Understanding the Evolution of Linear Regions in Deep Reinforcement Learning

Setareh Cohen
Department of Computer Science
University of British Columbia
Vancouver, BC, Canada
setarehc@cs.ubc.ca

Nam Hee Kim
Department of Computer Science
Aalto University
Espoo, Finland
namhee.kim@aalto.fi

David Rolnick
School of Computer Science
McGill University
Montreal, QC, Canada
drolnick@cs.mcgill.ca

Michiel van de Panne
Department of Computer Science
University of British Columbia
Vancouver, BC, Canada
van@cs.ubc.ca

Abstract

Policies produced by deep reinforcement learning are typically characterised by their learning curves, but they remain poorly understood in many other respects. ReLU-based policies result in a partitioning of the input space into piecewise linear regions. We seek to understand how observed region counts and their densities evolve during deep reinforcement learning using empirical results that span a range of continuous control tasks and policy network dimensions. Intuitively, we may expect that during training, the region density increases in the areas that are frequently visited by the policy, thereby affording fine-grained control. We use recent theoretical and empirical results for the linear regions induced by neural networks in supervised learning settings for grounding and comparison of our results. Empirically, we find that the region density increases only moderately throughout training, as measured along fixed trajectories coming from the final policy. However, the trajectories themselves also increase in length during training, and thus the region densities decrease as seen from the perspective of the current trajectory. Our findings suggest that the complexity of deep reinforcement learning policies does not principally emerge from a significant growth in the complexity of functions observed on-and-around trajectories of the policy. We further provide other empirically-driven observations regarding the number of visited regions with respect to the number of neurons, changing trajectory lengths during training, and the impact of network depth on region density.

1 Introduction

Deep reinforcement learning (RL) utilizes neural networks to represent the policy and to train this network to optimize an objective, typically the expected value of time-discounted future rewards. Deep RL algorithms have been successfully applied to diverse applications including robotics, challenging games, and an increasing number of real-world decision-and-control problems [François-Lavet et al., 2018]. For a given choice of task, RL algorithm, and policy network configuration, the performance is commonly characterised via the learning curves, which provide insight into the learning efficiency and the final performance. However, little has been done to understand the detailed structure of the state-to-action mappings induced by the control policies and how these evolve over time.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

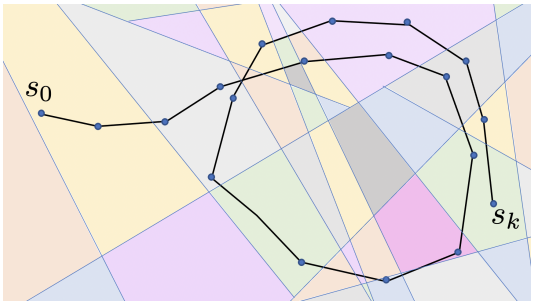


Figure 1: Schematic illustration of a trajectory traversing the piecewise linear regions in the policy state space. S_0 and S_k indicate the initial and final states of the trajectory.

In this work, we aim to further understand deep feed-forward neural network policies that use rectified linear activation functions (rectifier linear units or ReLUs). ReLUs [Nair and Hinton, 2010] are among the most popular choices of activation functions due to their practical successes [Montúfar et al., 2014]. For RL, these activations induce a piecewise linear mapping from states to actions, where the input space, i.e, the state space, is divided into distinct *linear regions*, where within each region, the actions are a linear function of the states. Note that the regions are formally an affine function of the input, due to the constant-valued bias terms. For simplicity and convenience, these are more commonly described as linear regions. Figure 1 provides a schematic illustration of these regions, along with a policy trajectory.

The number of distinct regions into which the input space is divided is a natural measure of network expressivity. Learned functions with many linear regions have the capacity to build complex and flexible decision boundaries. Thus, the problem of counting the number of linear regions has been extensively studied in recent literature [Montúfar et al., 2014, Raghu et al., 2017, Hanin and Rolnick, 2019a, Serra et al., 2018]. While the maximum number of regions is exponential with respect to network depth [Montúfar et al., 2014], recent work has demonstrated that the number of regions is instead typically proportional to the number neurons [Hanin and Rolnick, 2019a].

For RL, we are interested in the local granularity (density) of linear regions along trajectories arising from the policy. Fine-grained regions afford fine-grained control, and thus we may hypothesize that region density *increases* in regions frequently visited by the policy, in order to afford better control. Recent work in supervised-learning of image classification is inconsistent with regard to findings about the region density seen in the vicinity of data points, with some reporting a decrease [Novak et al., 2018] to provide better generalization and robustness to perturbation, and others not [Hanin and Rolnick, 2019a]. For the RL setting, we note that counting regions visited along an episode trajectory arguably provides a meaningful and task-grounded measurement in contrast to line-segments and ellipses that pass through randomly sampled points sampled from training data, which have been used in the prior works mentioned above. We further note that piecewise-affine control strategies are commonly designed into control systems, e.g., via gain scheduling. Understanding how these regions are designed and distributed by deep RL thus helps establish bridges with these existing methods.

To the best of our knowledge, our work is the first to investigate the structure and evolution of linear regions of ReLU-based deep RL policies in detail. We seek to answer several basic empirical questions :

Q1 Do findings for network expressivity, originally developed in supervised learning settings, apply to RL policies? How are the region densities affected by the policy network configuration? Do deeper policy networks result in finer-grained regions and hence an increased expressivity?

Q2 How do the linear regions of a policy evolve during training? Do we see a significantly greater density of regions emerge along the areas of the state space frequently visited by the episodic trajectories, thereby allowing for finer-grained control? Do random-action trajectories see different densities?

The key results can be summarized as follows, for policies trained using proximal policy optimization (PPO) [Schulman et al., 2017], and evaluated on four different continuous control tasks. Q1: There

is a general alignment with recent theoretical and empirical results for supervised learning settings. Region density is principally proportional to the number of neurons, with an additional small observed increase in density for deeper networks. Q2: Only a moderate increase of density is observed during training, as measured along fixed final-policy trajectories. Therefore, the complexity of a final learned policy does *not* come principally from increased density on-and-around the optimal trajectories, which is a potentially surprising result. In contrast, as measured along the evolving current-policy trajectories, a decrease in region density is observed during training. Across all settings, we also observe that the region-transition count, as observed during fixed time-duration episodes, grows during training before converging to a plateau. However, the trajectory length, as measured in the input space, also grows towards a plateau, although not at the same rate, and this leads to variations in the mean region densities as observed along current trajectories during training.

2 Related Work

Understanding the expressivity of a neural network is fundamental to better understanding its operation. Several works study exprressivity of deep neural networks with piecewise linear activations by counting their linear regions [Arora et al., 2016, Bianchini and Scarselli, 2014]. On the theoretical side, Pascanu et al. [2013] show that in the asymptotic limit of many hidden layers, deep ReLU networks are capable of separating their input space into exponentially more linear regions compared with shallow networks, despite using the same number of computational units. Following this, Montúfar et al. [2014] also explore the complexity of functions computable by deep feed-forward neural networks with ReLU and maxout activations, and provide tighter upper and lower bounds for the maximal number of linear regions. They show that the number of linear regions is polynomial in the network width and exponential with respect to the network depth. Furthermore, Raghu et al. [2017] improve the upper bound for ReLU networks by introducing a new set of expressivity measures and showing that expressivity grows exponentially with network depth for ReLU and tanh activated neural networks. Serra et al. [2018] generalizes these results by providing even tighter upper and lower bounds for the number of regions for ReLU networks and show that the maximal number of regions grows exponentially with depth when input dimension is sufficiently large.

More recent works that touch on expressivity of depth show that the effect of depth on the expressivity of neural networks is likely far below that of the theoretical maximum proposed by prior literature. Hanin and Rolnick [2019b] study the importance of depth on expressivity of neural networks in practice. They show that the average number of linear regions for ReLU networks at initialization is bounded by the number of neurons raised to the input dimension, and is independent of network depth. They also empirically show that this bound remains tight during training. Similarly, Hanin and Rolnick [2019a] find that the average distance to the boundary of the linear regions depends only on the number of neurons and not on the network depth – both at initialization and during training for supervised-learning tasks on ReLU networks. This strongly suggests that deeper networks do not necessarily learn more complex functions in comparison to shallow networks. Prior to this, a number of works have shown that the strength of deep learning may arise in part from a good match between deep architectures and current training procedures [Mhaskar and Poggio, 2016, Mhaskar et al., 2016, Zhang et al., 2021]. Notably, Ba and Caruana [2014] show that, once deep networks are trained to perform a task successfully, their behavior can often be replicated by shallow networks, suggesting that the advantages of depth may be linked to easier learning.

Another line of work studies function complexity in terms of robustness to perturbations to the input. Sokolić et al. [2017] theoretically studies the input-output Jacobian, which is a measure of robustness and also relates to generalization. Similarly, Zahavy et al. [2016b] propose a sensitivity measure in terms of adversarial robustness, and provides theoretical and experimental insights on how it relates to generalization. Novak et al. [2018] also study robustness and sensitivity using the input-output Jacobian and number of transitions along trajectories in the input space as measures of robustness. They show that neural networks trained for image classification tasks are more robust to input perturbations in the vicinity of the training data manifold, due to training points lying in regions of lower density. Several other recent works have also focused on proposing tight generalization bounds for neural networks [Bartlett et al., 2017, Dziugaite and Roy, 2017, Neyshabur et al., 2017].

There are a number of works that touch on understanding deep neural networks by finding general principles and patterns during training. Arpit et al. [2017] empirically show that deep networks prioritize learning simple patterns of the data during training. Xu et al. [2019] find a similar

phenomenon in the case of 2-layer networks with Sigmoid activations. Rahaman et al. [2019] study deep ReLU activated networks through the lens of Fourier analysis and show that while deep neural networks can approximate arbitrary functions, they favour low frequency ones and thus, they exhibit a bias towards smooth functions. Samek et al. [2017] present two approaches in explaining predictions of deep learning models in a classification task, with the first method computing the sensitivity of the prediction with respect to input perturbations and the second method that meaningfully decomposes the decision in terms of input variables.

While deep RL methods are widely used and extensively studied, few works focus on understanding the policy structure in detail. Zahavy et al. [2016a] propose a visualization method to interpret the agent's actions by describing the Markov Decision Process as a directed graph on a t-SNE map. They then suggest ways to interpret, debug and optimize deep neural network policies using the proposed visualization maps. Rupprecht et al. [2019] train a generative model over the state space of Atari games to visualize states which minimize or maximize given action probabilities. Luo et al. [2018] adapt three visualization techniques to the domain of image-based RL using convolutional neural networks in order to understand the decision making process of the RL agent.

3 Piecewise Linear Regions

Throughout this work, we consider RL policies based on *ReLU networks*. A ReLU network is a ReLU-activated feed-forward neural network, or a multi-layer perceptron (MLP) which can be formulated as a scalar function $f : \mathbb{R}^d \to \mathbb{R}^o$ defined by a neural network with L hidden layers of width $d_1, ..., d_L$ and a o-dimensional output. Assuming the output to be 1-dimensional, and following the notation of Rahaman et al. [2019], we have:

$$f(\mathbf{x}) = (T^{(L+1)} \circ \sigma \circ T^{(L)} \circ ... \circ \sigma \circ T^{(1)})(\mathbf{x}), \tag{1}$$

where $T^{(k)} : \mathbb{R}^{d_{k-1}} \to \mathbb{R}^{d_k}$ computes the weighted sum $T^{(k)}(\mathbf{x}) = W^{(k)}\mathbf{x} + \mathbf{b}^{(k)}$ for some weight matrix $W^{(k)}$ and bias vector $\mathbf{b}^{(k)}$. Here, $\sigma(\mathbf{u}) = \max(0, \mathbf{u})$ denotes the ReLU activation function acting element-wise on vector $\mathbf{u} = (u_1, ..., u_n)$.

Given the ReLU network f from Equation 1, again following Rahaman et al. [2019], piecewise linearity can be explicit written by

$$f(\mathbf{x}) = \sum_{r \in R} 1_{P_r}(\mathbf{x})(W_r\mathbf{x} + \mathbf{b}_r) \tag{2}$$

where r is the index for the linear region P_r and 1_{P_r} is the indicator function on P_r. The $1 \times d$ matrix W_r is given by:

$$W_r = W^{(L+1)}W_r^{(L)}...W_r^{(1)} \tag{3}$$

where $W_r^{(k)}$ is obtained from the original weight $W^{(k)}$ by setting its jth column to zero whenever neuron j of layer $k-1$ is inactive for $k > 1$.

4 Counting Linear Regions in RL Policies

To answer the key questions posed in the introduction, we need a method for counting the linear regions encountered during the episodic trajectories taken during RL. For each input $\mathbf{x}$, we encode each neuron of the policy network with a binary code of 0 if its pre-activation is negative, and with a binary code 1 otherwise. The linear region of the input $\mathbf{x}$ can thus be uniquely identified by the concatenation of binary codes of all the neurons in the network called an *activation pattern*. Figure 2 illustrates the activation pattern construction for a 2D input space.

Figure 1 provides a schematic illustration of the linear regions for a 2D state space, together with an example trajectory that consists of sequential states encountered by the policy, connected with

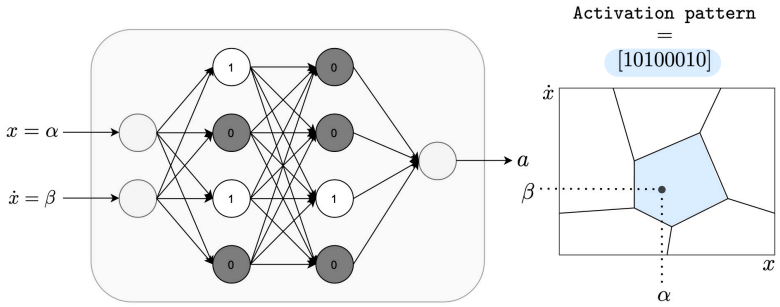


Figure 2: An overview of a ReLU-activated policy network and the binary labeling scheme of the linear regions. State $\mathbf{s} = [x, \dot{x}]$ is within the linear region uniquely identified by the activation pattern computed by concatenating the binary states of the activations of the policy network given input $\mathbf{s}$.

straight-line segments. As seen in the figure, regions can be revisited, which leads to a distinction between the number of transitions between regions and the number of unique regions visited along a trajectory. We compute both of these metrics along episodic trajectories, as enabled by keeping a list of the regions already visited at any point in time when processing a given episode trajectory. Our region-counting method also includes regions that are crossed by the straight-line segments between successive trajectory states, although the policy itself does not explicitly encounter these regions due to the discrete-time nature of typical RL environments.

To represent the k-th line segment of the episodic trajectories in the input space, we develop a parameterized line segment given by $s(u) = (1-u)s_{k-1} + us_k$, where $u \in [0, 1]$ and $s_{k-1}, s_k \in \mathbb{R}^d$ denote the endpoints of the line segment. We then calculate the exact number of the linear regions over this line segment by considering the hidden layers of the policy network one at a time, starting from input towards the output, and observe how each region can be split by the neurons. Starting with the first layer, we consider neurons one by one, and identify the point in the domain of u, if any, that induces a change in the binary labeling for that neuron. Each such point subdivides the domain of u into two new regions where the pre-activation will be zero for one of them in the next layer. By maintaining a list of these regions and the linear functions defined over them, and whether their pre-activation vanishes in the next layer, we proceed to the neurons of the next layer. This process repeats for each of the regions resulting from the previous layer. We record the activation patterns of all the final regions, to track state visits during all segments of episode trajectories. In the end, for each trajectory τ, we compute the total region transitions, $R_T(\tau)$, as well as the number of unique visited regions, $R_U(\tau)$, where $R_T(\tau) + 1 \geq R_U(\tau)$. We further compute the trajectory length in the input space, $L(\tau)$. This allows us to compute normalized region densities according to $\rho(\tau) = R_T(\tau)/(NL(\tau))$ where N is the total number of neurons in the policy network, in accordance with Hanin and Rolnick [2019a].

We use the above metrics along two types of trajectories. First, we consider *fixed* trajectories, τ^* as sampled from the final fully-trained policy. Second, we consider *current* trajectories, τ, as sampled from the current policy during training. Both of these trajectories offer informative views of the evolution of the policy. The former offers a direct picture of the linear-region density along a meaningful, fixed region of the state space, i.e., that of the final optimized policy. The latter offers a view of what the policy trajectories actually encounter during training. In order to better understand the evolution of τ, we also track its length, $L(\tau)$, given that the density is determined by the region-transition count as well as the length of the trajectory. Figure 3 shows the evolution of linear regions and the types of trajectories τ^* and τ for a simple 2D toy environment, and a ReLU-activated policy network of depth 2 and width 8.

5 Experimental Results

We conduct our experiments on four continuous control tasks including HalfCheetah-v2, Walker-v2, Ant-v2, and Swimmer-v2 environments from the OpenAI gym benchmark suits [Brockman et al., 2016]. We use the Stable-Baselines3 implementations of the PPO algorithm [Schulman et al., 2017] in all of our experiments throughout this work. We run each experiment with 5 different random

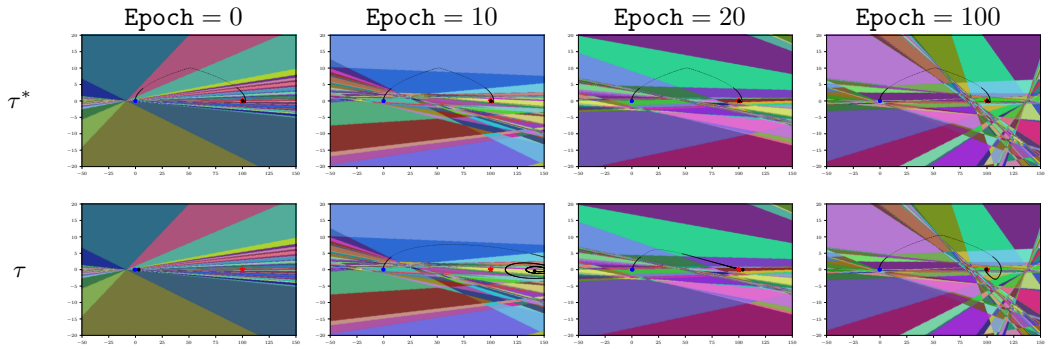


Figure 3: State space visualization with current (τ) and fixed trajectories (τ^*) of a ReLU-activated policy network with depth 2 and width 8 trained on a 2D toy environment during training. The first row, plots a fixed trajectory sampled from the final policy over the state space. The second row plots a trajectory sampled from the current snapshot of the policy. Columns indicate the number of training epochs. $\texttt{Epoch} = 0$ corresponds to the policy at initialization and $\texttt{Epoch} = 100$ corresponds to the fully-trained policy. In this toy environment, an agent starts at the origin indicated with a blue dot and the goal is for the agent to reach a target state located at $x = 100$. States are the position x (horizontal axis) and velocity $\dot{x}$ (vertical axis) of the agent, and action is the acceleration.

seeds and the results show the mean and the standard deviation across the random seeds. The policy network architecture is an MLP with ReLU activations in all hidden layers in our experiments.

We train 18 policy network configurations with $N \in \{32, 48, 64, 96, 128, 192\}$ neurons, widths $w \in \{8, 16, 32, 64\}$, and depths $d \in \{1, 2, 3, 4\}$. We use a fixed value function network structure of $(64, 64)$ in all of our experiments. Details of the network configurations used for the policy network are available in Appendix C. These network configurations are chosen such that the network is fully-capable of learning the task and achieves near state-of-the-art cumulative reward on the particular task it is trained on. This allows us to study the evolution of linear regions disentangled from properties such as trainability, which is outside the scope of this work. The table showing the range of the mean cumulative reward per task, across all network configurations can be found in Appendix B.

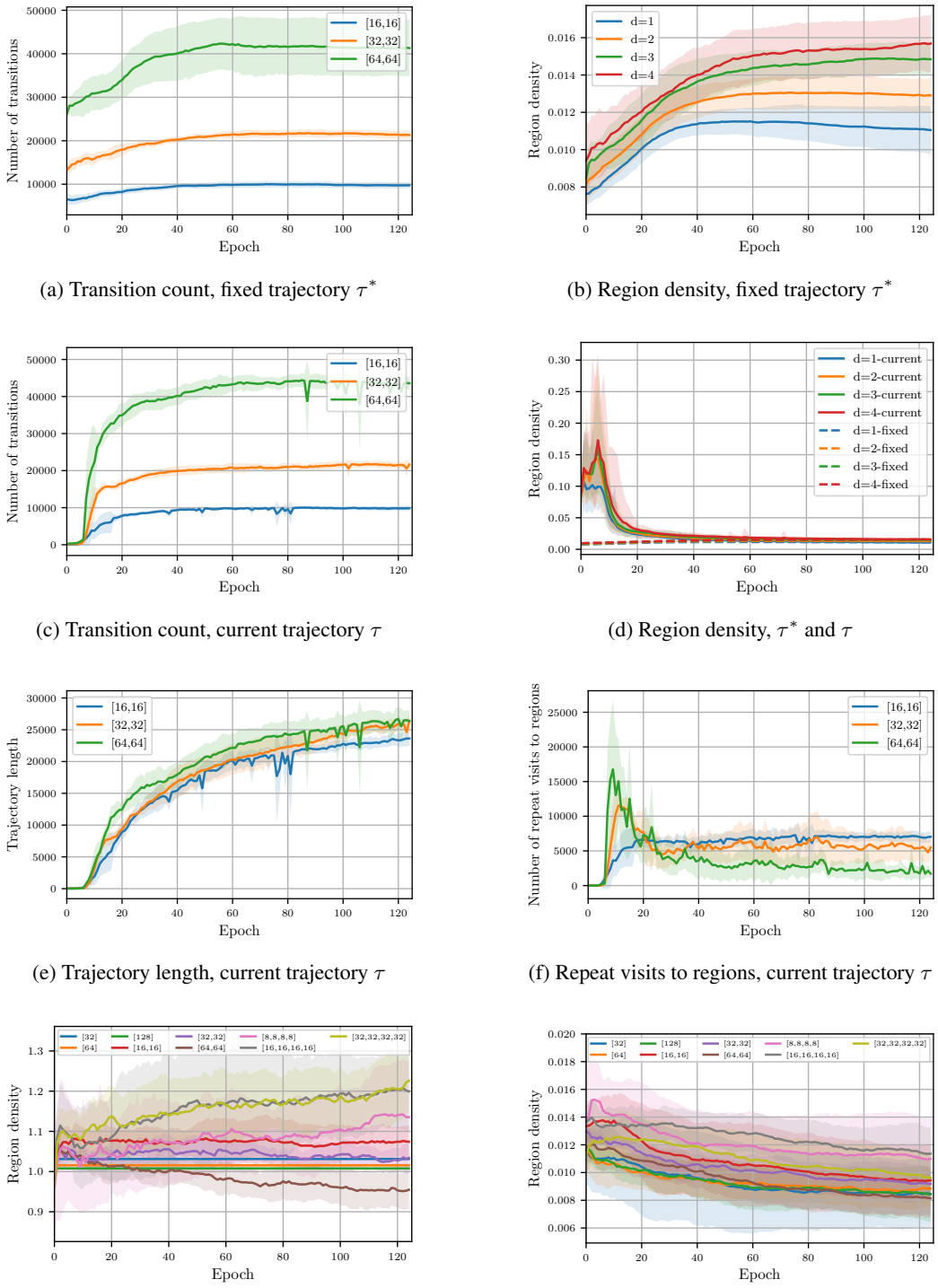
We adopt the network initialization and hyperparameters of PPO from StableBaselines3 [Raffin et al., 2021] and train our policy networks on 2M samples (i.e. 2M timesteps in the environment). This initialization scheme also matches the best-practice recommendations for on-policy RL, e.g., [Andrychowicz et al., 2020], although it differs from that used in Hanin and Rolnick [2019a] for theoretical and empirical linear region analysis. A detailed explanation of the initialization and choice of hyperparameters is available in Appendix E [1].

In what follows, we focus on the results for the HalfCheetah-v2 environment as a representative example. Similar results for the other environments are provided in Appendix J.

5.1 Evolution of the Density of Linear Regions during Training

We begin by examining the number of transitions encountered as we sweep along the fixed final-policy trajectory, $R_T(\tau^*)$, as shown in Figure 4(a). For clarity, we present results for only three policy network configurations. This plot shows a moderate and gradual increase of 50% in the number of region transitions observed in fixed-duration RL episodes during training, with larger policy networks having more transitions. Computing the region density requires dividing by number of neurons, N, and the trajectory length, $L(\tau^*)$, which is constant for the fixed final trajectory, τ^*. The result is given in Figure 4(b) which shows the mean and standard deviation of the normalized region densities across policy networks with the same depth. This shows a result that is consistent across all policy network dimensions to within $\pm 30\%$, and reflects an identical moderate-and-gradual increase during training, regardless of the policy network configuration. Thus, in response to **Q2**, RL policies trained with

[1] Our code is available at https://github.com/setarehc/deep_rl_regions.



(a) Transition count, fixed trajectory τ^*

(b) Region density, fixed trajectory τ^*

(c) Transition count, current trajectory τ

(d) Region density, τ^* and τ

(e) Trajectory length, current trajectory τ

(f) Repeat visits to regions, current trajectory τ

(g) Density for random lines through the origin

(h) Density for random-action trajectories

Figure 4: Evolution of the number of transitions, linear region densities, and length of trajectories for HalfCheetah, during training. Results for the rest of the environments are provided in Appendix J. The ranges indicate the standard error across 5 random seeds. $[n_1, ..., n_d]$ in the legend corresponds to a network architecture with depth d and n_i neurons in each layer. Summary results are grouped by network depth, d.

PPO see a moderate increase in region density along the parts of the state space that are frequently visited by the policy. We later discuss the observed impact of policy-network depth (§5.2).

We next examine the number of transitions encountered as we sweep along the evolving set of current-policy trajectories, $R_T(\tau)$, shown in Figure 4(c). These begin near zero and then rapidly increase as the policy succeeds at exploration. Again, for clarity, we show a representative sample of three policy network dimensions here. The corresponding region densities are shown in Figure 4(d), together with the fixed-region densities which are repeated from Figure 4(b) (see the bottom dashed curve). We speculate that the observed densities are higher early on due to the unstructured nature of early exploration and the form of network initialization, and then decrease over time as the trajectory lengths begin to increase, as shown in Figure 4(e).

We compute the number of repeat visits to regions as $R_T - R_U$, i.e., the region transition count minus the unique region count. In Figure 4(f), we track this for the current trajectory during training. We speculate that repeat visits are high early in training because of limited exploration. Repeat visits remain later in training because of the cyclic nature of the learned locomotion for HalfCheetah.

5.2 Impact of Policy Network Structure on Expressivity

Our normalized density computation assumes that the number of transitions scales linearly with the number of neurons, N. However, we empirically observe that there remains a moderate dependence on the network depth, as seen in Figure 4(b), with deeper networks resulting in moderately denser regions in the learned policies. We further examine the policy network expressivity using random infinite lines. Following Hanin and Rolnick [2019a], for each policy network, we sample 100 random lines by randomly sampling 100 points from the final trajectory τ^*, and connecting each randomly sampled point to the origin of the policy input space. Figure 4(g) shows the mean and standard deviation of the normalized number of transitions, R_T/N, as computed along these infinite lines in random directions passing through the origin, over the 5 random seeds of each network configuration. These produce approximately uniform results throughout training, showing no overall changes in mean density in various directions. These results are also consistent with the findings of Hanin and Rolnick [2019a] as they show normalized region densities are close to 1 at initialization and remain roughly constant during training. We repeated the same experiment with 100 random lines through the mean point instead of the origin, by connecting each randomly sampled point to the mean of the 100 sampled points. The difference between the density over lines through the origin and through the mean point is negligible.

To further examine the behavior of the policy network on areas that it does not focus on exploring during training, we compute the region density along random trajectories created by taking random actions in the environment. For each policy network, we sample a set of 10 such random trajectories τ^R and compute the normalized region density over these. Figure 4(h) shows the mean and standard deviation of this metric across this set of 10 random trajectories, over the 5 random seeds of each network configuration. This shows a very moderate decrease during training, with fairly consistent densities for all policy network configurations. We speculate that the slight decrease in density over time arises from the trained policy no longer visiting the same regions as the random-action policy. Comparing the results of random trajectories with fixed and current (Figure 4(b) and Figure 4(d)), we can see that the normalized region density over random-action trajectories and a fixed final trajectory are within the same range. Also, although this metric is higher over current trajectories early on during training, it eventually decreases and converges to the same range of region densities as the fixed trajectory. In Appendix G we further directly visualize the regions of a policy network for HalfCheetah, as viewed via an embedding plane that either passes through three points sampled from the final fixed trajectory, or three points sampled from a random-action trajectory. We do not observe significant differences between these two visualizations.

5.3 Additional Experiments

We further study the evolution of linear-region densities for the stochastic actor-critic (SAC) [Haarnoja et al., 2018] RL algorithm, to see how it may differ from PPO. The resulting comparison between PPO and SAC results is documented in detail in Appendix F. Region densities observed early on during training are quite different for SAC than they are for PPO, exhibiting high observed densities which then rapidly drop, before rising like the PPO case. We hypothesize this is due to a combination of (i) the entropy bonus for SAC, which is then annealed away, and (ii) the different network initialization used for the baseline SAC and PPO implementations. Despite this initial difference, the evolution pattern of densities are consistent with the PPO results later on during the course of training. Moreover,

similar to PPO results, densities over fixed, current and random-action trajectories are within the same range supporting our surprising finding that the complexity of RL policies is not principally captured by increased density on-and-around the optimal trajectories.

We examine whether a non-cyclic task, such as LunarLander, exhibits different linear-region evolution behavior compared to our four default tasks, which are naturally biased towards cyclic locomotion. In Appendix H, we show that the transition counts and densities along the fixed trajectory are similar in structure to those of the cyclic locomotion tasks, showing that a gradual increase in density appears to be a general property for the PPO setting, even for non-cyclic tasks.

Lastly, we also examine the behavior of linear regions in the value function, for the case of HalfCheetah and PPO. We find that linear-regions in the value functions largely evolve in a very similar way to those in the policy. Full set of results for this experiment are available in Appendix I.

6 Conclusions

The structure of ReLU-based reinforcement learning policies remains poorly understood. In this work, we explored a key property of such policies, the density of linear regions observed along policy trajectories, as a proxy for the complexity of functions they can learn. We performed a set of experiments to see how this evolves during training and whether and how changes in depth, and width of the policy network affect it. The hypothesis that the final policy trajectory sees an increase in region density is supported to a limited degree, as we observe a moderate increase in density during training, i.e., 50%. This result is consistent across tasks and across policy networks of different configurations. When observed from the vantage point of the evolving current trajectory, the region density is high to begin with, due to the limited length of early trajectories, and then decreases asymptotically towards the observed final density. For the tasks we consider, trajectory lengths are observed to grow and asymptotically approach a plateau during training. Overall, the surprising implication, for our experimental setting, is that the complexity of RL policies appears to be captured by fine-tuning during learning, rather than a significant growth in the complexity of the functions observed along trajectories of the final policy.

The key properties of linear regions as demonstrated for supervised learning of image classifiers, e.g., [Hanin and Rolnick, 2019a], are shown to generally also hold true for reinforcement learning policies. In particular, their expressivity is well modeled as being proportional to the number of neurons, and largely independent of their width and depth, for ranges of these parameters where good policies can still be learned. The number of regions observed for randomly oriented line segments through the origin of the input space is also in line with earlier theoretical and empirical results, and does not change significantly during training.

Our work comes with a number of limitations. Our experiments are mainly focused on locomotion-based tasks which produce cyclic trajectories. This could mean that the results may not generalize to other types of tasks. For example, some tasks may not result in transition counts that grow, as generally observed in our experiments. In addition, our work mainly studies deep RL policies trained with PPO. Thus, a broader analysis on more RL algorithms would shed light on the generality of the observations in this work. One challenge in comparing different RL algorithms lies in the inherent difference in the initialization schemes commonly used for different RL algorithms. This is because RL algorithms are usually sensitive to network initialization and changing it may result in instability. In addition, network initialization affects the linear regions. Currently, we compute only the mean region density for a trajectory; it may be useful to further understand the distribution of observed densities across trajectories. We note that it is common practice in RL to apply input (observation) normalization based on a rolling average of observations seen so far [Andrychowicz et al., 2020], and thus our results also employ this strategy. Applying RL without this normalization may impact both the performance of the RL algorithm and the resulting evolution of transition counts and region-density.

Our work suggests opportunities for improving our understanding of deep reinforcement learning by looking at what their mapping of states to actions look like and how they evolve during training. For instance, a number of questions for future work includes understanding the behavior of observed linear regions in the contexts of other RL algorithms, and behavior cloning. In addition, we envisage various future directions related to this work, including: (i) understanding the impact of non-IID data on decision regions more generally; (ii) consideration of RL algorithms that have less "learning

path dependence", e.g., via frequent (re)distillation onto randomly reinitialized policies; (iii) using visited regions to efficiently distill more interpretable non-parametric policies; and (iv) in real-world control settings, using knowledge of decision regions, their sizes, and their orderings to better understand exploration and safety issues. Lastly, we believe that exploring possible connections with the lottery-ticket hypothesis [Frankle and Carbin, 2018] will be worthwhile.

As foundational research related to reinforcement learning, this work could be used to develop improved reinforcement learning policy structures and algorithms, with many possible applications, both good and bad in terms of societal impact.

References

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*, 2020.

Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. *arXiv preprint arXiv:1611.01491*, 2016.

Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. In *International Conference on Machine Learning*, pages 233–242. PMLR, 2017.

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? *Advances in neural information processing systems*, 27, 2014.

Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. *Advances in neural information processing systems*, 30, 2017.

Monica Bianchini and Franco Scarselli. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE transactions on neural networks and learning systems*, 25(8):1553–1565, 2014.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Gintare Karolina Dziugaite and Daniel M Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *arXiv preprint arXiv:1703.11008*, 2017.

Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *arXiv preprint arXiv:1811.12560*, 2018.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. In *International Conference on Machine Learning*, pages 2596–2604. PMLR, 2019a.

Boris Hanin and David Rolnick. Deep relu networks have surprisingly few activation patterns. *Advances in neural information processing systems*, 32, 2019b.

Jieliang Luo, Sam Green, Peter Feghali, George Legrady, and Cetin Kaya Koç. Visual diagnostics for deep reinforcement learning policy development. *arXiv preprint arXiv:1809.06781*, 2018.

Hrushikesh Mhaskar, Qianli Liao, and Tomaso Poggio. Learning functions: when is deep better than shallow. *arXiv preprint arXiv:1603.00988*, 2016.

Hrushikesh N Mhaskar and Tomaso Poggio. Deep vs. shallow networks: An approximation theory perspective. *Analysis and Applications*, 14(06):829–848, 2016.

Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. *arXiv preprint arXiv:1402.1869*, 2014.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. *Advances in neural information processing systems*, 30, 2017.

Roman Novak, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. *arXiv preprint arXiv:1802.08760*, 2018.

Razvan Pascanu, Guido Montufar, and Yoshua Bengio. On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv preprint arXiv:1312.6098*, 2013.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 2021.

Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In *international conference on machine learning*, pages 2847–2854. PMLR, 2017.

Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019.

Christian Rupprecht, Cyril Ibrahim, and Christopher J Pal. Finding and visualizing weaknesses of deep reinforcement learning agents. *arXiv preprint arXiv:1904.01318*, 2019.

Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*, 2017.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. In *International Conference on Machine Learning*, pages 4558–4566. PMLR, 2018.

Jure Sokolić, Raja Giryes, Guillermo Sapiro, and Miguel RD Rodrigues. Robust large margin deep neural networks. *IEEE Transactions on Signal Processing*, 65(16):4265–4280, 2017.

Zhi-Qin John Xu, Yaoyu Zhang, and Yanyang Xiao. Training behavior of deep neural network in frequency domain. In *International Conference on Neural Information Processing*, pages 264–274. Springer, 2019.

Tom Zahavy, Nir Ben-Zrihem, and Shie Mannor. Graying the black box: Understanding dqns. In *International Conference on Machine Learning*, pages 1899–1908. PMLR, 2016a.

Tom Zahavy, Bingyi Kang, Alex Sivak, Jiashi Feng, Huan Xu, and Shie Mannor. Ensemble robustness and generalization of stochastic deep learning algorithms. *arXiv preprint arXiv:1602.02389*, 2016b.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 6.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code is available at: `https://github.com/setarehc/deep_rl_regions`.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 5.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We show mean/std bars. See Section 5.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No] Our experiments require modest compute resources.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

A Non-Linearity Definitions

The following activation functions are discussed or used in this work:

1. ReLU [Nair and Hinton, 2010]: $\max(x, 0)$
2. Tanh (Hyperbolic tangent): $(e^x - e^{-x})/(e^x + e^{-x})$

B Experimental Setup

All experiments are implemented in Pytorch [Paszke et al., 2019], using the RL codebase from Stable Baselines3 [Raffin et al., 2021]. We conduct our analysis on four continuous control tasks from the OpenAI gym benchmark suits [Brockman et al., 2016] including HalfCheetah-v2, Walker-v2, Ant-v2, and Swimmer-v2 environments, using PPO algorithm [Schulman et al., 2017]. Each of our experiments are performed over 5 trials with distinct random seeds.

All policy networks configurations are chosen such that the network is fully-capable of learning the task and achieve near state-of-the-art cumulative rewards on the particular task it is trained on. Except where specified, network initialization and hyperparameters are set to the defaults of the PPO implementation of Stable Baselines3. We train our policy networks on 2M samples (i.e. 2M timesteps in the environment). Table 1 shows the range of the return per environment, across all network configurations and random seeds. Return is measured as the mean trajectory cumulative reward across the batch collected at each epoch.

Table 1: Performance Metrics

1M Step Return	Average	Max	Min	Std
HalfCheetah	3920.5	5035.7	2576.1	423.2
Walker	1605.9	4527.7	135.5	1305.6
Ant	1898.1	2609.8	1255.9	337.2
Swimmer	82.3	128.8	30.8	28.8
2M Step Return	**Average**	**Max**	**Min**	**Std**
HalfCheetah	4876.6	6679.4	3406.3	494.3
Walker	3618.4	6066.1	683.3	1610
Ant	2626.6	4543.5	1535.3	672
Swimmer	112.8	132.7	41.5	19.7

C Network Dimensions

In all experiments, the policy and value function network architectures used are MLPs with ReLU activations in all hidden layers. The value function has $N = 128$ neurons with layer widths $64, 64$ in all experiments, while the policy network has 18 structures listed in Table 2.

Table 2: Network architectures

Neurons	Layer Widths		
N=32	32	16,16	8,8,8,8
N=64	64	32,32	16,16,16,16
N=128	128	64,64	32,32,32,32
N=48	48	24,24	16,16,16
N=96	96	48,48	32,32,32
N=192	192	96,96	64,64,64

D Plots and Error Bars

Each policy network architecture is trained with 5 random seeds. All metric plots are shown as the mean and standard deviation across these seeds. For the state space visualization of Figure 3, the sampling grid in input space is obtained by sampling a 300×300 grid with equally distanced points on a window of size 40×200 where $x \in [-50, 150]$ and $\dot{x} \in [-20, 20]$. Next, we feed each point of the sampling grid into the policy network to compute its activation pattern. Then, we group all points with the same activation pattern into a single linear region uniquely colored according to its activation pattern. Note that the way of encoding a linear region of a point as described in Section 4 guarantees that each region will be uniquely identified by an activation pattern. We create a color map for regions based on their activation patterns. For the state space visualization of Figure 21, we plot regions within a 2D plane that intersects the 17-dimensional input space of HalfCheetah-v2. To obtain the 2D plane, we sample three random points from a trajectory (final trajectory τ^* or random-action trajectory τ^R) and find the plane that passes through these points. We consider all the points within a square centered at the circumcenter of the three sampled points. Square is scaled such that it is slightly bigger than the circle. In order to avoid missing small regions, which are dominant in higher dimensional spaces, we do not use the subsampling method used for the visualization of the 2D environment. Instead, we compute the exact locations of the boundaries of the regions using our counting method describe in Section 4. Each region is then colored using a randomly sampled color.

E Initialization and Hyperparameters

The initialization of the policy network has a high impact on the training performance with PPO [Andrychowicz et al., 2020]. To be able to focus on the evolution of linear regions disentangled from properties such as trainability, which is outside the scope of this work, we use the default initialization of PPO from Stable Baselines3. This initialization scheme is consistent with the best-practice recommendations for on-policy RL [Andrychowicz et al., 2020], but it differs from that used in Hanin and Rolnick [2019a]. In this initialization, for both the policy and value function networks, weights of the hidden layers use orthogonal initialization with scaling $\sqrt{2}$, and the biases are set to 0. The weights of the policy network's output layer are initialized with the scale of 0.01, and the weights of the value function network's output layer are initialized with the scale of 1. In Hanin and Rolnick [2019a], networks are initialized with He normalization and biases are drawn i.i.d from a normal distribution centered at 0 with variance 10^{-6}.

We mainly adopt the default hyperparameters of PPO from the implementation of Stable Baselines3 in all of our experiments. We present the details of the choices of hyperparameters in Table 3.

Table 3: Hyperparameters

Hyperparameter	Value
Training steps	2×10^6
Learning rate	3×10^{-4}
Number of epochs	125
Minibatch size	64
Discount (γ)	0.99
GAE parameter (λ)	0.95
Init. log stdev.	-1.0
Clipping parameter (ϵ)	0.2
VF coeff. (c_1)	0.5
Entropy coeff. (c_2)	0.0
Hardware	CPU

F Comparison of PPO and SAC

To shed light on the effect of the choice of RL algorithm, we repeat our experiments with the Soft Actor Critic (SAC) algorithm [Haarnoja et al., 2018]. Similar to our main experimental setup for PPO, we use the Stable-Baselines3 implementation of the SAC algorithm and conduct our experiments on

the four continuous control tasks of HalfCheetah-v2, Walker-v2, Ant-v2, and Swimmer-v2 from the OpenAI gym benchmark suits. Again, we run each experiment with 5 different random seeds and the results show the mean and the standard deviation across the random seeds. The policy network architecture used is an MLP with ReLU activations in all hidden layers. To ensure full training capacity, we choose more parameterized networks in this setup. The value function has two hidden layers with 256 neurons in each layer, while the policy network has 12 structures as listed in Table 4. We adopt the network initialization and hyperparameters of SAC from Stable-Baselines3 and train our policy network on 1M samples. Table 5 shows the range of the random return per environment, across all network configurations and random seeds. Return is measured as the mean trajectory return of ten trajectories collected by running the deterministic policy without action noise once every 8000 steps. We present the details of the choices of hyperparameters in Table 6.

Figures 5-12 of this section, show the side-by-side comparison between the results of SAC trained on HalfCheetah and results of PPO trained on HalfCheetah previously shown in Figure 4. We can see that the observed region densities during training are quite different for SAC than they are for PPO, early on in training. We hypothesize that the many transitions (and therefore corresponding high densities) observed early on in training for SAC is due to a combination of (i) the entropy bonus for SAC, which is then annealed away, and (ii) the different network initialization used for the baseline SAC and PPO implementations.

Figures 13-20 of this section, show the same set of plots for all four environments, now grouped by metric type to enable easier comparison between environments. As can be observed the results for normalized region densities on the fixed final trajectory (Figure 13) are broadly consistent across environments (tasks), with Swimmer being an exception.

Table 4: Policy network architectures in SAC experiments

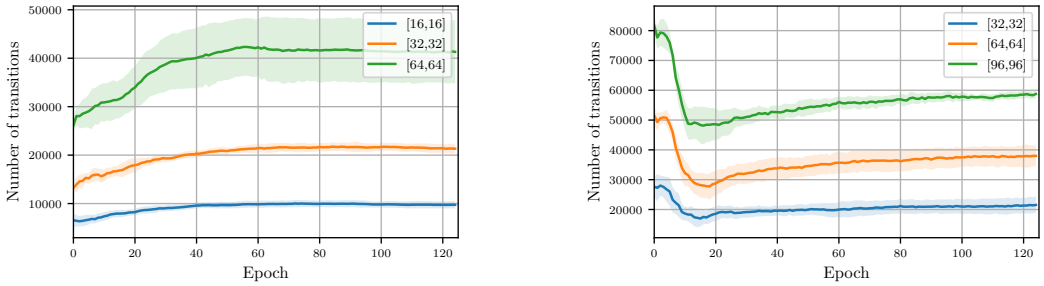
Neurons	Layer Widths		
N=32	-	16,16	-
N=64	-	32,32	-
N=96	-	-	32,32,32
N=128	128	64,64	32,32,32,32
N=192	192	96,96	64,64,64
N=256	256	128,128	64,64,64

Table 5: Performance Metrics of SAC experiments

500K Step Return	Average	Max	Min	Std
HalfCheetah	5752.8	8540.4	1573.8	1307.7
Walker	3746.2	4865.2	1207.2	812
Ant	1998.6	4982.4	-55	1110.9
Swimmer	48.2	75.1	-16.4	10.1
1M Step Return	**Average**	**Max**	**Min**	**Std**
HalfCheetah	6572.9	9436.1	1703.1	1540.6
Walker	4238.4	5131.5	1863	568.8
Ant	2501.6	5248.1	-115.5	1116.7
Swimmer	51.2	97.4	39.3	8.4

Table 6: Hyperparameters of SAC experiments

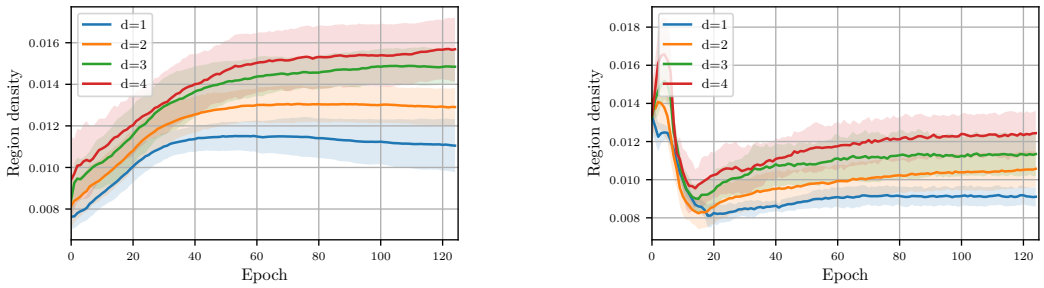
Hyperparameter	Value
Training steps	1×10^6
Learning rate	3×10^{-4}
Number of epochs	125
Batch size	256
Discount (γ)	0.99
Soft update coefficient (τ)	0.005
Entropy coeff. (initial)	0.1
Hardware	CPU



(a) HalfCheetah - PPO

(b) HalfCheetah - SAC

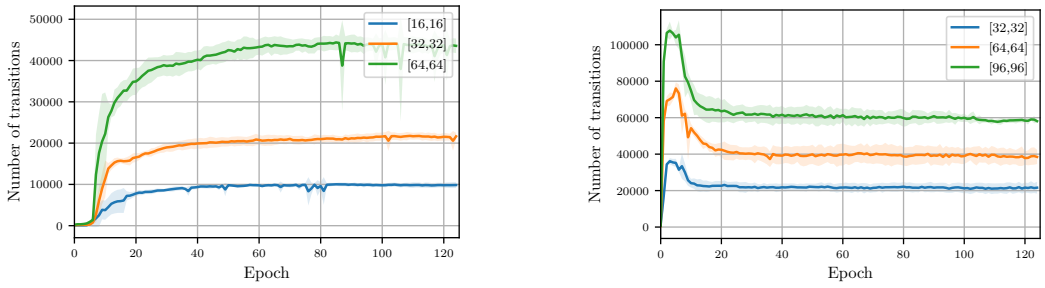
Figure 5: Evolution of the number of transitions over a fixed trajectory sampled from the final fully trained policy (τ^*) during training HalfCheetah with SAC and PPO algorithms. Plots show the mean and standard error across 5 random seeds. In the legend, $[n_1, ..., n_d]$ corresponds to a network architecture with depth d and n_i neurons in each layer.



(a) HalfCheetah - PPO

(b) HalfCheetah - SAC

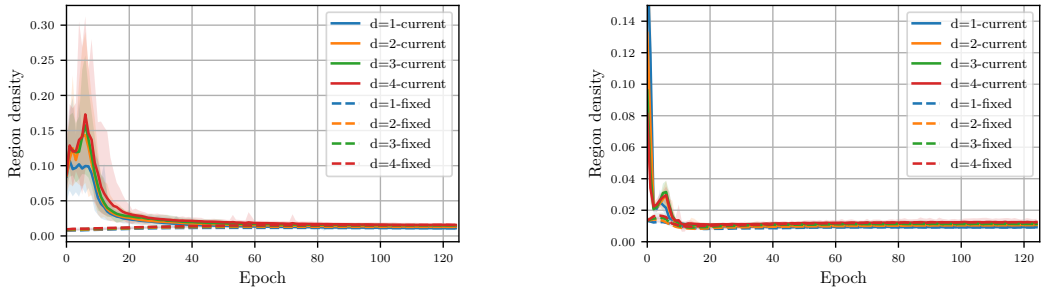
Figure 6: Evolution of the normalized region density over a fixed trajectory sampled from the final fully trained policy (τ^*) during training HalfCheetah with SAC and PPO algorithms.



(a) HalfCheetah - PPO

(b) HalfCheetah - SAC

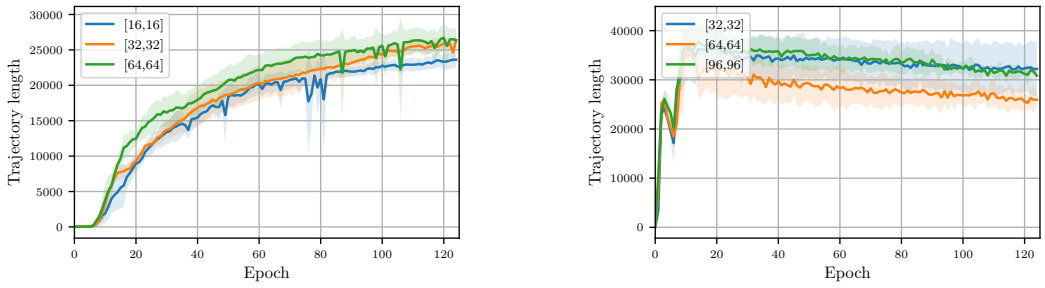
Figure 7: Evolution of the number of transitions over trajectories sampled from the current snapshot of the policy (τ) during training HalfCheetah with SAC and PPO algorithms.



(a) HalfCheetah - PPO

(b) HalfCheetah - SAC

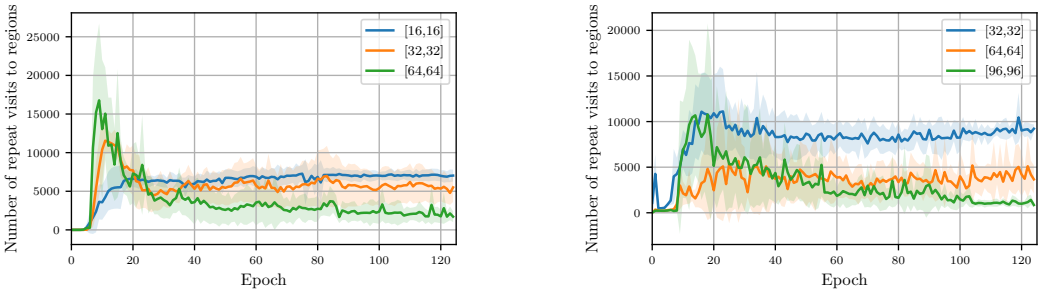
Figure 8: Evolution of the normalized region density over both a fixed trajectory sampled from the final fully trained policy (τ^*) and current trajectories sampled from the current snapshot of the policy (τ) during training HalfCheetah with SAC and PPO algorithms.



(a) HalfCheetah - PPO

(b) HalfCheetah - SAC

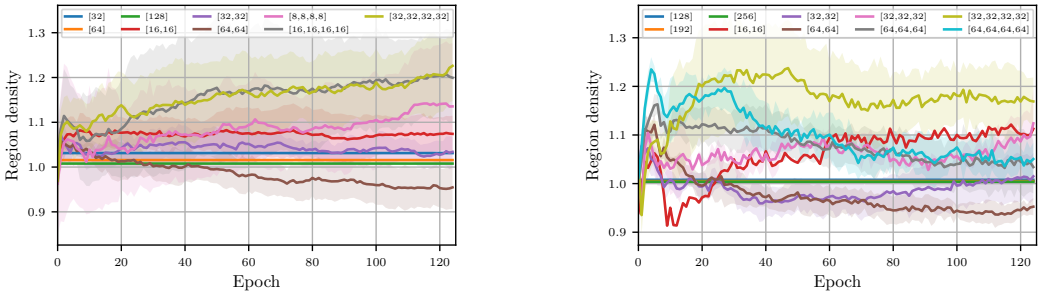
Figure 9: Evolution of the length of trajectories sampled from the current snapshot of the policy (τ) during training HalfCheetah with SAC and PPO algorithms.



(a) HalfCheetah - PPO

(b) HalfCheetah - SAC

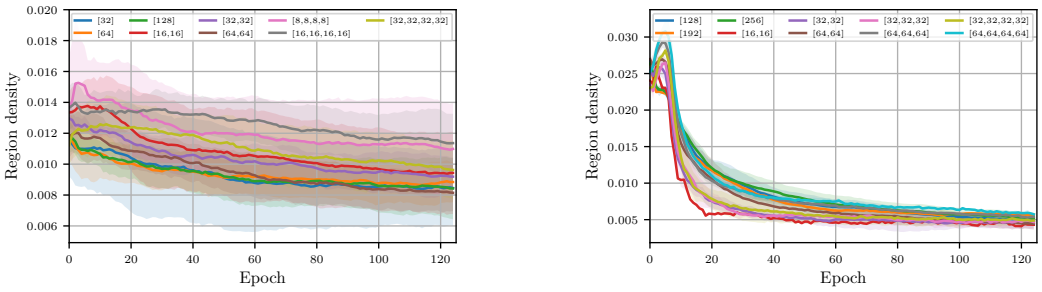
Figure 10: Evolution of the number of repeat visits to regions over trajectories sampled from the current snapshot of the policy (τ) during training HalfCheetah with SAC and PPO algorithms.



(a) HalfCheetah - PPO

(b) HalfCheetah - SAC

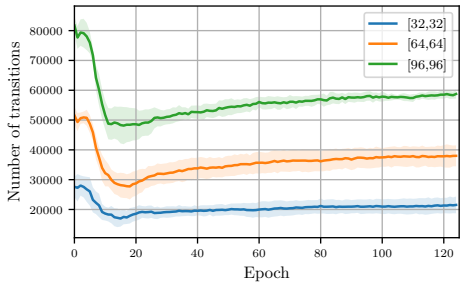
Figure 11: Evolution of the mean normalized region density over 100 random lines passing through the origin during training HalfCheetah with SAC and PPO algorithms. For each policy network configuration, we sample 100 random lines and compute the density of transitions as we sweep along these lines. We then report the mean density observed over these 100 lines.



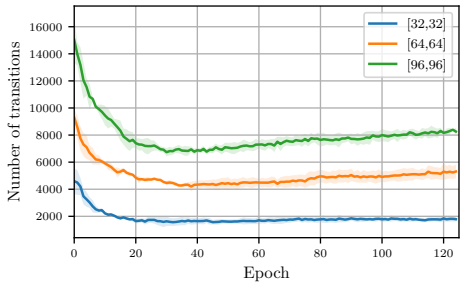
(a) HalfCheetah - PPO

(b) HalfCheetah - SAC

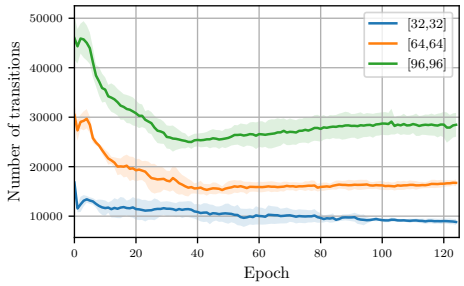
Figure 12: Evolution of the mean normalized region density over 10 random-action trajectories (τ^R) during training HalfCheetah with SAC and PPO algorithms. For each policy network configuration, we sample 10 random-action trajectories and compute the density of transitions as we sweep along these trajectories, and report the mean value of these trajectories.



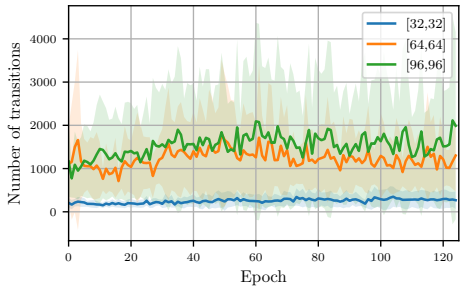
(a) HalfCheetah - SAC



(b) Walker - SAC

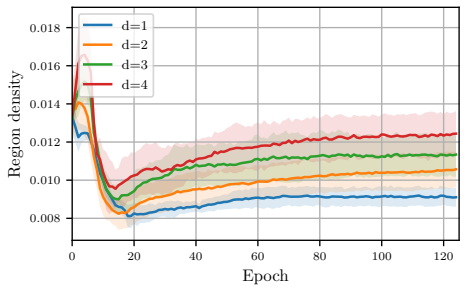


(c) Ant - SAC

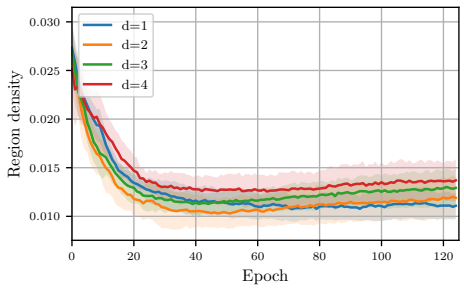


(d) Swimmer - SAC

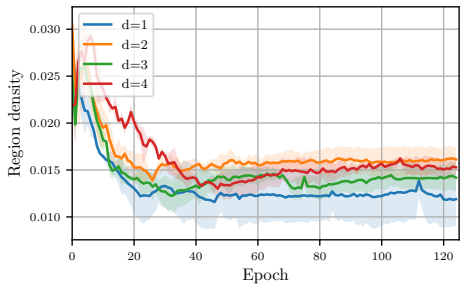
Figure 13: Evolution of the number of transitions over a fixed trajectory sampled from the final fully trained policy (τ^*) during training different tasks with SAC.



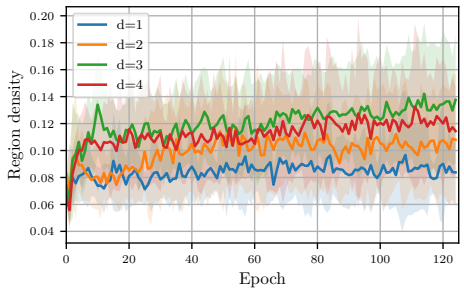
(a) HalfCheetah - SAC



(b) Walker - SAC

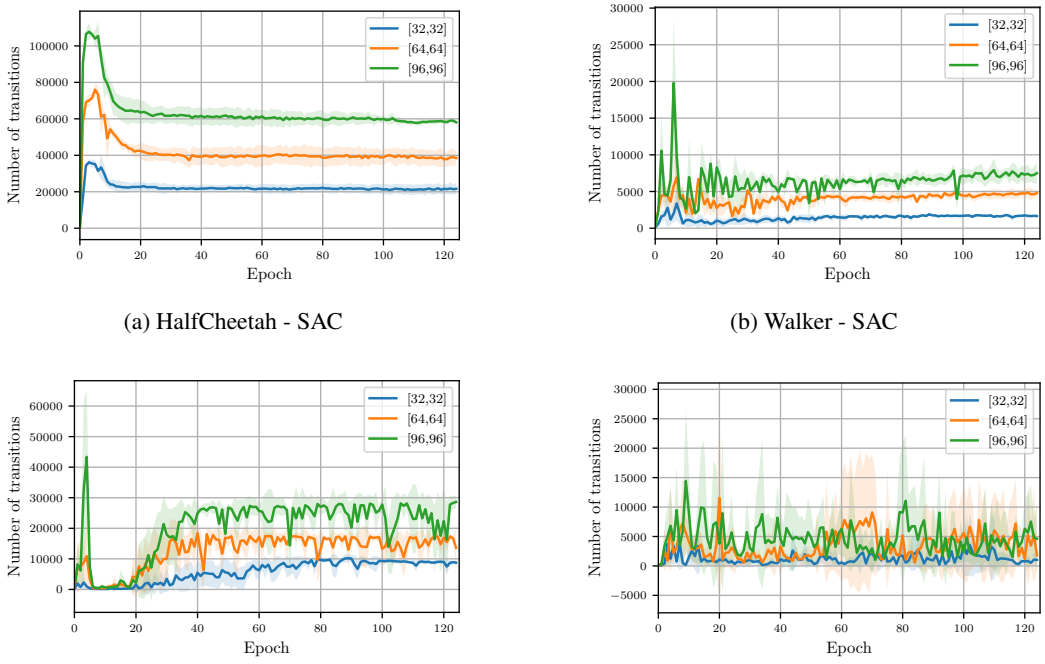


(c) Ant - SAC



(d) Swimmer - SAC

Figure 14: Evolution of the normalized region density over a fixed trajectory sampled from the final fully trained policy (τ^*) during training different tasks with SAC.



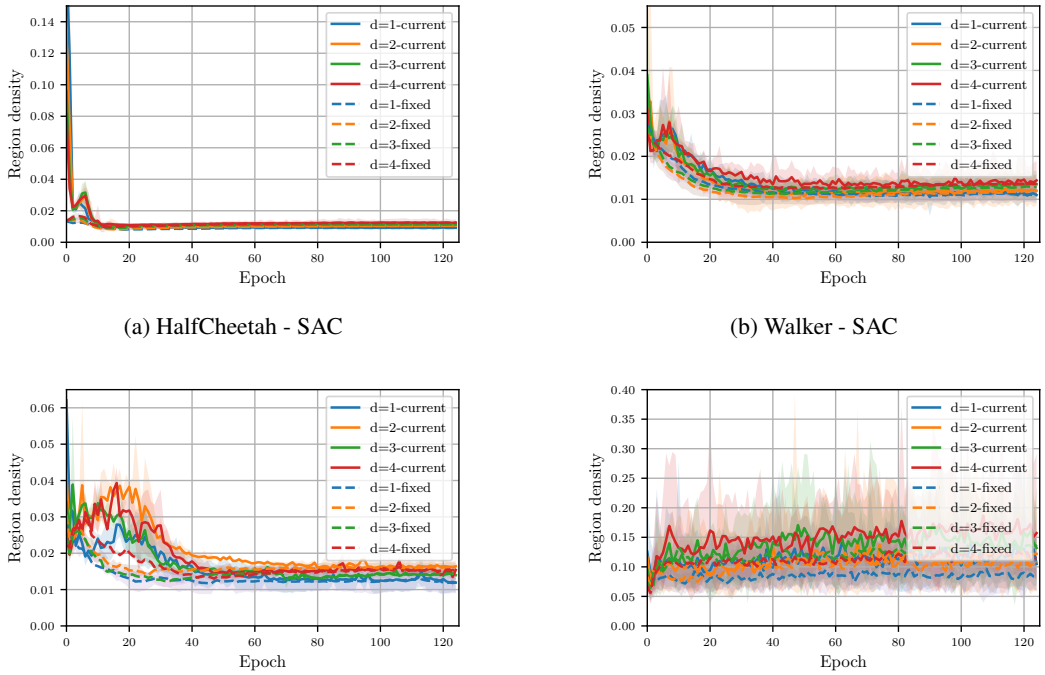
(a) HalfCheetah - SAC

(b) Walker - SAC

(c) Ant - SAC

(d) Swimmer - SAC

Figure 15: Evolution of the number of transitions over trajectories sampled from the current snapshot of the policy (τ) during training different tasks with SAC.



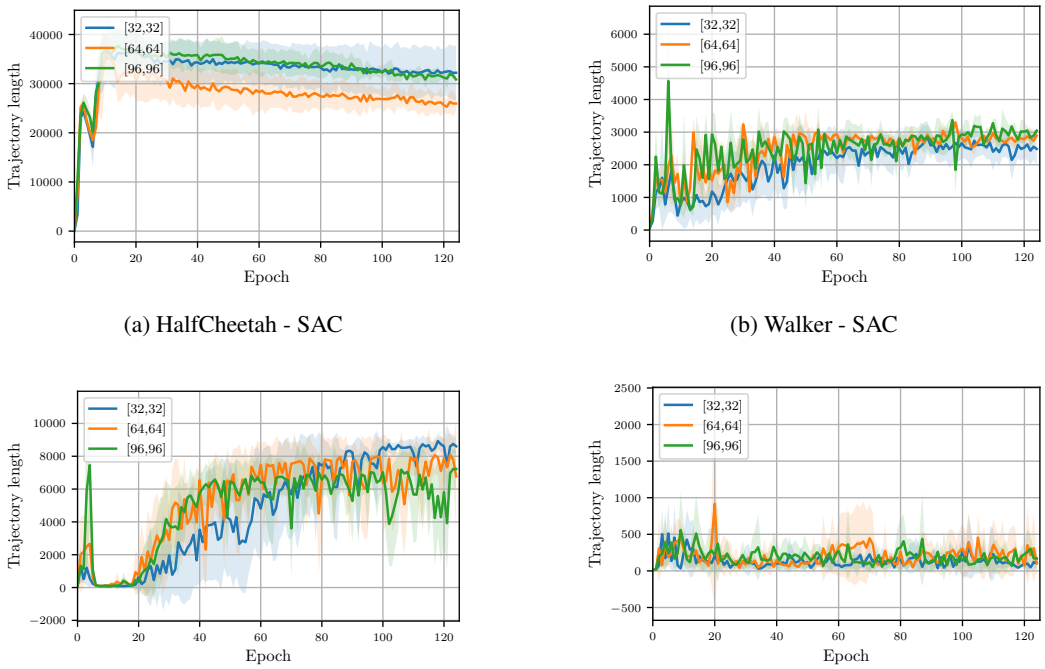
(a) HalfCheetah - SAC

(b) Walker - SAC

(c) Ant - SAC

(d) Swimmer - SAC

Figure 16: Evolution of the normalized region density over both a fixed trajectory sampled from the final fully trained policy (τ^*) and current trajectories sampled from the current snapshot of the policy (τ) during training different tasks with SAC.



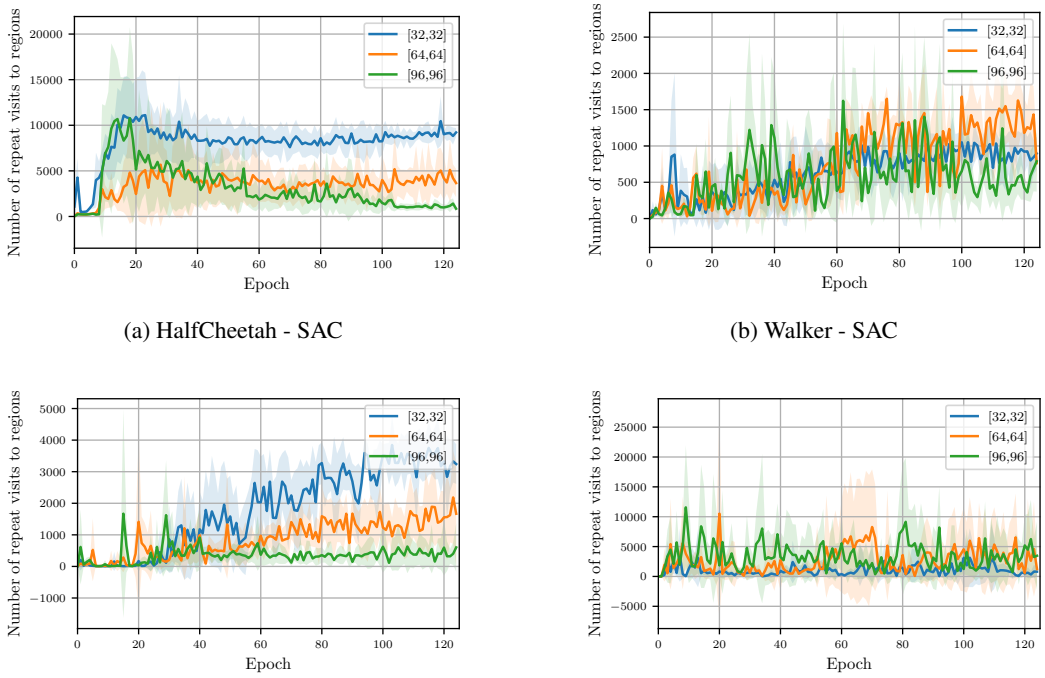
(a) HalfCheetah - SAC

(b) Walker - SAC

(c) Ant - SAC

(d) Swimmer - SAC

Figure 17: Evolution of the length of trajectories sampled from the current snapshot of the policy (τ) during training different tasks with SAC.



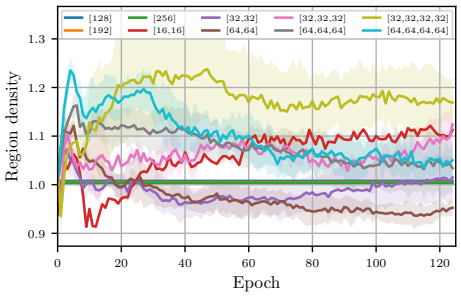
(a) HalfCheetah - SAC

(b) Walker - SAC

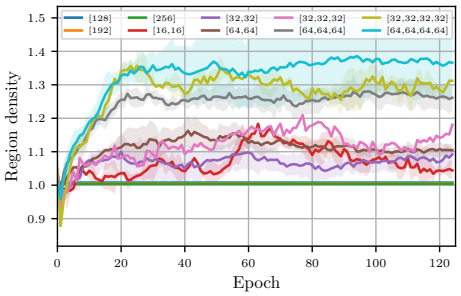
(c) Ant - SAC

(d) Swimmer - SAC

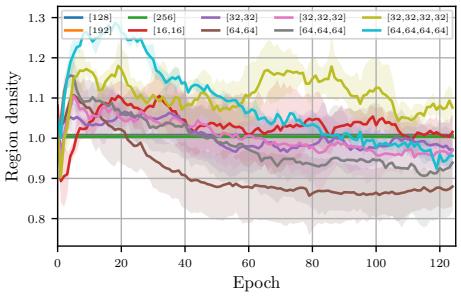
Figure 18: Evolution of the number of repeat visits to regions over trajectories sampled from the current snapshot of the policy (τ) during training different tasks with SAC.



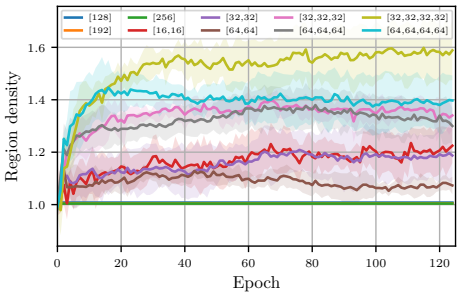
(a) HalfCheetah - SAC



(b) Walker - SAC

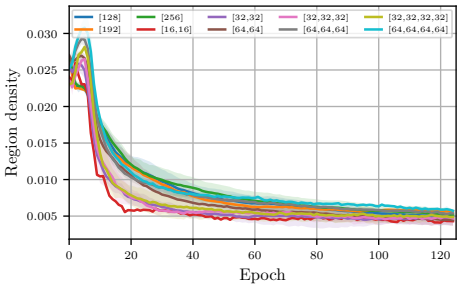


(c) Ant - SAC

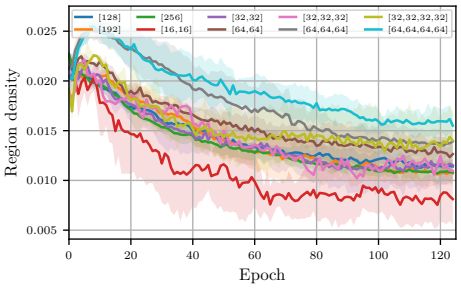


(d) Swimmer - SAC

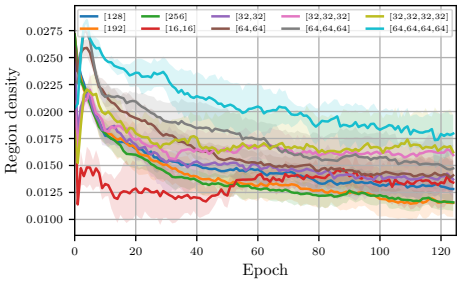
Figure 19: Evolution of the mean normalized region density over 100 random lines passing through the origin during training different tasks with SAC. For each policy network configuration, we sample 100 random lines and compute the density of transitions as we sweep along these lines. We then report the mean density observed over these 100 lines.



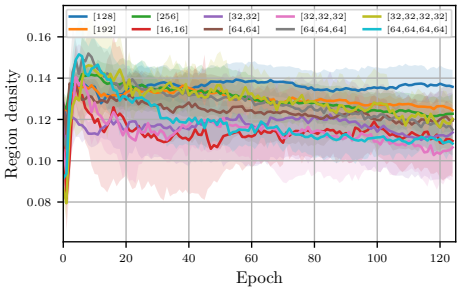
(a) HalfCheetah - SAC



(b) Walker - SAC



(c) Ant - SAC



(d) Swimmer - SAC

Figure 20: Evolution of the mean normalized region density over 10 random-action trajectories (τ^R) during training different tasks with SAC. For each policy network configuration, we sample 10 random-action trajectories and compute the density of transitions as we sweep along these trajectories, and report the mean value of these trajectories.

G Decision Regions viewed via Embedding Planes

To gain insights on what the linear regions may look like in our high dimensional setting, similar to Novak et al. [2018] and Hanin and Rolnick [2019a], we visualize the linear regions over a 2-dimensional slice through the input space defined by three points sampled from a trajectory emerged from the policy. Figure 21 shows the linear regions of a $(32, 32)$ policy network trained on HafCheetah in the 17-dimensional input space, over a 2-dimensional plane crossing three projection points sampled from the final trajectory (τ^*) in the first row, and sampled from the random-action trajectory (τ^R) in the second row. Comparing the visualizations of projected on points from τ^* and τ^R (first and second row), we do not observe a significant difference between the granularity of regions over points of these two trajectories. Moreover, surprisingly, our visualizations are more consistent with the findings of Novak et al. [2018] with the projection points lying in regions of lower density.

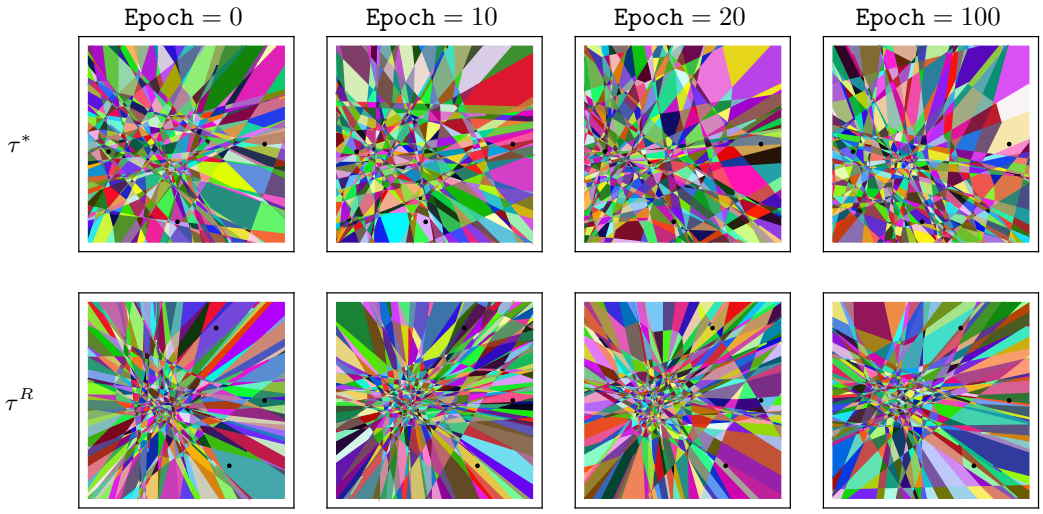
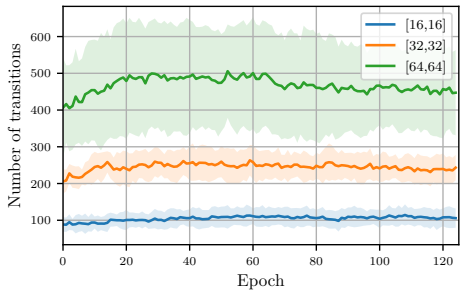


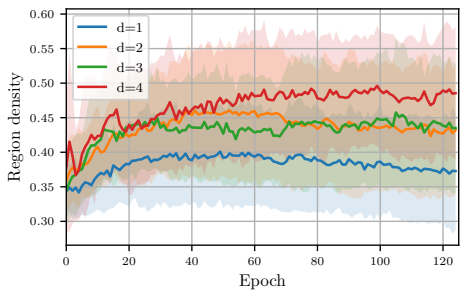
Figure 21: Linear regions that intersect a 2D plane through the input space for a network of depth 2 and width 32 trained on HalfCheetah-v2. Black dots indicate three points from the input space on which the plane is defined. In the first row, the three points are randomly sampled from the final trajectory trajectory τ^* whereas in the second row, the three points are randomly sampled from a random-action trajectory τ^R.

H Decision Regions for a Non-Cyclic Task: LunarLander

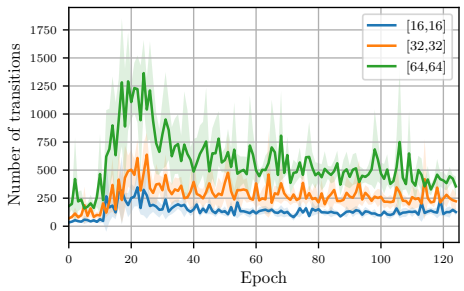
In this section, we aim to shed light on the possible effects of cyclic nature of locomotion tasks on our previous PPO results. Therefore, we repeat our experiments by training PPO on LunarLanderContinuous-v2 from OpenAI gym. Figure 22 shows the plots from this experiment. The linear-region evolution behavior is in fact still similar with the only discrepancy being the evolution pattern of observed density over current trajectories. For LunarLander, observed density over current trajectories does not decrease during training while having the same range of values as the density over fixed and random-action trajectories. This is due to the difference between the nature of non-cyclic and cyclic tasks. For locomotion tasks, the length of the cyclic trajectories increased during training, while for LunarLander, the trajectory length initially increases, then plateaus once the the agent converges.



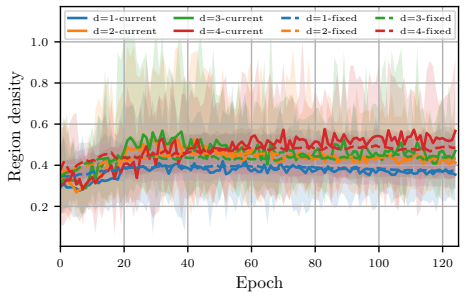
(a) Transition count, fixed trajectory τ^*



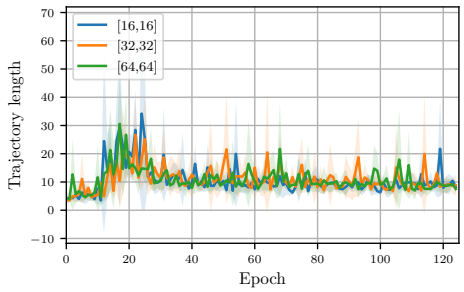
(b) Region density, fixed trajectory τ^*



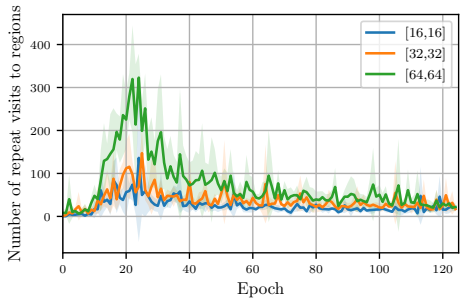
(c) Transition count, current trajectory τ



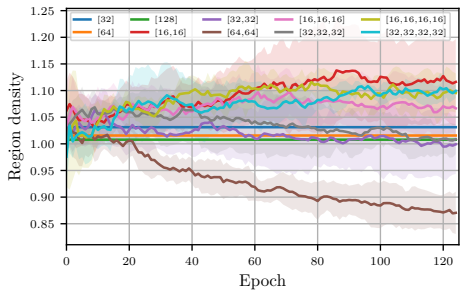
(d) Region density, τ^* and τ



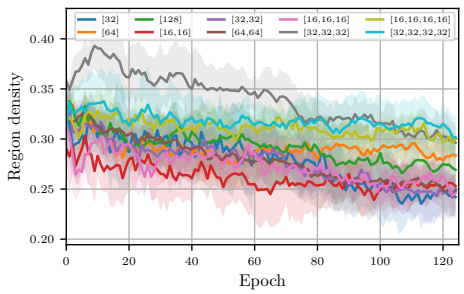
(e) Trajectory length, current trajectory τ



(f) Repeat visits to regions, current trajectory τ



(g) Density for random lines through the origin

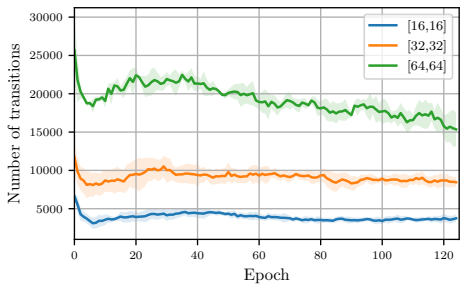


(h) Density for random-action trajectories

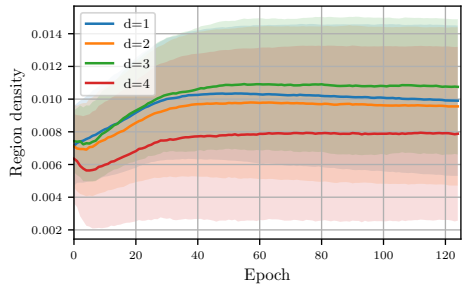
Figure 22: LunarLander Results. Training is with PPO and the ranges indicate the standard error across 5 random seeds. In the legend $[n_1, ..., n_d]$ describes a network architecture of depth d and n_i neurons in each layer. Summary results are grouped by network depth, d.

I Value Network Analysis

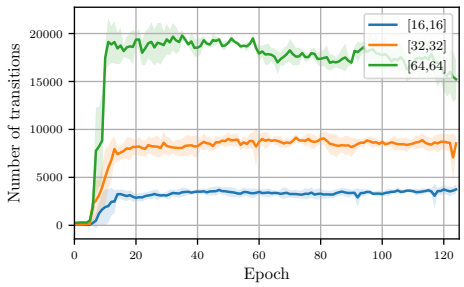
We also explore decision regions defined by the value-function network. We repeat our main experiments on the value network of PPO agents. More particularly, we train a policy network with a fixed network structure of $(64, 64)$ with 18 different value network configurations on HalfCheetah using the PPO algorithm. The value network configurations used here are the same as the ones used for the policy network and are listed in Table 2, having $N \in \{32, 48, 64, 96, 128, 192\}$ neurons, widths $w \in \{8, 16, 32, 64\}$, and depths $d \in \{1, 2, 3, 4\}$. These network configurations are chosen such that the network is fully-capable of learning the task and achieves near state-of-the-art cumulative reward on the particular task it is trained on. To compute the metrics, we now use the weights and biases of the value-function network, instead of the policy network. Figure 23 shows the resulting plots. In comparing Figure 23 and Figure 4, we see that the evolution of the region densities are remarkably similar in structure, for both fixed and current trajectories.



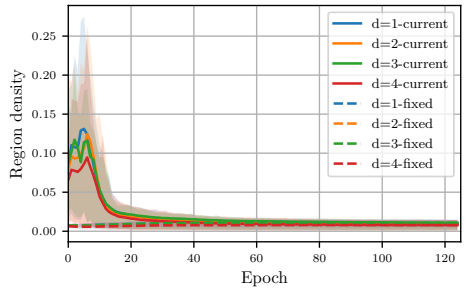
(a) Transition count, fixed trajectory τ^*



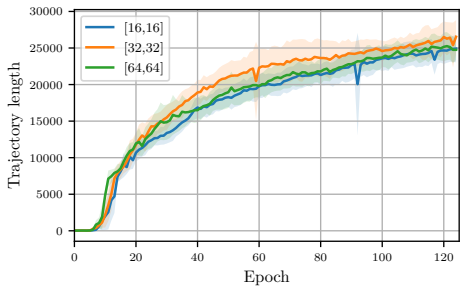
(b) Region density, fixed trajectory τ^*



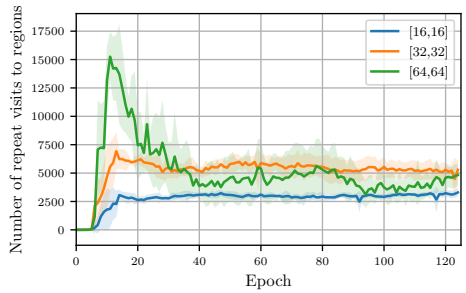
(c) Transition count, current trajectory τ



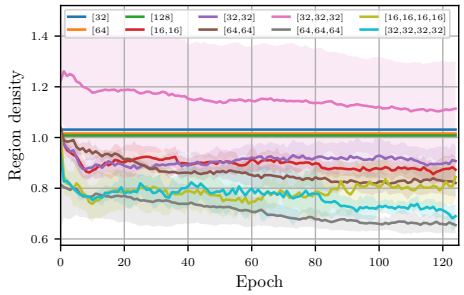
(d) Region density, τ^* and τ



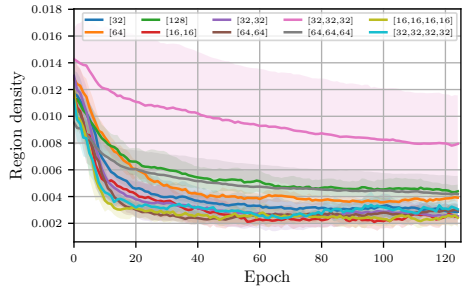
(e) Trajectory length, current trajectory τ



(f) Repeat visits to regions, current trajectory τ



(g) Density for random lines through the origin

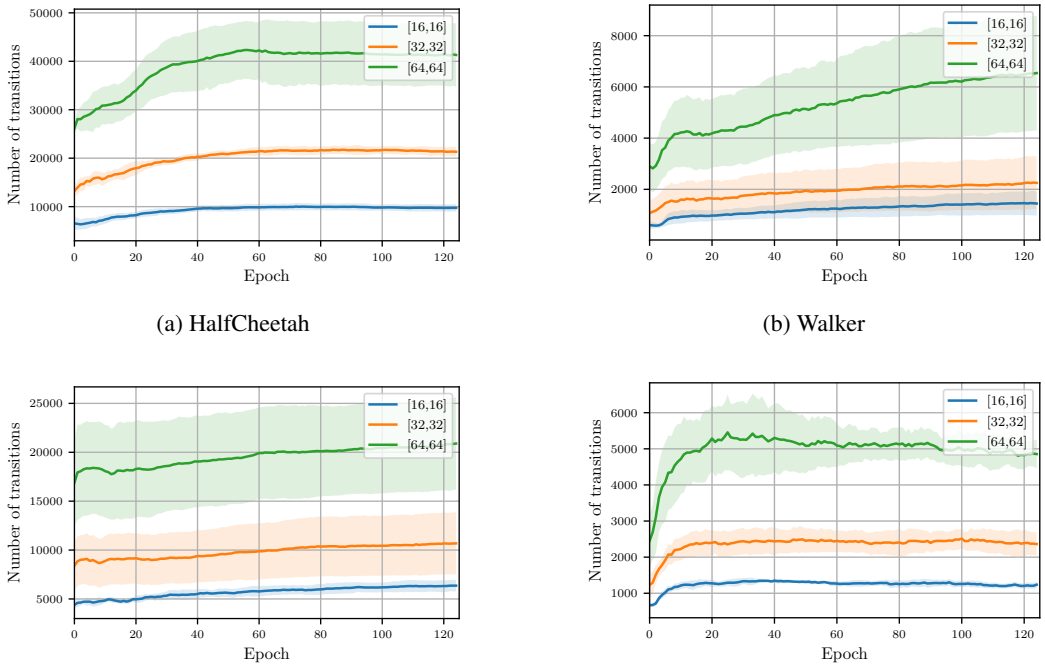


(h) Density for random-action trajectories

Figure 23: Evolution of the number of transitions, linear region densities, and length of trajectories for the value function, during training HalfCheetah with PPO. The ranges indicate the standard error across 5 random seeds. $[n_1, ..., n_d]$ in the legend corresponds to a network architecture with depth d and n_i neurons in each layer. Summary results are grouped by network depth, d.

J Additional Results

We previously provided the results for the HalfCheetah environment in Figure 4. Figures 24-31 of this section, show the same set of plots for the three remaining environments. Plots are now grouped by metric type instead of environment, to enable easier comparison between environments. Figure 32 shows the full set of results of Figures 4(a) and 4(c) for all policy network architectures trained on HalfCheetah. We previously provided the results for the HalfCheetah environment in Figure 4.



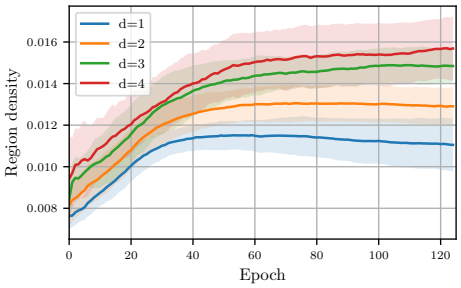
(a) HalfCheetah

(b) Walker

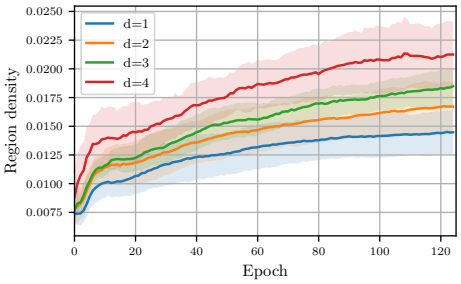
(c) Ant

(d) Swimmer

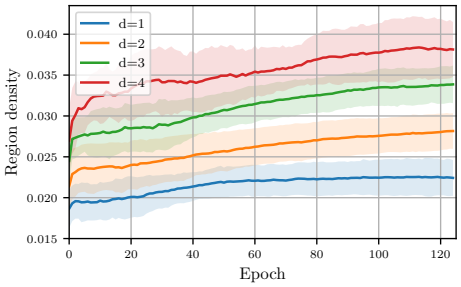
Figure 24: Evolution of the number of transitions over a fixed trajectory sampled from the final fully trained policy (τ^*) during training for different tasks. Plots show the mean and standard error across 5 random seeds. In the legend, $[n_1, ..., n_d]$ corresponds to a network architecture with depth d and n_i neurons in each layer. These plots show a moderate and gradual increase in the number of transitions over a fixed trajectory observed during training, with larger policy networks having more transitions.



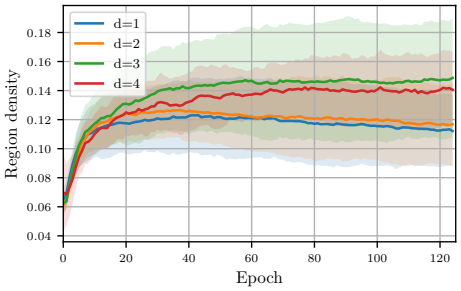
(a) HalfCheetah



(b) Walker

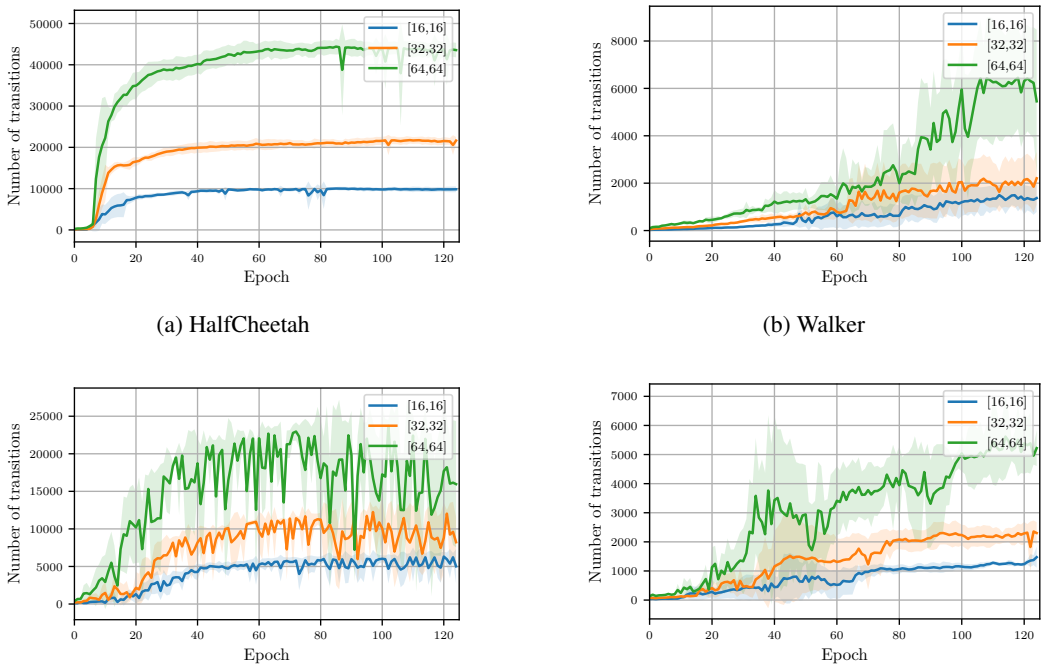


(c) Ant



(d) Swimmer

Figure 25: Evolution of the normalized region density over a fixed trajectory sampled from the final fully trained policy (τ^*) during training for different tasks. Plots show the mean and the standard error over all networks with equal depth. Note that the vertical axes do not begin at zero. In the legend, d corresponds to the depth of the policy network. These plots show a moderate and gradual increase in the density of transitions over a fixed trajectory during training. We can also observe that deeper policy networks results in moderately denser regions in the learned policies.



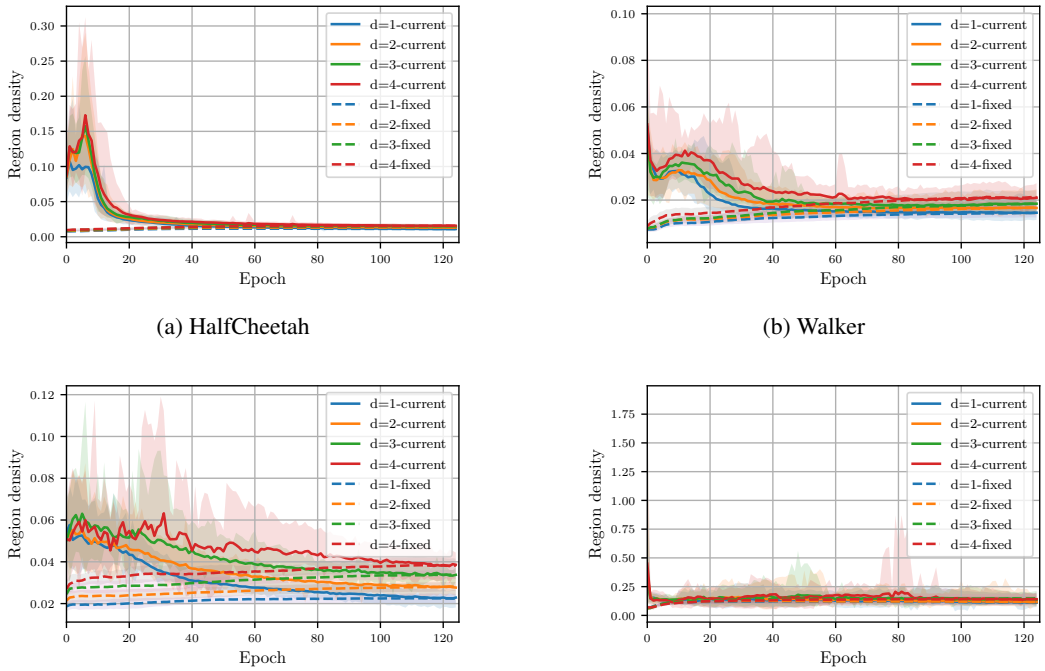
(a) HalfCheetah

(b) Walker

(c) Ant

(d) Swimmer

Figure 26: Evolution of the number of transitions over trajectories sampled from the current snapshot of the policy (τ) during training for different tasks. Plots show the mean and the standard error across 5 random seeds. In the legend, $[n_1, ..., n_d]$ corresponds to a network architecture with depth d and n_i neurons in each layer. These plots show an increase in the number of transitions observed on trajectories sampled from current snapshots of the policy during training.



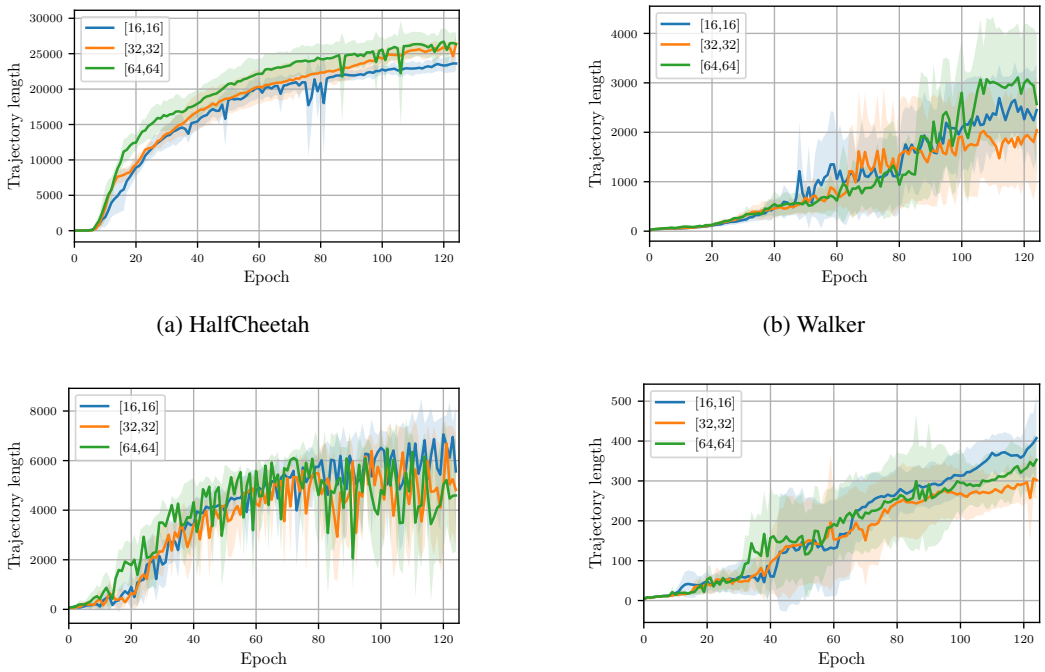
(a) HalfCheetah

(b) Walker

(c) Ant

(d) Swimmer

Figure 27: Evolution of the normalized region density over both a fixed trajectory sampled from the final fully trained policy (τ^*) and current trajectories sampled from the current snapshot of the policy (τ) during training for different tasks. Plots show the mean and the standard error over all networks with equal depth. In the legend, d corresponds to the depth of the policy network. These plots show that the density over fixed and current trajectories converges to the same values over training, while it increases for former and decreases for the latter. We speculate that the density is higher for current trajectories earlier during training due to early exploration and the form of network initialization.



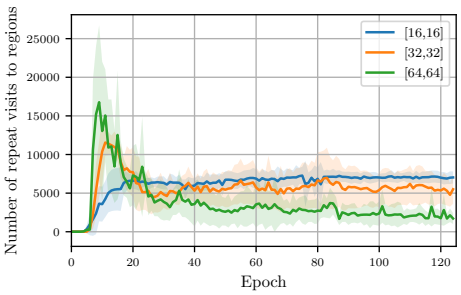
(a) HalfCheetah

(b) Walker

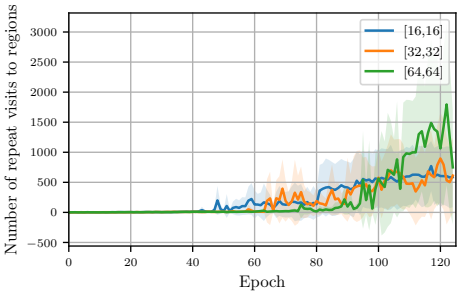
(c) Ant

(d) Swimmer

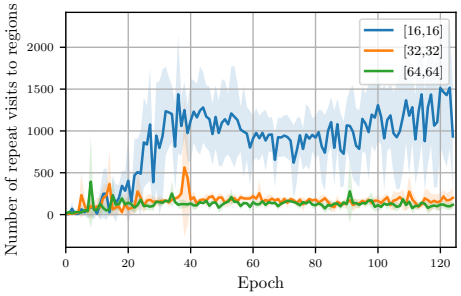
Figure 28: Evolution of the length of trajectories sampled from the current snapshot of the policy (τ) during training for different tasks. Plots show the mean and the standard error across 5 random seeds. In the legend, $[n_1, ..., n_d]$ corresponds to a network architecture with depth d and n_i neurons in each layer. These plots show that the length of the trajectories increase with training.



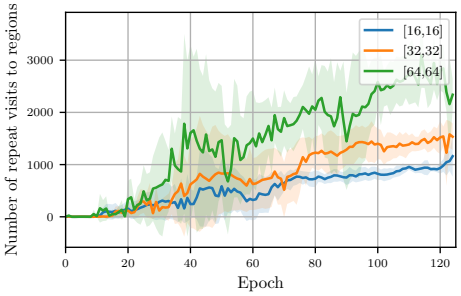
(a) HalfCheetah



(b) Walker

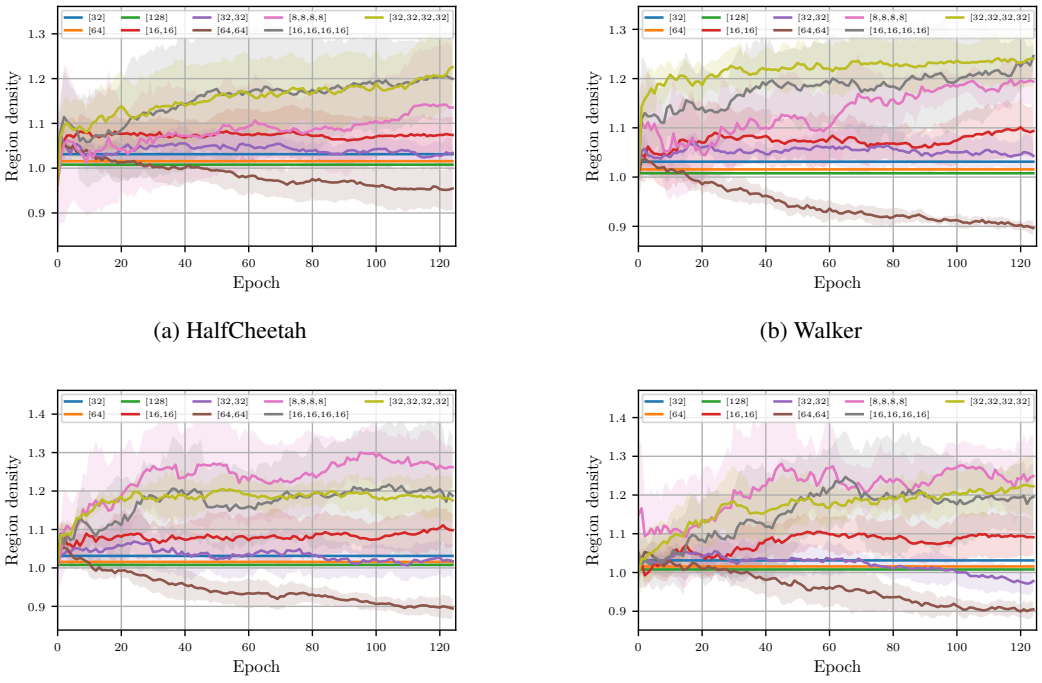


(c) Ant



(d) Swimmer

Figure 29: Evolution of the number of repeat visits to regions over trajectories sampled from the current snapshot of the policy (τ) during training for different tasks. Plots show the mean and the standard error across 5 random seeds. In the legend, $[n_1, ..., n_d]$ corresponds to a network architecture with depth d and n_i neurons in each layer. We can see that the number of repeat visits generally increase with training because of the cyclic trajectories resulting from locomotion-based tasks. We speculate that for HalfCheetah, repeat visits are high earlier during training because of limited exploration.



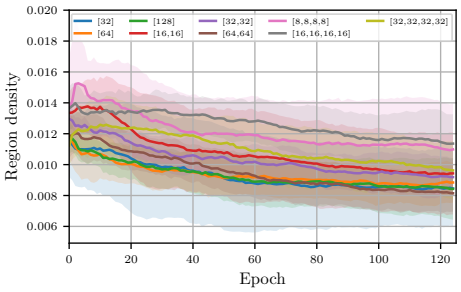
(a) HalfCheetah

(b) Walker

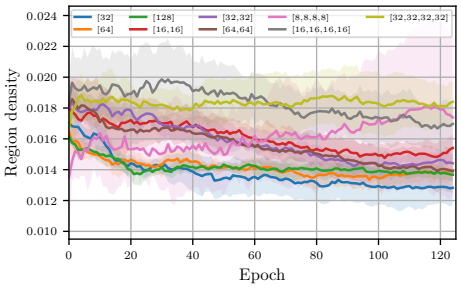
(c) Ant

(d) Swimmer

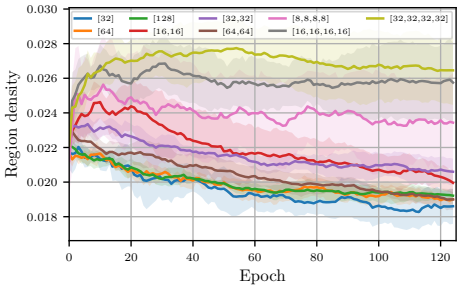
Figure 30: Evolution of the mean normalized region density over 100 random lines passing through the origin during training for different tasks. For each policy network configuration, we sample 100 random lines and compute the density of transitions as we sweep along these lines. We then report the mean density observed over these 100 lines. Plots show the mean and the standard error across 5 random seeds. In the legend, $[n_1, ..., n_d]$ corresponds to a network architecture with depth d and n_i neurons in each layer. These plots show that the mean normalized density starts close to 1 at initialization and remains roughly constant during training which is consistent with the findings of Hanin and Rolnick [2019a]. Note that because the vertical axis does not begin at zero, the variations around 1.0 is scaled.



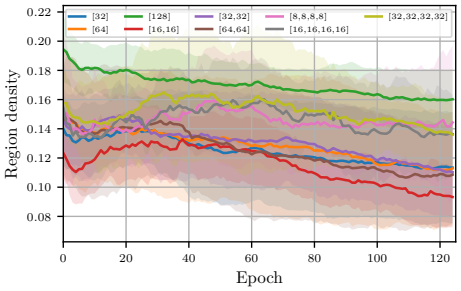
(a) HalfCheetah



(b) Walker

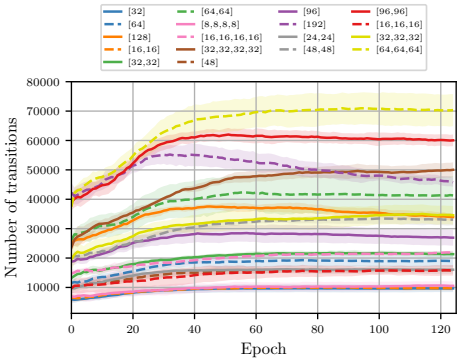


(c) Ant

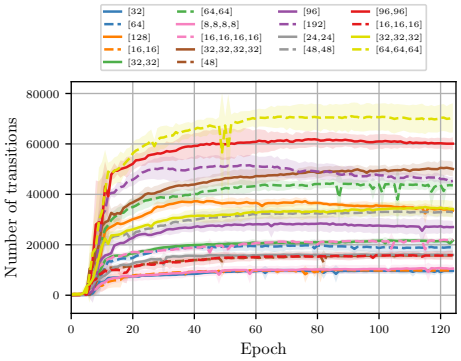


(d) Swimmer

Figure 31: Evolution of the mean normalized region density over 10 random-action trajectories (τ^R) during training for different tasks. For each policy network configuration, we sample 10 random-action trajectories and compute the density of transitions as we sweep along these trajectories, and report the mean value of these trajectories. Plots show the mean and the standard error across 5 random seeds. In the legend, $[n_1, ..., n_d]$ corresponds to a network architecture with depth d and n_i neurons in each layer. These plots show that the observed normalized density for random-action trajectories decreases slightly with training. When compared with the results of Figure 27, we observe that the observed densities are marginally less than that of fixed and current trajectories.



(a) Transition count, fixed trajectory τ^*



(b) Transition count, current trajectory τ

Figure 32: Evolution of the number of transitions over fixed and current trajectories during training for all policy network architectures trained on HalfCheetah. This figure is included for the sake of completeness.