Correlated Exploration in Deep Reinforcement Learning

Korrelierte Erkundung in Deep Reinforcement Learning
Bachelor-Thesis von Maximilian Hensel aus Bad Nauheim
Tag der Einreichung:

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Dr. Riad Akrour
3. Gutachten: M. Sc. Oleg Arenz

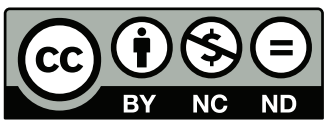
TECHNISCHE
UNIVERSITÄT
DARMSTADT

Correlated Exploration in Deep Reinforcement Learning
Korrelierte Erkundung in Deep Reinforcement Learning

Vorgelegte Bachelor-Thesis von Maximilian Hensel aus Bad Nauheim

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Dr. Riad Akrour
3. Gutachten: M. Sc. Oleg Arenz

Tag der Einreichung:



Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 31. Oktober 2019

(Maximilian Hensel)

Abstract

Reinforcement Learning (RL) provides a general framework for sequential decision making. Recently RL has shown success through the combination with deep learning [70, 2] by using massive computation and data budgets. The potential of deep RL to generalize to new situations and handle noisy and complex data make it a promising approach in creating intelligent autonomous robot systems. Unfortunately, real world data is expensive to collect and thus deep RL for robotics currently remains a research area.
It is possible to use a simulation in order satisfy the sample requirements of RL. However, real world and simulation differ and thus policies trained in simulation do not necessarily perform well in the real world. A possible way of overcoming this gap is optimizing the policy for the real by training on the real system. The training procedure requires a way of collecting data which is applicable to the real world. We will refer to this as *local exploration*. In the scope of this thesis we evaluate standard exploration methods w.r.t. their fitness for local exploration. In particular we are interested if the default Gaussian action noise is still the first choice for local exploration over correlated noises such as parameter noise or noise sampled from a stochastic process. For this we evaluate three tasks in simulation, two of which we also evaluate on a real robot system.

Contents

1 Introduction

Reinforcement Learning (RL) [1] provides a general framework for sequential decision-making. In RL an agent interacts with an environment and obtains rewards based on the quality of the interactions. Roughly speaking, the agents goal is to maximize some notion of rewards by acting in the best way possible.
Through the combination with Deep Learning, RL has recently yielded impressive results in certain areas. For example the seminal work of Deep RL by [70] plays Atari games with super-human performance from raw pixel observations. Similarly impressive is AlphaGO [2] where Deep RL is used to play Go at a super-human level.
However, currently these successes hinge on high computation budgets and enormous amounts of samples. Even if both these are present, RL is currently very brittle and can fail easily in unexpected ways. This is unfortunate since many problems may be expressed as RL problems but not be solved due to the aforementioned limitations.
An example of such a domain is Robot Learning (RoL). RoL aims to build intelligent and robust agents with learning based approaches [3]. Contrary to computer vision [4], natural language processing [5] and others, where purely learning based approaches recently surpassed handcrafted feature engineering, RoL has not surpassed classic control in terms of real world use. This is because collecting samples on real robot is expensive, not only because interaction causes wear and tear of the robot, but also the - compared to a simulation in wall time - slow generation of samples which might also require human supervision such as resetting the robot or the environment. Finding methods which do not face these limitations is a highly active area of research where multiple directions show some success [6, 7, 8, 9, 10].
While these approaches attack the problem from different angles, the fundamental problem of how to do exploration is still with all of them. Exploration describes how the agent acts to obtain new information about the environment. Since the agent starts off with limited knowledge about possibly rewarding situations or other characteristics of an environment, this is an integral part of the agents success. In addition to the sample constraints, robot learning requires the exploration to be safe for robot and environment.
Within scope of this thesis we want to evaluate exploration methods and see how suited they are for exploration on a real system. We will not require exploration from a random policy but rather start with a "decent" prior policy which we want to improve. The prior policy should be close to the optimal behaviour and thus short term exploration around the the prior should be sufficient. We refer to this problem as *local exploration* to distinguish it from the more common exploration based around sparse rewards.
There are numerous alternative ways of exploring [11, 12] which mostly focus on the orthogonal problem of sparse rewards. Nevertheless, some of these methods are more suited for local exploration than others and could provide benefits over the standard approach of uncorrelated noise. We will shortly discuss a selection of these approaches and then focus on action space noise sampled from a stochastic process [13] and noise applied parameter space of the policy [14]. We compare them to the default Gaussian action noise with our specific requirements in mind on simulated and real world tasks.

2 Reinforcement Learning (RL)

Reinforcement learning is a framework used to describe sequential decision making problems. At a high level it consists of an agent which interacts with an environment. The interactions take place at previously specified decision times where the agent observes the environment and chooses a valid action to execute. In this way the agent can manipulate the environment towards favorable outcomes. What states are favorable is expressed through numerical reward signal the agent obtains after each time an action is chosen. High reward correspond to good actions, low reward to a bad choice of actions. Typically the agent's goal is maximize a quantity depending on this reward signal.
This is a very general framework, which allows us to express a large variety of decision making problems. For example, we could express an industry robot arm performing some task as a reinforcement learning problem. The agent could observe its own joint positions and velocities and the positions the objects relevant for the task. A possible choice of actions would be the amounts of force applied to each of the robots joints. If the task was about assembling some parts, we could reward the agent every time it successfully assembles something.

2.1 Markov Decision Process (MDP)

Formally the Reinforcement Learning problem is expressed in a Markov Decision Process (MDP). The Markov Decision Process [15] $\mathcal{M}$ is a tuple $(\mathcal{S}, \mathcal{A}, P, \mathcal{R}, R, d_0)$ consisting of the set of all states $\mathcal{S}$, the set of all actions $\mathcal{A}$, a transition function $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \longmapsto [0,1]$, a set of all possible rewards $\mathcal{R} \subseteq \mathbb{R}$, a reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \longmapsto \mathcal{R}$ and a starting state distribution $d_0 : \mathcal{S} \longmapsto [0,1]$.
A MDP models the decision times as discrete points points called time steps. At each time step $t \in \mathbb{N}_{\geq 0}$ an observation of the state of the environment s_t is made and afterwards an action a_t is sampled from the policy $\pi : \mathcal{S} \times \mathcal{A} \longmapsto [0,1]$. After choosing an action, the MDP transitions to a successor state s_{t+1} for the next time step $t+1$ sampled according to $s_{t+1} \sim P(s' \mid s, a)$, where we conditioned P on the current state and action. Finally a reward $r_t = R(s_t, a_t, s_{t+1})$ is provided as feedback to the agent. We may choose to drop the dependence on s_{t+1} for the reward function if no confusion arises from it.
In the most general setting, no specific structure is imposed on $\mathcal{S}$ and $\mathcal{A}$. In the special cases where $\mathcal{S}/\mathcal{A}$ are finite, we will refer to them as discrete state spaces/action spaces. When $\mathcal{S}/\mathcal{A}$ are uncountable, we refer to them as continuous state spaces/action spaces.
We call a sequence of temporally successive states and actions $(s_0, a_0, s_1, a_1, \dots)$ with $s_0 \sim d_0$, $a_0 \sim \pi(\cdot \mid s_0)$, $s_1 \sim P(s_1 \mid s_0, a_0)$, $\dots$ trajectory and will usually denote it by τ. With slight abuse of notation we may write $\tau \sim \pi$ to express that τ is a trajectory generated from π as above.
An important aspect of the MDP is the Markov property. We assume that the current state contains the full information required to evaluate reward function and transition probabilities. Concretely, we make the assumption that

$$P(s_{t+1} \mid s_0, a_0, s_1, \dots) = P(s_{t+1} \mid s_t, a_t),$$

and

$$R(s_t \mid s_0, a_0, s_1, \dots) = R(s_t, a_t, s_{t+1}).$$

The goal of the RL problem is to find the optimal policy. The are many possible notions optimality which will produce optimal policies with different characteristics. In our setting we will use an objective function $J : \Pi \mapsto \mathbb{R}$ which assigns a value to our policy. We define the optimal policy π^* to be a policy which maximizes J, i.e. $\pi^* \in \arg\max_{\pi \in \Pi} J(\pi)$, where Π is the set of all policies. Note that there may be multiple optimal policies.
The most typical setting for Deep RL is the infinite horizon setting. For this we define the discounted return G_t starting from time step t by

$$G_t = \sum_{k=t}^{\infty} \gamma^{k-t} R(s_t, a_t),$$

where we call $\gamma \in (0,1)$ discount factor. The γ ensures convergence of the sum but can also be intuitively understood as valuing short term rewards higher than rewards long into the future. The agent's goal is to maximize the expected discounted reward

$$J(\pi) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} R_t\right].$$

We define the value function $V^\pi : \mathscr{S} \mapsto \mathbb{R}$ of policy π to map a state to its expected discounted reward, $V^\pi(s) = \mathbb{E}[G_1 \mid s_1 = s, \pi]$. The conditioning on π indicates that actions are chosen w.r.t. π. Similarly to V^π we define $Q^\pi : \mathscr{S} \times \mathscr{A} \mapsto \mathbb{R}$ as the expected discounted reward given a state and an action, $Q^\pi(s,a) = \mathbb{E}[G_1 \mid s_1 = s, a_t = a, \pi]$.
Under some conditions [15] an MDP admits a stationary state distribution. We denote by ρ_t the state distribution of the MDP at time step t.

2.2 Policy Gradients

RL algorithms based on policy gradients are perhaps the simplest and most direct RL algorithms. The basic idea of policy gradients is to find the optimal policy by updating the parameters of a policy π_θ iteratively in the direction of the gradient $\nabla_\theta J$.
The stochastic policy gradient theorem allows us to compute the gradient of a policy by

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \int_{\mathscr{S}} \rho^\pi(s) \int_{\mathscr{A}} \nabla_\theta \pi_\theta(a \mid s) Q^\pi(s,a) da ds \\ &= \mathop{\mathbb{E}}_{s\sim\rho^\pi, a\sim\pi_\theta} [\nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s,a)].\end{aligned}$$

Surprisingly, the policy gradient is independent of the gradient of the state distribution, allowing us to estimate it from samples. This theorem forms the basis of many policy gradient algorithms, the simplest one being REINFORCE [16]. A similar result can be derived for a deterministic policy μ_θ [17]. We have

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \int_{\mathscr{S}} \rho^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a) \mid_{a=\mu_\theta(s)} ds \\ &= \mathop{\mathbb{E}}_{s\sim\rho^\mu} [\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a) \mid_{a=\mu_\theta(s)}],\end{aligned}$$

where Q^μ is a learned state-action value function which must be differentiable w.r.t. the actions. The Q function is referred to as critic and algortihms which learn a policy and a critic are called actor-critic. The first algorithms based on the deterministic policy gradient is DPG [17] and does not use deep neural networks as policy or critic. DPG uses the deterministic policy gradient theorem to update the policy via the gradient and trains the critic with temporal differences. The updates take place after a fixed amount of time steps and uses the most recent data. Algorithms based DPG are off-policy which means they can use data from a arbitrary exploration policy. Exploration in the algorithms is usually conducted by adding noise on top the actions of the deterministic policy.

2.2.1 Twin Delayed Deep Deterministic Policy Gradient (TD3)

TD3 is the algorithm we use in this thesis. It is based on Deep DPG (DDPG) [13] which itself extends DPG to be usable with deep neural networks. Essentially DDPG applies the tricks of DQN to DPG. DDPG makes use of a replay buffer which stores past experience and a target network for the critic. The target network is updated each step by linear interpolation rather than a hard update after fixed amount of steps in DQN. Both actor and critic are updated from a minibatch of transitions at each time step.
Twin Delayed DDPG (TD3) further improves on DDPG by employing some tricks which increase stability of the algorithm. Firstly it addresses the overestimation of Q-values [18] by learning mutiple critic. In the calculation of the TD-Error the minimum value of all the critics is used resulting in less overestimation. Secondly, TD3 updates the actor at a lower rate than the critic. This is based on the observation that updates with a bad critic may be destructive to the learning progress. Updating the actor less frequently ensures that the critic has better fit. The third improvement is to enforce smoothness of the critic by introducing noise into the training data. The resulting TD3 algorithm can be seen in algorithm 1.

2.3 Exploration

In RL an agent has to collect its own training data to learn from unlike supervised learning where a dataset is fixed. Exploration methods address how to collect data in order to increase the agent's knowledge about the environment's dynamics and reward function. At any point the agent has to make a choice between using its current knowledge to gain the highest possible (to its current knowledge) cumulative reward or keep exploring. These two behaviours are conflicting ways of acting. *Exploration* is a long term endeavour where the agent tries to maximize the possibility of high rewards in the future, while *exploitation* is making use of the current knowledge and maximizing the expected rewards in the short term. The agent needs to strike a balance between these two contrasting behaviours, often referred to as "exploration-exploitation dilemma".

Algorithm 1 TD3, adapted from [19]

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta_1' \leftarrow \theta_1$, $\theta_2' \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \varepsilon$,
  $\varepsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \varepsilon, \quad \varepsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \arg\min_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$
    $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
  end if
end for

2.3.1 Action Noise

The simplest exploration methods are based on applying noise to the action space. For discrete action spaces, we may select an action randomly with chance ε and follow the policy with chance $1 - \varepsilon$. This simple exploration strategy is called ε-greedy [1] as the agent acts greedily up to ε. The random action itself is sampled usually sampled uniformly from all possible actions.
For continuous action spaces, noise usually is added to the action selection, i.e we select actions according to $\mu(s) + \epsilon$, where ϵ is some kind of noise and μ is a function which outputs the mean of the policy depending on state.
Algorithms based on the stochastic policy gradient [20, 21] require backpropagation through the noise distribution as they make use of the likelihood function, thus limiting the choice of the distribution for ϵ.
In deterministic off-policy gradient algorithms, such as TD3 [19], it is not necessary to evaluate the likelihood of actions, therefore we can employ more complex ways of generating noise which do not necessarily have a tractable likelihood function. Most prominently DDPG [13] uses the Ornstein-Uhlenbeck process [22] to generate noise which is correlated in time. We will now focus on noise applicable to continous action spaces.

Gaussian Noise

The default choice for ϵ is Gaussian noise, $\epsilon \sim \mathcal{N}(0, \sigma^2)$, where σ^2 may be trainable, fixed or follow other heuristics depending on the specific algorithm. We may also choose other distributions such as the beta distribution if we want to guarantee that the actions stay within some valid range. The perturbation of those approaches will not be correlated in time and this might make exploration hard in scenarios where specific sequences of actions need to be taken in row.

Ornstein-Uhlenbeck (OU) Process

Instead of adding uncorrelated noise onto the actions we can sample noise from a stochastic process such that the noise is correlated over multiple time steps. The most prominent stoachastic process used for this is the Ornstein-Uhlenbeck (OU) Process. The OU process can be described as continuous-time random walk with drift back to the origin. The process is defined by the stochastic differential equation (SDE)

$$dx_t = -\theta(\mu - x_t)dt + \sigma dW_t,$$

where $\theta > 0$ and $\sigma > 0$ are parameters and W_t is the Wiener process [23]. μ is the origin of the process, θ describes the strength of the drift back to μ and σ describes the magnitude of the Wiener process.
We can sample values from the process in discrete time intervals with width T by using the Euler–Maruyama method. This results in the following equation

$$x_t = x_t - \theta(\mu - x_t)T + \sigma\sqrt{T}\varepsilon$$

for the noise at time step t with $\varepsilon \sim \mathcal{N}(\prime, \infty)$. Since T can be chosen arbitrarily, we choose $T = 1$ such that we obtain

$$x_t = x_t - \theta(\mu - x_t) + \sigma\varepsilon.$$

This is equivalent to the $\mathcal{AR}(1)$ process and we can see the Ornstein-Uhlenbeck process as a continuous time analogue of $\mathcal{AR}(1)$.
There are two interesting parameter choices: In the case where $\theta = 1$, we obtain uncorrelated Gaussian noise and for $\theta = 0$, we obtain a Gaussian random walk. Figure 2.1a shows these some realizations of the OU process. We can see that low values for θ cause the process to wander off further from μ. Also note that the behaviour $\theta = 0.4$ seems similar to the uncorrelated Gaussian noise with $\theta = 1$. Caused by out choice of time interval T, the behaviour of the OU process we are interested in, the correlation in time, is only present at low θ.

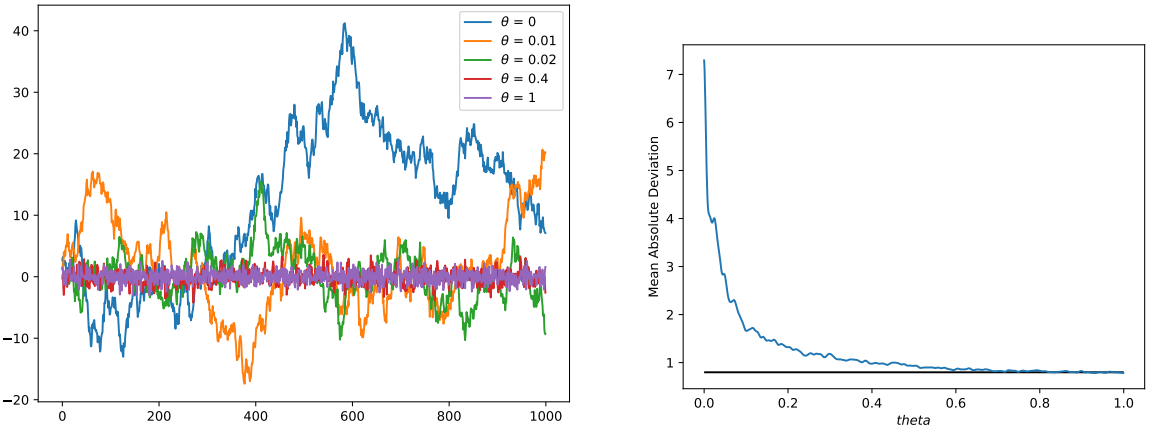


Figure 2.1: (left) OU process realizations for different θ and fixed $\mu = 0$, $\sigma = 1$ and $T = 1$. Low values of θ make the process wander away further from μ. For high θ behaviour is similar to uncorrelated Gaussian noise. (right) Mean absolute deviation (MAD) estimated from samples for different values of θ with fixed $\mu = 0$, $\sigma = 1$ and $T = 1$. The straight line is the MAD of a normal distribution which is constant at $\sqrt{2/\pi}$.

An interesting quantity to look at is the mean absolute deviation (MAD) of the OU process X_t discretized at times $t = 1 \ldots n$

$$\frac{1}{n}\sum_{t=1}^{n} E|X_t - EX_t|.$$

Figure 2.1b shows the MAD for different values of θ. We can again see that for $\theta > 0.4$ the behaviour is similar to the uncorrelated Gaussian and keep this in mind when looking for interesting parameter values.

2.3.2 Parameter Noise

Instead of applying noise in the actions space, we can instead apply noise to the parameter space of a policy. The basic idea is simple: We sample a noise vector ϵ from a distribution, usually $\epsilon \sim \mathcal{N}(0, \sigma^2)$, and add it onto the policy parameters θ to obtain the perturbed parameters $\hat{\theta} = \theta + \epsilon$. We then use the policy $\pi_{\hat{\theta}}$ for one episode to obtain new data. As we keep the perturbation fixed for multiple time steps, this should produce exploration behaviour that is correlated in time. To apply this to deep RL during the training process, it is necessary to adapt the amount of noise during the training process. This is because the impact of parameter perturbation depends of training progress, model architecture and other factors. The adaptation can be done by making σ backpropagatable and using gradient updates [24]. Another approach computes the difference in the action space the parameter noise causes and tunes the parameter noise in such a way that it remains fixed [14].

2.3.3 The Problem Of Naive Exploration In A General Setting

In practice simple action noise is the most commonly used form of exploration [21, 25, 20]. These approaches work in environments where random sequences of actions are likely to cause positive rewards or to “do the right thing”. In domains where rewards are infrequent, getting a random positive reward can become very unlikely, resulting in a worst case sample complexity exponential in the amount of states and actions [26, 27, 28, 29]. For example, figure 2.2 shows an case where random exploration suffers from exponential sample complexity (Figure 2.3).

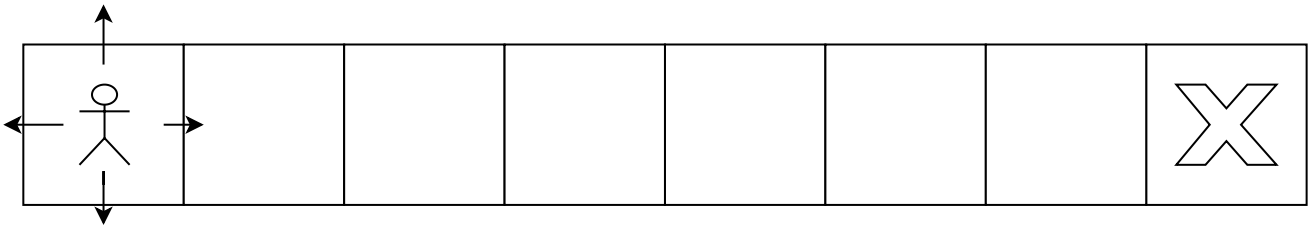


Figure 2.2: A simple problem where random exploration performs poorly. The agent can move in the directions indicated by the arrows. If the agent moves in a direction without next tile, the episode ends. The agent is rewarded if and only if it reaches the X. Purely random action selection at each time step requires $\Theta(4^N)$ episodes to reach the X, where N is the number of tiles. As no intermediate rewards are available, ϵ-greedy exploration will essentially be random, therefore requiring $\Theta(4^N)$ episodes as well. Optimistic approaches, which actively try to visit new states, only require $\Theta(N)$ episodes.

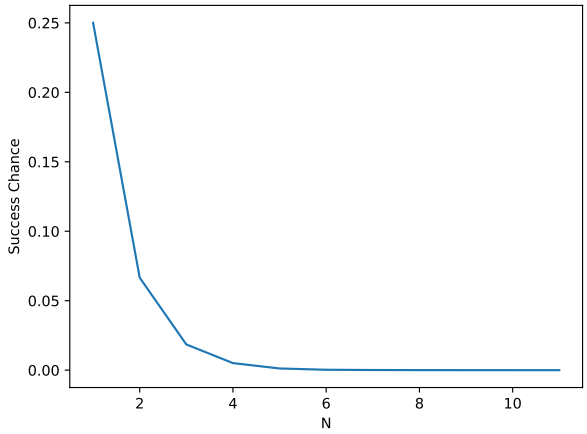


Figure 2.3: Success rate of an agent choosing random actions in the MDP seen in figure 2.2. The success rate is almost zero at 7 tries. We obtained the data from 200000 simulations of the environment.

2.3.4 More Complex Exploration Strategies

There is a wide array of different exploration strategies, many of which are based on optimism in the face of uncertainty (OFU) [30]. In these algorithms the agent acts greedily w.r.t. action values that are optimistic by including an exploration bonus depending on a visitation count, e.g. [31, 32, 26] for tabular MDPs. In continuous MDPs these algorithms are not directly applicable not only because storing the visitation counts becomes computationally intractable in terms of memory, but also since each state-action pair is likely to be visited only once or even never. Some approximations of counts which generalize between similar states include [33, 34, 35]. These algorithms augment the reward signal of the MDP with an exploration bonus and do not have theoretical guarantees. There are many other forms of intrinsic motivation in RL which are not based on counts [11, 36, 37, 38, 39]. Generally, the augmentation of the reward is called intrinsic reward, a term stemming from psychology [40].
Another approach is given by Bayesian RL methods. The goal of the methods is to minimize regret and therefore explore efficiently. This can achieve stronger performance in tabular MDP than UCB based approaches [41]. Again, these approaches do not scale to large or continuous domains as the belief state grows exponentially in the size of states [42, 27]. The posterior distribution over MDPs can be approximated to make these method work in large state spaces. Possible approximation include dropout [43, 44, 45] which is a bad approximation, bootstrapping [29, 12] and parameter noise [24, 46].

3 Robot Learning (RoL)

The high complexity of the real world with partial observations and unexpected events makes classic control unfit for tasks outside a controlled environment. A different approach at the intersection of machine learning and robotics is given by Robot Learning (RoL). RoL aims to build intelligent autonomous robot systems through learning based approaches. The learning based approaches show promise through their robustness to new situations stemming from the inherent generalization potential of machine learning. While there are multiple learning paradigms which are applicable to RoL, we will focus on Reinforcement Learning.

3.1 Difficulties Of Applying Reinforcement Learning To Robot Learning

RL has achieved impressive results when a simulation of the task is available. By using a massive computation budget it is possible to satisfy the large sample requirements of current RL methods. Unfortunately, in the real world, and especially in the context of Robot Learning, samples are expensive and the ideal conditions which are a prerequisite to the success of Deep RL are not a given. For this reason Deep RL is not yet at a level one might expect from AlphaGO [2] or other success stories. The following short discussion of these issues is based on [3] to which we refer for a more in depth treatise.

Real world samples are expensive due to hardware limitations and time constraints. Firstly, each sample is based on a real world interaction and thus contributes to the degradation of the expensive robot hardware. This hardware, unlike a simulation, has high acquisition cost and cannot be replicated easily. The high acquisition cost together with the inflicted costs from wear and tear limit the amount of real systems available and thus the rate sample collection. As the sample collection takes place in the real world, it cannot be sped up like a simulation. Thus each sample takes a high amount of wall clock time to collect, further limiting the rate of sample collection.

Furthermore, human supervision of the sample collection is necessary because of multiple reasons. The high instability of some RL methods can threaten to damage robot and surroundings, therefore great care and attention must be paid to how the policy develops during learning and if the behaviour it exerts is safe. This requires domain knowledge about the robot system and its safety limits. Resetting the environment may require human supervision as well and thus consumes valuable time and leads leads to another issue. The initial conditions of the task, e.g. robot arm position, cannot be reproduced as consistently as in simulation where the exactly same conditions can be ensured. In a similar way, the robot, task objects or other factors may vary slightly from time to time. One can argue that a well working learning based method should be able to generalize between such marginal differences; unfortunately this is not always the case. A reason for this is the vastly more complex nature of the real world. Simulations often choose to neglect or are simply not able to model some effects present in the real world, thus making simulated tasks easier.

Often in real world scenario observations are not complete, therefore requiring the agent to deal with incomplete knowledge. In addition to being partially observed, the states are mostly continuous for robotics tasks and the transition functions feature discontinuities. A natural example for this are collisions where upon collision with something the robot comes to a sudden stop. Similarly to the state spaces being continuous, the action space is usually continuous as well. This means that a broad range of RL algorithms, e.g. valued based algorithms as DQN), are not applicable due to their inability to handle continuous action spaces without discretization. By discretizing we can use these methods but lose the fine-grained control that a continuous action space offers. Furthermore, the discretization may suffer from the curse of dimensionality as action spaces for robotics control routinely have upward of 30 dimensions. The high dimensionality of the action space not only poses a problem for discretization but also for algorithms which are able to deal with continuous action spaces.

3.2 Deep RL Approaches To Robot Learning

There is a wide array of approaches to robot learning, e.g. imitation [94] or probabilistic movement primitives [93]. Non-Deep RL based approaches [74, 3] can be more sample efficient than Deep RL and therefore remain competitive in robot learning. Still, the large learning capacity of Deep RL can provide merit for RoL, as it can enable handling image observations [6] and fast generalization [76].

To overcome the massive samples requirements, if available, we can make use of a simulator to generate samples of the task. The simulator, though, is only an imperfect representations of the real world. The gap between simulation and reality is referred to as sim2real gap and causes policies to perform differently on real system than in simulation.

Approaches based on domain randomization [10] try to randomize the simulator such that the real world is contained within some of the randomizations. A policy is trained on the randomized environments in order to obtain a robust policy that identifies the environment and acts according to it. To the policy, the real world behaves like just another randomized environment. This approach can work but requires a high computation budget and a simulator which can represent the real world sufficiently well. Also, the policies learning capacity needs to allow good performance across all environments. In similar spirit, SimOpt [9] tunes the simulator parameters in such a way that it produces trajectory which are similar to real world trajectories.

4 Local Exploration For Robot Learning

4.1 Local Exploration as an Approach To Robot Learning

In this thesis we want to apply deep RL to robot learning by pre-training a policy π_{ref} in simulation and then optimizing it for the real world by training with RL on the real system. As we do not adapt or randomize the simulator, π_{ref} performs well in simulation but sub-optimally on the real system due to the existence of a sim2real gap. In this thesis we assume that π_{ref} is close to a local optimum that we are content with, allowing us to locally optimize the policy. This should also significantly reduce sample requirements when compared to learning from a random policy, possibly making learning on a real system feasible.
Training a policy on the real robot follows the standard procedure of reinforcement learning: We collect samples and then improve our policy as much as possible with the information contained in the samples. Due to the constraints on the amounts of available samples, using each sample to its full potential is important. This depends on the choice of learning algorithm and will not be of concern for us in this thesis. Our focus will be on the way the samples are generated, the exploration method, and the potential improvements we can obtain with these samples. The exploration method has to exert a special kind of exploration we will refer to as *local exploration*.

4.2 Defining Local Exploration

For the approach discussed in the previous section it will not be necessary for our exploration method to deal with sparse reward or other potentially hard-exploration problems. On the other hand we have special requirements from robot learning. Loosely speaking we can define local exploration as exploring shortsightedly around a reference policy. In the remainder of this section we will make the notion of local exploration more concrete by discussing the two conflicting main requirements of local exploration.

Sample Efficiency / Amount Of Exploration
The goal of local exploration is to collect data in such a way that we can improve our policy as much as possible in one step. This is different from the usual exploration problem as we do not aim to minimize regret or adhere to other metrics of efficient exploration. In a general setting this might lead to the policy getting stuck in a local maximum which might not be ideal. However, in our setting this is sufficient by assumption.

Closeness to Prior Policy
As we know that a sufficiently good policy will be close to the prior policy, staying close to the prior policy is a crucial component of local exploration. Exploring far away from the reference policy only poses additional risk with no possible pay-off. There are many possible notions of closeness to consider, e.g. KL-divergence of state distributions, expected KL-divergence of the policies on some state distribution, etc.

4.2.1 Formalizing These Requirements

Ideally we would like to have a relation between reference policy, exploration method, maximum return reduction and expected one step policy improvement. This would allow us to formally reason about different exploration methods in some simple scenarios. To achieve this we would have to find a way to reason about the expected amount samples required for a certain amount of improvement or the amount of improvement we can expect from a set of samples. Unfortunately, this proved difficult for multiple reasons.
Classical results which are able to establish regret bounds for tabular MDPs rely on the fact that each state can be uniquely identified and we can count visitations or other metrics of it. Often proofs use the pigeonhole principle [26] to argue that each state and action will be visited at some point, allowing to establish the regret bounds. However, as we are dealing with continuous action space, this argument does not work anymore unless we impose some structure on the state and action spaces. Without some structure assumption, we cannot say that states near each other in e.g. Euclidian distance have remotely similar properties and obviously exploring each state of an uncountable state space is not possible either. There is some literature which obtains results for different structure assumptions. However, none of these seemed to directly fit our use case. For example an often used assumption assumption are Lipschitz continuity [79, 78, 77] of

rewards and transitions. This is not directly suited to RoL as discontinuities are a common occurrence in the real world. For example, collisions cause the transition function to be discontinuous.
Aside from this we would need to find a way to relate the perturbations caused by the exploration method to the expected decrease in policy performance. This seems less desperate as there are bounds on the policy performance difference for a simple class of policies [56] and TRPO [21] generalizes it to stochastic policies. However, the bounds of TRPO are not applicable to our deterministic reference policy either as they depend on coupling to show that the different policies will act similarly some of the time. Unfortunately if we use a deterministic reference policy the chance that a stochastic policy with an absolutely continuous distribution will act exactly the same as the reference policy is 0.

4.3 Comparing Local Exploration Empirically

As formally reasoning about good local exploration proves difficult, we now switch to an empirical approach. In this section we want discuss metrics which we can use to compare different exploration methods in terms of how well they do local exploration.

4.3.1 Measuring The Amount Of Exploration

An important criterion for good local exploration was collecting the data in such a way that we can improve the policy as much as possible in one step. The natural way of evaluating this is to exactly collect a batch of samples and then seeing how much the policy can improve from this batch. Unfortunately we found this approach to suffer from too high variance within a reasonable sample budget and we were unable to draw any conclusion from it.
Another idea is to consider the entropy of the state distribution induced by the exploration policy. Intuitively, high entropy of the state distribution should mean that many different states are visited and therefore we should be able to improve the policy more compared to low entropy exploration methods. Some methods [61] explicitly maximize state distribution entropy to explore and show that some methods based on intrinsic reward maximize state distribution entropy. However, this approach approach lacks a theoretical foundation and high state distribution might not correlate with high policy improvement unless we make strong assumptions about the underlying MDP.
An additional problem of entropy as exploration metric is that we implicitly assume that each state is equally worth exploring since entropy does not weight states by their the exploration value. Consider this example [38, 37], where an environment contains a source of random noise such as a TV playing white noise. Employing exploration methods which try to maximize prediction uncertainty become attracted to the source of noise due to its inherent unpredictability. Similarly, some exploration methods may produce high state distribution entropy which may not necessarily be interesting behaviour.

4.3.2 Decrease In Return

We can measure the return of an exploration method by executing the policy and summing the return. Comparing it to the performance of the reference allows to get insight into how far away we explore from the reference policy.
Fortunately our focus on local exploration allows us to relate closeness to decrease in return in a simple way. By assumption in local exploration, it is not necessary to decrease reward in order to attain an optimal policy. Therefore we can directly conclude that an exploration policy which decreases the return has to act sub-optimally some of the time. How often the exploration policy acts sub-optimally correlates with the magnitude of the return decrease. Thus exploration methods which decrease return by the least amount are the most desirable for local exploration when focusing on this in isolation.
In the scope of this thesis, we will also assume that the reward signal of the environment rewards safe behaviour. For example in a robot task the reward signal should punish erratic movements and punish movement outside the desired zone. However, we need to be careful when making conclusions about the safety since only a part of the reward signal is attributed to safety.

4.3.3 Action Space Difference

While parameter noise and action space work at fundamentally different level, both eventually exert their influence over the behaviour through altering the action selection. Thus a natural way to compare the exploration methods would be the difference of actions selected by them compared to the reference policy. In local exploration for continuous control we require the difference of actions to be below some level since we assume the ideal behaviour is close to the reference policy by assumption, therefore this provides us with another way of measuring closeness to the reference policy.

Additionally this metric can act as point of comparison for different exploration methods with different parameters: Comparing the same amount of Gaussian noise applied to actions or parameters is meaningless but comparing parameter settings for Gaussian action/parameter noise which produce similar amounts of actions noise on the actions is meaningful. For a reference policy π_{ref} and exploration policy π_{ex}, we compute the difference between chosen actions

$$d(\pi_{ref}, \pi_{ex}) = E_{s\sim\nu}[d(\pi_{ref}(s), \pi_{ex}(a \mid s))]$$

in some metric $d : \mathcal{A} \times \mathcal{A} \mapsto \mathbf{R}$. We have multiple choices for d and the distribution of states ν used to evaluate the metric.
We choose the absolute difference $d(a, b) = |a - b|$ induced by the L_1 norm and the Euclidian distance $d(a, b) = (a - b)^2$ from L_2 and for the sake brevity refer to them by L_1 and L_2 action distances. L_1 weights all differences equally while L_2 weights large deviations differences more heavily. We expect difference to be higher in L_1 for the action noise based approaches since they are by design close to the reference policy, while the L_2 distance should be higher on parameter noise.
At a first glance it seems like choice of ν does not matter for the action space noises as we are just adding noise onto the deterministic policy independent of state. This is correct as long as the environment we are using does not clip the actions to be within a valid range. Our evaluation environment, Pendulum-v0, clips the actions, therefore our the amount of noise produced will be state dependent even for the pure action noises.
For the sake of comparison it would of course be beneficial to use the same ν to evaluate the distance for every noise. However, we have to consider the two possible sources of bias arising from a particular choice of ν. As previously pointed out, choosing ν such that the reference policy likely chooses actions close to either end of the action clipping range could reduce the effective amount of noise produced by the action space noises. Additionally, we have to consider that parameter noise might depend on ν in complex ways. For example it is imaginable that the parameters of the policy are more robust to small perturbations in states frequently visited during training compared to states not encountered during training. For now we turn a blind eye to these potential biases and choose ν to be the state distribution of the reference policy.
Overall low L_1 distance should signal low overall amounts of perturbation, resulting in lower jerkiness which is desirable for the safety of the hardware. Also L_1 distance means we are close to the reference policy on the fixed ν. However, running the policy even with a marginal difference in the action selection in a general MDP may result in arbitrarily large differences in states visited. Fortunately, the tasks we will use have transition function which are are mostly continuous w.r.t. to the actions, diminishing this possible source of error.

4.4 Exploration Methods We Evaluate

Due to the limited scope of this thesis we focus on only three exploration methods that represent the most common ways randomness for exploration is introduced in RL.

Gaussian Action Noise

This is the default choice for exploration in simulation and for this reason we want to investigate if it is still the first choice for exploration on real robotics. The lack of correlation in the Gaussian noise may be a problem for the application on a real system as the jerky movements produced by it might damage the robot.

Ornstein-Uhlenbeck (OU) Process

Noise that is correlated in time may not have the issue of jerkiness that we expect with the uncorrelated noise and therefore could be superior to Gaussian noise. We choose OU as a representative of the correlated noise as it is the most commonly used form of correlated action space noise [13].

Parameter Noise

We choose Gaussian parameter noise as it is an exploration method itself but also can be viewed as representative of some of the more sophisticated exploration methods in some aspects of comparison. We assume that more sophisticated exploration methods which perturb parameters, e.g. posterior based approaches or intrinsic motivation, behave similar to Gaussian parameter noise in terms of trajectories produced and other characteristics in the short term. This is still a crude approximation as these approaches perturb the parameters in a more sophisticated way even in one step.

5 Experimental Setup

We want to evaluate local exploration for Robot Learning, thus tasks with a real world system are of interest to us. However, the bulk of experiments will be performed in simulation due to their high sample requirements. We choose the following tasks:

5.1 Pendulum Swingup

The pendulum swingup is a classic task from control theory, where the goal is to swing up and balance an inverted pendulum. The agent can apply torque in either direction of the pendulum within a certain valid range. As the pendulum is underactuated, it is necessary to build up momentum in order to reach the top.

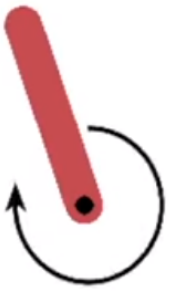


Figure 5.1: The Pendulum-v0 environment implemented in OpenAI Gym. The goal is to swing the underactuated pendulum up and balance it at the top position.

We use the implementation of OpenAI Gym [47], specifically the environment Pendulum-v0. The state of the pendulum is described by the tuple $(\theta, \dot{\theta})$, where θ is the angle between current pendulum position and the topmost position of the pendulum and $\dot{\theta} = \frac{d\theta}{dt}$ is the angle velocity of pendulum. The agent does not observe the tuple but a redundant observation of it: $\mathscr{S} = [-1,1] \times [-1,1] \times [-2,2]$ and for the pendulum with state $(\theta, \dot{\theta})$ the agent will observe $s \in \mathscr{S}$ with $s = (\cos\theta, \sin\theta, \dot{\theta})$.
The low dimensionality of the state space and the simplicity of the task allows us to gain some insights into how the exploration methods work. For this reason the majority of our empirical evaluations are performed on this task in simulation. We use a simulation rather than a real system due to the significantly faster sample collection. In the scenario we are interested in the sim2real gap is important. To emulate the gap we use a policy that is pre-trained with TD3 on the exact same simulation but not to an optimal degree. The optimal policy in simulation is able to achieve about -120 reward average reward while the policy we use achieves -200 reward. We do not execute any real world experiments on this task.

5.2 Furuta Pendulum

The Furuta Pendulum [48] also called rotational inverted pendulum has a similiar goal to the Pendulum Swingup.
However, the pendulum is attached to an arm in the horizontal plane. The agent can only control the pendulum indirectly by applying force to the horizontal arm. This makes the task highly non-linear and notably more complex than the standard Pendulum Swingup. The state of the pendulum is now a tuple $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2)$ consisting of the angles of the arms and their respective angle velocities. Again for observations the angles are represented by their sine and cosine resulting in a six-dimensional observation space.

Figure 5.2: Real world Furuta pendulum used in the experiments.

5.3 Robot Arm Control

As a final and most complex task we want to control a 7 joint robot arm by setting the goal positions and velocities of the robots default PD-controller. The goal of the task is to follow a prerecorded trajectory where the arm hits a drum multiple times in a row. We choose state space to consist of the angle and angle velocity of each of the joints. Additionally we include a number $t \in [0, 1]$ which indicates the running time of the episode. The action space is 14 dimensional, $\mathcal{A} = [-1, 1]^{14}$, with the first 7 dimensions indicating how to change the goal position and the remaining values affect the goal velocity of the default PD-Controller.
The simulation is implemented in SL [49] and the real system is the IIWA Kuka. We use a fixed control rate of 500 for simulation and real system. Initially we planned to use a neural network as controller but did not manage to obtain a working policy with TD3. Therefore we use a open-loop controller which at each time step t changes the goal position and velocity towards their respective value of the recorded trajectory, i.e. we update goal position x and goal velocity v of the robots PD controller by

$$[x, v]^T \leftarrow [x, v]^T + \text{clip}([x_t, v_t]^T - [x, v]^T, -\Delta, \Delta),$$

where Δ denotes the maximum change at one time step and clip clips the value elementwise. Unfortunately, this controller lacks an easy way of implementing parameter noise in a way that is comparable to parameter noise in a neural network.

5.4 Entropy Estimation

We want to measure the entropy [50] of the state distributions induced by different exploration methods. Entropy is a information theoretic quantity which quantifies the uncertainty of a random variable. Let X be a discrete random variable on $\mathcal{X}$ with probability mass function p. The entropy $H(X)$ of X is defined as

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x).$$

The analogue for a continuous random variable X with density f_X is called differential entropy and denoted by $h(X)$ defined by

$$h(X) = -\int_{\mathcal{X}} f_X(x) \log f_X(x) dx.$$

In our use case we only have access to the distributions through samples, therefore it is impossible to compute these quantities directly. However, we can still obtain an estimate of the entropy from the samples. An easy way of estimating differential entropy makes use of a histogram. We compute a histogram of the samples with equally sized buckets and then compute the discrete entropy of the density induced by the histogram. This approach is biased and suffers from the curse of dimensionality. As we increase the dimension of the distribution, we need exponentially more samples to populate each bucket. We tried this approach but obtained poor results.

An alternative is provided by non-parametric entropy estimators [51]. These methods estimate the density of the underlying distribution non-parametrically and use the estimate to obtain an entropy estimate. This usually scales better to high dimensions and might be less biased depending on the specific properties of the distribution.
The entropy estimator we use [67] uses the k-nearest neighbour distance to estimate the log probability density at each of the N samples. The log probability estimates of the samples are then averaged to obtain an entropy estimate. This is similar to estimators based on kernel density estimates but avoids the integration over all sample points to normalize the probabilities. Under the assumption that the probability density is constant within a norm ball encapsulating the k-nearest-neighbours of each sample, this yields a consistent estimator for large N. While for large N, this estimator should be unbiased, for small N bias may exist [52].
Special attention may be necessary if the data lies on a low-dimensional manifold [53]. For example if our state space $\mathscr{S}$ consists pixel of pixel of an image, this method of estimating entropy might fail. In our case the observations are mostly joint positions and velocities, which still may suffer from this issue. The underlying dynamical system restricts the states and most likely does not allow arbitrary combinations of the individual state components, therefore we cannot rule out that our state space is a low dimensional manifold. Keeping these potential pitfalls of the entropy estimation in mind, we validate the correctness of the entropy estimates with some experiments.
We use a open source implementation [53] of the nearest neighbour estimator. This implementation uses the max-norm for the nearest neighbour estimation.

6 Experiments

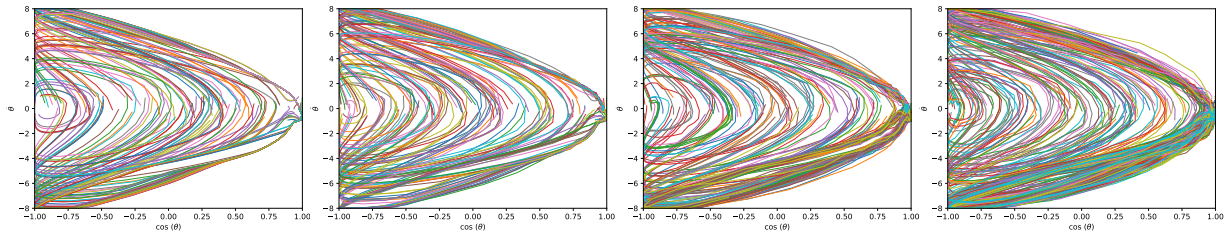
This section will present the results of the experiments with which we evaluate the exploration methods. We will refer to the reference policy as π_{ref} and to the exploration policies derived from π_{ref} by π_{gauss} for Gaussian action space noise, π_{param} for Gaussian parameter noise and π_{ou} for OU action noise. The parameter values of the exploration strategies are not included in this abbreviation and we just use these names as shorthands, e.g. instead of saying the exploration policy with Gaussian action space noise we simply say π_{gauss} for brevity.

6.1 Behaviour Of Exploration Methods

To get a first insight into how the different exploration methods actually explore, we inspect them by the trajectories they generate. Looking at the system itself or a rendering of it is time-intensive, therefore we visualize multiple trajectories in a single plot.
Our choice of environment for this purpose is Pendulum-v0 due its low-dimensional state space consisting of the angle θ and the angle velocity $\dot{\theta}$ of the pendulum. To be able to create a 2d plot, we ignore the time component and just visualize the states of a trajectory in a single line. Since the pendulum has some inertia and the dynamics of the environment are continuous, we can still get some insight into the temporal succession of the states.
To obtain the data, we use policy π_{ref} that is pretrained to act almost optimally. We combine π_{ref} with each exploration method to obtain the exploration policies π_{gauss} with gaussian action noise, π_{param} with parameter noise and π_{ou} with OU action noise. We record 100 trajectories for multiple parametrizations of each exploration policy.

Deterministic policy

Figure 6.1 shows the behaviour of Gaussian action space noise and the reference policy π_{ref} for $\sigma^2 = 0$. All trajectories of π_{ref} follow a similar pattern which we will briefly describe:
Each trajectory starts off at a random angle θ of the pendulum with angle velocity $\dot{\theta} = 0$. Then the pendulum accelerates due to gravity until it passes through the bottom position at $cos(\theta) = -1$ with varying amounts of speed. Afterwards the pendulum decreases in speed again due to gravity until a maximum point is reached. This repeats in growing intensity until the policy manages to swing the pendulum up and keep it at the balancing point $\cos(\theta) = 1, \dot{\theta} = 0$. The swing-up and balancing of π_{ref} visit similar states because the policy π_{ref} is deterministic and the environment Pendulum-v0 is mostly deterministic as well. The only source of randomness are the initial conditions of the pendulum which only influence the ideal swing up direction and the amount of cycles needed to swing up. We can observe the difference in ideal swing up direction from $\dot{\theta}$ as we have one cluster of trajectories with positive velocities and one with negative velocities.



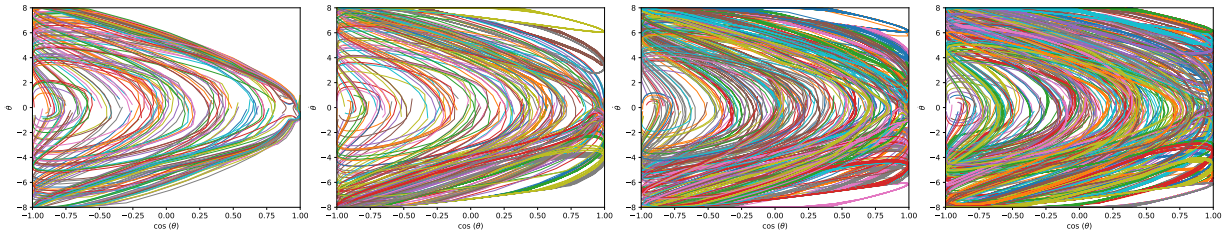
(a) $\pi_{ref}(\sigma^2 = 0.0)$ **(b)** $\sigma = 0.5$ **(c)** $\sigma = 1$ **(d)** $\sigma = 1.5$

Figure 6.1: Angles and velocities $\dot{\theta}$ of the Pendulum-v0 visited during 100 trajectories. This plot shows Gaussian action space noise with different standard deviations σ^2. The trajectories all follow a similar pattern similar to 6.1a with the addition of some spread. The spread of the trajectories increases as the amount of noise is increased.

Gaussian action noise

The trajectories of π_{gauss} have a pattern similar to π_{ref} with the addition of some local spread around the states visited by π_{ref}. No vastly different behaviour, such as spinning the pendulum in circles, emerges and all trajectories follow the usual swing up procedure. Considering local exploration, this is desirable as we make the assumption that the optimal behaviour is similar to π_{ref}.

Equally desirable is the seemingly smooth increase of the spread with increasing intensity of the noise. This is caused in part by the noise but also the nature of the environment. The Pendulum-v0 is a simple environment with - aside from action clipping - continuous dynamics. This means that bounded difference in action taken results in a bounded difference in next state. As we are using Gaussian noise, the difference in action taken is easily bounded with some probability depending on σ^2. This also explains why the spread increases with increased noise. However, the one-step state difference may still accumulate over multiple steps and then π_{gauss} could visit vastly different states. As this is not the case, we come up with two possible explanations.
First, the policy π_{ref} has learned to compensate Gaussian action space noise as we used Gaussian action space noise for exploration during training. Secondly, as the noise is uncorrelated, it might at each time step cancel out previous pertubations.

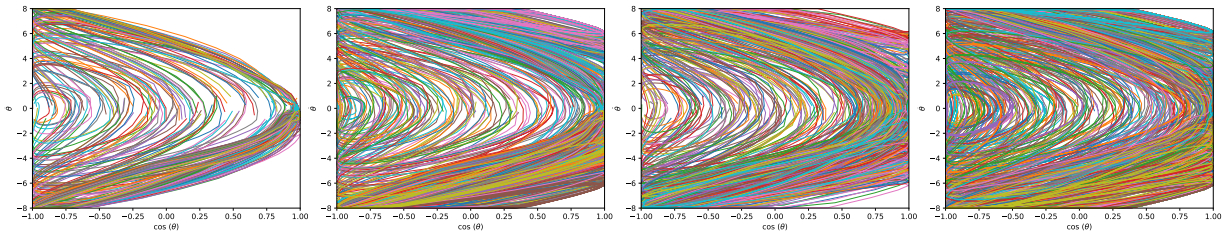


(a) $\sigma = 0.03$ **(b)** $\sigma = 0.09$ **(c)** $\sigma = 0.15$ **(d)** $\sigma = 0.21$

Figure 6.2: Angles and velocities $\dot{\theta}$ of the Pendulum-v0 visited during 100 trajectories. This plot shows Gaussian parameter space noise with different standard deviations σ^2.

Gaussian parameter noise

In figure 6.2 we see that parameter noise produces a different pattern of trajectories. The trajectories which follow the fall-swing-up-balance progression are further apart from each other when compared to Gaussian action noise. Also note that for $\sigma^2 >= 0.09$ the pendulum starts to spin around for some trajectories. The fraction of trajectories where the pendulum spins increases as the amount of parameter noise is increased. This can partially explained by the policy losing some of the robustness to perturbations due to the altered weights, but also the increasingly different behaviour the policy exerts due to weight altering.
Overall parameter noise seems to visit larger portion of the state space when compared to Gaussian noise. This might lead us to believe that the state distribution entropy of π_{param} should be higher than the one of π_{gauss}, but this will turn out to be false.



(a) $\sigma = 0.3$ $\theta = 0.1$ **(b)** $\sigma = 0.3$ $\theta = 0.01$ **(c)** $\sigma = 0.9$ $\theta = 0.1$ **(d)** $\sigma = 0.9$ $\theta = 0.01$

Figure 6.3: Angles and velocities $\dot{\theta}$ of the Pendulum-v0 visited during 100 trajectories. This plot shows OU action space noise with different standard deviations σ and correlations θ.

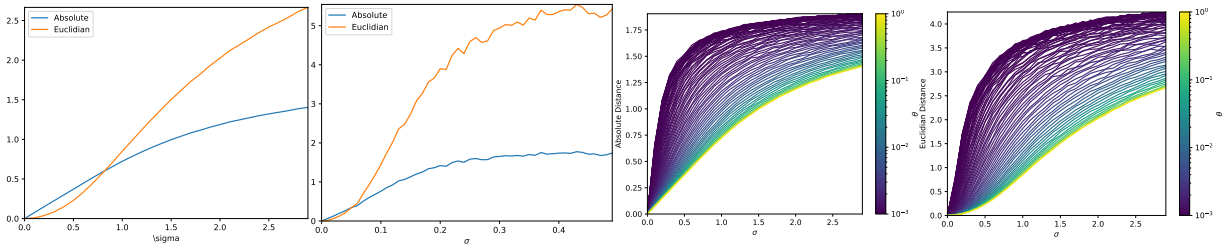
OU action noise

The behaviour of π_{ou} seems similar to π_{param}. Spinning takes place for all but the lowest noise setting. The most spinning occurs for $\theta = 0.01$ and $\sigma = 0.3$, a high amount of correlation and low amount of noise. With $\theta = 0.01$ and $\sigma = 0.9$ spinning occurs less frequently as the noise will likely exceed the action space which results in constantly choosing the same action, thus not managing to swing up. In turn we expect π_{ou} to have higher entropy than π_{param}.
π_{ou} seems to cover significantly more of the state space than π_{gauss} similar to π_{param}. The only notable difference from π_{param} is the increased spread within a trajectory. For π_{param} one trajectory mostly visits the same set of states resulting in distinct lines with some gaps in between, while for π_{ou} the states visited are slightly varied, therefore resulting in less states that are not visited at all.

6.2 Measuring Action The Space Difference

In the previous section we saw some unique characteristics of each exploration method and how they change depending on the exploration parameters. However, comparison between the exploration methods is not possible since we are missing a common reference point. While both Gaussian parameter and action noise are parameterized by their standard deviation, they act in completely different ways and thus are incomparable by their parameters.

To illustrate why this is an issue, consider the spinning behaviour we observed for the parameter noise. We attributed this to the decreased robustness of the policy and the increased correlation of the noise. However, we cannot be certain that this the reason. For example the raw amount action space noise induced by the parameter noise could be magnitudes Gaussian action space noise we tried. If this was the case the spinning behaviour might start to occur if we just turn the action space noise up high enough. If our exploration methods were comparable in some metric depending the action space noise, we could rule out this alternate explanation. Thus the first goal of this section is to find some metric in which we can compare the exploration methods.



(a) π_{gauss} **(b)** π_{param} **(c)** π_{ou}, L_1 **(d)** π_{ou}, L_2

Figure 6.4: Comparison of both action space distances for different exploration methods in dependence of the exploration parameters. π_{gauss} achieves a higher ratio between absolute and Euclidian distance compared to π_{gauss}. Note the logarithmic scale for the θ of π_{ou}. For low correlation at $\theta > 0.1$ of π_{ou} the behaviour of π_{ou} and π_{param} is similar.

Figure 6.4 shows an evaluation of the previously discussed action space distances on the Pendulum-v0 environment. We can see that the action noise generated by the different methods are in the same range due to the clipping from the environment.

We would expect the L_1 distance for π_{gauss} to be given by $\sqrt{\frac{2}{\pi}}\sigma$ and the L_2 distance to be the variance σ^2 if the environment did not clip actions. We can see that this is the case when σ and actions are clipped infrequently. As we increase the noise, actions become clipped more often leading to the drop in action space noise generated. For the other noises this causes similar drop offs.

For high values of θ we can see that OU reaches an absolute difference of 2 even for low σ. When θ is low, the correlation is high and the OU process will likely wander off in either direction without returning to the origin 0. Since the amount of noise sampled from the process will be so large that it overwrites the policy behaviour leading to π_{ou} choosing one action constantly.

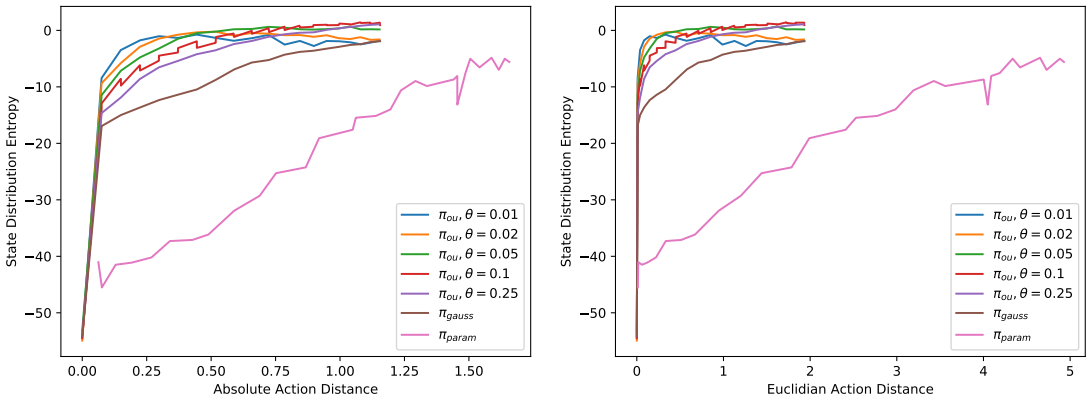
For the parameter noise we observe a ratio between L_1 and L_2 that is larger when compared to the other noises. The L_1 distance seems to cap out at 1 while the L_2 distance reaches values of 5. This can be explained by the parameter mostly choosing actions which are close (low in L_1 distance) to the reference policy while sometimes choosing actions that are completely different which are more heavily weighted by the L_2 distance. Later on we will see that this happens on a per episode basis where, depending on the particular parameter perturbation for the episode, the policy will act like the reference policy or just choose one action constantly resulting in the pendulum spinning.

6.3 Evaluation of the Exploration Methods

We have established a common ground for comparison in the action space distances introduced in the previous section. In addition to making the exploration methods more comparable, the action space distances allowed us to think about the action space exploration behaviour of the individual exploration strategies. we want to complement this by looking at the amount of exploration in the state space by measuring the state distribution entropy resulting from the exploration methods. We then compare the different exploration methods by their state space exploration, using the previously obtained action space distances as reference points. Afterwards we compare the decrease of return these exploration methods produce in a similar way since we defined this to be a crucial aspect of local exploration. As previously introduced, we use the entropy of the state distribution of the exploration policies as a proxy for the amount of exploration a exploration

strategy exerts. We estimate the entropy with the estimator discussed in **??** from a fixed amount of samples to keep bias consistent.

6.3.1 Pendulum-v0 Simulation



(a) Entropy vs. L_1 action distance

(b) Entropy vs. L_2 action distance

Figure 6.5: Estimated state distribution entropy from 25000 samples for different exploration methods compared by their action space distances. As expected applying noise to the actions increases exploration and therefore entropy. π_{ou} noise behaves similar to π_{gauss} with π_{ou} becoming more similar to π_{gauss} as we decrease correlation by letting θ go to 1. π_{gauss} and π_{ou} increase entropy immedeately while π_{param} steadily increases the entropy. The choice of x-axis only affects scaling and the overall pattern is similar.

In figure 6.5 we see that as expected entropy of the state distribution increases with more action space noise. However, the relation of these quantities depends on the exploration method at hand. We see that the action space noises come close to their maximum entropy for low amounts of action space distance. OU noise is able to create higher entropy than Gaussian noise through the correlation. We can also see that the entropy drops off for $\theta = 0.01$ and $\theta = 0.02$ at some point. This can again can be explained by the OU process wandering off in some direction of the action space and making π_{ou} select a fixed action. We also see that the behaviour of π_{ou} becomes more similar to π_{gauss} as the correlation decreases.

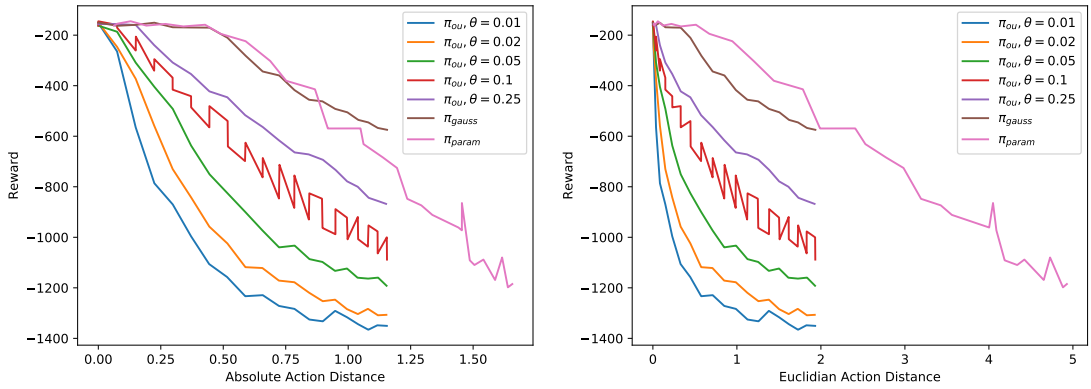
Parameter noise follows a completely different trade-off. The entropy for π_{param} increases more steadily but is overall at a lower ratio between entropy and action distance. The parameter noise is able to attain significantly higher L_2 distance than the action noise for similar amounts of L_1 distance. For this environment the the action space noises are superior to π_{param} with π_{ou} providing a slightly better trade-off than π_{gauss}. We have to keep in mind that this experiment was executed with strong policy which was trained with Gaussian noise, therefore the policy is resilient to the action space noise and thus return is impacted less severely.

In figure 6.6 we see a the downside that comes with the high entropy of π_{ou}. The high exploration of π_{ou} decreases the policy performance more than π_{gauss} and π_{param} which behave similarly. Surprisingly π_{gauss} and π_{param} do not decrease performance up to a certain amount of action distance. From this perspective π_{gauss} and π_{param} seem to be a better choice than π_{ou}.

Considering both return and entropy, π_{param} seems to be dominated by the other exploration methods. If the priority is high exploration, π_{ou} is the best choice. On the other hand if low decrease in return is important, π_{gauss} performs similar to π_{param} but provides more exploration.

The results so far have shown that increasing the amount of noise increases exploration but reduces the policy performance with π_{param} performing the worst in both of these aspects. Of course both of these aspects are important for local exploration, therefore we now want to examine if one exploration method gives us a better trade-off between exploration and decrease in return. We expect this to be either π_{ou} or π_{gauss} since π_{param} was dominated by π_{gauss} in terms of exploration amount.

Figure 6.7 visualizes this trade-off on two environments. All exploration policies share a common point at the bottom right, where the parameter are chosen in such a away that no exploration is happening. This point is also where π_{ref} is located in the diagram. As π_{ref} is deterministic it has low entropy but achieves the highest reward. Moving leftwards from the reference point, we see an increase in entropy which initially does not sacrifice reward for all exploration methods. Then reward decreases while entropy still increases up to a point where it will start to decrease again. The entropy starts to decrease for high amounts of noise again due to the clipping of the environment. For high intensity π_{ou}



(a) L_1 action distance **(b)** L_2 action distance

Figure 6.6: Return of the different exploration strategies. Increased noise degrades policy performance but π_{gauss} and π_{param}do not decrease return initially. π_{ou} decreases the performance the most of all exploration methods. The fact that π_{param} is able to produce more action space distance in both metrics is partly due the parameter range we tried.

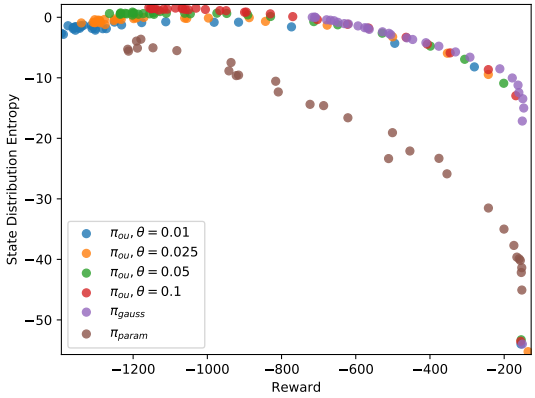


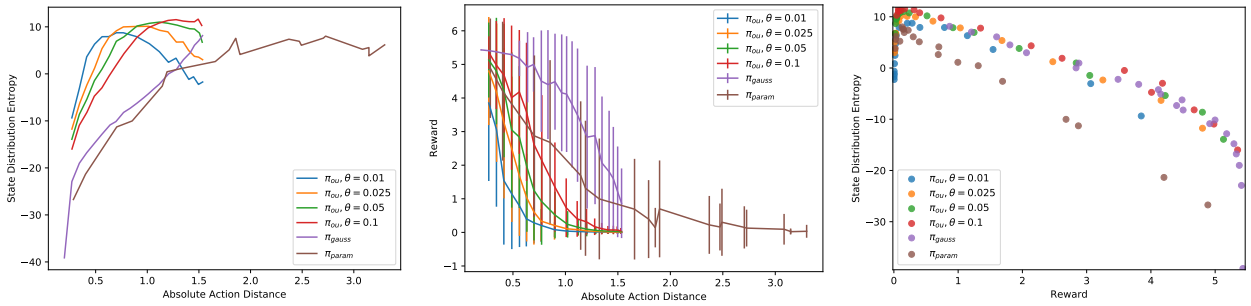
Figure 6.7: Trade-off between exploration and performance of exploration policy. The points show which trade-off between exploration and decrease in reward is attainable with a certain exploration strategy. The color of each point indicates which exploration strategy was used to obtain the point. The ideal exploration strategy would be located at the top right, where performance is optimal and exploration is high. The deterministic reference policy is located at the bottom right, where the performance is almost optimal but exploration is low.

and π_{param} action are consistently outside the valid range and therefore become clipped to the same value. This results in the pendulum not being to swing up anymore and thus a lower state distribution entropy.
We can see that surprisingly π_{ou} and π_{gauss} seem to lie on a similar curve. The similarity at the right end can be explained by OU behaving like uncorrelated noise.
As expected the trade-off curve for π_{param} lies below the others confirming the previous observation that π_{param} explores poorly in this instance. Gaussian action space noise or OU with low correlation are superior to π_{param}in our experimental setup. However, we cannot say this in general as our setup should highly favor action space noise.

6.3.2 Qube-v0

Optimal Policy in Sim

The first experiments in this section are run with a policy that achieves the almost optimal reward of 5.4 in simulation on average. Our goal is to see if parameter noise still outperformed by Gaussian noise in simulation and if similar trends are still visible on the real system.
Figure 6.8 shows that behaviour on the qube is slightly different. Entropy seems to peak at some point for π_{ou} while π_{gauss} and π_{param} follow a similar trend. Still, the π_{param} causes the reward to drop off more than π_{gauss}, therefore it is dominated by π_{gauss}. The trade-off plot confirms this with by all points of π_{param} being below the others. However, while π_{param} still performs worse on Qube-v0, it does so to a lesser degree when compared to the Pendulum. Possibly the



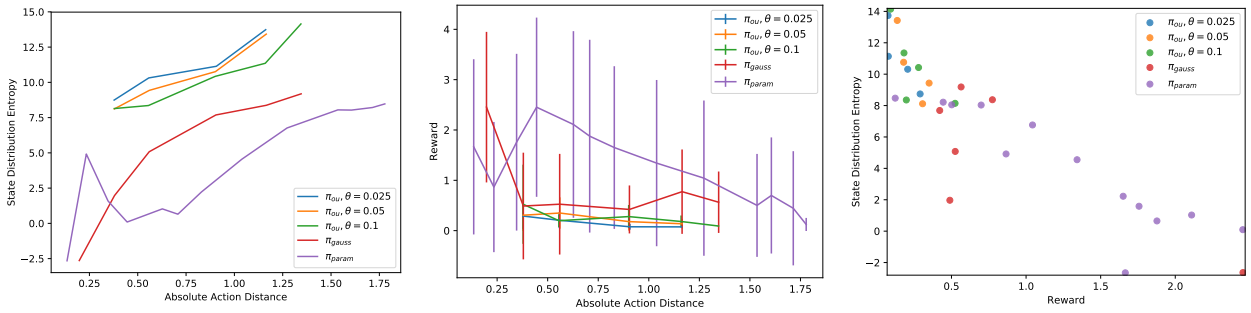
(a) Entropy **(b)** Return **(c)** Trade-off

Figure 6.8: Experiment results in simulation with the optimal policy.

increased complexity of the environment benefits the parameter noise. Perhaps on an even more complex environment this gap could close even further.

Sim2Real of Optimal Policy

Transitioning the ideal policy from simulation to the real world works surprisingly well if we use a high control rate on the real Qube. For a control rate of 500 and 250 Hertz we obtain optimal performance on the real robot even when the pre-trained policy was trained with 100 Hertz in simulation. If we use 100 Hertz on the real system, the performance notably degrades to about 2.5 on average but still manages to swing up and balance a fraction of times. Managing to swing up results in about 4 to 4.5 reward for the episode and shows us that task is solvable and the policy lacks reliability. Since the goal of local exploration is alleviating a sim2real gap, we choose 100 Hertz on the real system exactly for the worse performance and the existence of the sim2real gap which is not impossible to overcome.



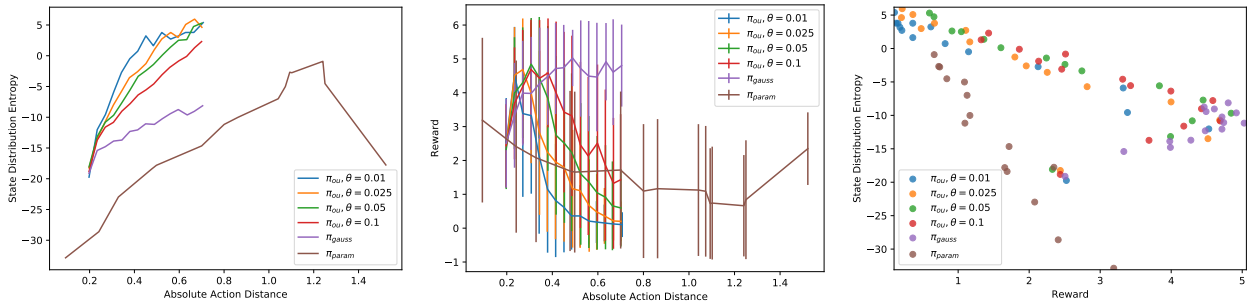
(a) Entropy **(b)** Return **(c)** Trade-off

Figure 6.9: Experiment results on the real Qube with a control rate of 100. The policy was optimal in simulation, achieving an average reward of 5.4. On the real system it achieves an average reward of 2.5.

In this scenario (Figure 6.9), the noises behave different from the simulation. Again entropy increases with noise intensity for all noises. Return is reduced by π_{gauss} and $\pi_{ou}$and reaches almost zero at a low level of action distance. On the other hand π_{param} increases even increases reward initially. The standard deviation of the π_{param} data is high, showing that the parameters noise sometimes helps the pendulum to swing up and in other instances does does nothing. In the trade-off plot no obvious pattern emerges which can probably be attributed to the high variance and the generally low policy performance.

Sub-optimal Policy in Sim

Lastly we want to see if different behaviour of the noises is a property of the real system or if it a result applying the noises to good or bad policies. For this we obtain a policy that achieves an average reward of 2.5 in simulation, therefore making comparable it terms of performance of the optimal policy transferred to the real system.
In Figure 6.10 we can see that the noises exhibit behaviour which is unlike the previous two situations. All the noises increase entropy but surprisingly the action noises also increase reward up to some point. As the standard deviation in the retunr is again high, this can probably be explained by the policy sometimes managing to swing up and getting good reward while failing to get any reward at other times. Interestingly, only the action noise increases reward while the paramter noise directly decreases reward. This is opposite from the behaviour on the robot and is probably a result of the policies not being the same even when they achieve the same reward on average.

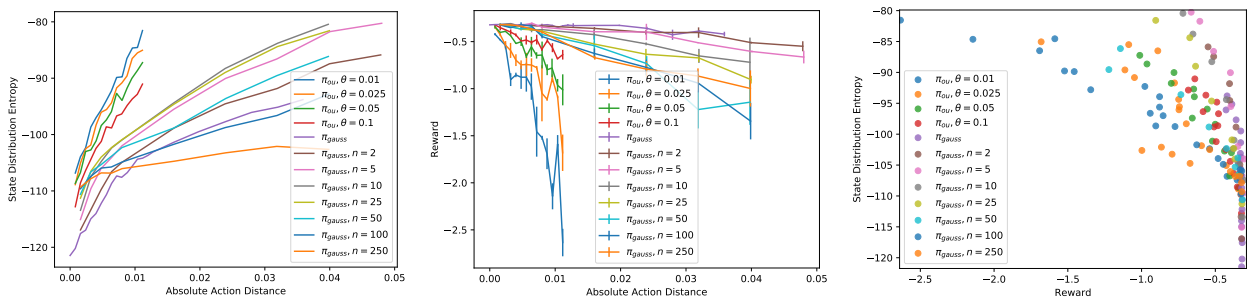


(a) Entropy **(b)** Return **(c)** Trade-off

Figure 6.10: Experiment results in Qube-v0 simulation with policy that performs similar to the sim2real of the optimal policy.

Also note that the sim2real and the simulation experiments this section used poor performing policies. The question if local exploration can be applied in this scenario is warranted. Unfortunately, we cannot answer this question decisively either as we do not have a rigorous definition of local exploration.

6.3.3 IIWA



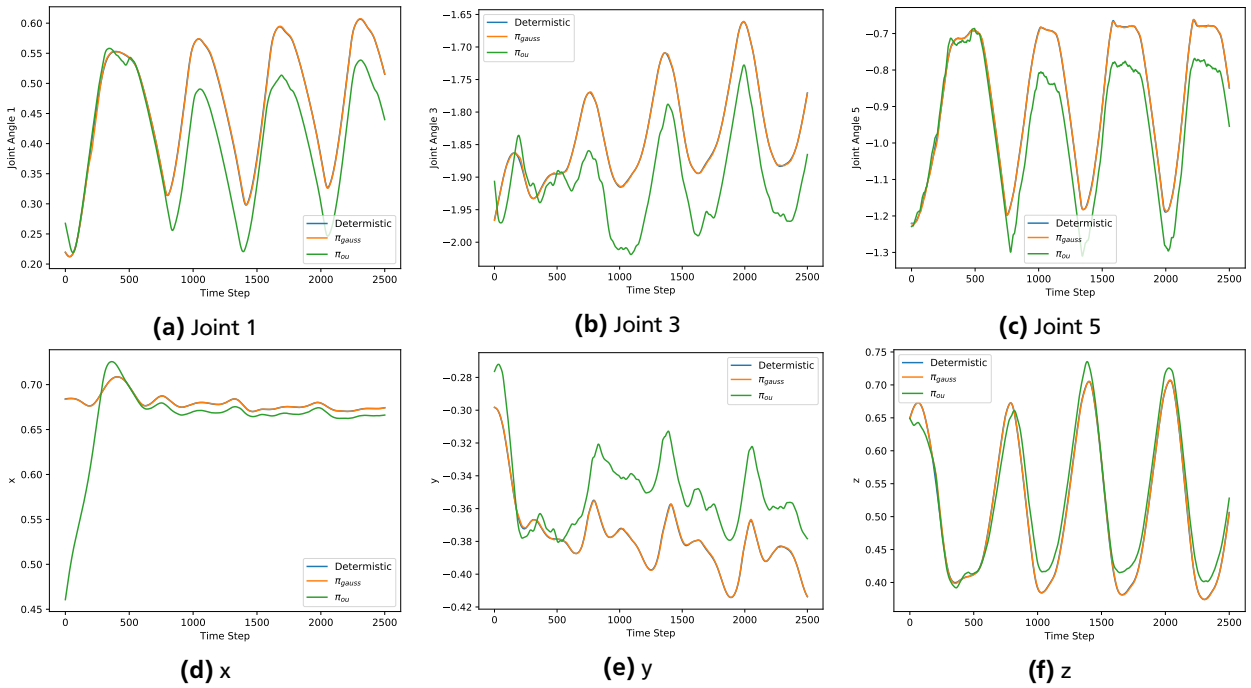
(a) Entropy **(b)** Return **(c)** Trade-off

Figure 6.11: Experiment results in simulation with policy that performs similar to the sim2real of the optimal policy.

The jerkiness of the noise poses a safety risk for the robot and thus we were only able to try far lower intensity noise on the real system. The maximum safe amount of noise was judged by an expert based on the perceived jerkiness. As the jerkiness of π_{gauss} and π_{ou} depends on only σ in both cases, we want to compare which noise gives us the most exploration for a similar value of σ.
In figure 6.12 we see trajectories produced by the different exploration methods. The trajectory produced by π_{gauss} is similar to the one from the deterministic policy, meaning that π_{gauss} barely explores. On the other hand, the π_{ou} trajectory differs from the deterministic trajectory and therefore is the better choice for exploration. The poor performance of π_{gauss} can in part be attributed to the high control rate. With a high control rate subsequent perturbations are likely to cancel each other out, reducing the overall impact these perturbations have. Our policy is able to compensate these small perturbations in my instances. Nevertheless, this does not reduce jerkiness as we are still possibly applying noise in the completely opposite direction from each time step to the next. While the mean of OU is still 0, for one realization of the process noise only drafts back to 0 and therefore is is less likely cancel out previous perturbations which makes it harder to compensate for our policy.
In the plots we do not see the jerkiness of the Gaussian noise and it seems like OU noise is jerkier. However, this is the actual exploration that we can see and the jerkiness is not visible at all in the plots. Looking at torques applied to each joint reveals that Gaussian and OU exhibit similar levels of jerkiness.

6.4 Validation of Entropy Estimation

The result that π_{param} produces less entropy than other methods seems counter-intuitive considering the trajectory plots where π_{param} seemed to visit the whole state space. In this section we address the possible issues regarding the entropy estimation.



(a) Joint 1 **(b)** Joint 3 **(c)** Joint 5

(d) x **(e)** y **(f)** z

Figure 6.12: Angles for the main joints in the movement and endeffector position for representative trajectories produced by the deterministic policy, π_{gauss} and π_{ou}. There is no discernible difference between the deterministic trajectory and the trajectory of π_{gauss}.

Bias of the estimate

The first issue was the possible bias of the entropy due to the structure of the state space and the finite amount of samples used. To see how far this biases the estimate, we estimate the entropy for different amounts of samples multiple times each and see how much the average entropy estimate depends on the amount of samples used.

We see in figure 6.13a that the amount of samples used for the estimation influences the entropy estimate. The entropy estimate decreases as we increase the amount of samples. Unfortunately - at least in the range of samples that we tried out - there does not seem to be a value to which the estimate converges. But it seems unlikely that the amount of samples used is not enough to cover the whole state space, therefore the bias might be caused by the structure of the state space.

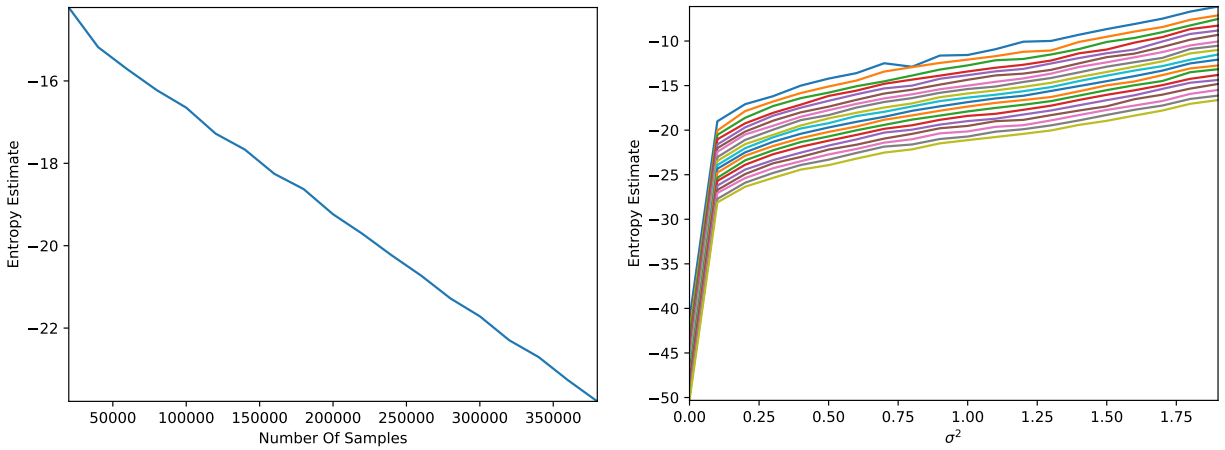
Fortunately, we are not interested in the entropy value itself but rather the entropy values of different exploration strategies compared to each other. As long as the bias only depends on the amounts of samples and the general structure of the state space, all values should have the same fixed bias and therefore be comparable. We see that this is the case in figure 6.13b where the entropy is estimated from different amounts of samples and samples obtained with different noise settings. All estimates have the same shape and are shifted by their respective biases along the y-axis. The order of the lines is preserved for all different datasets, therefore we can conclude the bias of the estimate must only depend on the amount of samples used for the estimate.

Are we actually estimating entropy?

While we have observed the entropy estimate is only biased to some degree which we can avoid by taking a look at the difference between estimates, it is still questionable if a difference in entropy actually corresponds to meaningful difference in exploration behaviour.

The result of the previous section tells us Gaussian action noise explores more than Gaussian action parameter space noise. This seems counter-intuitive when considering the trajectory plot (figure 6.2) where parameter noise seemed to visit a larger portion of the state space. The cause of this confusion is the trajectory plot which turns out to be misleading. The trajectory plot shows which states were visited at least once but only gives limited insight into how often the state were visited. A look at the histogram of the state distribution provides us with a better insight into the entropy we can expect.

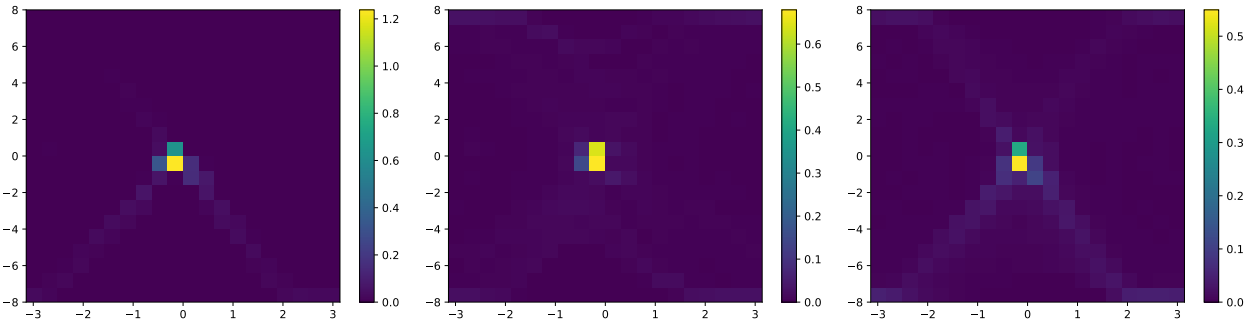
The histograms in figure 6.14 reveal that while parameter noise visited many different states, it did not visit them consistently. Depending on the particular parameter perturbation, π_{param} still seems to be able to still act optimally most of the time and otherwise just spin around at the highest possible speed, which we can barley see at the top and bottom of the histogram. There is some variation in the acceleration up to full speed, π_{param} seem to have higher entropy in the



(a) Entropy estimate depending on amount of samples used for the estimation on fixed data.

(b) Entropy estimates evaluated from different sets of samples with varying amount of samples.

Figure 6.13: In a we see that using a low amount samples increases the entropy estimate. In (b) we estimate the entropy from samples from π_{gauss} with different σ indicated on the x-axis. Each line corresponds to entropy estimation resulting from $25000n, 0 < n < 20$ samples subsampled from each of the datasets. The topmost line is corresponds to the highest entropy estimatation, therefore the lowest amount of samples as we saw in (a).



(a) π_{gauss}, $\sigma = 1.$

(b) π_{param}, $\sigma = 0.15$

(c) π_{ou}, $\sigma = 0.5, \theta = 0.03$

Figure 6.14: State distribution histograms of different exploration policies. The point in the middle with high density is the balancing point of the pendulum. For the parameter noise π_{param} almost all the density is concentrated in the middle, while for Gaussian π_{gauss} and OU π_{ou} action noise we can see some density across an X pattern.

trajectory plot. Since the acceleration is only takes only a few time steps and the balancing and spinning are low entropy behaviours, it makes sense that π_{param} has low entropy compared the action noise.
This complements the observation we made about the entropy of the parameter noise. We reasoned that the entropy was low because the parameter noise mostly stays close to the reference policy but sometimes acts completely different only by spinning at a constant speed. However, since there was some variation during the speed-up part, these behaviour are inherently low entropy, therefore the entropy of the induced state distribution was low.

6.5 Conclusion

We have analyzed three exploration methods in simulation and in the real world. Gaussian noise seemed to perform the best on tasks where the reference policy is almost optimal, whereas parameter noise seemed to perform better with worse policies. Still, measuring entropy and return and looking at their trade-off unfortunately did only provide limited insight. These metrics are highly task dependent which ultimately defeats their purpose of reasoning about good exploration in a more general setting. Furthermore, entropy does not necessarily correlate with policy improvement, therefore it is unclear if high entropy exploration actually translates to high policy improvement. The entropy estimate itself is biased to a large degree and only meaningful when compared to other values which are collected in the exact same circumstances. Also the estimate depends on time horizon we choose for an episode, e.g. if one exploration method induces great variation in the first n steps but does the same afterwards, its entropy might be low in spite of actually exploring. The good performance of Gaussian can be attributed due to the policies being trained with Gaussian noise and thus our

comparison for many of the experiments was not the fairest. All in all, reasoning about good local exploration in the way we did remains unconclusive and does not translate well between simulation and real system.
The inadequacy of these metrics can be seen in our own results. When testing the noise on IIWA task, we were not guided by the entropy or the trade-off which the different explorations offer in simulation. Instead, we determined a maximum amount of jerkiness and observed empirically that OU noise gives more exploration for the same jerkiness than Gaussian noise. However, there is no generality in this result and it might only apply to the IIWA task. We were not able to try out parameter noise on the IIWA but this probably would have been too risky anyway due to parameter noises' tendency to completely change the behaviour of the policy.
Future work could try to find assumptions which rigorously define local exploration in a way that is applicable to deep RL for robot learning. Possibly this would allow us to more generally reason about which exploration methods perform good local exploration.

Bibliography

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," vol. 529, pp. 484–503.

[3] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," vol. 32, pp. 1238–1274.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,"

[6] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-End Training of Deep Visuomotor Policies,"

[7] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection,"

[8] D. Morrison, P. Corke, and J. Leitner, "Closing the Loop for Robotic Grasping: A Real-time, Generative Grasp Synthesis Approach,"

[9] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience,"

[10] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World,"

[11] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros, "Large-Scale Study of Curiosity-Driven Learning,"

[12] I. Osband, J. Aslanides, and A. Cassirer, "Randomized Prior Functions for Deep Reinforcement Learning," in *Advances in Neural Information Processing Systems 31* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), pp. 8617–8629, Curran Associates, Inc.

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

[14] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter Space Noise for Exploration,"

[15] M. L. Puterman, *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

[16] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," vol. 8, no. 3-4, pp. 229–256.

[17] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms,"

[18] H. V. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems 23* (J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, eds.), pp. 2613–2621.

[19] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods,"

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms,"

[21] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization,"

[22] G. E. Uhlenbeck and L. S. Ornstein, “On the Theory of the Brownian Motion,” vol. 36, no. 5, pp. 823–841.

[23] R. Durrett, *Probability: Theory and Examples*, vol. 49. Cambridge university press.

[24] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg, “Noisy Networks for Exploration,”

[25] S. Fujimoto, D. Meger, and D. Precup, “Off-Policy Deep Reinforcement Learning without Exploration,”

[26] S. Kakade, “On the Sample Complexity of Reinforcement Learning,”

[27] C. Szepesvári, “Algorithms for reinforcement learning,” vol. 4, no. 1, pp. 1–103.

[28] I. Osband, B. Van Roy, and Z. Wen, “Generalization and Exploration via Randomized Value Functions,”

[29] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, “Deep Exploration via Bootstrapped DQN,”

[30] T. Lai and H. Robbins, “Asymptotically efficient adaptive allocation rules,” vol. 6, no. 1, pp. 4–22.

[31] M. Kearns, “Near-Optimal Reinforcement Learning in Polynomial Time,” p. 24.

[32] R. I. Brafman and M. Tennenholtz, “R-max - a General Polynomial Time Algorithm for Near-optimal Reinforcement Learning,” vol. 3, pp. 213–231.

[33] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, “Unifying Count-Based Exploration and Intrinsic Motivation,”

[34] H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, “#Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning,”

[35] G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos, “Count-Based Exploration with Neural Density Models,”

[36] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, “VIME: Variational Information Maximizing Exploration,”

[37] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, “Curiosity-driven Exploration by Self-supervised Prediction,”

[38] J. Schmidhuber, “Formal Theory of Creativity, Fun, and Intrinsic Motivation (1990–2010),” vol. 2, no. 3, pp. 230–247.

[39] B. C. Stadie, S. Levine, and P. Abbeel, “Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models,”

[40] R. M. Ryan and E. L. Deci, “Self-Determination Theory and the Facilitation of Intrinsic Motivation, Social Development, and Well-Being,” p. 11.

[41] A. L. Strehl, L. Li, and M. L. Littman, “Reinforcement Learning in Finite MDPs: PAC Analysis,” p. 32.

[42] D. J. Russo, B. Van Roy, A. Kazerouni, I. Osband, and Z. Wen, “A Tutorial on Thompson Sampling,” vol. 11, no. 1, pp. 1–96.

[43] Y. Gal and Z. Ghahramani, “Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning,”

[44] Y. Gal, R. T. McAllister, and C. E. Rasmussen, “Improving PILCO with Bayesian Neural Network Dynamics Models,” p. 7.

[45] I. Osband, “Risk versus Uncertainty in Deep Learning : Bayes , Bootstrap and the Dangers of Dropout,”

[46] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, “Parameter Space Noise for Exploration,”

[47] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI Gym,”

[48] B. S. Cazzolato and Z. Prime, “On the Dynamics of the Furuta Pendulum.”

[49] J. Williams, “The SL simulation and real-time control software package | Autonomous Motion - Max Planck Institute for Intelligent Systems.”

[50] T. M. Cover and J. A. Thomas, *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience.

[51] J. Beirlant, E. J. Dudewicz, and L. Gyorfi, “Nonparametric entropy estimation: An overview,” p. 14.

[52] J. D. Victor, “Approaches to Information-Theoretic Analysis of Neural Activity,” vol. 1, no. 3, pp. 302–316.

[53] G. V. Steeg, “Gregversteeg/NPEET.”

[54] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, “Evolution Strategies as a Scalable Alternative to Reinforcement Learning,”

[55] E. Conti, V. Madhavan, F. Petroski Such, J. Lehman, K. Stanley, and J. Clune, “Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents,” in *Advances in Neural Information Processing Systems 31* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), pp. 5027–5038, Curran Associates, Inc.

[56] S. Kakade and J. Langford, “Approximately Optimal Approximate Reinforcement Learning,” in *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML ’02, pp. 267–274, Morgan Kaufmann Publishers Inc.

[57] J. Achiam, D. Held, A. Tamar, and P. Abbeel, “Constrained Policy Optimization,”

[58] M. Bouton, A. Nakhaei, K. Fujimura, and M. J. Kochenderfer, “Safe Reinforcement Learning with Scene Decomposition for Navigating Complex Urban Environments,”

[59] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh, “A Lyapunov-based Approach to Safe Reinforcement Learning,” p. 10.

[60] J. García, Fern, and O. Fernández, “A Comprehensive Survey on Safe Reinforcement Learning,” vol. 16, no. 42, pp. 1437–1480.

[61] L. Lee, B. Eysenbach, E. Parisotto, E. Xing, S. Levine, and R. Salakhutdinov, “Efficient Exploration via State Marginal Matching,”

[62] S. Lange, T. Gabel, and M. Riedmiller, “Batch Reinforcement Learning,” in *Reinforcement Learning* (M. Wiering and M. van Otterlo, eds.), vol. 12, pp. 45–73, Springer Berlin Heidelberg.

[63] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, “The Arcade Learning Environment: An Evaluation Platform for General Agents,” vol. 47, pp. 253–279.

[64] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller, “Maximum a Posteriori Policy Optimisation,”

[65] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor,”

[66] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, “Geometric deep learning: Going beyond Euclidean data,” vol. 34, no. 4, pp. 18–42.

[67] A. Kraskov, H. Stögbauer, and P. Grassberger, “Estimating mutual information,” vol. 69, no. 6, p. 066138.

[68] T. M. Cover and J. A. Thomas, “ELEMENTS OF INFORMATION THEORY,” p. 774.

[69] T. M. Cover and J. A. Thomas, “ELEMENTS OF INFORMATION THEORY,” p. 774.

[70] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” vol. 518, no. 7540, pp. 529–533.

[71] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing Atari with Deep Reinforcement Learning,” p. 9.

[72] "[1810.04805] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding."

[73] J. Kober and J. R. Peters, "Policy search for motor primitives in robotics," in *Advances in Neural Information Processing Systems*, pp. 849–856.

[74] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2219–2225, IEEE.

[75] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-End Training of Deep Visuomotor Policies,"

[76] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine, "One-Shot Visual Imitation Learning via Meta-Learning,"

[77] J. Ok, A. Proutiere, and D. Tranos, "Exploration in Structured Reinforcement Learning," p. 9.

[78] K. Asadi, D. Misra, and M. L. Littman, "Lipschitz Continuity in Model-based Reinforcement Learning,"

[79] M. Pirotta, M. Restelli, and L. Bascetta, "Policy gradient in Lipschitz Markov Decision Processes," vol. 100, no. 2, pp. 255–283.

[80] "Silver deterministic policy gradient - Google Search."

[81] J. Ok, A. Proutiere, and D. Tranos, "Exploration in Structured Reinforcement Learning,"

[82] R. Ortner and D. Ryabko, "Online Regret Bounds for Undiscounted Continuous Reinforcement Learning,"

[83] R. Dubey, P. Agrawal, D. Pathak, T. L. Griffiths, and A. A. Efros, "Investigating Human Priors for Playing Video Games,"

[84] I. Osband, J. Aslanides, and A. Cassirer, "Randomized Prior Functions for Deep Reinforcement Learning,"

[85] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros, "Large-Scale Study of Curiosity-Driven Learning,"

[86] B. O'Donoghue, I. Osband, R. Munos, and V. Mnih, "The Uncertainty Bellman Equation and Exploration,"

[87] C. Riquelme, G. Tucker, and J. Snoek, "Deep Bayesian Bandits Showdown: An Empirical Comparison of Bayesian Deep Networks for Thompson Sampling,"

[88] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," p. 30.

[89] B. Efron, *The Jackknife, the Bootstrap and Other Resampling Plans*. CBMS-NSF Regional Conference Series in Applied Mathematics, Society for Industrial and Applied Mathematics.

[90] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," vol. 518, p. 529.

[91] J. Z. Kolter and A. Y. Ng, "Regularization and Feature Selection in Least-squares Temporal Difference Learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 521–528, ACM.

[92] J. Z. Kolter and A. Y. Ng, "Near-Bayesian Exploration in Polynomial Time," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 513–520, ACM.

[93] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic Movement Primitives," in *Advances in Neural Information Processing Systems 26* (C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds.), pp. 2616–2624, Curran Associates, Inc.

[94] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," vol. 57, no. 5, pp. 469–483.