem (Rulla, 1996), which has a unique solution if ∇F is Lipschitz continuous. When F is non-differentiable, we can replace the gradient with the subgradient, defined as $\partial F(\mathbf{x}) \triangleq \{\mathbf{p}' \in \mathbb{R}^n : F(\mathbf{y}) \geq F(\mathbf{x}) + \mathbf{p}\cdot(\mathbf{y}-\mathbf{x}), \forall \mathbf{y} \in \mathbb{R}^n\}$. Note $\partial F(\mathbf{x}) = \{\nabla F(\mathbf{x})\}$ if F is differentiable at $\mathbf{x}$. In this case, the gradient flow formula above is replaced as: $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} \in -\partial F(\mathbf{x}(t))$.

Numerical solution Exact solution to the gradient-flow problem (16) is typically intractable. Numerical methods is a default choice. A standard method to solve (16) is called *Minimizing Movement Scheme* (MMS), which is an iterative scheme that evolves $\mathbf{x}$ along the gradient of F on the current point for a small step in each iteration. Specifically, let the current point to be $\mathbf{x}_k$, the next point is defined as $\mathbf{x}_{k+1} = \mathbf{x}_k - \nabla F(\mathbf{x}_{k+1})h$, with h being the stepsize. Note $\mathbf{x}_{k+1}$ is equivalent to solving the following optimization problem:

$$\mathbf{x}_{k+1} = \arg\min_{\mathbf{x}} F(\mathbf{x}) + \frac{\| \mathbf{x} - \mathbf{x}_k \|^2}{2h} .$$

To explicitly spell out the dependency of $\mathbf{x}_k$ w.r.t. h, we rewrite $\mathbf{x}_k$ as $\mathbf{x}_k^{(h)}$. The numerical scheme can be proved to be accurate. Specifically, define $\mathbf{v}_{k+1}^{(h)} \triangleq \frac{\mathbf{x}_{k+1}^{(h)} - \mathbf{x}_k^{(h)}}{h}$. Also define two curves $\mathbf{x}^h, \tilde{\mathbf{x}}^h : [0, T] \to \mathbb{R}^n$ for $t \in (kh, (k+1)h]$ as:

$$\mathbf{x}^h(t) = \mathbf{x}_{k+1}^{(h)}, \;\; \tilde{\mathbf{x}}^h(t) = \mathbf{x}_k^{(h)} + (t - kh)\, \mathbf{v}_{k+1}^{(h)} \;\; .$$

The MMS is proved to converge to the original gradient flow (Ambrosio et al., 2005), stated in the following theorem.

[§]We will focus on the convex case since this the case for many gradient flows on the space of probability measures, detailed later.

Theorem 9 *Let* $\mathbf{v}^h(t) \triangleq \mathbf{v}_{k+1}^{(h)}$ *defined above. Suppose* $F(\mathbf{x}_0) < +\infty$ *and* $\inf F > -\infty$. *If*[¶] $h \to 0$, $\tilde{\mathbf{x}}^h$ *and* $\mathbf{x}^h$ *converge uniformly to a same curve* $\mathbf{x}(t)$, *and* $\mathbf{v}^h$ *weakly converges in* $\mathcal{L}^2$ *to a vector function* $\mathbf{v}(t)$, *such that* $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{v}$. *Furthermore, if the partial derivatives of* F *exist and are continuous, we have* $\mathbf{v}(t) = -\nabla F(\mathbf{x}(t))$ *for all* t.

B. Sketch Proofs for RL with WGF

Proof [Proof of Proposition 5]

We provide two methods for the proof.

The first method directly uses property of gradient flows. Note that the WGF is defined as

$$\begin{aligned} \partial_\tau \mu_\tau &= -\nabla \cdot (\mathbf{v}_\tau\, \mu_\tau) = \nabla \cdot \left(\mu_\tau \nabla (\frac{\delta F}{\delta \mu_\tau}(\mu_\tau)) \right) \\ &\triangleq -\nabla_W F(\mu_\tau) \, . \end{aligned}$$

Denote the inner product in the probability space induced by W_2 as $\langle \cdot, \cdot \rangle_W$, we have

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}\tau} F(\mu_\tau) &= \langle \nabla_W F(\mu_\tau), \frac{\mathrm{d}}{\mathrm{d}\tau} \mu_\tau \rangle_W \\ &= -\langle \nabla_W F(\mu_\tau), \nabla_W F(\mu_\tau) \rangle_W \, . \end{aligned} \tag{17}$$

For any $\tau_1 \geq \tau_0$, integrating (17) over $[t_0, t_1]$, we have

$$\begin{aligned} & F(\mu_{\tau_1}) - F(\mu_{\tau_0}) \\ =& -\int_{\tau_0}^{\tau_1} \langle \nabla_W F(\mu_\tau), \nabla_W F(\mu_\tau) \rangle_W \mathrm{d}\tau \leq 0 \, , \end{aligned}$$

where the last inequality holds due to the positiveness of the norm operator. Consequently, we have $F(\mu_{\tau_1}) \leq F(\mu_{\tau_0})$, which means the energy functional F decreases over time. In our case, the energy functional is defined as the KL divergence, which is convex in terms of distributions. As a result, evolving μ_τ along the gradient flow would reach the global minimum of the energy functional, *i.e.*, $\lim_{\tau \to \infty} \mu_\tau = e^{J(\pi_{\boldsymbol{\theta}})}$.

The second way uses property of the Fokker-Planck equation for diffusions. Since the WGF with energy functional in (12) is equivalent to a Fokker-Planck equation. Specifically,

[¶]h can also be a decreasing-stepsize sequence $\{h_k\}$ such that $h_k \to 0$.

according to Section 2.2, the solution of the gradient flow is described by the following Fokker-Planck equation:

$$\partial_\tau \mu_\tau = \nabla \cdot (-\mu_\tau \nabla J(\pi_{\boldsymbol{\theta}}) + \nabla \cdot (\mu_\tau)) \ , \qquad (18)$$

On the other hand, it is well known that the unique invariant probability measure for the FP equation (18) is:

$$\mu = e^{J(\pi_{\boldsymbol{\theta}})} = \lim_{\tau \to \infty} \mu_\tau \ .$$

This completes the proof. ■

Proof [Proof of Proposition 8] The first part of Proposition 8, stating that $\pi(\mathbf{a} \,|\, \mathbf{s})$ converges to $p_{s,\pi}(\mathbf{a}) \propto e^{Q(\mathbf{a},\mathbf{s})}$, follows by the same argument as the proof of Proposition 5. Now we derive the soft Bellman equation.

This follows from the definition of $Q(\mathbf{a}_t, \mathbf{s}_t)$, *i.e.*,

$$\begin{aligned}
&Q(\mathbf{a}_t, \mathbf{s}_t) \\
=&r(\mathbf{a}_t, \mathbf{s}_t) + \mathbb{E}_{(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}, \cdots) \sim (\rho_\pi, \pi)} \sum_{l=1}^{\infty} \gamma^l r(\mathbf{s}_{t+l}, \mathbf{a}_{t+l}) \\
=&r(\mathbf{a}_t, \mathbf{s}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim \rho_\pi} \mathbb{E}_{\mathbf{a}_{t+1} \sim \pi} \left[r(\mathbf{a}_{t+1}, \mathbf{s}_{t+1}) \right. \\
&\left. + \mathbb{E}_{(\mathbf{s}_{t+2}, \mathbf{a}_{t+2} \cdots) \sim (\rho_\pi, \pi)} \sum_{l=1}^{\infty} \gamma^l r(\mathbf{s}_{t+1+l}, \mathbf{a}_{t+1+l}) \right]
\end{aligned}$$

Since $\pi(\mathbf{a} \,|\, \mathbf{s}) = e^{Q(\mathbf{a},\mathbf{s}) - V_\pi(\mathbf{s})}$ where $V_\pi(\mathbf{s}) = \int_{\mathcal{A}} Q(\mathbf{a}, \mathbf{s}_{t+1}) \mathrm{d}\, \mathbf{a}$, we have

$$\begin{aligned}
Q(\mathbf{a}_t, \mathbf{s}_t) &= r(\mathbf{a}_t, \mathbf{s}_t) \\
&+ \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim \rho_\pi} \left[V_\pi(\mathbf{s}_{t+1}) - \mathcal{H}(\pi(\cdot \,|\, \mathbf{s}_{t+1})) \right]
\end{aligned}$$

■

C. More Details on Related Works

For reference, in addition to Section 5, we provide more details on the connection of our framework compared to existing methods.

Connections with trust region methods Trust Region methods can stabilize policy optimization in RL (Nachum et al., 2017; Kakade, 2002). We can also show the connection between the proposed method and TRPO (Schulman et al., 2015). In TRPO, an objective function is maximized subjected to a constraint on the size of policy update. Specifically,

$$\max_{\boldsymbol{\phi}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi^\phi(\cdot \,|\, \mathbf{s})}{\pi^{\phi_{k-1}}(\cdot \,|\, \mathbf{s})} \hat{A}_k \right] \qquad (19)$$

$$\text{subject to} \quad \hat{\mathbb{E}}_t \left[\mathrm{KL}[\pi^\phi(\cdot \,|\, \mathbf{s}), \pi^{\phi_{k-1}}(\cdot \,|\, \mathbf{s})] \right] \leq \delta \qquad (20)$$

Here, π_ϕ is a stochastic policy; ϕ_{k-1} is the vector of policy parameters before the k-th update; $\hat{A}_k$ is an estimator of the advantage function at timestep k. The theory of TRPO suggests using a penalty instead of a constraint, *i.e.*, solving the unconstrained optimization problem,

$$\max_{\boldsymbol{\phi}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi^\phi(\cdot \,|\, \mathbf{s})}{\pi^{\phi_{t-1}}(\cdot \,|\, \mathbf{s})} \hat{A}_t - \beta \mathrm{KL}[\pi^\phi(\cdot \,|\, \mathbf{s}), \pi^{\phi_{t-1}}(\cdot \,|\, \mathbf{s})] \right] \qquad (21)$$

In our proposed framework, the Wasserstein distance between $\pi^\phi(\cdot \,|\, \mathbf{s})$ and $\pi^{\phi_{k-1}}(\cdot \,|\, \mathbf{s})$, a weaker metric than the KL divergence, constrains the update of a policy on a manifold endowed with the Wasserstein metric, and potentially leads to more robust solutions. This is evidenced by the development of Wasserstein GAN (Arjovsky et al.). As a result, our framework can be regarded as a trust region based counterpart for solving the soft Q-learning problem (Haarnoja et al., 2017).

Connections with noisy exploration In our framework, adding noise to the parameters, leading to noisy exploration can be interpreted as a special case of indirect policy learning with single particle. As shown in (Fortunato et al., 2018; Plappert et al., 2018), independent Gaussian noisy linear layer is defined as.

$$y \overset{\text{def}}{=} \quad (\mu^w + \sigma^w \odot \epsilon^w) x + \mu^b + \sigma^b \odot \epsilon^b, \qquad (22)$$

The parameters $\mu^w \in \mathbb{R}^{q \times p}$, $\mu^b \in \mathbb{R}^q$, $\sigma^w \in \mathbb{R}^{q \times p}$ and $\sigma^b \in \mathbb{R}^q$ are learnable whereas $\epsilon^w \in \mathbb{R}^{q \times p}$ and $\epsilon^b \in \mathbb{R}^q$ are random noises, where p and q are the number of hidden units of connected layers.

It corresponds to the maximum a posterior (MAP) with a Gaussian assumption on the posterior distributions of parameters (weight uncertainty). In our framework, we can explicitly (Liu & Wang, 2016) or implicitly (Blundell et al., 2015) define the weight uncertainty. Especially, when employing SGLD (Welling & Teh, 2011) to approximate the posterior distributions of parameters, we will using noisy gradient instead of noisy weights in the parameter space.

Previous work, such as DDPG (Lillicrap et al., 2016), adding noise to the action to encourage exploration can be regarded as a special case of DP-WGF. Adding noise in parameter space has shown superiority with action space, but the computational cost of employing particles to approximate parameter distribution is much higher than that of directly approximate policy distribution. Previous work (Fortunato et al., 2018; Plappert et al., 2018) made a trade-off and optimize the MAP instead of the distribution.

D. Extensive Experiments

To optimize over the discretized WGF via the JKO scheme (6), to need to specify the discretized stepsize h. In addition, we have an additional hyparameter λ in the gradient formula of W_2^2. Also note that we can only evaluate the gradient of W_2^2 up to a constant, there needs to a parameter balancing the gradient of the energy functional F and the Wasserstein term. We denote this hyparameter as ϵ. In the experiments, if not explicitly stated, the default setting for these parameters are $\epsilon = 0.4$, and $\lambda = \texttt{med}^2/\log M$. Here med is the median of the pairwise distance between particles of consecutive policies. Adam (Kingma & Ba, 2015) optimizer is used for all experiments, except the BNN regression, for which we use RMSPorp (Hinton et al., 2012).

D.1. Comparative Evaluation

DP-WGF-V learns substantially faster than popular baselines on four tasks. In the Humanoid task, even though TRPO-GAE does not outperforms DP-WGF-V within the range depicted in the Figure 2, it achieves good final rewards after more episodes. The quantitative results in our experiments are also comparable to results reported by other methods in prior work (Duan et al., 2016; Gu et al., 2017; Henderson et al., 2018; O'Donoghue et al., 2016; Mnih et al., 2016), showing sample efficiency and good performance.

D.2. BNN for regression

For SVGD-based methods, we use a RBF kernel $\kappa(\boldsymbol{\theta}, \boldsymbol{\theta}') = \exp(-\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|_2^2/m)$, with the bandwidth set to $m = \texttt{med}^2/\log M$. Here med is the median of the pairwise distance between particles. We use a single-layer BNN for regression tasks. Following (Blundell et al., 2015), 10 UCI public datasets are considered: 100 hidden units for 2 large datasets (Protein and YearPredict), and 50 hidden units for the other 8 small datasets. We repeat the experiments 20 times for all datasets except for Protein and YearPredict, which we repeat 5 times and once, respectively, for computation consideration (Blundell et al., 2015). The experiment settings are almost identical to those in (Blundell et al., 2015), except that the prior of covariances follow $\mathsf{Inv\text{-}Gamma}(1, 0.1)$. The batch size for the two large datasets is set to 1000, while it is 100 for the small datasets. The datasets are randomly split into 90% training and 10% testing. Table 3 shows the complete results for different models on all the datasets.

D.3. Toy example in a multi-goal environment

We use the similar toy example as in soft Q-learning to show that our proposed , where the environment is defined as a multi-modal distribution,

Figure 3 illustrates a 2D multi-goal environment. The left one shows trajectories from a policy learned with DP-WGF-V. The x and y axes correspond to 2D positions (states). The agent is initialized near the origin, and the first step of trajectory is omitted. Red dots are depicted goals and the environment is terminated once the distance between the agents and some goal meets predefined threshold. The level curves show the distance to the goal.

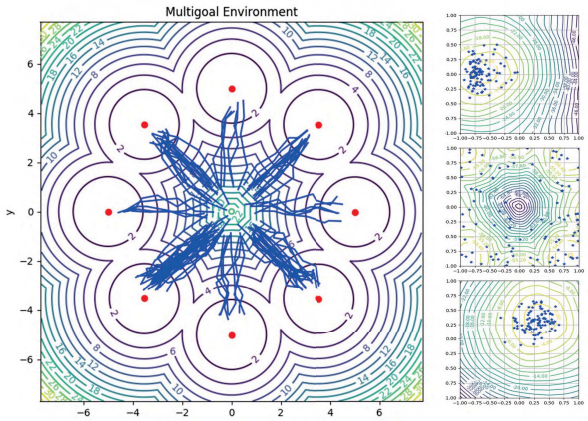

Figure 3. DP-WGF-V on multi-goal Environment.

Q-values at three selected states (-2.5, 0), (0, 0), (2.5, 2.5) are presented on the right, depicted by level curves (yellow: high values, red: low values). The x and y axes correspond to 2D velocity (actions) bounded between -1 and 1. Actions sampled from the policy are shown as blue stars. The experiments shows that our methods have the ability to learn multi-goal policies while achieving better stability and rewards than soft-Q learning.

D.4. Hyperparameter Sensitivity

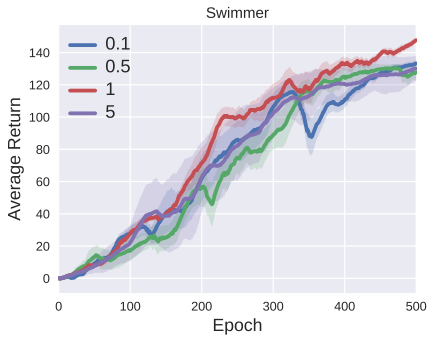

Figure 4. Sensitivity of Hyper-parameters

We further conduct experiments on Swimmer-v1 task to analysis the influence of different Wasserstein-2 scale ϵ. We run the algorithm for 500 epochs and 5 re-runs. Figure 4(a) shows the mean of average return against epoch. From the experiments, with appropriate ϵ, the learning curves become more stable and achieves higher final rewards; while

	Test RMSE ↓				Test Log likelihood ↑			
Dataset	Dropout	PBP	SVGD	WGF	Dropout	PBP	SVGD	WGF
Boston	4.32 ± 0.29	3.01 ± 0.18	2.96±0.10	$\mathbf{2.46 \pm 0.34}$	-2.90 ± 0.07	-2.57 ± 0.09	-2.50±0.03	$\mathbf{-2.40 \pm 0.10}$
Concrete	7.19 ± 0.12	5.67 ± 0.09	5.32±0.10	$\mathbf{4.59 \pm 0.29}$	-3.39 ± 0.02	-3.16 ± 0.02	-3.08±0.02	$\mathbf{-2.95 \pm 0.06}$
Energy	2.65 ± 0.08	1.80 ± 0.05	1.37±0.05	$\mathbf{0.48 \pm 0.04}$	-2.39 ± 0.03	-2.04 ± 0.02	-1.77±0.02	$\mathbf{-0.73 \pm 0.08}$
Kin8nm	0.10 ± 0.00	0.10 ± 0.00	$\mathbf{0.09 \pm 0.00}$	$\mathbf{0.09 \pm 0.00}$	0.90 ± 0.01	0.90 ± 0.01	$\mathbf{0.98 \pm 0.01}$	0.97 ± 0.02
Naval	0.01 ± 0.00	.01± 0.00	0.00±0.00	$\mathbf{0.00 \pm 0.00}$	3.73 ± 0.12	3.73 ± 0.01	4.09±0.01	$\mathbf{4.11 \pm 0.02}$
CCPP	4.33 ± 0.04	4.12± 0.03	4.03±0.03	$\mathbf{3.88 \pm 0.06}$	-2.89 ± 0.02	-2.80 ± 0.05	-2.82±0.01	$\mathbf{-2.78 \pm 0.01}$
Winequality	0.65 ± 0.01	0.64 ± 0.02	0.61±0.01	$\mathbf{0.57 \pm 0.03}$	-0.98 ± 0.01	-0.97 ± 0.01	-0.93±0.01	$\mathbf{-0.87 \pm 0.04}$
Yacht	6.89 ± 0.67	1.02 ± 0.05	0.86±0.05	$\mathbf{0.56 \pm 0.16}$	-3.43 ± 0.16	-1.63 ± 0.02	-1.23±0.04	$\mathbf{-0.99 \pm 0.15}$
Protein	4.84 ± 0.03	4.73 ± 0.01	4.61±0.01	$\mathbf{4.24 \pm 0.02}$	-2.99 ± 0.01	-2.97 ± 0.00	-2.95±0.00	$\mathbf{-2.88 \pm 0.01}$
YearPredict	$9.03 \pm$ NA	$8.88 \pm$ NA	8.68± NA	$\mathbf{8.66 \pm NA}$	$-3.62 \pm$ NA	-3.60±NA	$-3.58 \pm$ NA	$\mathbf{-3.57 \pm NA}$

Table 3. Averaged predictions with standard deviations in terms of RMSE and log-likelihood on test sets.

too large scale of ϵ will reduce the final rewards of policy, since the update size is excessively restricted. The results also show that the scale ϵ of Wasserstein trust-region is not parameter sensitive.

E. Implementation Details

E.1. Smoothing previous policy

Towards Wasserstein-2 distance, we need to use consecutive to compute policies $W_2^2(\pi^{\overline{\boldsymbol{\phi}}}(\cdot|\,\mathbf{s}_t), \pi^{\boldsymbol{\phi}}(\cdot|\,\mathbf{s}_t))$. For the previous policy $\pi^{\overline{\boldsymbol{\phi}}}(\cdot|\,\mathbf{s}_t)$, there are two strategy to get it. *i*) policy of last iteration, *i.e.* $\overline{\boldsymbol{\phi}} = \boldsymbol{\phi}_{k-1}$. *ii*) moving average of prior policy, *i.e.* $\overline{\boldsymbol{\phi}} = (1-\tau)\overline{\boldsymbol{\phi}} + \tau\boldsymbol{\phi}_{k-1}$. Empirically, when the learning curve is stable, (e.g. Half-Cheetah-v1), adopting strategy *i*) is helpful, and strategy *ii*) will reduce the speed of convergence and may lead lower final rewards; Otherwise, strategy *ii*) will help stabilize the training, and speed up the convergence.

Table 4. Shared parameters of direct policy learning

Parameter	Symbol	Value
horizon		500
batch size		5000
learning rate		5×10^{-3}
discount	γ	0.99
hidden units		[25, 16]
variance (prior)		0.01
temperature	α	$\{6, 7, 8, 9, 10, 11\}$

E.2. Indirect Policy learning

For the easy task, Cartpole, all agents are trained for 100 episodes. For the two complex tasks, Cartpole Swing-Up and Double Pendulum, all agents are trained up to 1000 episodes. SVPG and IP-WGF shared the same hyperparameters, except the temperature, for which we performed a grid search over $\alpha \in \{6, 7, 8, 9, 10, 11\}$.

E.3. Direct-Policy learning

We use OpenAI gym[‖] (Brockman et al., 2016) and rl-lab[**] (Duan et al., 2016) baselines implementations for TRPO and DDPG. SAC[††] and Soft-Q[‡‡] implementation are used, and we use recommended parameters.

Hyperparameters Table 5 lists the common DP-WGF-V, DP-WGF, SAC and Soft-Q parameters used in the comparative evaluation in Figure 2, and Table 2 lists the parameters that varied across the environments. For SAC, we use 4 components of mixture Gaussian. For DP-WGF and Soft-Q, 32 particles are used to approximate policy distributions.

Table 5. Shared parameters of indirect policy learning

Parameter	Symbol	Value
horizon		1000
batch size		64
learning rate		$3\cdot10^{-4}$
discount	γ	0.99
target smoothing coefficient	τ	0.01
number of layers (3 networks)		2
number of hidden units per layer		128
gradient steps		1
scale of Wasserstein trust-region		0.4

Table 6. Environment Specific Parameters

Environment	DoFs	Reward Scale	Replay Pool
Swimmer	2	100	10^6
Hopper-v1	3	1	10^6
Walker2d-v1	6	3	10^6
Humanoid	21	3	10^6

[‖] https://github.com/openai/baselines
[**] https://github.com/rll/rllab/tree/master/examples
[††] https://github.com/haarnoja/sac
[‡‡] https://github.com/haarnoja/softqlearning

F. Demos

Demos of our framework on a set of RL tasks can be accessed online via: https://sites.google.com/view/wgf4rl/ .

G. Algorithm Details

For completeness, we list the detailed algorithms for IP-WGF, DP-WGF and DP-WGF-V in Algorithms 1, 2 and 3, respectively.

Algorithm 1 DP-WGF

Require: $\mathcal{D} = \emptyset$; initialize $\boldsymbol{\theta}, \boldsymbol{\phi} \sim$ some (prior) distribution.
Target parameters: $\overline{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}, \overline{\boldsymbol{\phi}} \leftarrow \boldsymbol{\phi}$
for each epoch **do**
  for each t **do**
    % Collect expereince
    Sample an action $\mathbf{a}_t$ from policy $\pi^{\boldsymbol{\phi}}(\cdot|\,\mathbf{s}_t)$.
    Sample next state from the environment: $\mathbf{s}_{t+1} \sim p_{\mathbf{s}}(\mathbf{s}_{t+1}\,|\,\mathbf{s}_t, \mathbf{a}_t)$
    Save the new experience in the replay memory: $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1}\}$
    % Sample from the replay memory
    $\{(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)}, r_t^{(i)}, \mathbf{s}_{t+1}^{(i)})\}_{i=0}^N \sim \mathcal{D}$.
    % Update Q function
    Compute empirical values $\hat{V}^{\overline{\boldsymbol{\theta}}}(\mathbf{s}_{t+1}^{(i)})$
    Compute empirical gradient $\hat{\nabla}_{\boldsymbol{\theta}} J_Q(\boldsymbol{\theta})$
    Update $\boldsymbol{\theta}$ according to it using ADAM
    % Update policy
    Compute $W_2^2(\pi^{\overline{\boldsymbol{\phi}}}(\cdot|\,\mathbf{s}_t), \pi^{\boldsymbol{\phi}}(\cdot|\,\mathbf{s}_t))$,
    Compute empirical gradient $\hat{\nabla}_{\boldsymbol{\phi}} J_\pi(\boldsymbol{\phi})$
    Update prior policy parameters: $\overline{\boldsymbol{\phi}} \leftarrow \boldsymbol{\phi}$
    Update $\boldsymbol{\theta}$ according to it using ADAM
    % Update target
    Update target Q function parameters:
    $\overline{\boldsymbol{\theta}} \leftarrow \tau\boldsymbol{\theta} + (1-\tau)\overline{\boldsymbol{\theta}}$
  end for
end for

Algorithm 2 IP-WGF

Require: Initialize policy particles $\boldsymbol{\Theta} \sim$ some (prior) distribution as a Bayesian neural network.
for each iteration **do**
  Reset FIFO replay pool R
  for each timestep t in episodes **do**
    Sample $\mathbf{a}_t$ from $\pi_\phi(\cdot|\,\mathbf{s}_t)$
    Sample next state from the environment: $\mathbf{s}_{t+1} \sim p_{\mathbf{s}}(\mathbf{s}_{t+1}\,|\,\mathbf{s}_t, \mathbf{a}_t)$
    Save experience in to FIFO replay pool R:
    for each particles $\boldsymbol{\theta}^{(i)} \in \boldsymbol{\Theta}$ **do**
      Compute $W_2^2(\overline{\boldsymbol{\Theta}}^{(i)}, \boldsymbol{\Theta}^{(i)})$
      Compute empirical gradient $\nabla_{\boldsymbol{\theta}^{(i)}} J(\pi_{\boldsymbol{\theta}^{(i)}})$
      Save current particles $\overline{\boldsymbol{\theta}}^{(i)} \leftarrow \boldsymbol{\theta}^{(i)}$
      Update policy particle $\boldsymbol{\theta}^{(i)}$
    end for
  end for
end for

Algorithm 3 DP-WGF-V

Require: $\mathcal{D} = \emptyset$; initialize $\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi} \sim$ some (prior) distribution. Target parameters: $\overline{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}, \overline{\boldsymbol{\phi}} \leftarrow \boldsymbol{\phi}$
for each epoch **do**
  for each t **do**
    % Collect expereince
    Sample an action $\mathbf{a}_t$ from policy $\pi^{\boldsymbol{\phi}}(\cdot|\,\mathbf{s}_t)$.
    Sample next state from the environment: $\mathbf{s}_{t+1} \sim p_{\mathbf{s}}(\mathbf{s}_{t+1}\,|\,\mathbf{s}_t, \mathbf{a}_t)$
    Save the new experience in the replay memory: $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1}\}$
    % Sample from the replay memory
    $\{(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)}, r_t^{(i)}, \mathbf{s}_{t+1}^{(i)})\}_{i=0}^N \sim \mathcal{D}$.
    % Update Q function
    Compute empirical gradient $\hat{\nabla}_{\boldsymbol{\theta}} J_Q(\boldsymbol{\theta})$
    Update $\boldsymbol{\theta}$ according to it using ADAM
    % Update value function
    Compute empirical gradient $\hat{\nabla}_{\boldsymbol{\psi}} J_V(\boldsymbol{\psi})$
    Update $\boldsymbol{\psi}$ according to it using ADAM
    % Update policy
    Compute $W_2^2(\pi^{\overline{\boldsymbol{\phi}}}(\cdot|\,\mathbf{s}_t), \pi^{\boldsymbol{\phi}}(\cdot|\,\mathbf{s}_t))$,
    Compute empirical gradient $\hat{\nabla}_{\boldsymbol{\phi}} J_\pi^\phi$
    Update prior policy parameters: $\overline{\boldsymbol{\phi}} \leftarrow \boldsymbol{\phi}$
    Update $\boldsymbol{\phi}$ according to it using ADAM
    % Update target
    Update target value parameters:
    $\overline{\boldsymbol{\psi}} \leftarrow \tau\boldsymbol{\psi} + (1-\tau)\overline{\boldsymbol{\psi}}$
  end for
end for