VISUALIZING AND UNDERSTANDING DEEP REINFORCEMENT LEARNING POLICIES

Setareh Cohan
Department of Computer Science
University of British Columbia
scohan@cs.ubc.ca

ABSTRACT

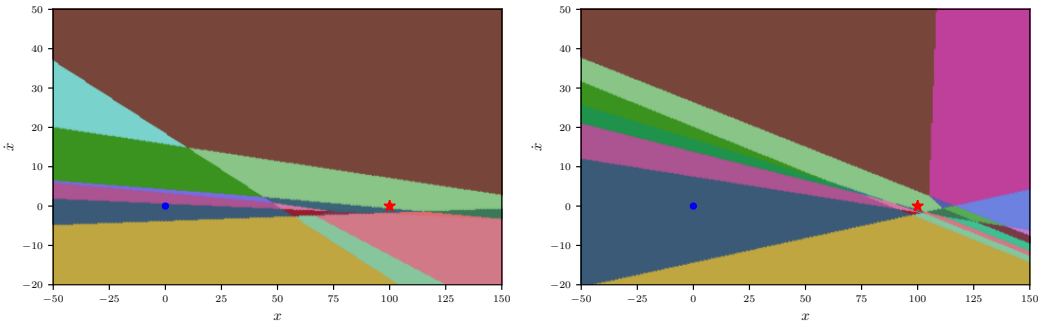
With the growing use of deep reinforcement learning, understanding how such agents behave and what their learned policies look like is of substantial importance. Understanding the decision making process such policies follow and how they perceive the state space can be valuable in identifying and solving problems in what is learned. However, little work has been done so far in exploring this area and learned policies are typically only characterised by their performance in the reinforcement learning community. In this work, we take a step towards understanding how neural network control policies map continuous states to continuous actions, how complex such mappings are, and how these mappings evolve during training. To this end, we consider a simple continuous 2-dimensional dynamical system, and a policy network with rectified linear activations. We visualize how the policy divides the state space into affine feedback regions, analyze the statistics of these regions, and then study how these statistics are affected by changes in the policy network configuration; mainly changes in depth of the network.

1 INTRODUCTION

Humans have the innate ability to learn a wide variety of skills and perform a large number of tasks. Moreover, we are capable of contemplating and justifying why we would make a decision in performing certain actions. Similarly, the decision making process followed by intelligent systems deployed in human environments must be understandable. Understanding how the model works enables us to build reliable systems, identify and solve flaws in the system, and achieve advances in our models. Following the success of deep neural networks in a variety of machine learning areas (LeCun et al., 1998; Krizhevsky et al., 2012; Long et al., 2015), the interest in understanding and analyzing the inner workings of the trained models have been growing.

The field of computer vision, for example, has developed a range of visualization techniques that have been successfully deployed to understand how convolutional neural networks (CNNs) work, what different layers of each network encode, and diagnose potential problems in the model (Zeiler & Fergus, 2014; Seifert et al., 2017; Zurowietz & Nattkemper, 2020). In addition, the field of theory has sought to explain the broad success of deep neural networks via a mathematical characterization of the expressivity of such networks (Hanin & Rolnick, 2019). For instance, Pascanu et al. (2013), Montúfar et al. (2014) and Hanin & Rolnick (2019) show that the expressive power of deep neural networks grows with depth.

Deep reinforcement learning (RL) utilizes deep neural networks to represent the policy and trains this network to optimize an objective (e.g. maximize the total reward). Deep RL algorithms have been successfully applied in a diverse set of applications including robotics, video games, and computer vision (François-Lavet et al., 2018). Although widely used and extensively studied, little work has so far been done in understanding and analyzing how RL policy networks work. Learned policy networks are commonly characterised solely by their training time and performance on the final task which is usually judged by learning curves. The expressive power of a deep RL agent, in the best case, is also solely evaluated on a validation set of scenarios (Rupprecht et al., 2019). In



(a) State-space divisions 5 steps into training (b) State-space divisions 50 steps into training

Figure 1: Figures show (a) state space divisions of a policy network with 2 hidden layers each with 16 hidden units early in training and (b) state space divisions of the same policy network after training. The colored regions correspond to linear regions where the output of the network is a linear function of the input.

this work, we take a step towards understanding how deep RL policies work and whether changes in depth, results in an improvement on the expressive power of such networks.

In this work, we focus on a class of deep feed-forward neural network policies with rectified linear activation funtions (rectifier linear units or ReLUs). There are two reasons why we study ReLU-activated deep RL agents. First, ReLUs (Nair & Hinton, 2010) are among the most popular choices of activation functions due to their practical successes (Montúfar et al., 2014). Second, linear or piecewise linear systems are beneficial in existing control systems. A piecewise linear controller has the advantage of solving complex control tasks by decomposing them into a number of linear sub-problems. This divide and conquer approach enables the application of well-established linear analysis and design methods to non-linear problems which are otherwise relatively difficult to analyse (Johansson, 1999; Leith & Leithead, 2000). And, ReLU-activated neural networks are piecewise linear

We propose a visualization framework to study how ReLU-activated deep RL policy networks divide the state space into affine feedback regions. To do so, we consider a simple continuous toy environment with a 2-dimensional state space and scalar actions.[1]. We also comute and log the statistics of the resulting linear regions since the number and complexity of such regions is a measure of network expressivity and has been a core argument for using "deep" networks as apposed to their shallow counterparts (Pascanu et al., 2013; Montúfar et al., 2014). The statistics studied in this work include the number of generated linear regions, the number of visited linear regions during training, and the ratio of the number of visited regions to the total number of generated regions.

Figure 1 shows linear regions generated by a policy network with two hidden layers. On the left, regions come from the policy network early in training and on the right, regions come from the trained policy network. As can be seen, linear regions become more fine-grained as training continues and this pattern is especially visible around the goal.

Overall, our results indicate that in practice and with our problem setup, the critical nature of depth appears different for deep RL continuous systems compared to deep learning models. From our experiments, we see that with an increase in network depth, unlike findings of (Pascanu et al., 2013; Montúfar et al., 2014; Hanin & Rolnick, 2019) in deep learning literature, the number of generated linear regions decreases. Next, our results show that the number of visited linear regions is correlated with the number of network parameters and depth does not directly affect it. Finally, our results show

[1] An affine region is a subset of the state space where the output is an affine function of the input. However, in what follows we refer to these regions as *linear regions*

that the ratio of the number of visited linear regions to the number of generated linear regions grows with depth. Therefore, we conclude that expressivity in deep RL is correlated with the ratio of the number of visited linear regions to the number of generated linear regions. Intuitively, this ratio shows how many of the generated regions are used by the policy network during training and it can be interpreted as how smart the policy network actually is.

2 Related Work

Deep RL is the combination of RL and deep learning which has been successfully applied to many complex tasks in recent years. Mnih et al. (2015) attain super-human level performance in playing Atari games from the pixels. Silver et al. (2016) have mastered the game of go, and Moravčík et al. (2017) have beaten the world's top professionals at the game of Poker. Deep RL also has potential for real-world applications such as robotics (Levine et al., 2016; Gandhi et al., 2017), self-driving cars (Pan et al., 2017) and finance (Deng et al., 2016), to name a few. At the highest level, deep RL algorithms are divided into two groups; model-based and model-free algorithms. In model-based RL, agent has access to (or learns) a model of the environment which indicates the state transitions and the rewards. In model-free RL, agent does not have access to a model of the environment. In comparison, model-free algorithms are less sample efficient compared with model-based models but they tend to be easier to implement and tune and are more popular (François-Lavet et al., 2018).

There are two main approaches to representing and training agents with model-free RL; *policy optimization* and *Q-learning*. Policy optimization represents a policy explicitly as a function of network parameters and optimize the parameters either directly by gradient ascent on the objective, or indirectly, by maximizing local approximations of the objective. Policy Gradient (Sutton et al., 2000), A2C and A3C (Mnih et al., 2016) , PPO (Schulman et al., 2017) and TRPO (Schulman et al., 2015) are examples of this family. Q-learning learns an approximator for the optimal action-value function. Typically they use an objective function based on the Bellman equation and perform off-policy optimization to find the optimal action-value function. The corresponding policy is then obtained via the connection between the optimal action-value function and optimal policy. DQN Mnih et al. (2013) is a popular Q-learning method, and DDPG (Lillicrap et al., 2015) and SAC (Haarnoja et al., 2018) lie in between policy optimization and Q-learning methods and keep a trade-off between the strengths and weaknesses of either side.

Although deep RL methods are widely used and extensively studied, small number of works focus on the importance of understanding how RL agents learn and perform. Zahavy et al. (2016) 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 CNN visualization techniques to the domain of CNN-based RL in order to understand the decision making process of the RL agent.

In deep learning literature, there is a line of work that on the success of deep neural networks with a focus on network expressivity. On the theory side, Pascanu et al. (2013) study the complexity of functions computable by ReLU-activated deep neural networks. They show that in the asymptotic limit of many hidden layers, deep 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 in terms of their number of linear regions. Their analysis of ReLU-activated networks, show that the number of linear regions grows polynomially with width and exponentially with depth. More specifically, they show that a ReLU-activated network with n_0 input units and L hidden layers of width $n \geq n0$ can compute functions that have $\Omega\left((\frac{n}{n_0})^{n_0(L-1)} n^{n_0}\right)$ linear regions. Raghu et al. (2017) generalize these results by introducing a new set of expressivity measures. They provide asymptotically tight bounds on the growth of these measures in depth of the ReLU and tanh activated neural networks.

Hanin & Rolnick (2019) study the importance of depth on expressivity of neural networks in practice. They show although effective, in practice, the effect of depth on expressivity of neural networks is likely far below that of the theoretical maximum proposed by prior literature. They found that the average size of 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. This strongly suggests that deeper networks do not learn more complex functions than 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. Notably, Ba & Caruana (2013) 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 focus 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. They showed 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; one method which computes the sensitivity of the prediction with respect to changes in the input and one approach which meaningfully decomposes the decision in terms of the input variables.

3 BACKGROUND

Throughout this work, we consider a class of models which we call *ReLU networks*. A ReLU network is a ReLU-activated feed-forward neural network, or a multi-layer perceptron (MLP), as shown in Figure 4. A ReLU network is essentially a scalar function $f : \mathbb{R}^d \to \mathbb{R}$ defined by a neural network with L hidden layers of width $d_1, ..., d_L$ and a single output neuron:

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

where each $T^{(k)} : \mathbb{R}^{d_{k-1}} \to \mathbb{R}^{d_k}$ computes the weighted sum $T^{(k)}(\mathbf{x}) = W^{(k)}\mathbf{x} + \mathbf{b}^{(k)}$ for some weight matrix $W^{(t)}$ and bias vector $\mathbf{b}^{(k)}$. Here, $\sigma(\mathbf{u}) = \max(0, \mathbf{u})$ denotes the ReLU activation function acting element-wise on a vector $\mathbf{u} = (u_1, ..., u_n)$.
ReLU activations are piecewise linear and thus, the output of ReLU networks is also piecewise linear, a consequence of the fact that composing piecewise linear functions results in a piecewise linear function (Raghu et al., 2017). A *linear region* of a piecewise linear function is a maximal connected subset of its input space, on which the function is linear (Montúfar et al., 2014). ReLU networks divide their input space into a number of these linear regions.

For any given input $\mathbf{x}$, each hidden unit is assigned an activation value $\epsilon \in \{0, 1\}$, conditioned on whether its input is negative or not. Following the terminology of Raghu et al. (2017), we can assign an *activation pattern* to the ReLU network at any point of time which is a string of 0s and 1s consisting of the collection of activation values of all hidden units in the network. Activation pattern of the network remains the same within each linear region since the output of the network is a linear function of the input inside that region. Therefore, each linear region corresponds to a unique activation pattern. Figure 2 shows how the complexity of these linear regions changes in a deep ReLU network with 2-dimensional inputs. Each neuron in the first layer splits the input space into two pieces along a hyperplane, fitting a different linear function to each of the pieces. Subsequent layers split the regions of the preceding layers resulting in a final division of the state space.

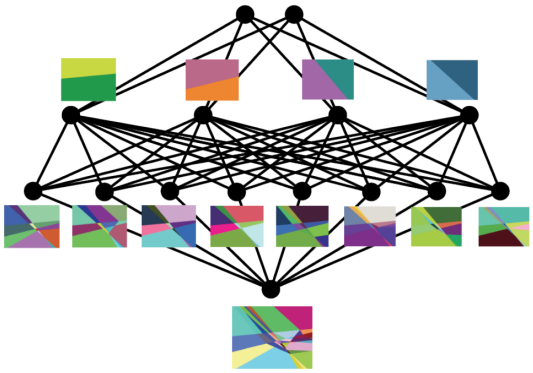


Figure 2: Evolution of linear regions of a ReLU network with 2-dimensional inputs. Each neuron in the first layer defines a linear boundary that partitions the input space into two regions. Neurons in the second layer combine and split these linear boundaries into higher level patterns of regions, and so on. Ultimately, the input space is partitioned into a number of regions, within each of which the output of the neural network is a linear function of its input. During training, both the partitioning into regions and the linear functions on them are learned. Figure is borrowed from Hanin & Rolnick (2019).

3.1 Piecewise Linearity

Let us consider the ReLU network f from Equation 1. By incorporating bias terms into the weight matrices, we have:

$$f(\mathbf{x}^{'}) = (K^{(L+1)} \circ \sigma \circ K^{(L)} \circ ... \circ \sigma \circ K^{(1)})(\mathbf{x}^{'}) \tag{2}$$

where $K^{(k)} = [\mathbf{b}^{(k)}|W^{(k)}]$ and $\mathbf{x}^{'} = [1|\mathbf{x}^{T}]^{T}$. Looking at a snapshot of the network, if neuron j of layer 1 is inactive, then j^{th} index of $\sigma(K^{(1)}\mathbf{x}^{'})$ should be zero and, if it is active then the j^{th} index would be equal to the j^{th} index of $K^{(1)}\mathbf{x}^{'}$. This operation can simply be obtained by a matrix multiplication $P^{(1)}\mathbf{x}^{'}$ such that:

$$P_j^{(1)} = \begin{cases} K_j^{(1)} & \text{neuron } j \text{ is active} \\ \mathbf{0} & \text{neuron } j \text{ is inactive} \end{cases} \tag{3}$$

where subscript j indicates j^{th} row of matrices.

Now, if we convert the calculations back to the original form where bias and weights were separate, for the general L-layered network, we can make piecewise linearity explicit by writing:

$$f(\mathbf{x}) = \sum_{r \in R} 1_{P_r}(\mathbf{x})(W_r\mathbf{x} + \mathbf{b}_r) \tag{4}$$

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

$$W_r = W^{(L+1)}W_r^{(L)}...W_r^{(1)} \tag{5}$$

where $W_r^{(k)}$ is obtained from the original weight $W^{(k)}$ by setting its j^{th} column to zero whenever neuron j of layer k is inactive. Figure 3 shows the graph of the output of a ReLU network with 1-dimensional inputs and outputs. As shown in the figure, output of the network is a collection of a number of linear functions $W_r\mathbf{x} + b_r$ defined on linear region $r \in R$ in the input space.

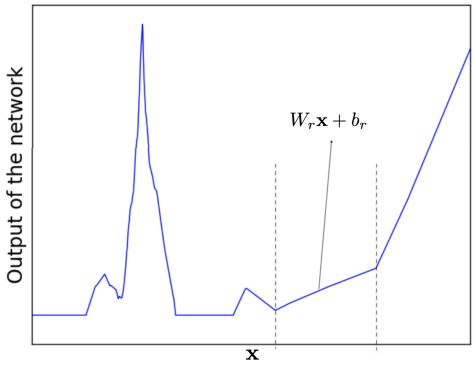


Figure 3: Graph of a function computed by a ReLU network with input and output dimension 1 at initialization. The weights of the network are He normal (i.i.d. normal with variance = 2/fan-in) and the biases are i.i.d. normal with variance 10^{-6}. Here, r indicates the linear region rvx belongs to. Figure is borrowed from Hanin & Rolnick (2019).

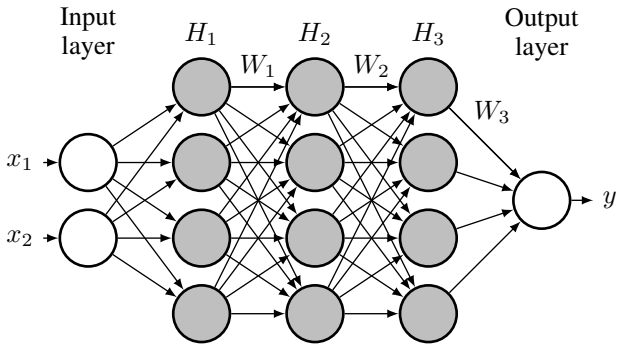


Figure 4: Illustration of a ReLU activated feed-forward neural network with three hidden layers of width 4, input dimension 2 and output dimension 1.

3.2 Reinforcement Learning

We formalize the RL problem in the standard Markov Decision Process (MDP) setting where an agent interacts with its environment in episodes, each consisting of sequences of observations, actions and rewards. We use s_t to denote the state at time step t, a_t to denote the action taken by the agent at time step t, and r_t to denote the reward received at time step t. Our goal is to train an agent whose expected cumulative reward in each episode is maximized. To do this, we can use policy gradient method to solve for a stochastic policy $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$ which optimizes the expected return:

$$J(\pi, \rho_0) = \mathop{\mathbb{E}}_{\tau \sim \pi}[\sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t) | \rho_0] \tag{6}$$

where $\mathcal{R}(s_t, a_t)$ is the reward at time t, $\gamma \in [0, 1]$ is the discount factor, and T is the length of the rollouts in the episode. In this setting, the value (V), action-value (Q), and advantage function (A) are defined as:

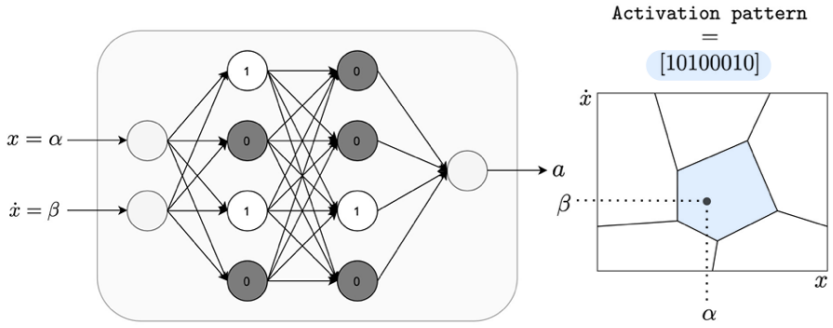


Figure 5: An overview of the proposed visualization framework. Policy network has 2 hidden layers with 4 units each. Input is 2-dimensional and the output is a scalar number. In our problem setup, first input value is the position of the agent x, the second input is the velocity of the agent $\dot{x}$, and the output is the acceleration of the agent a. Shaded hidden units have an activation value of 0 and non-shaded hidden units have an activation value of 1. The collection of these values is the activation pattern which is a unique binary code corresponding to each linear region. The collection of the activation patterns are then used to create a color map for visualizing the state space divisions.

$$V^{\pi}(s_t) = \mathop{\mathbb{E}}_{\tau \sim \pi}[\sum_{t'=t}^{T} \gamma^{t'-t} \mathcal{R}(s_{t'}, a_{t'}) | s_t] \tag{7}$$

$$Q^{\pi}(s_t, a_t) = \mathop{\mathbb{E}}_{\tau \sim \pi}[\sum_{t'=t}^{T} \gamma^{t'-t} \mathcal{R}(s_{t'}, a_{t'}) | s_t, a_t] \tag{8}$$

$$A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t). \tag{9}$$

4 METHOD

We develop a visualization framework to study the evolution of deep neural network policies during training. We also explore the effects of changes in network configuration, particularly, changes in network depth on the behavior of the deep RL agent. More specifically, we visualize how ReLU-activated deep RL policy networks divide the state space into linear regions and how these divisions evolve during training. To do so, we focus on a class of deep feed-forward neural network policies with ReLU activations. An overview of our visualization framework is shown in Figure 5 in which the policy network maps input states into output actions while dividing the state space into a number of linear regions.

To capture the linear regions, we focus our attention on the values of the ReLU activations in the policy network. We encode each hidden unit of the policy network with a binary code value of zero if its corresponding ReLU activation is in the flat region and with a one if its ReLU activation is in the sloped region. Therefore, the state of the policy network can be represented by a binary code, called an *activation pattern*, consisting of all the binary codes of its hidden units. By grouping states that produce the same activation pattern into a region, we then divide the state space into a number of linear regions. In order to visualize how the state space is divided into these linear regions, we assign a unique color code to each activation pattern (or region) and plot the state space with the created color map.

5 EXPERIMENTS

5.1 SETUP

We set up a simple continuous 2-dimensional dynamical system with scalar actions. In this environment, the agent starts at at the origin $x = 0$ with zero velocity $\dot{x} = 0$ and its goal is to move in a straight line in order to reach the goal state located at $x_g = 100$. Therefore, $s_t = \{x_t, \dot{x}_t\}$ is the state at time step t where x_t is agent's position and $\dot{x}_t$ is agent's velocity at that time step, and a_t, agent's acceleration, is the action at that time step. The input to our framework is the 2-dimensional state vector and the output is the scalar action.

We use proximal policy optimization (PPO) algorithm (Schulman et al., 2017) to train our model. The rewards at time step t is defined as below:

$$\begin{aligned}\mathcal{R}(s_t, a_t) = 1 - (&0.01 * \texttt{distance penalty} + \\ &0.001 * \texttt{action penalty} + \\ &0.01 * \texttt{max velocity penalty})\end{aligned}$$

where *distance penalty* is distance between current state and goal state ($|x_t - x_g|$), *action penalty* is the magnitude of acceleration ($|a_t|$), and *max velocity penalty* is the penalty of reaching the maximum velocity of 100 defined as:

$$\texttt{max velocity penalty} = \begin{cases} 1 & \texttt{if } ||\dot{x}_t| - \texttt{maximum velocity}| < 0.001 \\ 0 & \end{cases}$$

5.2 EVALUATION

As mentioned earlier, on top of visualizing how the state space is divided by the policy network, we also compute and analyze certain statistics of the generated linear regions. Metrics computed and evaluated here are:

- Number of generated linear regions: According to deep learning literature (Pascanu et al., 2013; Montúfar et al., 2014; Raghu et al., 2017; Hanin & Rolnick, 2019), the number of linear regions a model can generate is a measure of model's expressivity. Therefore, we count and report the number of generated linear regions of network policies.
- Number of visited linear regions: In every episode, the agent starts at the starting position of $x = 0$ and performs a number of actions until the episode terminates. By collecting the visited states and their corresponding visited regions during each episode, we count and report the number of these regions. To do so, we sample 10 stochastic trajectories and report the mean and the standard deviation of the number of visited regions from these 10 trajectories.
- Ratio of visited linear regions: This is the ratio of the number of visited regions divided by the number of generated regions. This can be intuitively interpreted as how *smart* the policy is and how efficient it is in generating cells that are actually visited and used during training.

Note that we consider a confined window of the state space for creating and counting the number of generated and visited linear regions. Figure 6 shows the state space divisions of a 2-layered policy network with 4 neurons in each layer. The black line on top of the divisions is the trajectory of visited states during training. In the confined window of the state space visualized, the number of generated linear regions for this trained policy is 14, and the number of visited linear regions for this trained policy is 9.

We only study certain network configurations here. We use the notation $d \times m$ to indicate a network configuration of a d-layered network with m hidden units in each layer. For all the experiments, we report the results from 5 trials of each experiment with 5 different random seeds. Therefore, the reported metrics are the mean value of the metrics over the 5 runs.

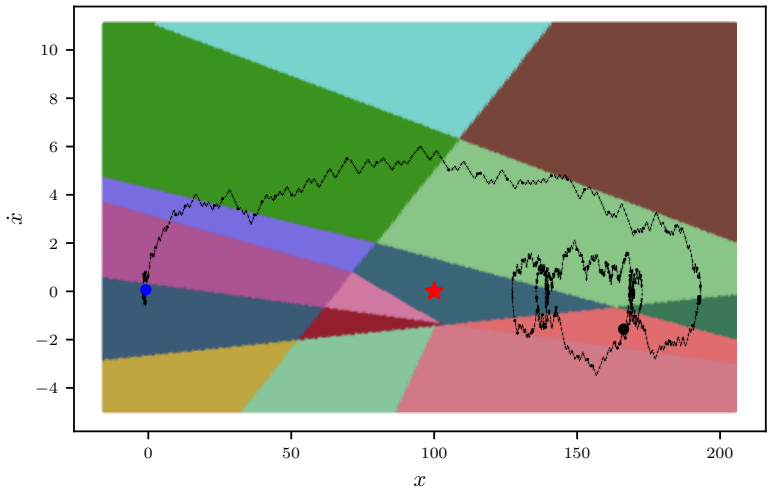


Figure 6: State space divisions of a two-layered policy network with 4 units in each layer which is trained for only 10 epochs for our simple 2-dimensional continuous control problem. Trajectory of visited states during training is visualized with a black line on top of the divisions. Starting state is marked with a blue circle, goal state is marked with a red star, and the stop point of the trajectory is marked with a black circle. Here, the number of generated regions is 14 while the number of visited regions is 9 (considering that the policy visits top right region which is colored with brown).

5.3 Qualitative Analysis

We start by visualizing the divisions of a 2-layered policy network with equal number of hidden units in each layer. Figure 7 shows the division plots of an untrained, partially trained and fully trained policy network with $n \in \{2, 4, 8, 16, 32\}$ neurons in each layer. The black line on top of the state space divisions shows the trajectory of visited states during training. Note that since we consider the confined window of $x \in [-50, 150]$ and $\dot{x} \in [-20, 20]$, there may exist regions outside of the boundaries we are visualizing. Moreover, when policy is not sufficiently trained, it may overshoot far passed the goal state or completely go to the wrong direction and away from the goal state. In conditions like these, the trajectory does not fully lie inside the boundaries visualized.

From these results, we can see that the state space becomes more fine-grained near the goal state and where the trajectory of visited states lies. This means that the number of visited regions along trajectories will increase which is intuitive in that the policy network focuses on where it actually goes.

5.4 Quantitative Analysis

In what follows, we study the affect of depth in practice for our RL problem and compute and report the number of generated linear regions, the number of visited linear regions during training as well as the ratio of the number of visited linear regions to the number of generated linear regions for a deep policy network.

5.4.1 Fixed Number of Hidden Units

In this section, we look at the effect of policy network configuration on the behavior of the agent while the number of hidden units in the policy network is kept fixed. The value network is 2×64. For the policy network, we have 4 sets of configurations. In each set, the total number of neurons ($n \in \{4, 8, 16, 32, 12, 24, 48, 64\}$) in the network is kept fixed while the depth changes from 1 to d such that the width of all layers remains equal. For instance, for $n = 4$, we have two configurations: $\{1 \times 4, 2 \times 2\}$ networks and for $n = 8$ we have 3 configurations: $\{1 \times 8, 2 \times 4, 4 \times 2\}$ and so on.

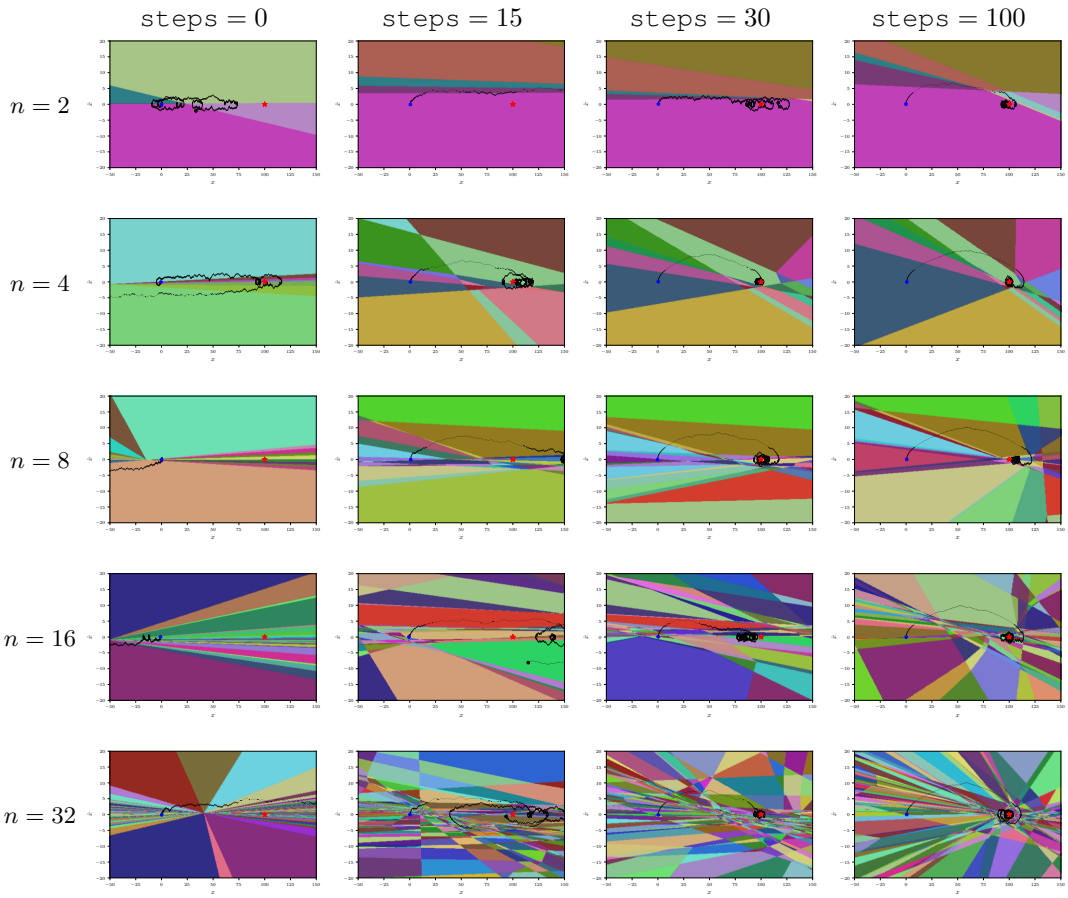
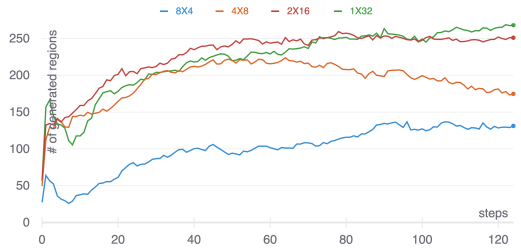


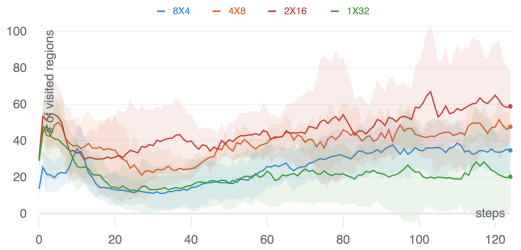
Figure 7: State space divisions of policy networks with 2 hidden layers having $n \in \{2, 4, 8, 16, 32\}$ hidden units in each layer. Plot at row $n = i$ and column $\texttt{steps} = j$, shows the results of a network with i neurons in each hidden layer which is trained for j steps. $\texttt{steps} = 0$ corresponds to the policy before training and $\texttt{steps} = 100$ corresponds to the fully-trained policy. Black line on top of the plots shows the trajectory of visited states during training. The red star indicates the goal state of $[100, 0]$.

We repeat training each policy network with 5 different random seeds and report the average of all the metrics over these 5 runs.

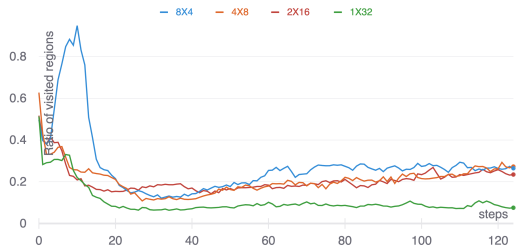
Figure 8 and Figure 9 show the metric evolution plots for a policy network with $n = 32$ and $n = 64$ hidden units respectively. The rest of the plots are provided at the Appendix section. From these results, we can see that as depth increases, in all cases, the number of generated linear regions decreases. This is in contrast with the findings of the deep learning literature (Pascanu et al., 2013; Montúfar et al., 2014; Raghu et al., 2017; Hanin & Rolnick, 2019) where the number of linear regions increases with depth. We can conclude that in practice, for the simple experimental setup we have with a 2-dimensional continuous control system, the critical nature of depth behaves differently from deep learning. In our setting, increasing the depth of the policy network causes a decrease in the number of generated linear regions. Looking back at the results, we can see that the number of visited linear regions increase as depth increases. Moreover, the ratio of visited to generated linear regions increases with depth. Here, we have a hypothesis that in deep RL, the notion of expressivity is different from deep learning and the number of generated linear regions does not correspond to network expressivity. We hypothesize that expressivity in deep RL is correlated with the ratio of number of visited linear regions to the number of generated linear regions. That is why we see an in increase in this ratio as the depth of the policy network increases. Intuitively, this ratio shows how



(a) Generated linear regions



(b) Visited linear regions



(c) Ratio of visited linear regions

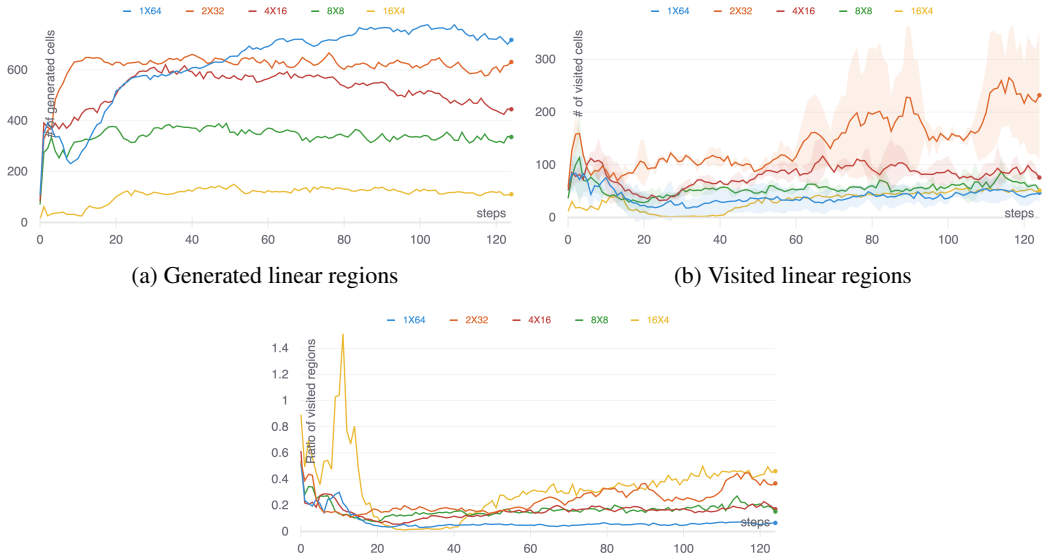
Figure 8: Evolution of expressivity metrics during training for different configurations of a policy with $n = 32$ hidden units. (a) shows the number of generated linear regions during training. We can see that the number of generated linear regions grows as training continues and that overall, deeper networks have less number of generated regions. (b) shows the mean and the standard deviation of the number of visited linear regions during training (considering 10 sampled trajectories). We can see that the number of visited linear regions grows during training. (c) shows the ratio of the number of visited linear regions to the number of generated linear regions during training. We can see that overall, this ratio is higher for deeper networks compared to the shallower ones.

many of the generated regions are used by the policy network and it can be interpreted as how smart the policy network actually is.

5.4.2 Fixed Number of Policy Network Parameters

We can argue that in the experiments of Section 5.4.1, results are not solely reflecting the effect of policy network depth on the metrics. That is because in that setting, changes in the network depth result huge changes in the number of policy network parameters. In this section, we look at the effect of policy network configuration on the behavior of the agent while the number of policy network parameters does not change as much. The value network is again 2×64. We start with training policy networks with different configurations that have close enough number of parameters. Policy networks studied here are $1 \times 8, 2 \times 4$ with 33 and 37 parameters respectively, $2 \times 6, 4 \times 4$ with 67 and 77 parameters respectively, $1 \times 12, 4 \times 3$ both with 49 parameters, $1 \times 24, 8 \times 3$ both with 97 parameters, $1 \times 64, 4 \times 8$ with 257 and 249 parameters respectively, and $16 \times 4, 8 \times 6$ with 317 and 319 parameters respectively. Figure 10 and Figure 11 show the metrics evolution plots for $1 \times 12, 4 \times 3$ policy networks, and $1 \times 64, 4 \times 8$ policy networks respectively during training. The rest of the plots are provided at the Appendix section.

From these, similar to Section 5.4.1, we can see that with equal or close number of parameters, increasing depth also causes a decrease in the number of generated linear regions. About the number of visited regions, here we see that the number of visited linear regions does not follow a particular pattern with respect to changes in policy network depth. We can say the number of visited regions remains almost the same as depth changes. Whereas in the results of Section 5.4.1, we saw a general increase in the number of visited linear regions with depth. This exactly shows that the increase seen in previous results of Section 5.4.1 was a result of an increase in the number of network parameters rather than the network depth. The results here are in agreement with the results of Section 5.4.1 as they both show an increase in the ratio of visited to generated linear regions as depth increases. We now make a new hypothesis that the number of visited cells is correlated with



(a) Generated linear regions

(b) Visited linear regions

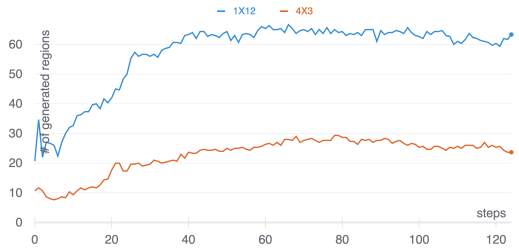
(c) Ratio of visited linear regions

Figure 9: Evolution of expressivity metrics during training for different configurations of a policy with $n = 64$ hidden units. (a) shows the number of generated linear regions during training. We can see that deeper networks have less number of generated regions. (b) shows the mean and the standard deviation of the number of visited linear regions during training (considering 10 sampled trajectories). We can see that the number of visited linear regions slightly grows during training. (c) shows the ratio of the number of visited linear regions to the number of generated linear regions during training. We can see that overall, this ratio is higher for deeper networks compared to the shallower ones.

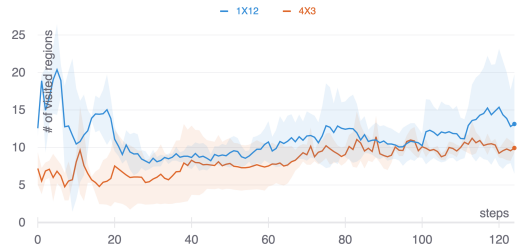
the number of parameters in the policy network.

To further study the effect of depth on policy network training, and to assess some of our hypotheses, we design another experiment. We start by considering a wide base policy and training it. To increase the depth, we add narrow layers to it such that the number of parameters does not change much while the depth changes. More specifically, we consider a policy network with 2 hidden layers where the first layer has 32 hidden units and the second has 2 hidden units. We then keep adding a new layer with 2 hidden units between the 32-unit layer and the output layer. In this setting, adding additional layers increases the number of parameters by 3 which is negligible comparing to the total number of parameters which was 165 (for a policy with 32 units in first layer followed by a 2-unit layer) before adding the first new layer. We also repeated this experiment for a base policy with 64 hidden units in the first, and similar to before, we keep adding layers of width 2 before the output layer to this network.

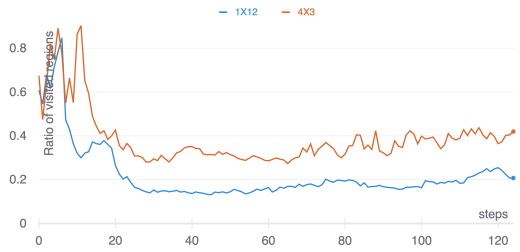
In this experiment, instead of focusing on the evolution of metrics during training like we did in previous parts, we look at fully trained snapshots of the policy and study how the metrics change with respect to the depth of the policy network. Figure 12 shows the results for the case where the base policy has a single 32-unit layer while layers with width 2 keep being added as the final hidden layer (before the output layer) of the network. Figure 13 shows the results for the case where the base policy has a 64-unit layer while layers with width 2 keep being added as the final hidden layer (before the output layer) of the network. We have repeated the same experiment while adding the narrow layers to the beginning of the policy network instead of the end and results are consistent with what we report here. These plots are available in the Appendix. Note that for these set of experiments, we repeat training each policy configuration 5 times with 5 different random seeds and report the mean value of the metrics over the 5 runs.



(a) Generated linear regions



(b) Visited linear regions



(c) Ratio of visited linear regions

Figure 10: Evolution of expressivity metrics during training for a 1×12 policy network and a 4×3 policy network both having a total of 49 parameters. (a) shows the number of generated linear regions during training. We can see that the deeper network has less number of generated regions. (b) shows the mean and the standard deviation of the number of visited linear regions during training (considering 10 sampled trajectories). We can see that the number of visited linear regions slightly grows during training. (c) shows the ratio of the number of visited linear regions to the number of generated linear regions during training. We can see that overall, this ratio is higher for the deeper network compared to the shallower one.

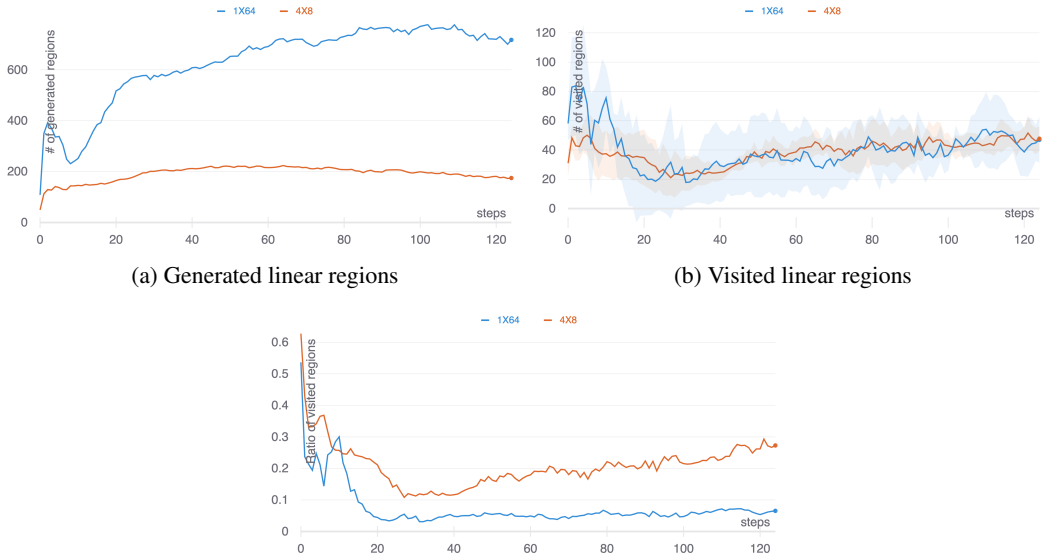
The relation seen in these results between the number of generated linear regions and depth is consistent with previous experiments in that depth decreases the number of generated linear regions. And, unlike previous results, where the number of visited cells usually increased with depth in Section 5.4.1 and remained almost constant with depth in Section 5.4.2, here we see a general pattern of decrease in the number of visited cells as depth increases. Similar to previous experiments, we see an increase in the ratio of the number of visited linear regions to the number of generated linear regions as depth of the policy network increases.

5.4.3 Value Network Configuration

In this section, we study the effect of value network configuration on the behavior of the agent. For this, policy network configuration is kept fixed while the value function network configuration changes. Particularly, we have set the policy network to be 2×8. For the value function network, we have 4 sets of configurations. In each set, the total number of neurons ($n \in \{32, 64, 128, 256\}$) in the network is kept fixed while the depth changes from 1 to 4 while the width of all layers is kept fixed. For instance, for $n = 32$, we have 4 configurations: $\{1 \times 32, 2 \times 16, 4 \times 8, 8 \times 4\}$ and for $n = 64$ we have 4 configurations: $\{1 \times 64, 2 \times 32, 4 \times 16, 8 \times 8\}$ and so on. We repeat training each policy network with 5 different random seeds and report he average of all the metrics over these 5 runs.

Figure 14 and Figure 15 show the metric evolution plots for a value function network with $n = 32$ and $n = 64$ hidden units respectively. The rest of the plots are attached to the Appendix section.

From the results, we can see that when value network is expressive enough, changes in width and depth of the value network do not change the statistics of the linear regions and that these statistics are only affected by the policy network configuration.



(a) Generated linear regions

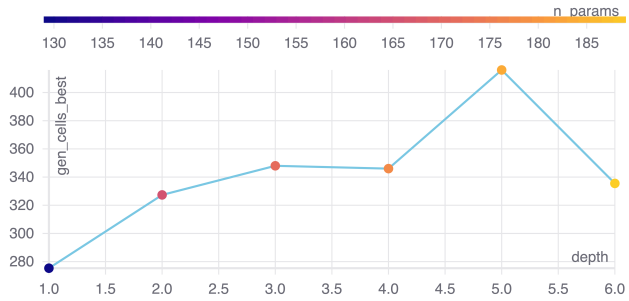
(b) Visited linear regions

(c) Ratio of visited linear regions

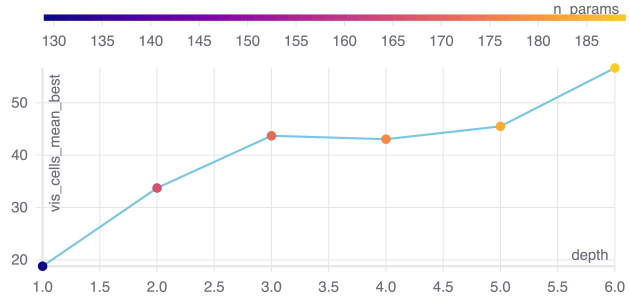
Figure 11: Evolution of expressivity metrics during training for a 1×64 policy network and a 4×8 policy network having a total of 257 and 249 parameters respectively. (a) shows the number of generated linear regions during training. We can see that the deeper network has less number of generated regions. (b) shows the mean and the standard deviation of the number of visited linear regions during training (considering 10 sampled trajectories). We can see that the number of visited linear regions slightly grows during training. (c) shows the ratio of the number of visited linear regions to the number of generated linear regions during training. We can see that this ratio is higher for the deeper network compared to the shallower one.

6 Discussion

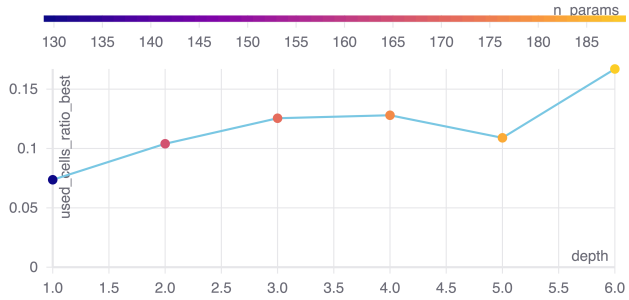
Characterizing the expressive power of deep RL models, and understanding how expressivity varies with policy network configuration, is an important problem that is fairly unexplored in the RL research community. In this work, we have presented a visualization framework to study how ReLU-activated deep RL policy networks view the state space, and how this view evolves during training. Our qualitative results show that the policy network focuses on the areas of the state space that it visits during training, and thus, produces more fine-grained linear regions over these areas compared with other areas of the state space. We also study the relation of policy network configuration and three metrics related to network expressivity. Our results show that increasing network depth causes a decrease in the number of generated linear regions. This is in contrast with the findings of the deep learning literature that advocate the use of deeper networks because of their higher expressive power (certain works show that the number of generated linear regions increase with depth specifically). We can conclude that in practice, for the simple experimental setup we have with a 2-dimensional continuous control system, the critical nature of depth behaves differently from deep learning and depth causes a decrease in the number of generated linear regions. Moreover, our results show that the ratio of the number of visited regions to the number of generated regions increases with depth. From these two findings, we conclude that expressivity in deep RL is correlated with the ratio of number of visited linear regions to the number of generated linear regions. Intuitively, this ratio shows how many of the generated regions are used by the policy network and it can be interpreted as how smart the policy network actually is. Our results also show that the number of visited regions during training is not correlated with policy network depth but rather with the number of parameters of the policy network. Our results finally show that as long as the value function network is expressive enough, the changes in the configuration of the value function network do not affect the three expressivity metrics.



(a) Number of generated regions v. depth v. number of parameters



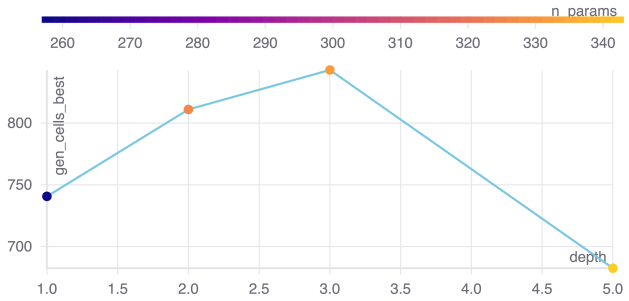
(b) Number of visited regions v. depth v. number of parameters



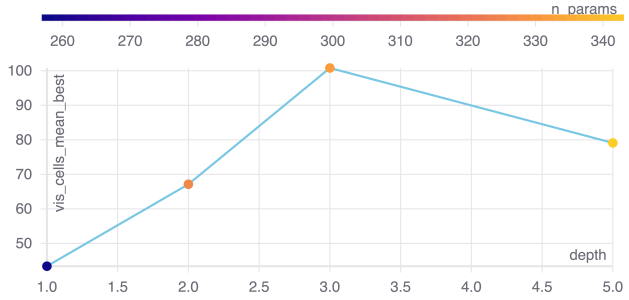
(c) Ratio of visited regions v. depth v. number of parameters

Figure 12: Results of fully-trained snapshots of 5 policies. The base policy has 32 units in a single layer and the following policies all have an additional 2-unit layer added to the previous policy, before the output layer, starting from the base policy. The blue point at $\texttt{depth} = 1$ corresponds to the base policy and the following points at $\texttt{depth} = 1 + i$ corresponds to the base policy plus i additional 2-unit layers added to it. The horizontal axis shows the depth of the network and the gradient line indicates the number of network parameters. The vertical axis shows in (a) the number of generated linear regions, in (b) the number of visited linear regions, and in (c) the ratio of visited linear regions to the number of generated regions. We can see that overall, the ratio of visited regions grows with depth.

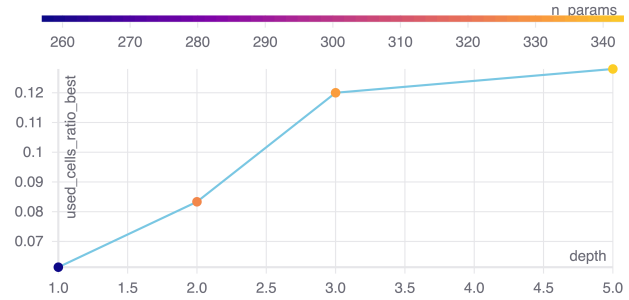
To conclude, this work provides a new visualization technique for deep RL problems. In addition, it helps us better understand deep RL policies and their optimization landscape. We generally believe this work could help spur discussion about techniques for better understanding deep RL algorithms and visualizing the learning process they follow. This work raises many interesting directions for future work as well. The first direction to explore would be to overcome some of the caveats of this work. For instance, all the metrics reported in this work are taken from a confined boundary of the state space. This may cause errors as some of the areas of the state space are not considered at all. Providing a solution to this limitation and repeating the experiments of this work will provide generalized results that are more insightful compared to ours. Another caveat is that we have experimented with only a simple 2-dimensional continuous RL problem. It would be interesting to repeat the same experiments on multiple higher dimensional problems to gain confidence about the findings of this work. In addition, to extract the linear regions, we sample points from the region



(a) Number of generated regions v. depth v. number of parameters



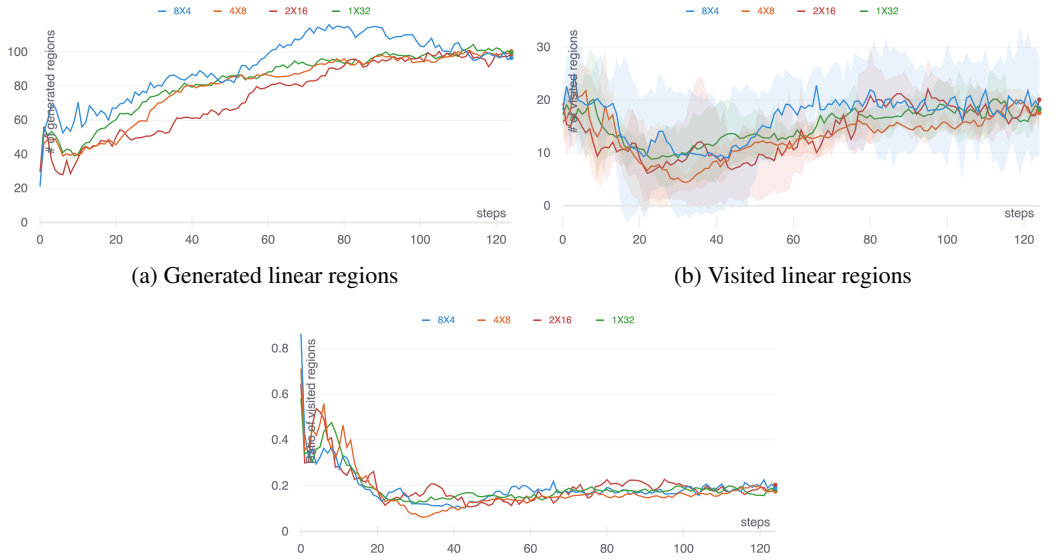
(b) Number of visited regions v. depth v. number of parameters



(c) Ratio of visited regions v. depth v. number of parameters

Figure 13: Results of fully-trained snapshots of 4 policies. The base policy has 64 units in a single layer and the following policies all have an additional 2-unit layer added to the previous policy, before the output layer, starting from the base policy. The blue point at $\texttt{depth} = 1$ corresponds to the base policy and the following points at $\texttt{depth} = 1 + i$ corresponds to the base policy plus i additional 2-unit layers added to it. The horizontal axis shows the depth of the network and the gradient line indicates the number of network parameters. The vertical axis shows in (a) the number of generated linear regions, in (b) the number of visited linear regions, and in (c) the ratio of visited linear regions to the number of generated regions. Note that the policy with 64 units in first layer followed by 3, 2-unit layers ($\texttt{depth} = 4$) is not included in the plots. That is because this configuration failed to converge in all 5 repeats of the experiments with 5 different random seeds. We can see that overall, the ratio of visited regions grows with depth.

boundaries, extract the activation patterns for each state, and group states with similar activation pattern into a linear region. With this method, our calculations are not exact and we may miss small regions. It is necessary to use an alternate method for calculating the linear regions to overcome this caveat. One interesting direction to explore is to see whether we can characterize a policy by the statistics of its linear regions (e.g. the number of generated linear regions, the number of visited regions, the shape of the regions or the complexity of the regions). If yes, then we can compare two policies using the statistics of their resulting regions rather than just their final performances.



(a) Generated linear regions

(b) Visited linear regions

(c) Ratio of visited linear regions

Figure 14: Evolution of expressivity metrics during training for different configurations of a value function network with $n = 32$ hidden units and a 2×8 policy. (a) shows the number of generated linear regions during training. We can see that the number of generated linear regions grows during training. (b) shows the mean and the standard deviation of the number of visited linear regions during training (considering 10 sampled trajectories). We can see that the number of visited linear regions slightly grows during training. (c) shows the ratio of the number of visited linear regions to the number of generated linear regions during training.

ACKNOWLEDGMENTS

REFERENCES

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*, pp. 233–242. PMLR, 2017.

Lei Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? *arXiv preprint arXiv:1312.6184*, 2013.

Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*, 28(3):653–664, 2016.

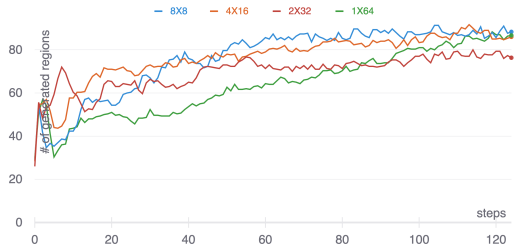
Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *arXiv preprint arXiv:1811.12560*, 2018.

Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. Learning to fly by crashing. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3948–3955. IEEE, 2017.

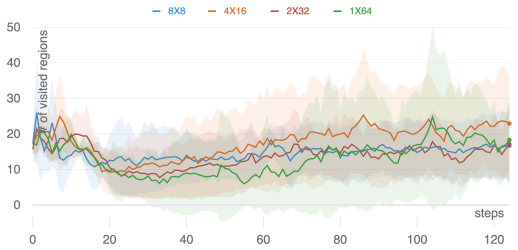
Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

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

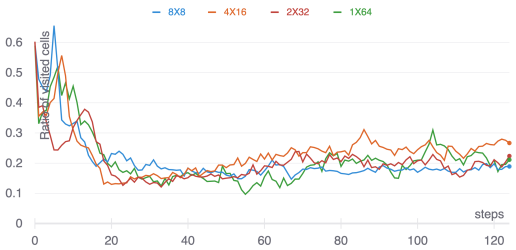
Mikael Johansson. *Piecewise linear control systems*. PhD thesis, Ph. D. Thesis, Lund Institute of Technology, Sweden, 1999.



(a) Generated linear regions



(b) Visited linear regions



(c) Ratio of visited linear regions

Figure 15: Evolution of expressivity metrics during training for different configurations of a value function network with $n = 64$ hidden units and a 2×8 policy. (a) shows the number of generated linear regions during training. We can see that the number of generated linear regions grows during training. (b) shows the mean and the standard deviation of the number of visited linear regions during training (considering 10 sampled trajectories). We can see that the number of visited linear regions slightly grows during training. (c) shows the ratio of the number of visited linear regions to the number of generated linear regions during training.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Douglas J Leith and William E Leithead. Survey of gain-scheduling analysis and design. *International journal of control*, 73(11):1001–1025, 2000.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

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.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PMLR, 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.

Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

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

Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952*, 2017.

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.

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*, pp. 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*, pp. 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, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015.

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

Christin Seifert, Aisha Aamir, Aparna Balagopalan, Dhruv Jain, Abhinav Sharma, Sebastian Grottel, and Stefan Gumhold. Visualizations of deep neural networks in computer vision: A survey. In *Transparent data mining for big and small data*, pp. 123–144. Springer, 2017.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

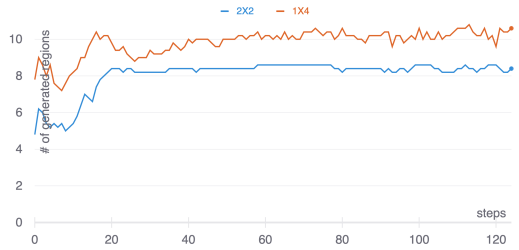
Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pp. 1057–1063, 2000.

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*, pp. 264–274. Springer, 2019.

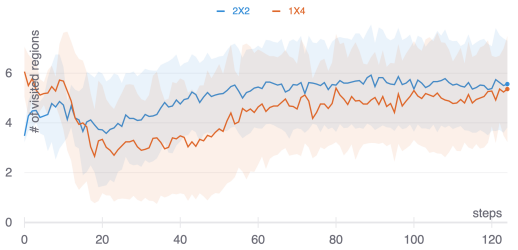
Tom Zahavy, Nir Ben-Zrihem, and Shie Mannor. Graying the black box: Understanding dqns. In *International Conference on Machine Learning*, pp. 1899–1908. PMLR, 2016.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.

Martin Zurowietz and Tim W Nattkemper. An interactive visualization for feature localization in deep neural networks. *Frontiers in Artificial Intelligence*, 3(49), 2020.



(a) Generated linear regions



(b) Visited linear regions

(c) Ratio of visited linear regions

Figure 16: Evolution of expressivity metrics during training for different configurations of a policy with $n = 4$ hidden units. (a) shows the number of generated linear regions during training. We can see that the deeper networks have less number of generated regions. (b) shows the mean and the standard deviation of the number of visited linear regions during training (considering 10 sampled trajectories). We can see that the number of visited linear regions grows during training. (c) shows the ratio of the number of visited linear regions to the number of generated linear regions during training. We can see that this ratio is higher for the deeper networks compared to the shallower ones.

A Appendix

A.1 Additional Experiments from Section 5.4.1

Figures 16-20 show the results of additional experiments from Section 5.4.1 where we study the effect of depth on the expressivity metrics while the number of hidden units is kept fixed.

A.2 Additional Experiments from the First Set of Experiments in Section 5.4.2

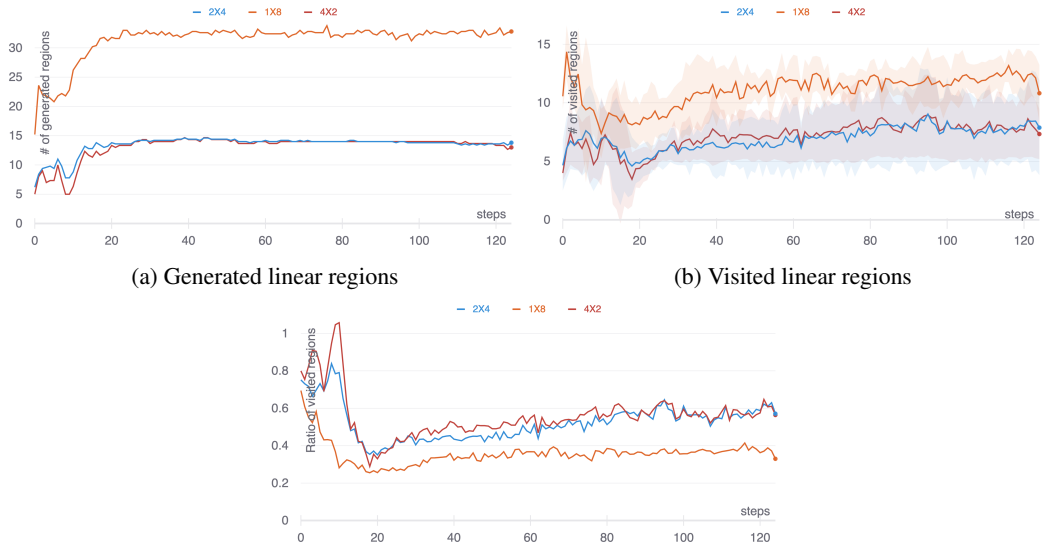
Figures 21-24 show the results of additional experiments from the first set of experiments in Section 5.4.2 where we compare the evolution of expressivity metrics for policy networks with close number of network parameters but different depths.

A.3 Additional Experiments from the Second Set of Experiments in Section 5.4.2

Figures 25 and 26 show the results of additional experiments from the second set of experiments in Section 5.4.2 where we compare the value of expressivity metrics of a policy network while increasing the depth of the network while keeping its number of parameters from growing too much. In this experiment, the goal is to study the effect of depth on the expressivity metrics.

A.4 Additional Experiments from Section 5.4.3

Figures 27 and 28 show the results of additional experiments from Section 5.4.2 where we study the effect of value function network configuration on the behaviour of the policy during training.

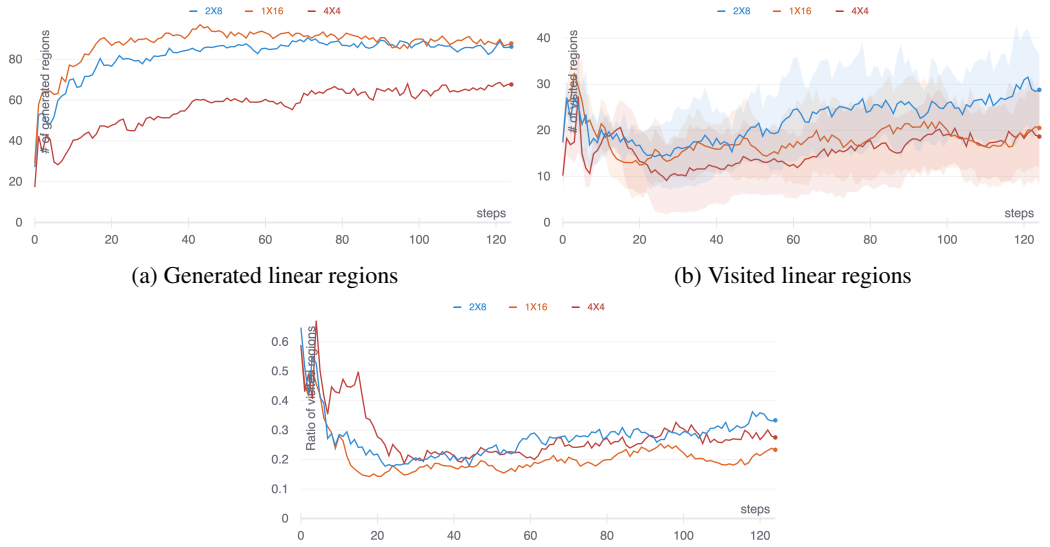


(a) Generated linear regions

(b) Visited linear regions

(c) Ratio of visited linear regions

Figure 17: Evolution of expressivity metrics during training for different configurations of a policy with $n = 8$ hidden units. (a) shows the number of generated linear regions during training. We can see that the deeper network have less number of generated regions. (b) shows the mean and the standard deviation of the number of visited linear regions during training (considering 10 sampled trajectories). We can see that the number of visited linear regions grows during training. (c) shows the ratio of the number of visited linear regions to the number of generated linear regions during training. We can see that this ratio is higher for the deeper networks compared to the shallower ones. The reason why the results of the 4×2 policy are not expressive enough is that with a deep network with narrow layers, we usually see that policy fails to converge.

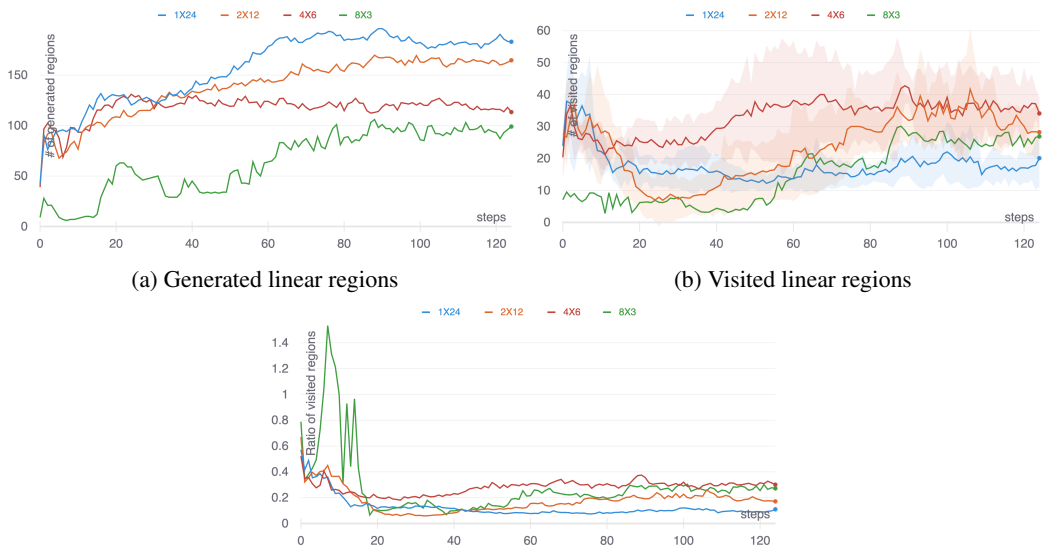


(a) Generated linear regions

(b) Visited linear regions

(c) Ratio of visited linear regions

Figure 18: Evolution of expressivity metrics during training for different configurations of a policy with $n = 16$ hidden units. (a) shows the number of generated linear regions during training. We can see that the deeper networks have less number of generated regions and that the number of generated regions grows with training. (b) shows the mean and the standard deviation of the number of visited linear regions during training (considering 10 sampled trajectories). We can see that the number of visited linear regions grows during training. (c) shows the ratio of the number of visited linear regions to the number of generated linear regions during training. We can see that overall, this ratio is higher for the deeper networks compared to the shallower ones.

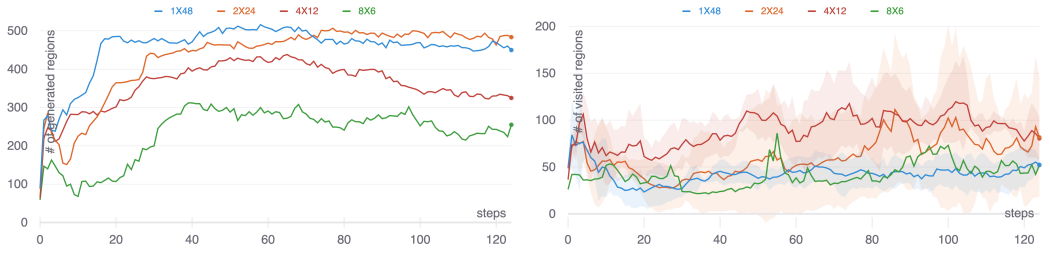


(a) Generated linear regions

(b) Visited linear regions

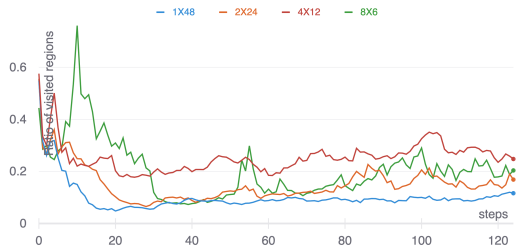
(c) Ratio of visited linear regions

Figure 19: Evolution of expressivity metrics during training for different configurations of a policy with $n = 24$ hidden units. (a) shows the number of generated linear regions during training. We can see that the deeper networks have less number of generated regions and that the number of generated regions grows with training. (b) shows the mean and the standard deviation of the number of visited linear regions during training (considering 10 sampled trajectories). (c) shows the ratio of the number of visited linear regions to the number of generated linear regions during training. We can see that overall, this ratio is higher for the deeper networks compared to the shallower ones.



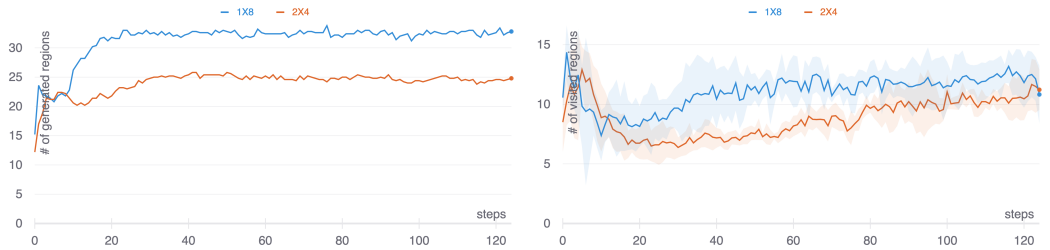
(a) Generated linear regions

(b) Visited linear regions



(c) Ratio of visited linear regions

Figure 20: Evolution of expressivity metrics during training for different configurations of a policy with $n = 48$ hidden units. (a) shows the number of generated linear regions during training. We can see that the deeper networks have less number of generated regions and that the number of generated regions grows with training. (b) shows the mean and the standard deviation of the number of visited linear regions during training (considering 10 sampled trajectories). (c) shows the ratio of the number of visited linear regions to the number of generated linear regions during training. We can see that overall, this ratio is higher for the deeper networks compared to the shallower ones.

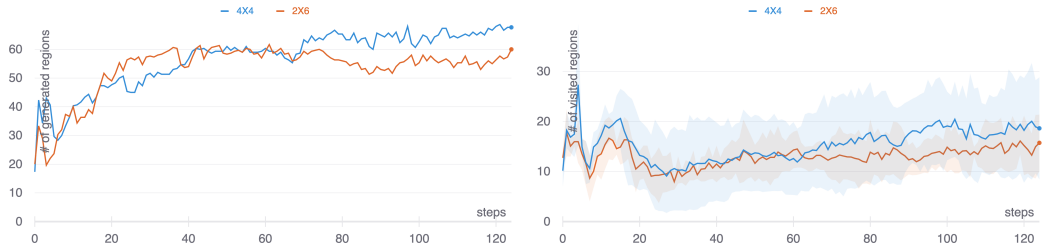


(a) Generated linear regions

(b) Visited linear regions

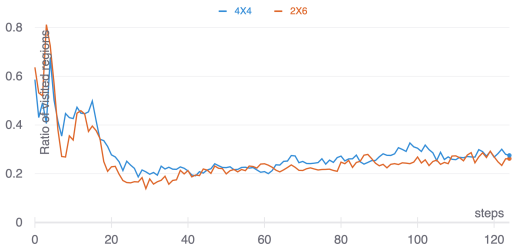
(c) Ratio of visited linear regions

Figure 21: Evolution of expressivity metrics during training a 1×8 policy network and a 2×4 policy network having a total of 33 and 37 parameters respectively. (a) shows the number of generated linear regions during training. We can see that the deeper network has less number of generated regions. (b) shows the mean and the standard deviation of the number of visited linear regions during training (considering 10 sampled trajectories). We can see that the number of visited regions grows with training. (c) shows the ratio of the number of visited linear regions to the number of generated linear regions during training. We can see that this ratio is slightly higher for the deeper network compared to the shallower one.



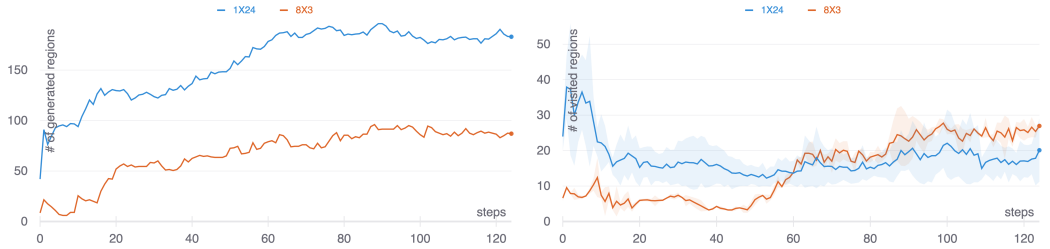
(a) Generated linear regions

(b) Visited linear regions



(c) Ratio of visited linear regions

Figure 22: Evolution of expressivity metrics during training a 2×6 policy network and a 4×4 policy network having a total of 67 and 77 parameters respectively. (a) shows the number of generated linear regions during training. We can see that the deeper network has less number of generated regions. (b) shows the mean and the standard deviation of the number of visited linear regions during training (considering 10 sampled trajectories). We can see that the number of visited regions slightly grows with training. (c) shows the ratio of the number of visited linear regions to the number of generated linear regions during training. We really do not see this ratio being higher for the deeper network compared to the shallower one in this example.

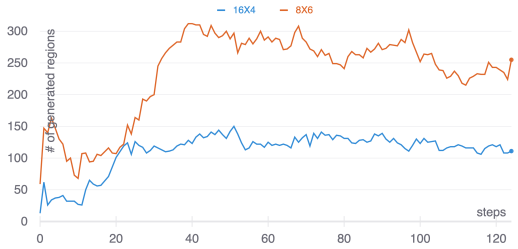


(a) Generated linear regions

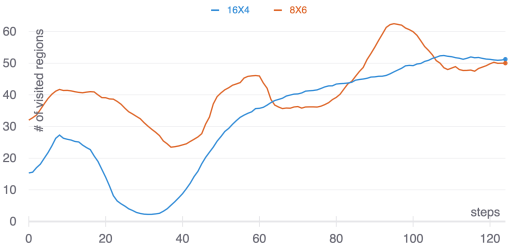
(b) Visited linear regions

(c) Ratio of visited linear regions

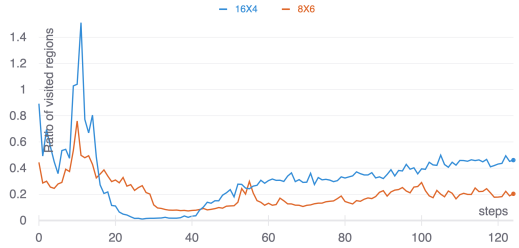
Figure 23: Evolution of expressivity metrics during training a 1×24 policy network and a 8×3 policy network both having a total of 97 parameters. (a) shows the number of generated linear regions during training. We can see that the deeper network has less number of generated regions and that the number of generated regions grows with training. (b) shows the mean and the standard deviation of the number of visited linear regions during training (considering 10 sampled trajectories). We can see that the number of visited regions slightly grows with training. (c) shows the ratio of the number of visited linear regions to the number of generated linear regions during training. We can see this ratio is higher for the deeper network compared to the shallower one.



(a) Generated linear regions

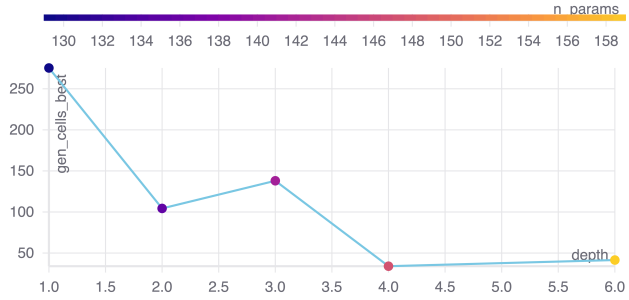


(b) Visited linear regions

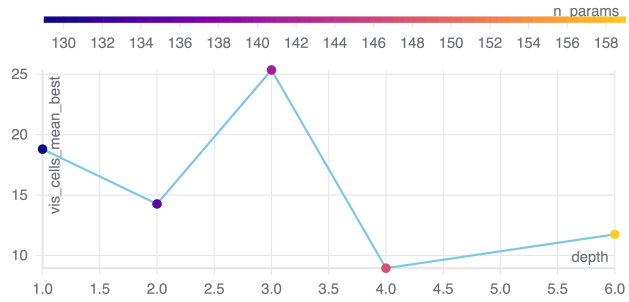


(c) Ratio of visited linear regions

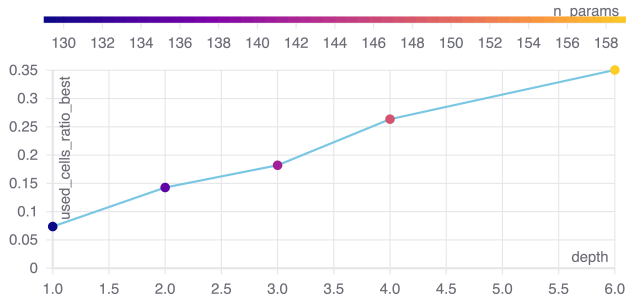
Figure 24: Evolution of expressivity metrics during training a 8×6 policy network and a 16×4 policy network having a total of 317 and 319 parameters respectively. (a) shows the number of generated linear regions during training. We can see that the deeper network has less number of generated regions. (b) shows the mean number of visited linear regions during training (considering 10 sampled trajectories). We can see that the number of visited regions grows with training. (c) shows the ratio of the number of visited linear regions to the number of generated linear regions during training. We can see this ratio is higher for the deeper network compared to the shallower one.



(a) Number of generated regions v. depth v. number of parameters

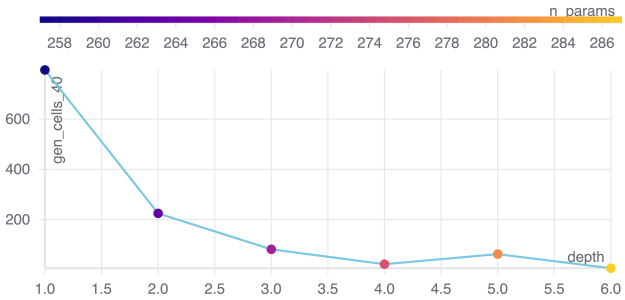


(b) Number of visited regions v. depth v. number of parameters

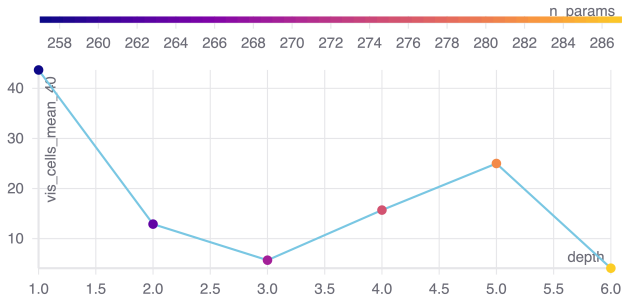


(c) Ratio of visited regions v. depth v. number of parameters

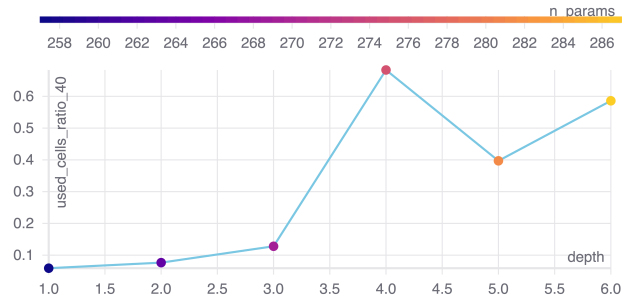
Figure 25: Results of fully-trained snapshots of 5 policies. The base policy has 32 units in a single layer and the following policies all have an additional 2-unit layer added to the previous policy, after the input layer, starting from the base policy. The blue point at $\texttt{depth} = 1$ corresponds to the base policy and the following points at $\texttt{depth} = 1 + i$ corresponds to the base policy plus i additional 2-unit layers added to it. The horizontal axis shows the depth of the network and the gradient line indicates the number of network parameters. The vertical axis shows in (a) the number of generated linear regions, in (b) the number of visited linear regions, and in (c) the ratio of visited linear regions to the number of generated regions. We can see that the ratio of visited regions grows with depth.



(a) Generated linear regions v. depth v. number of parameters

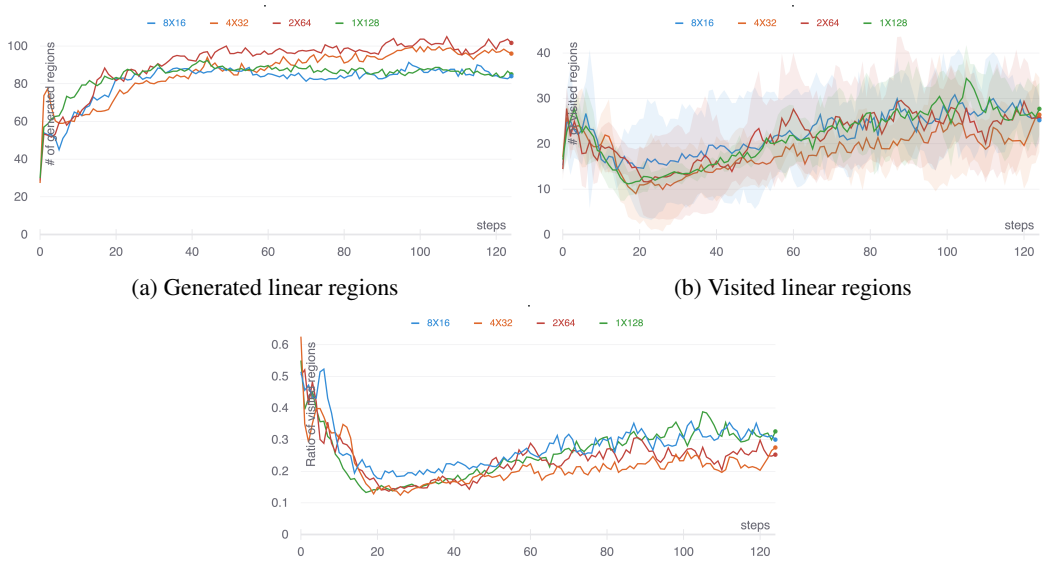


(b) Number of visited regions v. depth v. number of parameters



(c) Ratio of visited regions v. depth v. number of parameters

Figure 26: Results of fully-trained snapshots of 5 policies. The base policy has 64 units in a single layer and the following policies all have an additional 2-unit layer added to the previous policy, after the input layer, starting from the base policy. The blue point at $\texttt{depth} = 1$ corresponds to the base policy and the following points at $\texttt{depth} = 1 + i$ corresponds to the base policy plus i additional 2-unit layers added to it. The horizontal axis shows the depth of the network and the gradient line indicates the number of network parameters. The vertical axis shows in (a) the number of generated linear regions, in (b) the number of visited linear regions, and in (c) the ratio of visited linear regions to the number of generated regions. We can see that overall, the ratio of visited regions grows with depth.

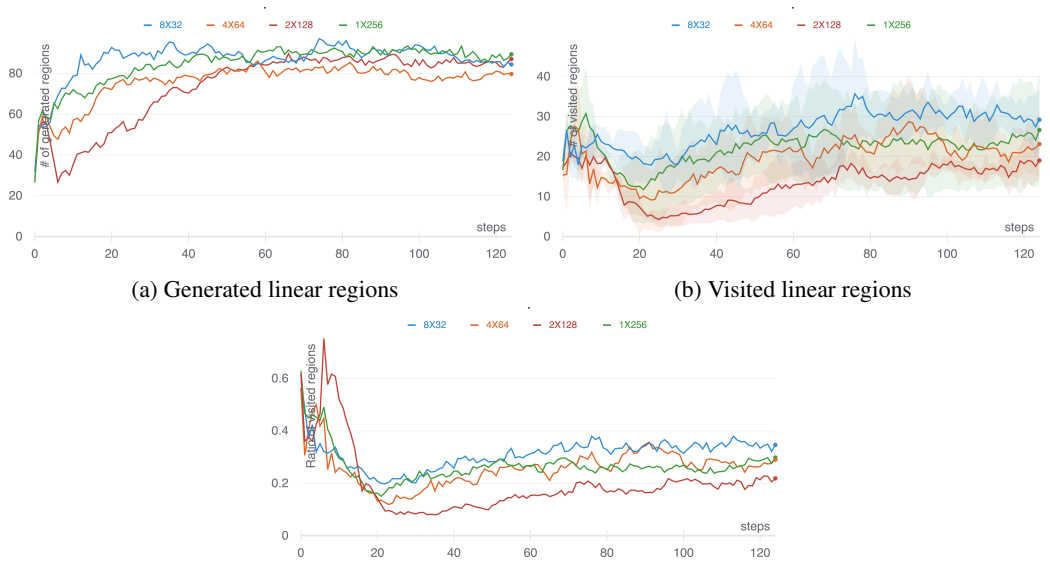


(a) Generated linear regions

(b) Visited linear regions

(c) Ratio of visited linear regions

Figure 27: Evolution of expressivity metrics during training for different configurations of a 2×8 policy and a value function network with $n = 128$ hidden units. (a) shows the number of generated linear regions during training. We can see that the number of generated linear regions grows during training. (b) shows the mean and the standard deviation of the number of visited linear regions during training (considering 10 sampled trajectories). We can see that the number of visited linear regions slightly grows during training. (c) shows the ratio of the number of visited linear regions to the number of generated linear regions during training.



(a) Generated linear regions

(b) Visited linear regions

(c) Ratio of visited linear regions

Figure 28: Evolution of expressivity metrics during training for different configurations of a 2×8 policy and a value function network with $n = 256$ hidden units. (a) shows the number of generated linear regions during training. We can see that the number of generated linear regions grows during training. (b) shows the mean and the standard deviation of the number of visited linear regions during training (considering 10 sampled trajectories). We can see that the number of visited linear regions slightly grows during training. (c) shows the ratio of the number of visited linear regions to the number of generated linear regions during training.