Balloon Estimators for Improving and Scaling the Nonparametric Off-Policy Policy Gradient

Fabio d'Aquino Hilt [1] **JanNiklas Kolf** [1] **Christian Weiland** [1] **João Carvalho** [2] **Samuele Tosatto** [2]

Abstract

The Nonparametric Off-Policy Policy Gradient (NOPG) introduces an algorithm to solve reinforcement learning tasks in continuous environments with low sample complexity. NOPG uses nonparametric regression and Kernel Density Estimation (KDE) methods to model the reward and state transition function. In previous work, only fixed bandwidth Gaussian Kernels and KDE was used. In this work, we investigate the use of an adaptive bandwidth estimator - the Balloon Estimator. This estimator has higher performance in estimating sparse and multi-modal data in comparison to Gaussian KDE. We provide a proof of concept for using Balloon Estimators with NOPG and compare the results and performance to Gaussian KDE in the continuous mountain car task.

1. Introduction

Reinforcement Learning (RL) for low dimensional and discrete environments has been successful for a long time. In recent years, a lot of progress has been made to create approaches for high dimensional, continuous environments, e.g. the Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2019). The majority of these algorithms are on-policy and with high sample complexity, which limits the real world applicability. The Nonparametric Off-Policy Policy Gradient (NOPG) (Tosatto et al., 2020) was introduced to tackle some of these issues. In order to make Reinforcement Learning strategies applicable to the real world, the policy needs to be learned offline. Offline policy training allows to decouple training and data sampling from the real world application and is therefore much safer to use. NOPG derives the gradient of deterministic and stochastic policies from a nonparametric Bellman equation, which allows the computation of the closed form gradient of the policy. Using gradient ascent, the policy is updated in order to increase the expected return of the policy. The paper demonstrates the usage of the algorithm with a generic fixed-bandwidth Gaussian kernel for the system estimations. In this paper, we investigate the usage of an adaptive bandwidth estimator for NOPG. An adaptive bandwidth estimator computes the bandwidth either per evaluation point or per sample point in order to adapt to the density of the training data. We propose to use the Balloon Estimator (Terrell & Scott, 1992), which computes an individual bandwidth for each evaluation point and use it in for Gaussian kernel. In this paper, we introduce the Balloon Estimator for NOPG and investigate the advantages and disadvantages of such an estimator in comparison to the fixed bandwidth Gaussian Kernel Density Estimation. We show that the Balloon Estimator can be used for NOPG, but introduces other difficulties such as the need for continuous state and action spaces and data preprocessing.

2. Related Work

The Balloon Estimator is just one example of adaptive bandwidth estimators, which seems to deliver good estimations, since the bandwidths change continuously with the test point. This means that on one hand the bandwidths are more precisely tailored to the point that is evaluated, but on the other hand this might create a pathological non-integrable density function. The use of the distance to the kth nearest neighbor as bandwidth for density estimation was first introduced by Loftsgaarden and Quesenberry (Loftsgaarden & Quesenberry, 1965). Terrell and Scott picked up on the idea and collected a variety of Kernel Density Estimators with variable bandwidth (Terrell & Scott, 1992). A different adaptive bandwidth estimator that could work with NOPG is the Sample Point Estimator, which computes the bandwidths for each test point only among the test data. This means that on evaluation, the bandwidths are already fixed for each data point and do not have to be recomputed for each evaluation and the resulting density function is integrable. The downside of this Estimator is that far away training points

[1]Technical University of Darmstadt, Darmstadt, Hesse, Germany [2]Intelligent Autonomous Systems Lab, Technical University of Darmstadt, Darmstadt, Hesse, Germany. Correspondence to: Fabio d'Aquino Hilt <fabio.daquinohilt@stud.tu-darmstadt.de>, Jan Niklas Kolf <janniklas.kolf@stud.tu-darmstadt.de>, Christian Weiland <christian.weiland@stud.tu-darmstadt.de>, João Carvalho <joao.correia_carvalho@tu-darmstadt.de>, Samuele Tosatto <samuele.tosatto@tu-darmstadt.de>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

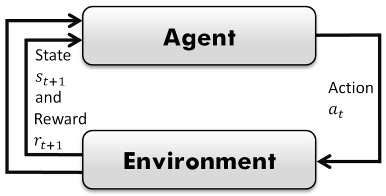


Figure 1. The Agent-Environment Interaction Cycle. The agent executes action a_t at time step t. The environment returns the new state s_{t+1} with reward r_{t+1} which are based on the executed action. Graphic based on (Sutton & Barto, 2018).

with a large bandwidth can have a bigger impact on the density at the evaluation point than close-by points with small bandwidths. This effect is called non-locality (Terrell & Scott, 1992). That could become an issue with Reinforcement Learning tasks, as the data is often clustered around the start and the goal position.
Even though there are a few options of adaptive bandwidth Kernel Density Estimators, the use of any of them in combination with Reinforcement Learning is not known to us.

3. Preliminaries

3.1. Reinforcement Learning

Reinforcement learning describes the learning of appropriate action sequences in an environment to achieve a given goal as successfully as possible (Sutton & Barto, 2018). This is modeled by having an agent take the position of the learner, whose task is to interact with the environment and achieve the goal. The environment in which the agent interacts has a state, which models how the agent's environment is constructed and can be changed by the interactions. These interactions are called actions and are available for the agent to choose from. At a given time, the agent can choose an appropriate action from the set of actions. This action is executed in the environment and changes its state. Likewise, the agent receives a reward after interacting successfully with the environment. This specifies how well a certain action is suited for a given state to achieve the given goal. Performing one-step optimal actions does not always have a direct positive impact on what happens. It is also often necessary to try suboptimal decisions in order to find better paths to the goal in the long run.

To model the environment with its states, actions and the rewards the agent receives, a Markov Decision Process (MDP) is used (Hao Dong, 2020). An MDP consists primarily of a set of states $\mathcal{S}$ of the environment, a set of actions $\mathcal{A}$ that an agent can perform, a transition function $p\left(s_{t+1}|s_t, a_t\right)$ that gives the probability of transitioning to state $s_{t+1} \in \mathcal{S}$ when performing action $a_t \in \mathcal{A}$ in state $s_t \in \mathcal{S}$. The reward function $r\left(s_t, a_t\right)$ describes the reward received for choosing action a_t in state s_t.
The agent that chooses an appropriate action for a given state is usually called a policy. The goal of reinforcement learning is to learn an appropriate policy for the environment modeled to describe the task. The policy is denoted as π and can be deterministic ($a_t = \pi\left(s_t\right)$), or it can be stochastic, i.e. a matching action is sampled from the policy with a certain probability ($a_t \sim \pi\left(\cdot|s_t\right)$). To compute the future possible, or future achievable reward R under a policy π^*, one can either compute the final reward based on the current state only, or based on the current state and the chosen action for this state.
The value function $V_\pi\left(\mathbf{s}\right)$ describes the long term reward received when starting in state $\mathbf{s}$ and following the policy π. The state-action value function $Q_\pi\left(\mathbf{s}, \mathbf{a}\right)$ describes the long-term reward received when starting in state $\mathbf{s}$, taking action $\mathbf{a}$, and subsequently following the policy π. This state-action value function describes how good it would be if the policy were to to perform that action in the current state.
With the help of these two functions a policy can be learned. It is not always helpful to use only known states, but also to explore the environment to see if there are unknown states, which allows the collection of a higher reward. The method of optimizing the policy to take the best action given the current knowledge is called exploitation, the trying out of unknown states and actions exploration. Both procedures are necessary and must be used appropriately. To use them, it is possible to use the last learned policy to make decisions for individual states and to navigate through the environment. The states visited and the experience gained then only fit that policy exactly, as another policy might have decided differently. Therefore, in this procedure the current policy must always be used for navigating the environment and only current experience and rewards can be used to improve the policy. This procedure is called on-policy, because the policy is used directly for exploration and exploitation. In contrast, there are also off-policy procedures, where there two different policies are used for exploration (sampling the environment) and exploitation (learning to maximize the expected reward). With the off-policy approach, the data collected by the exploration policy can be reused.

3.2. Kernel Density Estimation

In many areas of machine learning, data samples X arise from unknown distributions $p_X\left(\cdot\right)$ (Murphy, 2012). Kernel Density Estimation (KDE) is concerned with the reconstruction of the original probability density function $p_X\left(\cdot\right)$ with nonparametric methods. However, in scenarios where the sample density is low, it can be very difficult if not impossible to reconstruct a suitable distribution (Bishop, 2011), which gets even worse in high-dimensional or continuous

contexts.
To estimate the probability $p_X(x)$ of an arbitrary point x, the already known samples x_i can be used to determine whether the point x is close to other points or not. A higher density of data points around x indicates towards a higher probability density in the unknown distribution. To realize this, kernel functions are used which model the likelihood $K(x - x_i)$ that x is sampled near a given point x_i. They are symmetric, non-negative and integrable functions $K(\cdot)$. Often normalized kernels are used, whose integral value is 1. The approximated probability function can now be written as a sum of kernels (Murphy, 2012):

$$\hat{p}_X(x) = \frac{1}{n}\sum_{i=1}^{n} K(x - x_i).$$

This sums up for all given points the weighting by the kernel based on the distance from x to all other data points and normalizes this based on the size of the data set. What value the kernel returns is based on the distance between the two points. To increase or decrease the range of the influence of the distance on the density, the bandwidth h can be chosen. With a small bandwidth, only points that are very close to x are considered for the density calculation. Points further away have very little influence and the value of the kernel for these points goes towards 0. Smaller bandwidths tend to create wiggly density estimations. If a very high bandwidth value is used, the influence of more distant points is increased. Thus the resulting probability density function is smoother. In Figure 2 an example using Gaussian Kernels with different bandwidths is shown. The smoothness of the resulting KDE is dependent on the chosen bandwidth h and the smoothness of the training data. Kernel Density Estimation with a bandwidth h can be written as follows (Bishop, 2011):

$$\hat{p}_X(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right).$$

Choosing the right bandwidth h is a hard task and there are several heuristics and selection methods using e.g. assumptions over the underlying density of the data. This can sometimes lead to mediocre tradeoffs, e.g. if the unknown density contains steep peaks and heavy tails. To prevent such tradeoffs there are approaches to change the bandwidth depending on the location of the evaluation point (balloon estimators) or data point (sample point estimator), the former being introduced in more detail in chapter 4. In Figure 3 we can see a fixed bandwidth and the balloon estimator (both with Gaussian kernels) estimating the density of the same data.

3.3. NOPG

NOPG is a recent nonparametric off-policy policy gradient algorithm to solve reinforcement learning problems (Tosatto et al., 2020). It learns from off-policy samples and does not need any knowledge about the behavioural policy used to collect them, which makes it possible to learn from human demonstrations. It reuses those samples for sample efficiency. It has a lower variance than other semi-gradient methods using importance sampling, since it uses the full-gradient to update the policy, and is also unbiased. It is based on the closed-form solution of a nonparametric Bellman equation.

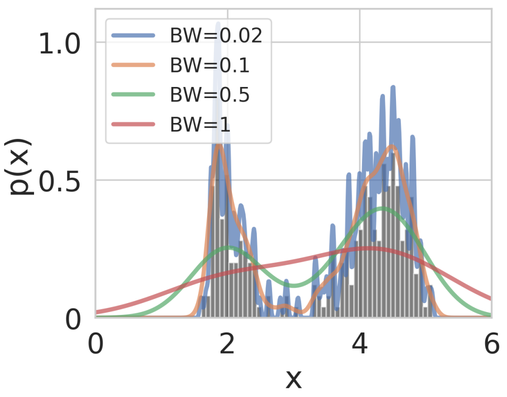


Figure 2. Given samples are located on x axis in range between 1 and 5. The grey bars show the histogram of the underlying data. Each of the four Gaussian Kernel Density Estimators has a different bandwidth. The lower the bandwidth value h, the less smooth and more bumpy is the KDE curve. If the bandwidth is higher, the KDE is smoothed out and is not capable of covering two peaks of the dataset.

The environment is modeled as a Markov decision process (MDP) with a state space $\mathcal{S} \equiv \mathbb{R}^{d_s}$, an action space $\mathcal{A} \equiv \mathbb{R}^{d_a}$, a state transition probability $p(s'|s,a)$, a discount factor γ, and a mean reward $r(s,a)$. The policy $\pi_\theta(s,a)$ can be stochastic or deterministic and is parameterized by parameters θ. Since we have focused our work only on deterministic policies, we will omit the stochastic formulations of the following equations. The value function resulting from the Bellman equation

$$V_\pi(s) = r(s, \pi_\theta(s)) + \gamma \int_{\mathcal{S}} V_\pi(s')\, p(s'|s, \pi_\theta(s))\, \mathrm{d}s'$$

is used to define the expected return

$$J_\pi = \int_{\mathcal{S}} \mu_0(s)\, V_\pi(s)\, \mathrm{d}s$$

where $\mu_0(s)$ is defined as the probability of starting in state $s \in \mathcal{S}$. To solve the Bellman equation in closed-form, the mean reward $r(s,a)$ and the transition conditional $p(s'|s,a)$ are approximated by the Nadaraya-Watson regression and kernel density estimation, respectively

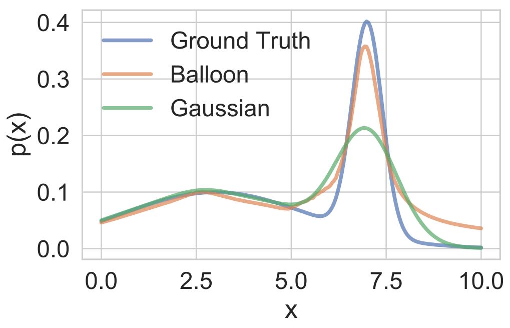


Figure 3. Balloon and Gaussian Estimators estimate a Gaussian mixture ground truth (means= $(3, 7)$, stddevs= $(2.5, 0.4)$). The Balloon Estimator uses the distance to the kth nearest neighbor from the evaluation point, where $k = \lfloor 3\sqrt{n} \rfloor$. The Gaussian Estimator uses a fixed bandwidth of $h = 0.7$. The Balloon Estimator covers the right peak of the data better than the Gaussian Estimator. It is evident that the former is a better estimation, since the shortcomings are common: they both oversmooth the well at $x = 6$ and after the peak where $x > 8$.

$$r(\mathbf{s}, \mathbf{a}) \approx \hat{r}(\mathbf{s}, \mathbf{a}) := \frac{\sum_i \psi_i(\mathbf{s}) \varphi_i(\mathbf{a}) r_i}{\sum_i \psi_i(\mathbf{s}) \varphi_i(\mathbf{a})}$$
$$p(\mathbf{s}'|\mathbf{s}, \mathbf{a}) \approx \hat{p}(\mathbf{s}'|\mathbf{s}, \mathbf{a}) := \frac{\sum_i \phi_i(\mathbf{s}') \psi_i(\mathbf{s}) \varphi_i(\mathbf{a})}{\sum_i \psi_i(\mathbf{s}) \varphi_i(\mathbf{a})}.$$

where $\psi : S \times S \to \mathbb{R}^+$, $\varphi : A \times A \to \mathbb{R}^+$ and $\phi : S \times S \to \mathbb{R}^+$ are normalized, symmetric and positive definite kernel functions. Assuming a dataset of observations s_i, a_i, r_i, s_i', we also define $\psi_i(\mathbf{s}) := \psi(s, s_i)$, $\varphi_i(\mathbf{s}) := \varphi(a, a_i)$, and $\phi_i(\mathbf{s'}) := \phi(s', s_i')$. By inserting the approximations into the value function, we obtain the non-parametric Bellman equation for deterministic policies, where the vector of responsibilities $\varepsilon_i^\pi(\mathbf{s})$ is introduced to encapsulate the dependence on the parameterized policy:

$$\begin{aligned}\hat{V}_\pi(\mathbf{s}) &= \sum_i \frac{\psi_i(\mathbf{s}) \varphi_i(\pi_\theta(\mathbf{s}))}{\sum_j \psi_j(\mathbf{s}) \varphi_j(\pi_\theta(\mathbf{s}))} \Bigg(r_i + \\ &\gamma \int_{\mathcal{S}} \phi_i(\mathbf{s}') \hat{V}_\pi(s') \mathrm{d}s' \Bigg) \\ &= \sum_i \varepsilon_i^\pi(\mathbf{s}) \left(r_i + \gamma \int_{\mathcal{S}} \phi_i(\mathbf{s}') \hat{V}_\pi(s') \mathrm{d}s' \right) \\ &= \boldsymbol{\varepsilon}_\pi^\intercal(\mathbf{s}) \left(\mathbf{r} + \gamma \int_{\mathcal{S}} \boldsymbol{\phi}(\mathbf{s}')(\mathbf{s}) \mathbf{q}_\pi \mathrm{d}s' \right)\end{aligned}$$

From Theorem 1 in (Tosatto et al., 2020) we know that:

$$\hat{V}_\pi^*(s) := \boldsymbol{\varepsilon}_\pi^\intercal(\mathbf{s}) \boldsymbol{\Lambda}_\pi^{-1} \mathbf{r},$$

where the following definitions are used

$$\begin{aligned}\boldsymbol{\Lambda}_\pi &:= I - \hat{P}_\pi \\ \hat{P}_\pi &:= \gamma \int_{\mathcal{S}} \phi(s') \boldsymbol{\varepsilon}_\pi^\intercal(\mathbf{s}') \mathrm{d}s' \\ q_\pi &:= \boldsymbol{\Lambda}_\pi^{-1} \mathrm{r} \\ \boldsymbol{\varepsilon}_{\pi,0}^\intercal &:= \int_{\mathcal{S}} \mu_0(s) \boldsymbol{\varepsilon}_\pi^\intercal(\mathbf{s}) \mathrm{d}s \\ \boldsymbol{\mu}_\pi &:= \boldsymbol{\Lambda}_\pi^{-\intercal} \boldsymbol{\varepsilon}_{\pi,0}\end{aligned}$$

The analytical gradient of the expected return can be computed as

$$\nabla_\theta \hat{J}_\pi = \left(\frac{\partial}{\partial \theta} \boldsymbol{\varepsilon}_{\pi,0}^\intercal \right) + \gamma \boldsymbol{\mu}_\pi^\intercal \left(\frac{\partial}{\partial \theta} \hat{\mathbf{P}}_\pi \right) \mathrm{q}_\pi.$$

To compute the gradients of $\boldsymbol{\varepsilon}_{\pi,0}^\intercal$ and $\hat{\mathbf{P}}_\pi$ w.r.t. θ, by the chain rule the kernels have to be differentiated w.r.t. θ as well. Actually it is only the action kernel $\varphi_i(\pi_\theta(s))$ which depends on θ but it is assumed that all kernels have the same kernel function, which has to be differentiable.

4. Balloon Estimator for NOPG

Adaptive bandwidth estimators are generally better at estimating the density of multimodal data with different heights, since it is difficult to find a single bandwidth working well for a steep and a shallow peak (Minnotte, 1993). As NOPG is designed to work on multi dimensional data, an adaptive bandwidths density estimator should be able to improve the density estimation additionally compared to the Gaussian kernel density estimator with a fixed bandwidth, since it could take advantage of the multi-dimensionality of the data. We use the Balloon Estimator to compute the bandwidth for a given data point, given the training data. This bandwidth is computed as the distance to the kth nearest data point using k-nearest neighbors (Loftsgaarden & Quesenberry, 1965). For the Balloon estimator proposed by Loftsgaarden for one-dimensional data, a rectangular kernel is used, which creates very spiky density estimations. Instead, we chose to use a Gaussian kernel. This creates smoother densities and provides an informative derivative. The resulting kernel density estimation follows the equation:

$$f(x) = \frac{1}{n h_k(x)^d} \sum_{i=1}^n K\left(\frac{x - x_i}{h_k(x)} \right)$$
$$K(x) = \mathcal{N}(x|0, 1)$$

where $h_k(x)$ is the euclidean distance to the kth nearest neighbor, n number of training points and d the dimensions of the system.
The number of neighbors k used to calculate the bandwidth is an important hyper parameter. Generally a good order for k is a multiple of the square root of the number of

training samples n, therefore $k \in \mathcal{O}\left(\sqrt{n}\right)$ (Loftsgaarden & Quesenberry, 1965). In order to have better control over the bandwidth and to test the impact of different values of k, we introduce a bandwidth-factor m such that $k = \lfloor m\sqrt{n} \rfloor$.

Importance of Data Preprocessing - The Balloon Estimator has one major flaw: it fails with duplicate data points. If too many data points are at the same location, the distance to the kth nearest neighbor h_k will collapse to almost zero, which creates a huge spike in the estimated density, which obfuscates the densities elsewhere. Therefore we had to preprocess the data, in order to remove duplicate entries, which is counter intuitive for a density estimation and possibly introduces a bias.

Implementation Details - To our best knowledge there exists no common implementation of the Balloon Estimator yet, so we implemented it ourselves. To do so, we used the PyTorch framework (Paszke et al., 2019), which generally provides Tensor computations for Deep Learning, but also an implementation to retrieve the distances to the kth nearest neighbors. To compute the gradient of the not differentiable bandwidth, because it uses kth nearest neighbors, we implemented a backwards function using the finite first order forward difference to approximate it. The calculation of the bandwidth is done for every test point in every iteration. In our case, this was easier to integrate into the existing codebase. The downside of this implementation is the increase in time complexity from computing the kth nearest neighbor distances in every iteration and has a significant impact on execution time compared to the fixed bandwidth kernel.

5. Investigations

In this section we will describe our experimental setup, then present our observations and measurements and finally analyze them in context.

5.1. Experimental Setup

Environment - To be able to compare and test different algorithms efficiently and in a standardised way, various frameworks have been developed. In this work we use the OpenAI Gym framework (Brockman et al., 2016) to test NOPG with different density estimators. For our experiments, we used the continuous mountain car environment. In this environment the agent controls an underactuated car that is initially placed in a valley between two mountains. The goal is to reach the flag located on top of the right hill. Since the slope of the mountains are too steep to simply drive uphill, the car has to gain momentum by swinging up and down the mountains. The environment is set to return a reward of $r = -1$ for each state-action pair, that has not reached the goal. The reward accumulated over time should be maximised or, put in other words, the penalty should be minimized by reaching the goal with as few actions as possible.

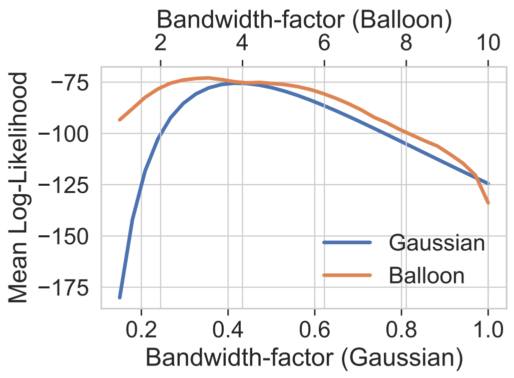


Figure 4. Analysis of mean log-likelihood for different bandwidth-factors for Balloon Estimator and Gaussian (bandwidth) for the action density estimation, calculated with 5-Fold-Cross-Validation. The analysis shows that the mean log-likelihood is greatest at bandwidth-factor $m = 4$ for the Balloon estimator and $h = 0.4$ for the Gaussian KDE.

Data generation - Since NOPG is an offline learning procedure, trajectories of the environment must be created in advance. To obtain smooth trajectories, we mapped the mouse location on the screen to an action in the simulation. To get a better generalisation of the data and to avoid duplicate actions when the mouse stops moving, we applied Gaussian noise to the actions. We created a total of 10 trajectories, from which 5 were randomly selected for all learning processes. The 5 selected trajectories contained ~ 505 data points, out of 994 total data points in all 10 trajectories. Each of the 10 generated trajectories bring the car to the goal state, which is a critical requirement, as there is no other indication of how good the current state is, since the reward function is constant.

Workflow details - In order to be able to test the density estimation procedure, we tried various parameters. We selected different values based on Figure 4. For the Balloon Estimator we tested the bandwidth-factors $m \in \{1, 4, 7\}$. Bandwidth-factor $m = 1$ serves as a baseline, while $m = 4$ is the best value based on the cross-validation and $m = 7$ serves as additional test. For Gaussian KDE we used $h \in \{0.1, 0.4, 0.7\}$. These bandwidth-factors are applied to the action estimation only, as the action estimation has the biggest impact on the optimization. For the state and next state estimators, we chose a fixed bandwidth factor of $m = 1$ for the Balloon estimator and $h = 0.3$ for the Gaussian KDE. For all methods, we chose the learning rates $0.01, 0.001, 0.0001, 0.00001$ and optimized for 50000 iterations.

Evaluation metrics - To test the quality of the solution, we compared the loss and the required iterations to reach the goal. The undiscounted *accumulated return* is returned from the environment and is equal to the number of iterations needed to reach the goal, since the reward r of any state-action pair is constant at $r = -1$. We also consider the convergence of the individual methods based on the number of iterations required until the goal is reached by the agent.

Gradients - As seen before, the bandwidth of the Balloon Estimator $h_k(x)$ is dependent on the current evaluation point x. When when computing the gradient of the kernel $K(x)$ with respect to x, the derivative of $h_k(x)$ w.r.t. x should be incorporated. But this involves taking the gradient of kth nearest neighbors, which is not differentiable everywhere (i.e. when changing neighbors) and the gradient itself might be non-informative (either 1 or -1). But not considering the gradient of the bandwidth might create an incomplete gradient and worsen the performance of the gradient based optimization method. Therefore we denote the Balloon Estimator where the gradient of the kernel incorporates the gradient of the bandwidth as Balloon-∇. As mentioned in section 4 currently the gradient of the bandwidth is computed via forward finite differences.

Investigations - We divide our investigations into three parts:

(a) We show that the log-likelihood of the Balloon Estimator is higher than that of the Gaussian Estimator and that parameter estimation is important for Balloon Estimators.

(b) We investigate the performance of Balloon Estimators in comparison to Gaussian Estimators. We also look at the stability and dispersion of the solutions found.

(c) We investigate to what extent incorporating the derivative of the Balloon Estimator's bandwidth for the individual data points influences the stability of the learning procedure.

5.2. Log-Likelihood and crucial parameter estimation

Since Balloon Estimators can approximate multi-modal data much better and can vary the weighting per data point to a great extent, the probability density function can change a lot. This leads to problems in a gradient-based learning procedure, as the procedure gets stuck in deep valleys. This can also be seen in Figure 6 and Figure 7, as for $m = 1$, NOPG does not make any progress with the Balloon Estimator and generally does not find a solution. If the bandwidth factor m and thus the smoothness of the kernels is increased, the method can influence the learning process much better. The Gaussian kernel does not have this problem, as a much smoother result is produced even at smaller bandwidths.

5.3. Performance Comparison

Figure 6 shows that Balloon and Gaussian Estimators have similar performance in reducing the loss. However, the Balloon Estimator has a much higher dispersion of the loss and does not manage to keep the loss smoothly at one level like the Gaussian Estimator. However, the policies found are more stable than those of Gauss. With suitable learning rates of 0.001 and 0.0001, the Balloon Estimator produces stable policies and reaches the target much faster on average than Gaussian, whose trajectories and policy results vary greatly, as seen in Figure 7. Likewise, the Balloon Estimator arrives at an acceptable solution much faster than the Gaussian Estimator. Gaussian finds a solution within 20000 iterations that is comparable to the Balloon Estimator solution found between 5000 and 10000 iterations. This also shows that the calculated loss does not always perfectly reflect the quality of the solution. Based on the solutions found, the Balloon Estimator has shown better performance despite more unstable loss curves and there lower convergence speed.

5.4. Gradient Influence

As can be seen in Figure 8, the balloon estimator using the gradient of the bandwidth determination reaches higher return values faster than the balloon estimator without gradient. Especially at the learning rates 0.001 and 0.0001 the learning process stagnates without gradient calculation. At the learning rate of 0.00001, the influence of the gradient on the learning performance is also clearly visible. Therefore, it seems advantageous to include the gradient of the bandwidth in the learning procedure.

6. Conclusion and Future Work

We implemented the adaptive bandwidth Balloon KDE variant which is able to optimize sub-optimal trajectories for the Nonparametric Off-Policy Policy Gradient and compared the results to the fixed bandwidth Gaussian KDE. We showed that the Balloon Estimator is able solve the mountain car environment on par with the Gaussian Estimator and even has some slight advantage in the resulting performance.
The Balloon Estimator should theoretically perform even better in higher dimensional environments (Terrell & Scott, 1992), but the way NOPG is currently implemented the dimensions of the environment are split into individual kernels. To boost the performance of the Balloon Estimator, the current implementation could be adapted, to support higher dimensional environments natively. In the same way, the

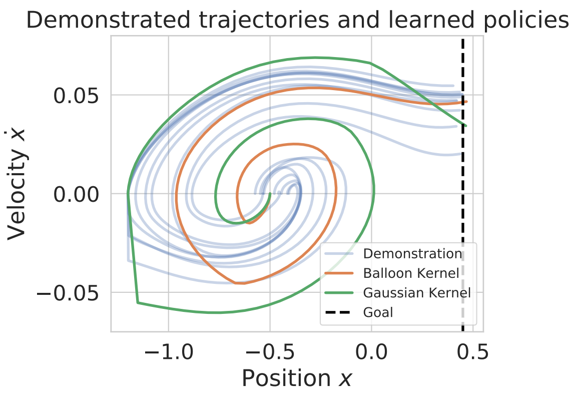


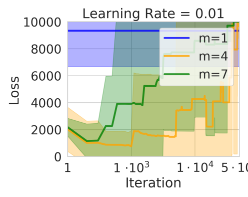
Figure 5. With 10 demonstrations, NOPG is able to learn a policy that brings the mountain car to the goal for the Gaussian kernel as well as the Balloon kernel. The x-axis shows the Position of the car, with the starting position being $x = -0.5$. At the left side, there is a border at $x = -1.2$, while the goal state is at $x = 0.45$. The y-axis shows the velocity of the car, with positive values being a velocity in positive/right direction and negative values in negative/left direction. The Balloon-learned policy chooses smaller actions at the beginning and stays longer in the valley, while the Gaussian-learned policy takes greater actions which results in running against the border on the opposite side of the goal.

Balloon Estimator could be applied to other higher dimensional environments, for example the LunarLander from OpenAI Gym. It is however notoriously difficult to choose an appropriate bandwidth matrix. The bandwidth of multi-dimensional KDE is generally a matrix, often chosen to be a scaled identity of diagonal matrix for simplicity (Terrell & Scott, 1992).

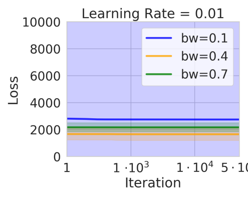
Choosing a good bandwidth factor is crucial to a good density estimation, both for Balloon and Gaussian Estimators, and should not be left to chance. To tackle this NOPG could be complemented by a cross-validation or similar approach to choose a good bandwidth. This would simplify the setup and remove a hyper-parameter. To create confidence in the bandwidth selection it might also be interesting to further analyze the influence of the bandwidth in the optimization process.

The approximation of the bandwidth gradient is currently being computed with first order forward differences, which might be too simplistic. Before expanding it to a more sophisticated numerical approximation, it might be worth to analyze which values the bandwidth gradient takes and the effect this has on the full gradient of the policy. It could also be worth trying to see if it is possible to have PyTorch compute the exact gradient of the bandwidth with its automatic gradient computation.

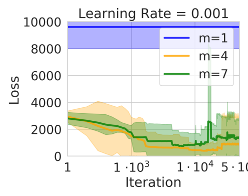
Another point of interest could be the use of other adaptive bandwidth methods. Investigating the Sample Point Estimator could be interesting as the bandwidth for this type of estimator depends only on the training data and does



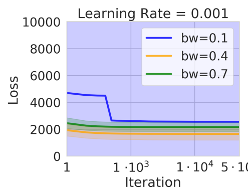
(a) Balloon - ∇



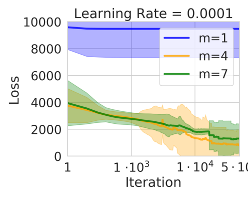
(b) Gaussian



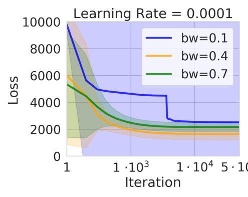
(c) Balloon - ∇



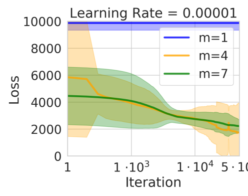
(d) Gaussian



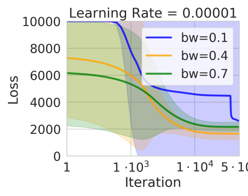
(e) Balloon - ∇



(f) Gaussian



(g) Balloon - ∇



(h) Gaussian

Figure 6. Comparison of the loss between Balloon Estimator with gradient calculation (left column) and Gaussian Estimator (right column) with different learning rates. Learning rates used: (a)-(b)= 0.01, (c)-(d)= 0.001, (e)-(f)= 0.0001, (g)-(h)= 0.00001

not change with the evaluation point, thus the kth nearest neighbor can be pre-calculated, which accelerates training with the Sample Point Estimator.

To accelerate the execution of the Balloon Estimator it should be possible to store the data in a k-d tree or a ball tree to accelerate the kth nearest neighbors computation. As NOPG uses offline learning methods, the dataset would be valid across all iterations. In return this strategy would increase the space complexity and might not be easily executable for huge datasets, though this depends on the data structure chosen and would have to be analyzed specifically. With this approach, Balloon Estimator might not be significantly slower than NOPG with Gaussian KDE anymore.

Right now the preprocessing removes duplicate data points to prevent collapsing bandwidths and thus exploding densities. Arguably this removes points which could help describe the unknown density. It could be interesting to analyze if this creates a bias in the density estimation and

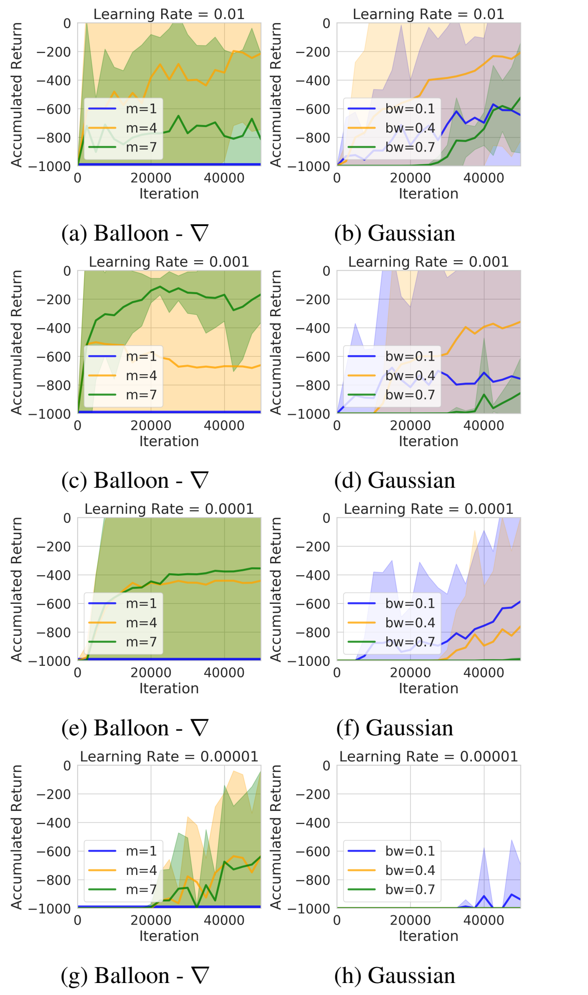


Figure 7. Comparison of accumulated return between Balloon Estimator with gradient calculation (left column) and Gaussian Estimator (right column) with different learning rates. Learning rates used: (a)-(b)= 0.01, (c)-(d)= 0.001, (e)-(f)= 0.0001, (g)-(h)= 0.00001

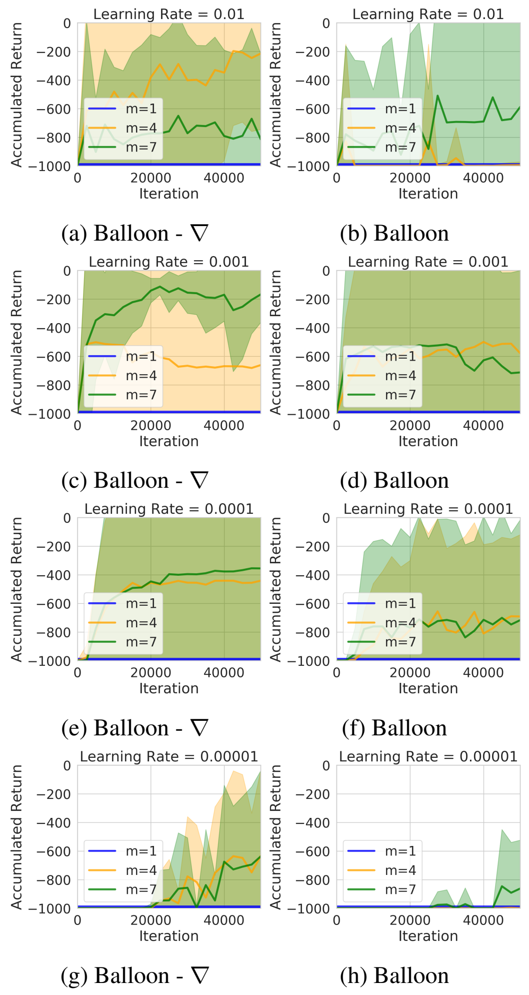


Figure 8. Comparison of accumulated return between Balloon Estimator with gradient calculation (left column) and Balloon Estimator without gradient calculation (right column) with different learning rates. Learning rates used: (a)-(b)= 0.01, (c)-(d)= 0.001, (e)-(f)= 0.0001, (g)-(h)= 0.00001

how this bias scales with the number of dropped points relative to the total points. Note however that this should have no noticeable impact on our results, since the used dataset contains less than 2% duplicates.

References

Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, hardcover edition, 2011. ISBN 0387310738,9780387310732.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016. URL https://arxiv.org/abs/1606.01540.

Hao Dong, Zihan Ding, S. Z. *Deep Reinforcement Learning: Fundamentals, Research and Applications*. Springer-Nature New York Inc, 2020. ISBN 9811540942,9789811540943.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning, 2019.

Loftsgaarden, D. O. and Quesenberry, C. P. A Nonparametric Estimate of a Multivariate Density Function. *The Annals of Mathematical Statistics*, 36(3):1049 – 1051, 1965. doi: 10.1214/aoms/1177700079. URL https://doi.org/10.1214/aoms/1177700079.

Minnotte, M. C. A test of mode existence with applications to multimodality. diss., rice university., 1993. URL https://hdl.handle.net/1911/16652.

Murphy, K. P. *Machine learning: a probabilistic perspective*. MIT press, 2012.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library, 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Terrell, G. R. and Scott, D. W. Variable Kernel Density Estimation. *The Annals of Statistics*, 20(3):1236 – 1265, 1992. doi: 10.1214/aos/1176348768. URL https://doi.org/10.1214/aos/1176348768.

Tosatto, S., Carvalho, J., Abdulsamad, H., and Peters, J. A nonparametric off-policy policy gradient. 2020. URL https://arxiv.org/pdf/2001.02435.pdf.