LET OFFLINE RL FLOW: TRAINING CONSERVATIVE AGENTS IN THE LATENT SPACE OF NORMALIZING FLOWS

Anonymous authors
Paper under double-blind review

ABSTRACT

Offline reinforcement learning aims to train a policy on a pre-recorded and fixed dataset without any additional environment interactions. There are two major challenges in this setting: (1) extrapolation error caused by approximating the value of state-action pairs not well-covered by the training data and (2) distributional shift between behavior and inference policies. One way to tackle these problems is to induce conservatism - i.e., keeping the learned policies closer to the behavioral ones. To achieve this, we build upon recent works on learning policies in latent action spaces and use a special form of Normalizing Flows for constructing a generative model, which we use as a conservative action encoder. This Normalizing Flows action encoder is pre-trained in a supervised manner on the offline dataset, and then an additional policy model - controller in the latent space - is trained via reinforcement learning. This approach avoids querying actions outside of the training dataset and therefore does not require additional regularization for out-of-dataset actions. We evaluate our method on various locomotion and navigation tasks, demonstrating that our approach outperforms recently proposed algorithms with generative action models on a large portion of datasets.

1 INTRODUCTION

Offline Reinforcement Learning (ORL) addresses the problem of training new decision-making policy from a static and pre-recorded dataset collected by some other policies without any additional data collection Lange et al. (2012); Levine et al. (2020). One of the main challenges in this setting is the extrapolation error Fujimoto et al. (2019) – i.e. inability to properly estimate values of state-action pairs not well-supported by the training data, which in turn leads to overestimation bias. This problem is typically resolved with various forms of conservatism, for example, Implicit Q-Learning Kostrikov et al. (2021) completely avoids estimates of out-of-sample actions, Conservative Q-Learning Kumar et al. (2020) penalizes q-values for out-of-distribution actions and others Fujimoto & Gu (2021); Kumar et al. (2019) put explicit constraints to stay closer to the behavioral policies.

An alternative approach to constraint-trained policies was introduced in PLAS Zhou et al. (2020), where authors proposed to construct a latent space that maps to the actions well-supported by the training data. To achieve this, Zhou et al. (2020) use Variational Autoencoder (VAE) Kingma et al. (2019) to learn a latent action space and then train a controller within it. However, as was demonstrated in Chen et al. (2022), their specific use of VAE leads to a necessity for clipping the latent space. Otherwise, the training process becomes unstable and the optimized controller is able to exploit the newly constructed action space, arriving at the regions resulting in out-of-distribution actions in the original space. While the described clipping procedure was found to be effective, this solution is rather ad-hoc and discards some of the in-dataset actions which could potentially limit the performance of the trained policies.

In this work, inspired by the recent success of Normalizing Flows (NFs) Singh et al. (2020) in the online reinforcement learning setup, we propose a new method called Conservative Normalizing Flows (CNF) for constructing a latent action space useful for offline RL. First, we describe why a naive approach for constructing latent action spaces with NFs is also prone to extrapolation error, and

then outline a straightforward architectural modification that allows avoiding this without a need for post-hoc clipping. We benchmark our method against other competitors based on generative models and show that it performs favorably on the large portion of D4RL Fu et al. (2020) locomotion and maze2d datasets.

2 METHOD

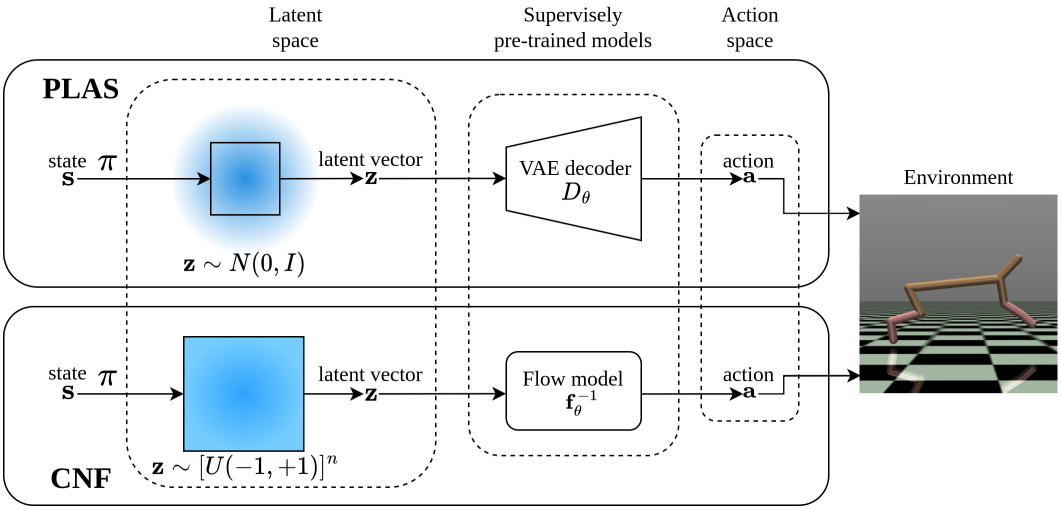


Figure 1: Schematic visualization and comparison of PLAS and CNF (ours) approach. Both methods use an action encoder-decoder model trained in a supervised manner on an offline dataset and a controller model to select actions from the latent space of the encoder. PLAS algorithm uses VAE with normal latent distribution with unbounded support (represented as the blue circle) and restricts latent policy outputs to only a part of the latent space (represented as the black borders inside of latent space). Our algorithm uses Normalizing Flow instead of VAE and bound base distribution itself, allowing the latent policy to use the whole latent space.

In this section, we describe our approach named Conservative Normalizing Flows (CNF) in detail. We start by delineating the proposed architecture and then describe both the pre-training and policy optimization phases.

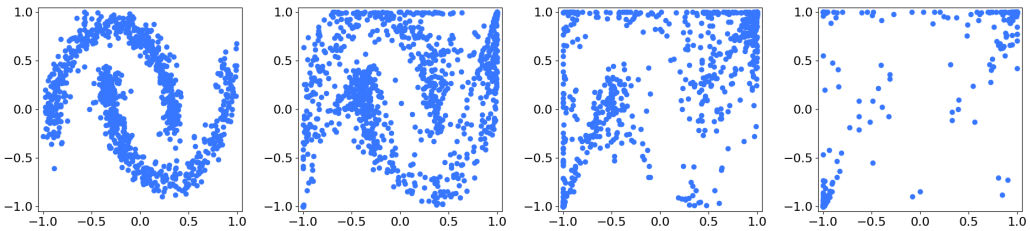
2.1 CONSERVATIVE NORMALIZING FLOWS

Normalizing Flows models are trainable bijective mappings, composed of fully invertible layers, that transform the original data space to the base distribution space. The latter is often referred to as the latent space. Typically, the base distribution is modeled as normal since it provides a tractable density and is easy to sample from Kingma & Dhariwal (2018). However, similar to the PLAS Zhou et al. (2020), this distribution would result in unbounded support. Therefore, when learning a policy in the latent space to maximize q-values it may exploit the regions that lead to the out-of-distribution actions in the original space exaggerating the extrapolation error.

To visualize this problem, we consider a toy example plotted in Figure 2, where we gradually increase the amplitude a of the latent samples drawn from the $a \cdot N(0, I)$ distribution for a NFs model based on the normal distribution (NF-Normal). One can see that, as expected, the higher amplitudes result in out-of-distribution data points.

One solution to avoid exploiting regions leading to out-of-distribution actions is to restrict policy output amplitude by some value. This approach was proposed in Zhou et al. (2020); Chen et al. (2022), where deterministic policy output was modeled as $z_{max} \cdot \tanh \pi_\theta(\mathbf{s})$ and z_{max} was set to 2. However, ideally, we would like to avoid this post-hoc clipping and construct a latent action space model that would prohibit the exploitation of out-of-distribution actions by design. To do so, we make use of NFs versatility, and add invertible $\tanh$ activation after the last layer of the Normalizing

Flows model - it makes NFs outputs lie in a bounded $n-$dimensional[1] interval $(-1, +1)^n$, which in turn allows us to substitute normal base distribution with $n-$dimensional uniform $[\mathbf{U}(-1, +1)]^n$. With these changes, the potential actor model should not be capable of generating actions outside of the training distribution.



(a) Amplitude = 2 (b) Amplitude = 4 (c) Amplitude = 10 (d) Amplitude = 30

Figure 2: A toy example demonstrates that normal-based NF can be manipulated by a controller to produce out-of-distribution training points. We model this by increasing the amplitude of the latent space samples, as we found this to happen in the preliminary experiments when the controller tries to maximize q-values. Note that the model with uniform latent space does not suffer from this problem because it already uses the whole latent distribution support during the training and sampling processes.

2.2 Latent Action Space Pre-Training

We start by pre-training conditional Normalizing Flows model $\mathbf{f}_\theta(\mathbf{a}|\mathbf{s})$ on the actions and states from the offline dataset, this is essentially a supervised learning problem. We use the same normalizing flow conditioning scheme as in the PARROT paper Singh et al. (2020): for each NFs layer, we add additional input for the state vector. Given a conditional NFs $\mathbf{f}$ with parameters θ_1, we use it to compute log-likelihood $\log p_{\theta_1}(\mathbf{a}|\mathbf{s})$ of actions $\mathbf{a}$ conditioned on states $\mathbf{s}$ from the offline dataset $\mathcal{D}$, minimizing the following loss:

$$L_{\mathbf{f}}(\theta_1) = \mathbb{E}_{(\mathbf{s},\mathbf{a})\sim\mathcal{D}}[-\log p_{\theta_1}(\mathbf{a}|\mathbf{s})] \tag{1}$$

The final result of this optimization process is a mapping $\mathbf{f}_\theta : A \times S \to Z$, where Z is the latent action space - a bounded n-dimensional region $(-1, 1)^n$. As this mapping is invertible, we can further transform latent vectors into the original action space for policy optimization.

2.3 Policy Optimization

Given a pre-trained latent action space model, we freeze its parameters and add it as an action encoder to the Critic Regularized Regression Wang et al. (2020) for policy optimization. This leads to the following loss function:

$$L_\pi(\theta_2) = \mathbb{E}_{(\mathbf{s},\mathbf{a})\sim\mathcal{D}}[\omega(\mathbf{s},\mathbf{a}) \cdot |\mathbf{a} - \mathbf{f}^{-1}(z \sim \pi(\cdot|\mathbf{s})|\mathbf{s})|], \tag{2}$$

where weights ω comes from exponentiation of the Advantage function:

$$\omega(\mathbf{a},\mathbf{s}) = \exp\left(\left(Q(\mathbf{s},\mathbf{a}) - Q(\mathbf{s}, \mathbf{f}^{-1}(z \sim \pi(\cdot|\mathbf{s})|\mathbf{s}))\right)/\lambda\right) \tag{3}$$

Together with controller and flow we optimize two critics Fujimoto et al. (2018) with standard Q-learning loss:

$$L_Q(\theta_3) = \mathbb{E}_{(\mathbf{s},\mathbf{a},r,\mathbf{s}')\sim\mathcal{D}}[(Q_1(\mathbf{s},\mathbf{a}) - y)^2 + (Q_2(\mathbf{s},\mathbf{a}) - y)^2] \tag{4}$$

where y is the Bellman target:

$$y = r + \gamma\mathbb{E}_{\mathbf{a}'\sim\mathbf{f}^{-1}(\pi(\cdot|\mathbf{s}'))}[\min_{i=1,2} Q_i(\mathbf{s}', \mathbf{a}')] \tag{5}$$

[1] n is the action space dimensionality.

Since NFs are differentiable, we can compute the gradient of the given loss functions for the policy model weights using the chain rule and reparametrization trick Kingma et al. (2019). Note, that one could bypass the differentiation through the NFs model, however, we will show in the ablations (Appendix A.4), that this results in obtaining worse policies.

3 EXPERIMENTS

To show how the proposed method works, we benchmark it on various locomotion and navigation tasks from the popular D4RL benchmark Fu et al. (2020), comparing it to the other methods based on generative models - PLAS Zhou et al. (2020) and LAPO Chen et al. (2022). We also include the variation of the CRR Wang et al. (2020) algorithm in our comparisons – AWAC Nair et al. (2020), as we use this method as the policy optimization backbone.

3.1 D4RL BENCHMARK

Locomotion We focus on 3 locomotion environments from the D4RL dataset in this section, including Walker2d-v2, Hopper-v2, and Halfcheetah-v2. We select 3 datasets for each environment: medium, medium-replay, and medium-expert. The first one consists of trajectories generated by a "medium" performance policy trained with the Soft Actor-Critic algorithm (SAC) Haarnoja et al. (2018b) up to a certain performance, the second one is the saved replay buffer from the SAC agent during training "medium" policies, and the third one is a combination of rollouts generated by "expert" and "medium" policies.

For the CNF model pre-training phase we divide the training dataset into 2 parts by randomly leaving 10% of the data for validation. We set the model hidden size to 256 and the number of layers to 12 for medium and medium-expert datasets and change them to 64 and 4 correspondingly for medium-replay datasets, as we observed that the larger models were prone to overfitting. We run a random search over the hyperparameter grid (see Table 3 in the Appendix B.1) limited by 50 trials, and then select the best model according to log-likelihood on the validation dataset for RL training. We train CNF models using Adam optimizer Kingma & Ba (2014) with standard beta values.

During the policy optimization phase, we did one million training steps for actor and critic models on all environments except the Halfcheetah-v2, where we did only 200.000 training steps as we found it sufficient for convergence. Once per 1000 (200 for Halfcheetah-v2 environment) training steps we evaluate the agent by running 10 episodes. We train policy and critic models with Adam optimizer with standard beta values and a learning rate $3e-4$. Hyperparameters for the policy optimization stage are listed in the Appendix B.1 (see Table 2).

For comparison, we re-implemented AWAC and IQL algorithms and use publically available PLAS implementation. We run AWAC and IQL algorithms with hyperparameters found in the papers Kostrikov et al. (2021); Nair et al. (2020); Zhou et al. (2020) and PLAS with hyperparameters recommended in Zhou et al. (2020). We could not reproduce the LAPO results, therefore we rely on the values reported in the original paper. Final scores of benchmarked algorithms are summarized in Table 1. Missing environments are dubbed as '-' in the table.

To highlight the performance of our method we include training curves on the Halfcheetah-v2 environment in Figure 3. Training curves for rest locomotion environments can be found in Appendix A.3 in Figure 4. It can be seen that the proposed algorithm exhibits stable training and preferable performance on all 3 Halfcheetah-v2 datasets.

Maze2d Next, we benchmark CNF on three maze2d datasets with increasing layout complexity: umaze-v1, medium-v1, and large-v1. The task in this environment is to navigate a 2D agent to reach the fixed goal location and it measures the ability of the ORL algorithm to stitch together sub-optimal trajectories to find a path to the desired goal. Datasets consisted of trajectories collected by some PD controller which navigated the labyrinth with random goals. After trajectories were collected, the reward was relabeled to correspond to the fixed evaluation goal position.

We use the same flow pre-training scheme as before, i.e. running a random hyperparameter search limited by 20 values, and then select a model with the highest log-likelihood on the validation dataset. Width and depth of the CNF architecture were fixed to four layers with hidden size of 64.

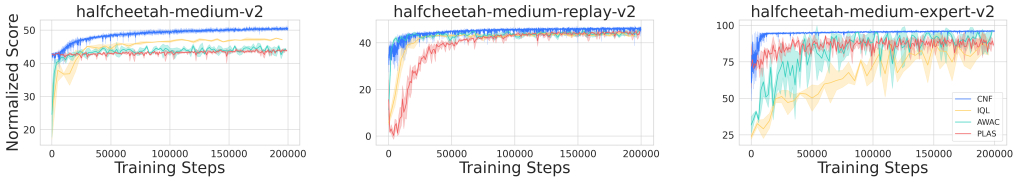


Figure 3: Average normalized performance on Halfcheetah-v2 environment. The x-axis denotes the training steps. Each curve is averaged over 3 random seeds. The shaded area represents one standard deviation.

For the policy optimization stage, we did not run a hyperparameter search and used three random seeds per dataset. We train policy and critics models for one million steps using Adam optimizer with default beta parameters and a learning rate of $3e-4$ (all the hyperparameters can be found in the Appendix B.1 Table 2).

For comparison we run our implementation of IQL in this benchmark, use publically available PLAS implementation and report scores from the LAPO paper. As can be seen in Table 1, our method outperform baselines in two of the most difficult environments and demonstrates a competitive performance in the other one.

Table 1: **Normalized performance on D4RL benchmark.** The scores are averaged over 10 final evaluations and 3 random seeds. The results are reproduced for all of the algorithms except LAPO, for which we take the values stated in the original paper. CNF outperforms all the other methods on a large portion of the considered datasets.

Environment	AWAC	IQL	PLAS	LAPO	CNF
walker2d-medium-v2	78.20	78.80	71.2	80.75	**83.60** $\pm$ 3.01
walker2d-medium-replay-v2	76.76	74.49	2.74	-	**81.96** $\pm$ 1.98
walker2d-medium-expert-v2	112.97	111.96	108.13	-	71.03 $\pm$ 37.24
hopper-medium-v2	62.59	63.7	52.93	51.63	**69.32** $\pm$ 1.04
hopper-medium-replay-v2	73.12	87.72	3.17	-	67.22 $\pm$ 8.50
hopper-medium-expert-v2	109.64	109.05	106.5	-	46.08 $\pm$ 5.45
halfcheetah-medium-v2	43.15	47.4	43.78	45.97	**50.55** $\pm$ 0.53
halfcheetah-medium-replay-v2	42.00	43.2	44.8	-	**45.84** $\pm$ 0.31
halfcheetah-medium-expert-v2	87.40	78.95	86.63	-	**96.23** $\pm$ 0.20
maze2d-umaze-v1	-	37.69	53.9	118.9	62.9 $\pm$ 10.36
maze2d-medium-v1	-	35.45	66.4	142.8	**155.89** $\pm$ 10.49
maze2d-large-v1	-	49.64	107.2	200.6	**212.8** $\pm$ 2.23

4 Conclusion

In this work, we presented a new deep offline RL method called Conservative Normalizing Flow (CNF). It constructs a latent action space with the use of the NFs model and then runs a policy optimization within it. This approach makes trained policies capable of fully utilizing the latent space as opposed to the post-hoc clipping procedures in PLAS Zhou et al. (2020) and LAPO Chen et al. (2022). We benchmarked our method against other competitors based on generative models and showed that it performs favorably on a large portion of D4RL Fu et al. (2020) locomotion and maze2d datasets.

References

Xi Chen, Ali Ghadirzadeh, Tianhe Yu, Yuan Gao, Jianhao Wang, Wenzhe Li, Bin Liang, Chelsea Finn, and Chongjie Zhang. Latent-variable advantage-weighted policy optimization for offline rl.

arXiv preprint arXiv:2203.08949, 2022.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018. URL `https://arxiv.org/abs/1802.09477`.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pp. 2052–2062. PMLR, 2019.

Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pp. 1851–1860. PMLR, 2018a.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31, 2018.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in Neural Information Processing Systems*, 32, 2019.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement learning*, pp. 45–73. Springer, 2012.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

Avi Singh, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine. Parrot: Data-driven behavioral priors for reinforcement learning. *arXiv preprint arXiv:2011.10024*, 2020.

Ziyu Wang, Alexander Novikov, Konrad Zolna, Josh S Merel, Jost Tobias Springenberg, Scott E Reed, Bobak Shahriari, Noah Siegel, Caglar Gulcehre, Nicolas Heess, et al. Critic regularized regression. *Advances in Neural Information Processing Systems*, 33:7768–7778, 2020.

Wenxuan Zhou, Sujay Bajracharya, and David Held. Plas: Latent action space for offline reinforcement learning. *arXiv preprint arXiv:2011.07213*, 2020.

A APPENDIX

A.1 BACKGROUND

Offline RL The goal of offline RL is to find a policy that maximizes the expected discounted return given a static and pre-recorded dataset $\mathcal{D}$ consisting of state-action-reward tuples. Normally, the underlying decision-making problem is formulated via Markov Decision Process (MDP) that is defined as a 4-elements tuple, consisting of state space $\mathbf{S}$, action space $\mathbf{A}$, transition probability $\mathbf{P} : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \to [0, 1]$, reward function $r : \mathbf{S} \times \mathbf{A} \to \mathbb{R}$ and a scalar discount factor $\gamma \in [0, 1]$. We denote the reward $r(\mathbf{s}_t, \mathbf{a}_t)$ as r_t. The discounted return is defined as $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$. Also, the notion of the advantage function $A(\mathbf{s}, \mathbf{a})$ is introduced – a difference between state-action value $Q(\mathbf{s}, \mathbf{a})$ and state value $V(\mathbf{s})$ functions:

$$
\begin{aligned}
A^\pi(\mathbf{s}, \mathbf{a}) &= Q^\pi(\mathbf{s}, \mathbf{a}) - V^\pi(\mathbf{s}) \\
Q^\pi(\mathbf{s}_t, \mathbf{a}_t) &= r_t + \mathbb{E}_\pi \sum_{k=0}^{\infty} \gamma^k r_{t+k} \\
V^\pi(\mathbf{s}) &= \mathbb{E}_{\mathbf{a} \sim \pi} Q^\pi(\mathbf{s}, \mathbf{a})
\end{aligned}
\tag{6}
$$

Critic Regularized Regression One way to learn a policy in an offline RL setting is by following the gradient of the expected discounted return estimated via importance sampling Levine et al. (2020), however methods employing estimation of the Q-function were found to be more empirically successful Kumar et al. (2020); Nair et al. (2020); Wang et al. (2020). Here, we describe one of them Wang et al. (2020) – Critic Regularized Regression, where the policy is trained by optimization of advantage re-weighted log-probabilities of the actions from the data buffer.

In practice, there are two trained models: policy π_θ with parameters θ and critic Q_ϕ with parameters ϕ. The training process consists of two alternating phases: policy evaluation and policy improvement. During the policy evaluation phase, the critic $Q^\pi(s, a)$ estimates the action-value function for the current policy, and during the policy improvement phase, the actor π is updated based on the current estimation of advantage. Combining all together, two following losses are minimized using the gradient descent:

$$
\begin{aligned}
L_\pi(\theta) &= \mathbb{E}_{(\mathbf{s},\mathbf{a}) \sim \mathcal{D}}[-\log \pi_\theta(\mathbf{a}|\mathbf{s}) \cdot \mathbf{f}(A_\phi(\mathbf{s}, \mathbf{a}))] \\
L_{TD}(\phi) &= \mathbb{E}_{(\mathbf{s},\mathbf{a},r,\mathbf{s}') \sim \mathcal{D}}[(r + \gamma Q_\phi(\mathbf{s}', \mathbf{a}' \sim \pi_\theta(\cdot|\mathbf{s}')) - Q_\phi(\mathbf{s}, \mathbf{a}))^2]
\end{aligned}
\tag{7}
$$

Where $A_\phi(\mathbf{s}, \mathbf{a})$ is computed according to Equation 6 and $\mathbf{f}$ is some function applied to advantage Wang et al. (2020).

Normalizing Flows As our method is built on top of Normalizing Flows, we include a brief description of NFs based on recent works Dinh et al. (2016); Kingma & Dhariwal (2018). Given a dataset $D = \{x^{(i)}\}_{i=1}^N$, from unknown distribution $\mathbf{X}$ the goal is to train an invertible mapping $\mathbf{z} = \mathbf{f}_\theta(\mathbf{x})$ to a simpler base distribution $\mathbf{Z}$ with tractable density, typically spherical Gaussian: $\mathbf{z} \sim N(0, I)$. This mapping is required to be invertible by design to sample new points from data distribution by applying the inverse mapping to samples from the base distribution: $\mathbf{x} = \mathbf{f}^{-1}(\mathbf{z})$. A full flow model is a composition of K invertible functions $\mathbf{f}_i$ and the relationship between $\mathbf{z}$ and $\mathbf{x}$ can be written as:

$$
\mathbf{x} \overset{\mathbf{f}_1}{\longleftrightarrow} \mathbf{h}_1 \overset{\mathbf{f}_2}{\longleftrightarrow} \mathbf{h}_2 \cdots \overset{\mathbf{f}_K}{\longleftrightarrow} \mathbf{z}
\tag{8}
$$

Log-likelihood of a data point x is obtained by using the *change of variable* formula and can be written as:

$$
\begin{aligned}
\log p_\theta(\mathbf{x}) &= \log p_{\mathbf{Z}}(\mathbf{z}) + \log |\det(d\mathbf{z}/d\mathbf{x})| \\
&= \log p_{\mathbf{Z}}(\mathbf{z}) + \sum_{i=1}^{K} \log |\det(d\mathbf{h}_i/d\mathbf{h}_{i-1})|
\end{aligned}
\tag{9}
$$

Normalizing Flows models are optimized to directly maximize log-likelihood of data points using Equation 9.

A.2 ALGORITHMS

Algorithm 1 CNF, Pre-training

1: **Input:** offline data replay buffer $\mathcal{D}$
2: Initialize flow action encoder parameters θ_1
3: **repeat** until convergence
4: Sample a mini-batch $B = \{(\mathbf{s}, \mathbf{a})\}$ from $\mathcal{D}$
5: Compute flow loss using Equation 1, compute loss gradient and update flow model:

$$\nabla_{\theta_1} \frac{1}{|B|} \sum_{(\mathbf{s},\mathbf{a})\in B} [-\log p_{\theta_1}(\mathbf{a}|\mathbf{s})]$$

Algorithm 2 CNF, Policy Optimization

1: **Input:** Pre-traine flow action encoder model $\mathbf{f}(\mathbf{a}|\mathbf{s})$ with parameters θ_1, offline data replay buffer $\mathcal{D}$
2: Initialize actor π with parameters θ_2 and two critics Q_1 and Q_2 with parameters θ_3
3: **repeat** for a given number of train-ops
4: Sample a mini-batch $B = \{(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')\}$ from $\mathcal{D}$
5: Sample next-state actions $\mathbf{a}'$ using policy $\pi(\cdot|\mathbf{s}')$ and flow $\mathbf{f}(\cdot|s')$ models, compute Q-target:

$$\mathbf{a}' = \mathbf{f}^{-1}(z \sim \pi(\cdot|\mathbf{s}')) \text{ for all } \mathbf{s}' \in B$$

$$y = r + \gamma(\min_{i=1,2} Q_i(\mathbf{s}', \mathbf{a}'))$$

6: Compute critics loss using Equation 4, compute loss gradient and update models:

$$\nabla_{\theta_3} \frac{1}{|B|} \sum_{(\mathbf{s},\mathbf{a},r,\mathbf{a}')\in B} [(Q_1(\mathbf{s}, \mathbf{a}) - y)^2 + (Q_2(\mathbf{s}, \mathbf{a}) - y)^2]$$

7: Sample actions $\mathbf{a}$ using policy $\pi(\cdot|\mathbf{s})$ and flow $\mathbf{f}(\cdot|s)$ models, compute advantage weights 3:

$$\hat{\mathbf{a}} = \mathbf{f}^{-1}(z \sim \pi(\cdot|\mathbf{s})) \text{ for all } \mathbf{s} \in B$$

$$A(\mathbf{s}, \mathbf{a}) = Q(\mathbf{s}, \mathbf{a}) - Q(\mathbf{s}, \hat{\mathbf{a}})$$

$$\omega = \exp(A(A(\mathbf{s}, \mathbf{a}))/\lambda)$$

8: Compute policy loss using Equation 2, compute loss gradient and update policy model:

$$\nabla_{\theta_2} \frac{1}{|B|} \sum_{(\mathbf{s},\mathbf{a})\in B} \exp(A(\mathbf{s}, \mathbf{a})/\lambda) \cdot |\mathbf{a} - \hat{\mathbf{a}}|$$

A.3 D4RL LOCOMOTION TRAINING CURVES

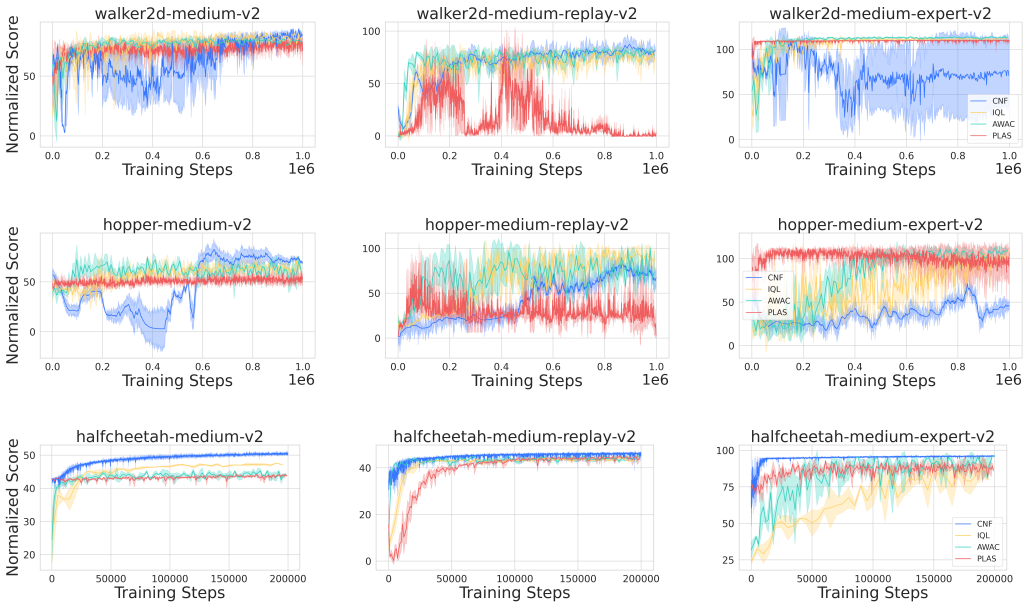


Figure 4: Average normalized performance on all locomotion environments from D4RL benchmark. The x-axis denotes the training steps. Each curve is averaged over 3 random seeds. The shaded area represents one standard deviation.

A.4 ABLATIONS

To study the importance of components of the proposed CNF method we conduct two additional experiments and describe them in this section.

Training controller in latent space In the first one, we train policy purely in the latent space of the Normalizing Flows model and didn't utilize gradients from it. To do so, we simply convert actions from the original environment action space to the latent space by encoding them with our pre-trained action encoder. After that, we run the AWAC algorithm with no additional changes on the dataset with converted actions. We conduct this experiment on 3 Halfcheetah-v2 environment datasets: medium, medium-replay, and medium-expert. Results are presented in Figure 5. It can be seen that this approach shows promising results, but convergence speed and a final score are slightly lower on all datasets. We conjecture that the performance is lower because a metric in the latent space is induced from the original action space by the flow model, and optimization of the distance between latent vectors corresponds to optimization of the distance between actions only implicitly. When training policy using metrics from the original action space, as suggested by Equation 2, the Normalizing Flows model provides gradients to the policy model that guide it to minimize the actual distance between actions.

Normalizing Flows with normal latent distribution In the second experiment, we use a conventional Normalizing Flows encoder without $\tanh$ activation after the last layer and with the normal latent distribution. We pre-train action encoders the same as before using only the best hyperparameters from previous experiments. For the latent policy model, we also remove $\tanh$ activation after the last layer. We again select the Halfcheetah-v2 environment and run 3 experiments with different random seeds per dataset. We plot results in Figure 6. It can be seen that in this experiment performance degraded substantially, indicating that both making latent distribution uniform and restricting policy outputs to correct bounds are crucial components of our algorithm.

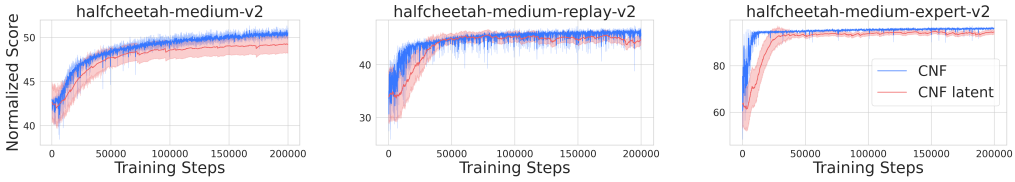


Figure 5: Comparison of CNF trained in original (blue) and latent (red) action spaces. Final performance is slightly lower for the training directly in latent space.

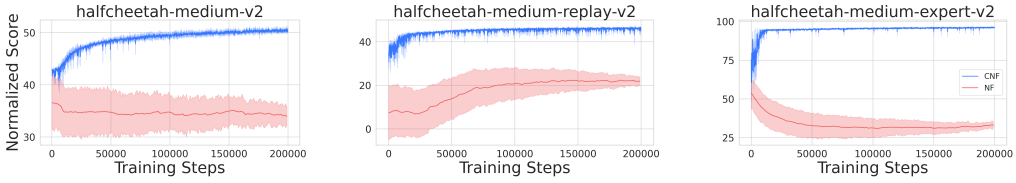


Figure 6: Comparison of CNF with uniform (blue) and normal (red) latent spaces. Policy performance is significantly worse when latent space is normal.

B Related Work

Normalizing Flows in RL Several prior works applied Normalizing Flows models for reinforcement learning tasks. In the work Haarnoja et al. (2018a) Normalizing Flows were used for training hierarchical policies using the Soft Actor-Critic RL algorithm. They choose Normalizing Flows because it provides an expression for exact likelihood computation and has an intuitive way to stack a sequence of models to construct one hierarchical model. In contrast to our work, they did not modify NF's latent space, they did not use offline pre-training of flow models and train models in the online RL framework.

The PARROT paper Singh et al. (2020) proposes to use the Normalizing Flows action encoder during the behavioral cloning pre-training phase before running reinforcement learning on the target task. After pre-training, they freeze Normalizing Flow and run reinforcement learning on a new and unseen task. They aimed to develop a method leveraging near-optimal demonstrations during Normalizing Flows pre-training and speed up the convergence of RL by better and meaningful exploration, while we use Normalizing Flows in purely offline RL setting and to make agent conservative.

Offline RL A large portion of recently proposed deep offline RL algorithms focuses on addressing the extrapolation issue, trying to impose a certain degree of conservatism limiting the deviation of the final policy from the behavioral one. Researchers approached this problem from multiple angles. For example, Kumar et al. (2020) proposed to directly penalize out-of-distribution actions, while Kostrikov et al. (2021) avoids estimating values for out-of-sample actions completely. Others Fujimoto & Gu (2021); Kumar et al. (2019) put explicit constraints to stay closer to the behavioral policy. Here, we took a different approach by constructing a latent action space that allows us to bypass the need for explicit regularizations.

Offline RL with generative models In the method named PLAS Zhou et al. (2020) proposed to pre-train conditional variational autoencoder on actions from offline dataset. This idea resembles ours, but to make agent conservative authors restrict policy outputs in the latent space to a fixed range. In our work, we firstly aimed to make a better action encoder model by switching from VAEs to Normalizing Flows. And secondly, we develop a new approach for constructing latent spaces and make agents conservative by design, rather than squeezing policy outputs. This allows us to utilize the whole latent space of the action encoder and we experimentally demonstrate that our approach leads to better performance on the popular offline RL benchmark.

One recent approach called LAPO Chen et al. (2022) proposes to train an action encoder together with a reinforcement learning agent. They motivated this by observing that the action distribution does not match the return distribution in the training data set and therefore actions that lead to higher returns are more important for action encoder training. However, we show experimentally that Normalizing Flows models with high enough capacity can capture whole action distribution and can be controlled by an additional MLP model to select the best action within the training dataset.

B.1 HYPERPARAMETERS

Table 2: Hyperparameters for latent policy and critics models for RL training phase on locomotion and maze2 tasks. All hyperparameters are fixed for all environments except the batch size.

Hyperparameter	value
number of layers	4
hidden size	256
learning rate	3e-4
batch size (locomotion)	256
batch size (maze2d)	10240

Table 3: Hyperparameters for NF models for supervised pre-training phase on locomotion and maze2 tasks. Medium, medium-replay, and medium-expert datasets are marked as m, m-r, and m-e correspondingly. We sample learning rate and weight decay from the continuous uniform distribution. We sample batch size from three available values, presented in the table.

Hyperparameter	value
number of layers (locomotion m and m-e datasets)	12
number of layers (locomotion m-r and maze2d datasets)	4
hidden size (locomotion m and m-e datasets)	256
hidden size (locomotion m-r and antmaze datasets)	256
learning rate	$\min$ = 1e-5, $\max$ = 3e-3
weight decay	$\min$ = 0.0, $\max$ = 1e-2
batch size	[512, 1024, 2048]