Exploring Model-free Lyapunov Density Models

Will Lavanakul, Vishal Raman

December 15, 2022

Abstract

Offline reinforcement learning (RL) is an important step towards integrating RL methods into real world applications. One major problem with current offline methods is the problem of "distribution shift", where a policy trained on the offline dataset fails to generalize well on states that are out of the training distribution. This can cause the trained policy to not only have poor performance after training, but can lead to unsafe states that are dangerous for the agent to explore. In Kang et. al [KGC+22], they introduce the *Lyapunov Density Model* (LDM) which acts as a mechanism to constrain the agent to states with high density in the training data. LDMs allow us to make guarantees on which actions will keep the agent within high density states, over a long horizon. This mitigates the problem of distribution shift as the agent will only encounter states it has trained on. In the original paper, they focus on the offline model-based setting. Specifically, they apply an LDM constraint over actions on a short-horizon MPC controller. This guaruntees that any action taken will keep the controller in high density states.

In our work, we extend the idea of Lyapunov Density Models into the model-free setting. We mainly focus on an offline DQN implementation in both discrete and continuous environments. We introduce 3 new methods for integrating LDM regularization into offline DQN learning. The first method involves regularizing the Q-learning process with violations of the LDM constraint. The second method uses the LDM constraint violation as an additive term during test time to affect the actions taken. The third method also uses the LDM constraint during test time, but instead of an additive term, it zeros out Q-values for actions violating the LDM constraint.

We ran 3 sets of experiments. The first two experiments use a discrete gridworld environment to exactly solve for the LDM. We chose to use a tabular environment with DQN so that we can explicitly see the effects of our LDM regularization without the inaccuracy of a learned LDM.

In our first experiment, we closely follow the same setup in Schweighofer et. al [SRD+21], specifically for their offline-DQN gridworld experiments. We train an expert policy until near optimal behavior, then sample a dataset from the expert policy using ϵ-greedy exploration and train a DQN agent only on transitions sampled from the dataset. We found similar results in that low values of ϵ caused distribution shift to cause standard offline DQN to fail frequently. For high values of ϵ, we get better reward, but fail to do as well or better than the expert policy. When applying our LDM regularization, we found that in almost all cases, failure rates decreased. In some cases, we achieve 0 failure rate, but end up in a suboptimal policy. Our next set of experiments involve testing how the data threshold parameter k affects failure rates. This parameter controls what level of training density we want to stay in. A higher k allows our agent to enter lower density states, risking more out of distribution problems while a lower k constrains our agent more. Our main goal is to verify that using our LDM regularization methods, a lower k results in less failure rates while a higher k results in higher failure rates. We see verify this in our experimental results. We also find that a hard constrained LDM method can completely remove failure rates while also achieving better performance than standard offline-DQN.

Our final experiments explore learning an LDM from the dataset instead of solving for it exactly (which is necessary for most environments), then using it on the continous Pointmass Environment. First, we show the efficacy of the LDM learning method by performing it on a gridworld environment where it can be solved for analytically. Finally, we compare the performance of our LDM regularization methods with CQL, AWAC, and IQL, where we overall see mixed results.

1 Introduction

In offline deep reinforcement learning, the goal is to learn a policy from a fixed dataset sampled from a previous expert policy. It has been previously shown that the learned policy has the capability of outperforming the expert policy. This can be extremely useful for situations where interacting with the environment is too costly or hard to do. However, many offline learning methods have their own challenges. One of the biggest challenges is handling "distribution shift" when training or testing the learned policy. Since the policy is trained under the fixed distribution of the training data, encountering states outside of a high training distribution will lead to sub-optimal decisions. This can compound over time as the policy makes worse decisions the further it ventures out of the training distribution. This can be an extremely pressing problem in the safety of training real-life systems. For example, distribution shift can lead a robot to unsafe states that might cause damage to the physical system. This makes it hard to train, as we might break or damage the robot early on in the training. Being able to ensure systems always stay within a high training distribution will prevent such behavior, further improving the training process.

A previous approach uses a density model to greedily choose actions that directly lead to high training density states. However, since the actions are greedily chosen, it is not guaranteed to stay within distribution over a long horizon. A more recent approach [KGC+22] solves the problem of staying in-distribution over a long horizon by learning a *Lyapunov Density Model* (LDM). LDMs combine density models and dynamics-aware Lyapunov functions to guarantee a system stays in-distribution. Their work focuses only on model-based MDP approaches. They can directly filter out actions that disagree with the LDM constraint.

The goal of our project is to extend the idea of Lyapunov Density Models into an offline model-free setting. We specifically focus on an offline-DQN framework where we train an expert policy using standard online DQN, then train an offline policy using a dataset sampled from the expert policy using ϵ-greedy exploration. We present 3 different methods for applying LDMs in an offline DQN setting. Using a gridworld environment, we provide empirical results that relate dataset diversity to reward, failure rate, stall rate, and success rate. We also provide experiments that explore how different density thresholds affect reward, failure rate, stall rate, and success rate. Furthermore, we show experiments that compare how a learned-LDM constrained offline DQN agent compares to AWAC, IQL, and CQL.

2 Problem Statement

Consider a deterministic, continuous-state, discrete-timed dynamical system $s_{t+1} = f(s_t, a_t)$ with state space $\mathcal{S}$ and action space $\mathcal{A}$. We have an offline dataset of transitions $D = \{(s_t^i, a_t^i, s_{t+1}^i)\}_{i=1}^N$ generated from some distribution $P(s, a)$. If $P(s_t, a_t) \geq c$ for some density $c > 0$, then the state-action tuple (s_t, a_t) is known as *in-distribution*. Let $\mathcal{D}_c = \{(s_t, a_t) | P(s_t, a_t) \geq c\}$ be the set of all state-action tuples that are in-distribution. We wish to train a Q-learning policy on D such that the policy only uses actions in states such that $(s_t, a_t) \in D_c$.

3 Background on Lyapunov Density Models

In this section, we review basic definitions and properties related to Lyapunov density models (LDMs). We refer the interested reader to [MLS94] and [KGC+22] for further details regarding LDMs and control Lyapunov functions (CLFs) from which LDMs take inspiration from.

First, we reframe the problem in terms of the following property:

Definition 3.1 (State-Action Control-Invariance)**.** A set $\mathcal{G} \subset \mathcal{S} \times \mathcal{A}$ is *state-action control-invariant* if for all $(s_0, a_0) \in \mathcal{G}$, there exists a sequence of actions $(a_t)_{t=1}^{\infty}$ so that $(s_t, a_t) \in \mathcal{G}$ for all $t \geq 0$.

In particular, if we have a state-action control-invariant set $\mathcal{G}_c \subset \mathcal{D}_c$, then there exists a trajectory $(s_t, a_t)_{t=1}^{\infty}$ satisfying the constraint $P(s_t, a_t) \geq c$ for all $t \geq 0$.

In order to construct such state-action control-invariant sets $\mathcal{G}_c$, we derive a function satisfying the properties that 1) each sub-level set is state-action control-invariant and 2) the value of the function is associated with a density level of a probability distribution. This leads to the definition of the Lyapunov density model:

Definition 3.2 (Lyapunov Density Model)**.** A coordinate-wise continuous function $G(s_t, a_t) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a *Lyapunov density model* for the dynamical system $s_{t+1} = f(s_t, a_t)$ and a density function $P : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^+$ if the following conditions hold:

- for all $(s_t, a_t) \in \mathcal{S} \times \mathcal{A}$, there exists $a_{t+1} \in \mathcal{A}$ such that
$$G(s_t, a_t) \geq G(f(s_t, a_t), a_{t+1}),$$
- for all $(s_t, a_t) \in \mathcal{S} \times \mathcal{A}$,
$$G(s_t, a_t) \geq -\log P(s_t, a_t).$$

The key sub-level set property (which is inspired by control Lyapunov functions) is given by the following theorem:

Theorem 3.1. *Any sub-level set of an LDM G,*
$$\mathcal{G}_c = \{(s_t, a_t) : G(s_t, a_t) \leq -\log(c)\},$$
is state-action control-invariant.

Next, in order to produce state-action control-invariant sets that are as large as possible (for the sake of downstream task-solving), we define the notion of the maximal LDM:

Definition 3.3 (Maximal LDM)**.** An LDM $G(s_t, a_t)$ is *maximal* if each of its sublevel sets $\mathcal{G}_c = \{(s_t, a_t) : G(s_t, a_t) \leq -\log(c)\}$ is the largest state-action control-invariant set contained inside $\mathcal{D}_c$.

While this definition is slightly unwieldly, it can be shown that if G is continuous, then the maximal LDM can be represented as
$$G(s_0, a_0) = \min_{(a_t)_{t=1}^{\infty}} \max_{t \geq 0} -\log P(s_t, a_t).$$
This motivates the following algorithm: first, construct a density model $E(s_t, a_t) = -\log P(s_t, a_t)$. Next, define the LDM backup operator
$$\mathcal{T} G(s, a) = \max\{E(s, a), \min_{a' \in \mathcal{A}} \gamma G(f(s, a), a')\} \tag{3.1}$$
where $\gamma \in [0, 1]$. Analogously to with Q-learning, the maximal LDM is given exactly by first taking $G_k = \mathcal{T}^k G_0$ where G_0 is the initial density and $\mathcal{T}^k$ represents iterating the backup operator k times, and taking the limit $G_\infty = \lim_{k \to \infty} G_k$.

In practice, we don't have access to the true dynamics $f(s_t, a_t)$ but instead the set of transitions $D = \{(s_t^i, a_t^i, s_{t+1}^i)\}_{i=1}^N$. In order to develop a practical algorithm for real datasets, we first learn the density model $E(s, a)$. In [KGC+22], this is done by representing $E_\theta(s_t, a_t)$ as a neural network and training it as a flow model [DBM+19]. In our experiments (see section 5.3 for details), we estimate this by discretizing the environment and returning the density of the visitation distribution. Then, we train neural network models of the LDM $G_\phi(s_t, a_t)$ in condition with a policy $\pi_\psi(s_t)$ as
$$\psi = \arg\min_{\psi} \mathbb{E}_{s_t \sim p_D}[G_\phi(s_t, \pi_\psi(s_t))],$$
$$\phi = \arg\min_{\phi} \mathbb{E}_{s_t, a_t, s_{t+1} \sim p_D}[(G_\phi(s_t, a_t) - \max\{E(s_t, a_t), \gamma G_\phi(s_{t+1}, \pi_\psi(s_t))\})^2].$$
There are a handful of other deep RL techniques that are used to stabilize the training such as

- Double value and target networks $\phi_1, \phi_2, \bar{\phi}_1, \bar{\phi}_2$
- Entropy term with automatic tuning $\alpha \mathcal{H}(\pi_\psi(\cdot|s_t))$ [HZA+18]
- Conservative regularization term CQL($\mathcal{H}$) in the LDM update [KZT+20]

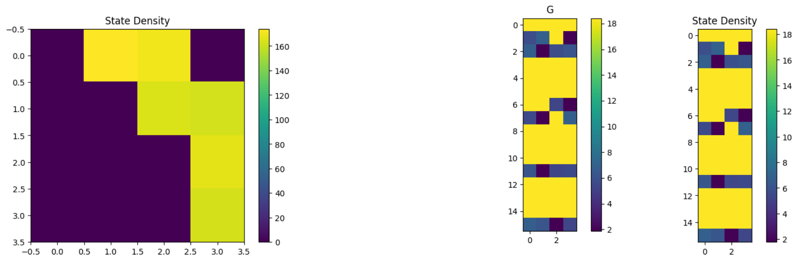

Figure 1: State density, dataset density (with states enumerated 1 to 16 and actions 1 to 4) and evaluated LDM with $\epsilon = 0.05$.

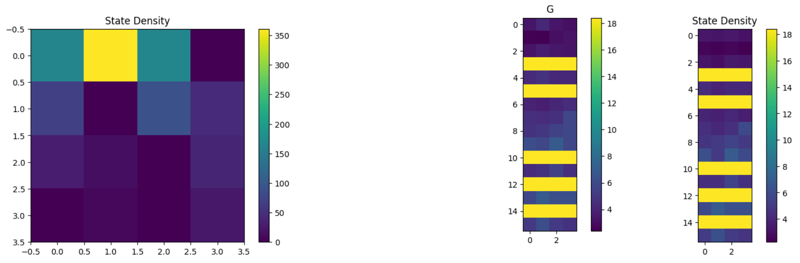

Figure 2: State density, dataset density (with states enumerated 1 to 16 and actions 1 to 4) and evaluated LDM with $\epsilon = 0.9$

.

4 Model-free LDMs

In our experiments, we apply LDM regularization in 3 ways to on offline DQN policy. Each method uses violation of the LDM constraint to augment the learned Q-function in hopes to keep the policy in-distribution. Both the LDM and the policy are trained on the same dataset, which we assume to be sampled from $P(s_t, a_t)$. LDM method 1 adds a regularization term in the target value used for Q-learning:

$$Q_{\text{target}}(s_t, a_t) = r(s_t, a_t) + \gamma_Q \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - \gamma_{LDM} \max(0, G(s_t, a_t) + \log(c)) \tag{4.1}$$

The third term in (4.1) penalizes thr Q-value only when (s_t, a_t) violates the LDM constraint. We decrease Q-values for state-action pairs that are out of distribution. This update equation encodes the LDM constraint into the learned Q-function and acts as a "soft" LDM constraint.

LDM method 2 uses the same term in (4.1), but only during action selection. When choosing an action, we augment the Q-function:

$$\pi(s_t) = \arg\max_{a_t} Q(s_t, a_t) - \gamma_{LDM} \max(0, G(s_t, a_t) + \log(c)) \tag{4.2}$$

During test time, the policy decreases Q values for state-action pairs that violate the LDM constraint.

LDM method 3 is most similar to model-based LDM constrained. In this method we "zero" out Q-values of actions that violate the LDM constraint:

$$\pi(s_t) = argmax_{a_t} T(s_t, a_t) \tag{4.3}$$

$$T(s_t, a_t) = \left\{ \begin{array}{ll} Q(s_t, a_t), & \text{if } G(s_t, a_t) \leq -\log(c) \\ \min_{a_t} Q(s_t, a_t) - 1, & \text{if } G(s_t, a_t) > -\log(c) \end{array} \right\} \tag{4.4}$$

This is an equivalent way of filtering out inputs that violate the LDM constraint. We only consider actions that will keep us in-distribution according to the LDM.

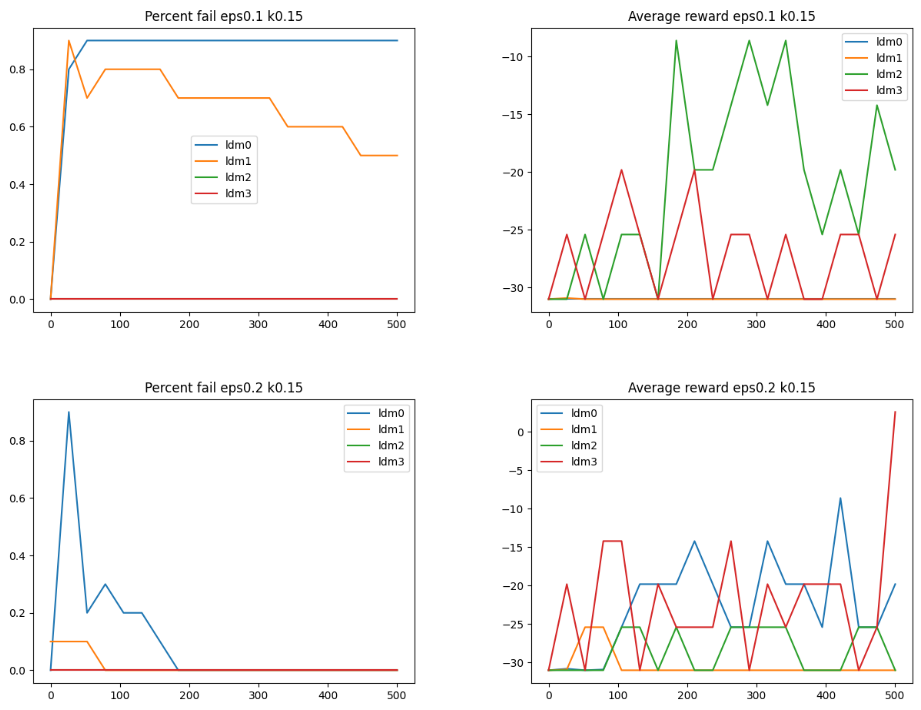

Figure 3: Percentage of failures and average reward for lower amounts of ϵ and fixed $k = 0.15$.

5 Experiments

We utilize two environments for our experiments. The first environment is a 4x4 gridworld environment with a player, holes, mountains, and a goal. The player must navigate to the goal state without falling into a hole within 30 moves. The player cannot travel through places with mountains. The reward signal is -1 for every step, -30 for falling into a hole, and 30 for reaching the goal state. For the gridworld environment, we solve the LDM exactly using the backup equation (3.1). We use an exact LDM instead of a learned LDM to clearly see the effects of our LDM regularization methods without the innacuracy of a learned LDM.

The second environment is the continuous Pointmass environment with a player, walls, and a goal. The goal is for the player to navigate to the goal state without traveling through walls. The reward signal is -1 for every step and 0 when the player is within ϵ of the goal. We use a learned LDM to compare to existing offline methods.

5.1 Exactly solved LDM on 4x4 Gridworld with varying ϵ

This experiment highlights the effect of distribution shift on offline DQN trained with datatsets generated from differenmt ϵ-greedy expert policies. This setup follows closely to the setup done in [SRD+21], where they find that low values of ϵ cause offline-DQN to perform poorly due to distribution shift and high values of ϵ give better, but sub-optimal, performance due to better diversity in the dataset. These experiments aim to see how our LDM regularization affects fail rate for low values of ϵ and reward for high values of ϵ. The experiment is conducted by first training an online expert policy on a fixed 4x4 gridworld environment for 200 rollouts. Then with varying values of ϵ, we create a dataset of transitions sampled by from the expert policy. With probability $1 - \epsilon$ we use the action the expert would have chosen. With probability ϵ, we choose a random action. We similarly find that with low values of ϵ, standard offline DQN suffers more from distribution shift. When ϵ is higher, the policy fails less but gets suboptimal rewards.

In Figures 1, 2, 3, and 4, we see the results of varying ϵ and keeping k fixed to a low value as well as the generated dataset for varying values of ϵ. With lower values of epsilon, the the expert policy used for

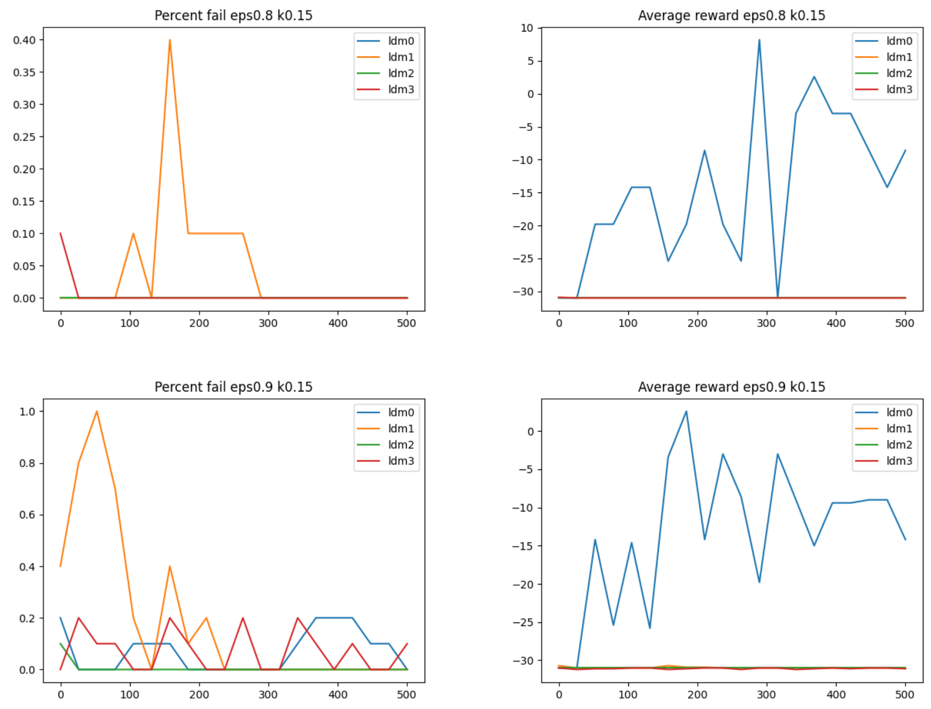

Figure 4: Percentage of failures and average reward for lower amounts of ϵ and fixed $k = 0.15$.

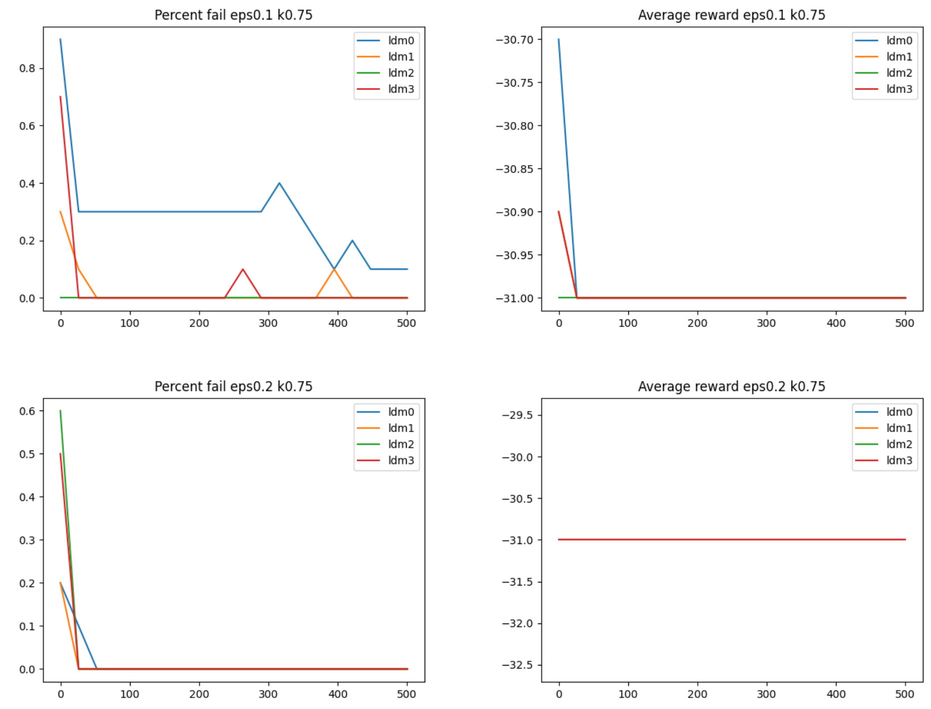

Figure 5: Above are plots percentage of failures and average reward for lower amounts of ϵ and fixed $k = 0.75$.

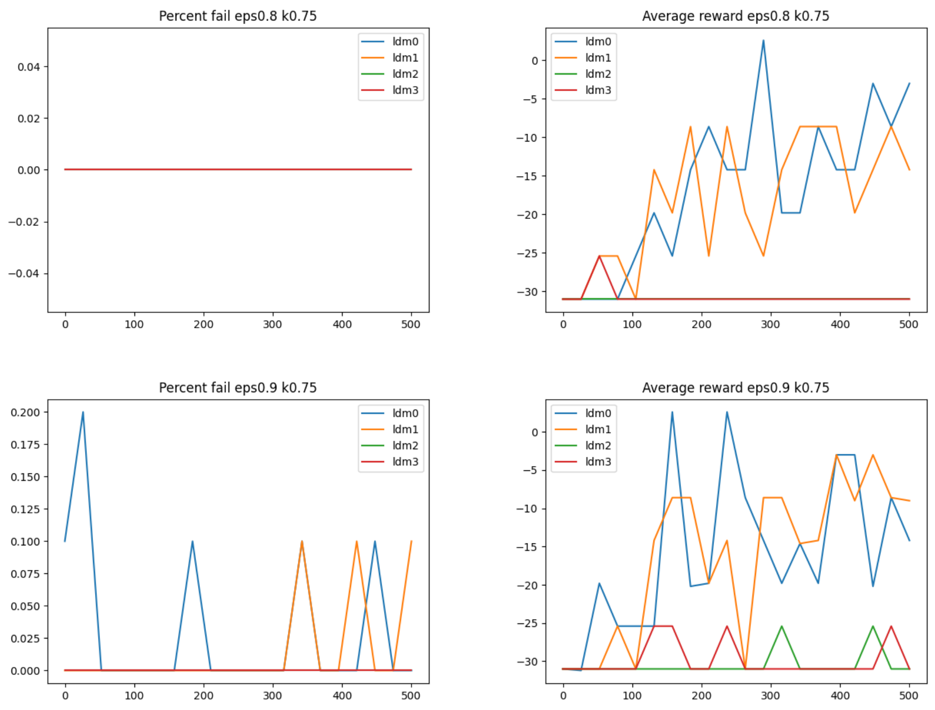

Figure 6: Above are plots percentage of failures and average reward for higher amounts of ϵ and fixed $k = 0.75$.

dataset generation has low exploration and the resulting dataset is made of a lot of the same transitions. As a result, standard offline-DQN (LDM0) suffers from distribution shift, leading to a large failure rate and low reward. All LDM regularized methods manage to mitigate failure rate. LDM1 being a more "soft" constraint learns to mitigate failure rate as the policy trains. LDM2 and LDM3 both regularize the policy during eval, which is why it is able to mitigate failure rate for all epochs. We also see that for a high value of epislon, offline-DQN is able to have good rewards. This is consistent with the findings in [SRD+21]. The LDM regularized methods are consistent in mitigating failure, but find sub-optimal policies as indicated with low reward.

In figures 5 and 6, we see the effect of considering more training data as in-distribution. We see that a higher k tends to cause LDM regularized methods to more frequently find sub-optimal policies. This is expected as we train to similar data as the standard offline DQN.

5.2 Exactly solved LDM on 4x4 Gridworld with varying k

This experiment highlights the effect of distribution shift on offline DQN trained with a fixed dataset over varying LDM thresholds k. We follow a similar experiment setup in [KGC+22] where they see the failure rates over different threshold values. The threshold value k controls what density of our training data we consider to be in distribution. Specifically, k denotes the kth percentile of the training data density to be in-distribution. In our experiment, lower values of k puts more constraint over what state-action pairs the LDM considers to be in-distribution. Our experiment setup consists of training an expert policy on a 4x4 gridworld then sampling a dataset of transitions using ϵ-greedy exploration for a fixed ϵ. Once this dataset is generated, we vary k and only re-evaluate the corresponding LDM. Then we train all 4 methods of LDM regularization on the dataset.

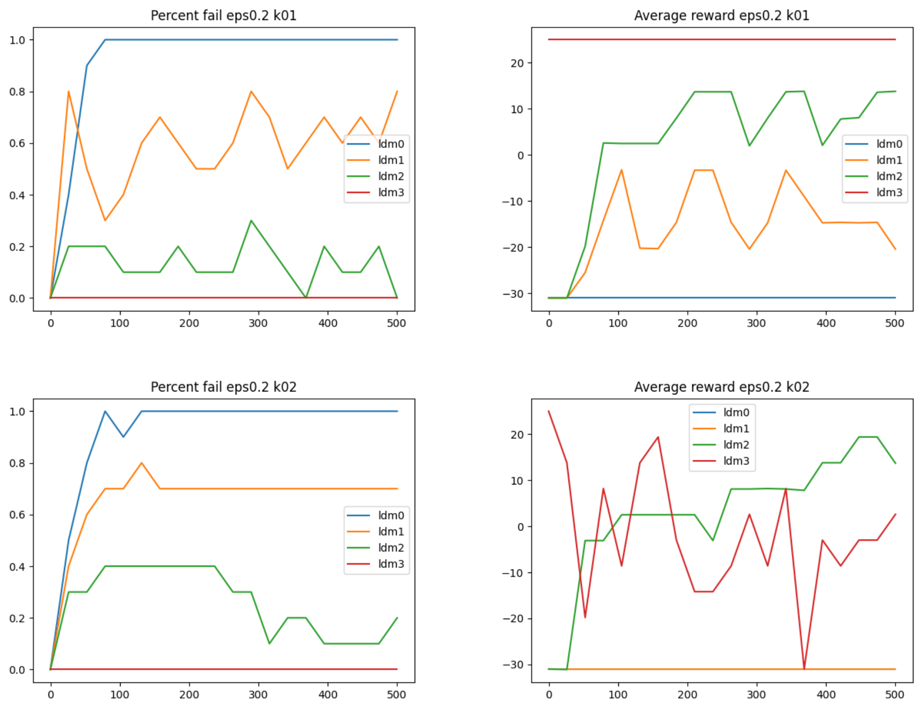

Figure 7: Above are plots percentage of failures and average reward for varying amount of fixed $\epsilon = 0.2$ and varying $k = [0.1, ..., 0.9]$.

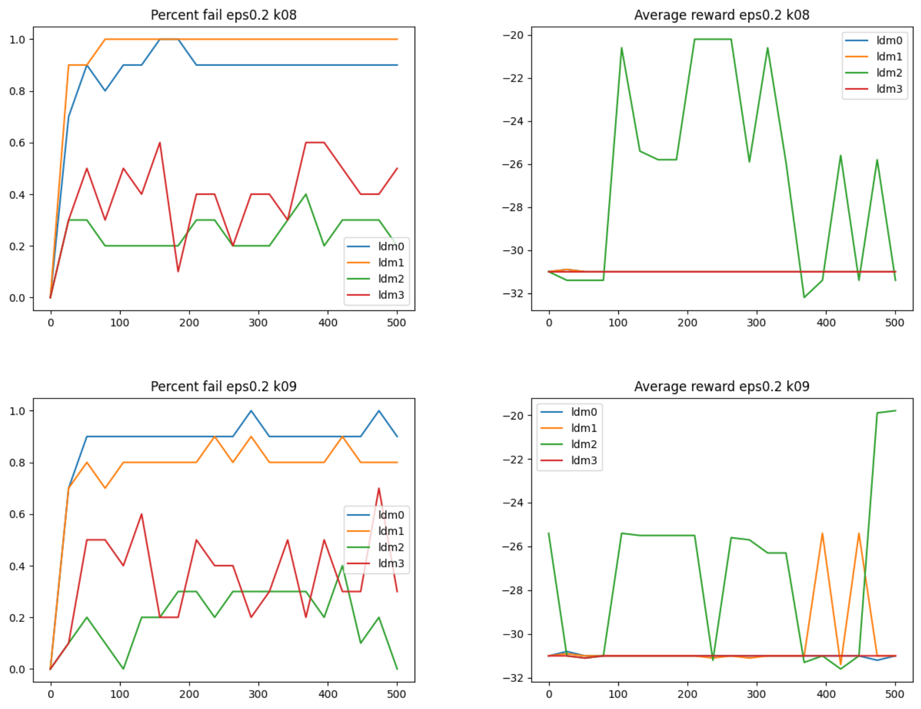

Figure 8: Above are plots percentage of failures and average reward for varying amount of fixed $\epsilon = 0.2$ and varying $k = [0.1, ..., 0.9]$.

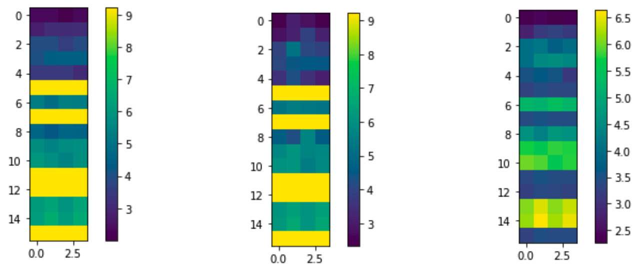

Figure 9: The left plot denotes the original buffer density, the middle denotes the solved LDM densities, and the right denotes the learned LDM densities.

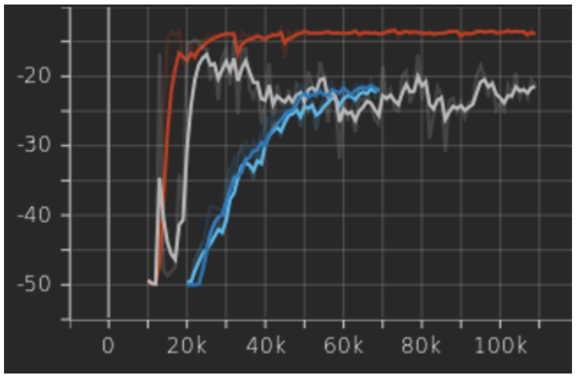

Figure 10: The red line denotes CQL, the white line denotes the LDM1 method, the dark blue line denotes AWAC with $\lambda = 10$, and the bright blue line denotes IQL with $\lambda = 10$ and an expectile value of 0.9.

The main goal of this experiment is to see how the failure rates of all four methods are affected with different threshold values. In figures 7 and 8, we see how the failure rate of the different LDM methods are affected by k. As expected for low threshold values, each LDM regularization method mitigates failure. As k increases, the failure rate for each method increases. Since LDM1 is a soft LDM constraint, it is expected that it mitigates failure rates worse than the hard constraints like LDM2 and LDM3.

5.3 Learned LDM vs AWAC, IQL, CQL on Pointmass

By learning the LDM on a more complex environment, we are able to do comparisons between the various model-free LDM methods and the current state-of-the-art (AWAC, IQL, CQL).

We first test the efficacy of the LDM learning procedure by solving for the LDM exactly in a Gridworld environment with a fixed density, and comparing it to a learned LDM, as shown in Figure 9. The overall L^2 error between the solved and learned LDMs are given by $||G_s - G_l||_2 \approx 25$. Next, we perform baseline tests on the PointmassEasy environment. In order to have a fair comparison, we first perform an exploration phase using random network distillation [BES+18] and start with the same buffer both for LDM training and for each of the different methods. In this experiment, we focus on the LDM1 method. A plot comparing the results is shown in Figure 10.

Overall, our method performs worse than CQL but seems to perform better than or similar to AWAC and IQL on the PointmassEasy environment. Our results are potentially limited by the performance of the LDM training method which we were not able to improve on the Gridworld or Pointmass environments. In the latter case, given a much finer level of discretization, we may be able to see improvements in the overall performance at the expense of computational complexity.

6 Conclusion

In this paper, we explored several integrations of LDMs into model-free offline settings. In our tests with the discrete Gridworld environment, we found that LDM1 as a soft LDM constraint slowly learns a Q-function that mitigates distribution shift problems. LDM2 and LDM3 completely mitigated failure rates during eval, but often resulted in sub-optimal policies. We also found that by increasing the threshold k, all LDM methods increase failure rates. This behavior is expected as a higher k allows more training data to be in-distribution. In the continuous Pointmass environment, we showed reasonable results for LDM learning and in comparison to the state-of-the-art baselines such as CQL, AWAC, and IQL, where our method performs similarly and better in some cases.

For further directions, it would be useful to explore other ways to integrate LDM into model-free deep RL methods. Furthermore, although we present overall solid performance with the our LDM methods, more complex architectures including integration with double networks and conservative regularization could lead to better performance. It would also be interesting to explore theoretical bounds on failure rate on each of our LDM methods if they exist.

In the overall report, Will contributed the code for solving the LDM through the backup equation. Will also integrated all the LDM regularization methods in the model-free DQN gridworld environment to provide experimental results for failure rates and rewards for different hyperparameters. Vishal implemented the LDM training method and contributed the experiments comparing the learned and analytical LDM as well as the comparisons to state-of-the-art baselines.

References

[MLS94] Richard M. Murray, Zexiang Li, and S. Shankar Sastry. *A mathematical introduction to robotic manipulation*. CRC, 1994.

[BES+18] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. *Exploration by Random Network Distillation*. 2018. eprint: arXiv:1810.12894.

[HZA+18] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. eprint: arXiv:1801.01290.

[DBM+19] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. *Neural Spline Flows*. 2019. eprint: arXiv:1906.04032.

[KZT+20] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. "Conservative Q-Learning for Offline Reinforcement Learning". *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 1179–1191.

[SRD+21] Kajetan Schweighofer, Andreas Radler, Marius-Constantin Dinu, Markus Hofmarcher, Vihang Patil, Angela Bitto-Nemling, Hamid Eghbal-zadeh, and Sepp Hochreiter. *A Dataset Perspective on Offline Reinforcement Learning*. 2021. eprint: arXiv:2111.04714.

[KGC+22] Katie Kang, Paula Gradu, Jason Choi, Michael Janner, Claire Tomlin, and Sergey Levine. *Lyapunov Density Models: Constraining Distribution Shift in Learning-Based Control*. 2022. eprint: arXiv:2206.10524.