university of groningen / faculty of science and engineering / artificial intelligence

On The Existence of Double Descent in Reinforcement Learning

Bachelor's Project Thesis

Richard Harnisch, s5238366, r.f.harnisch@student.rug.nl,
Supervisor: Dr. Matthia Sabatelli

Abstract: Double Descent (DD) is a test-performance phenomenon in which a deep model's performance on a test set worsens around the interpolation threshold (overfitting) but then improves again when increasing training episodes or model size. We implement an empirical experiment to test whether this phenomenon can appear in Reinforcement Learning. To this end, we train TRPO agents on a family of seeded grid-worlds with obstacles and evaluate the agent on held-out maps. Across model sizes and training duration, we observe a generalization gap appearing between performance on training and held-out testing maps. However, upon increasing the size of the model as well as the amount of training timesteps we do not observe a second descent regime in test performance. The results suggest that in this context, increased capacity and training do not recover generalization and motivate further experiments with different algorithms, architectures, and environment types.

1 Introduction

Double Descent is a well-studied phenomenon in supervised learning. When increasing model capacity, training time, or dataset size, the test risk of deep models begins to improve (decrease) with the train risk up until a "sweet spot" at which test risk is traditionally understood to be minimized. Past this point, increasing model capacity, training time, or dataset size leads to overfitting, a behavior where the model memorizes the training data and thus fails to generalize over unseen test data. In other words, a generalization gap grows between the test performance and the train performance. This behavior can be visualized as a U-shaped risk curve, as shown in Figure 1.1.

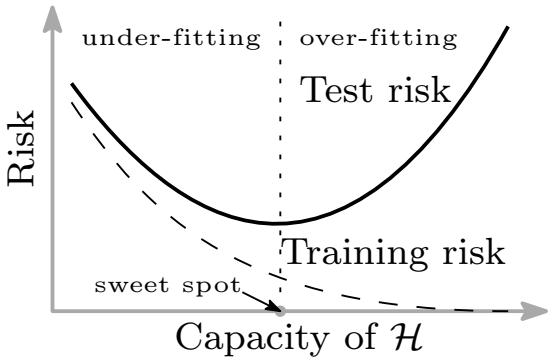


Figure 1.1: Traditional U-shaped risk curve. The training and test risk initially decrease together, until the model begins overfitting. Figure taken from Belkin et al. (2019).

However, work in the past years has shown that this traditional U-shaped risk curve is not the end of the story. Instead, after this initial overfitting phase, increasing model capacity, training time, or dataset size further leads to a second descent in test risk (Belkin et al., 2019; Nakkiran et al., 2019), as can be seen in Figure 1.2. This phenomenon has been coined "Double Descent" (DD) and has been observed across various architectures and datasets in supervised learning (Nakkiran et al., 2019).

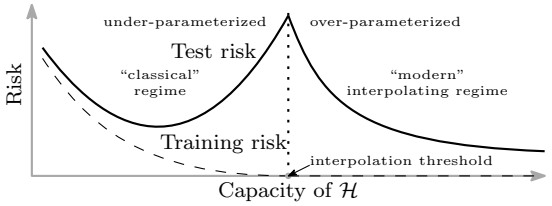


Figure 1.2: Double Descent risk curve. After the initial overfitting phase, increasing model capacity, training time, or dataset size further leads to a second descent in test risk. Figure taken from Belkin et al. (2019).

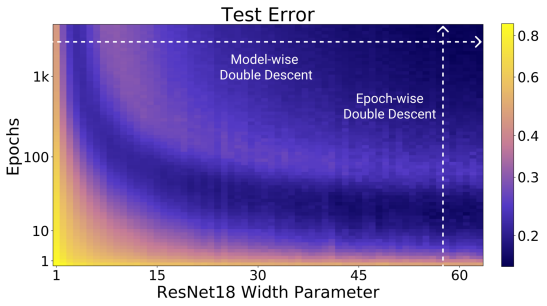


Figure 1.3: Double Descent observed in practice in Supervised Learning on CIFAR-10 with ResNets, varying model size (left), and training time (middle); showing Double Descent is possible under both regimes. Figure taken from Nakkiran et al. (2019).

The reason for this behavior remains unsolved, but with a number of complementary hypotheses.

The underlying mechanisms behind DD are still debated, but several complementary hypotheses have emerged in the supervised learning literature. A unifying perspective is that DD becomes most visible near the interpolation threshold: the regime where the model becomes expressive enough to fit the training data almost perfectly. Around this point, small changes in model size or training procedure can cause large changes in generalization, producing the characteristic peak in test risk (Belkin et al., 2019; Nakkiran et al., 2019).

One commonly cited explanation is that the interpolation threshold creates a variance spike in the learned function. As the model approaches the capacity to fit the training data exactly, it must contort its decision boundary or function representation to accommodate all training points, including noise or outliers. This leads to high variance in predictions on unseen data, harming generalization and producing the peak in test risk (Belkin et al., 2019). Adding more parameters allows the model to fit the data more smoothly, reducing variance and leading to the second descent. In linear regression, this can be shown explicitly because the variance of the estimator diverges at the interpolation threshold (Hastie et al., 2020).

Even when models are large enough to interpolate, training algorithms such as (stochastic) gradient descent do not pick arbitrary interpolating solutions. Instead, optimization exhibits implicit regularization. It tends to select solutions with particular simplicity biases (e.g., small norms or large margins), which can generalize well in overparameterized settings. From this view, the second descent occurs because, with sufficient capacity, the optimizer can find interpolating solutions that also satisfy these implicit biases (Belkin et al., 2019).

Finally, DD is often most pronounced when there is irreducible noise in the targets (e.g., label noise). In that case, interpolating the training set requires fitting noise as well as signal. Near the interpolation threshold, the model begins to fit this noise, harming generalization and producing the peak. With further overparameterization (and sufficient training), the model can fit noise in a way that is less damaging to the learned representation, yielding the second descent (Nakkiran et al., 2019).

1.1 Prior Work on DD in RL

While DD is well documented in supervised learning, its status in reinforcement learning is less clear and appears to depend strongly on the learning setup and on what is used as a proxy for generalization. Veselý et al. (2025) investigate DD in deep model-free RL by varying network capacity and analyze DD-like behavior through an information-theoretic lens, using policy entropy as a central diagnostic. They report that DD-style patterns can emerge in certain actor–critic settings, but emphasize that standard RL objectives may be unreliable indicators of generalization and call for evaluation on explicit out-of-distribution benchmarks. They note that further work testing whether their observations translate to superior generalization on out-of-distribution generalization. Complementing this empirical perspective, Brellmann et al. (2024) provide a theoretical account of DD in an RL context by studying policy evaluation with regularized LSTD and random features. In a regime where the number of parameters N and the number of distinct visited states m grow proportionally, they derive limiting expressions for the (empirical and true) mean-squared Bellman error and show that DD-like behavior is governed by the ratio N/m, with a peak near the interpolation threshold $N/m \approx 1$, and that stronger L_2 regularization and improved state coverage attenuate or remove the effect. Together, these works suggest that DD can arise in RL under specific conditions.

2 Methods

To replicate Double Descent as it occurs in Supervised Learning (SL) within Reinforcement Learning (RL), we need to define analogous concepts for test and training splits, and test and train risk. Since Double Descent is defined as a narrowing of the generalization gap between test and train risk, we need to be able to measure performance on both seen and unseen data. In SL, this is straightforward: the training split is the data the model is trained on, and the test split is held-out data the model has not seen during training. The train risk is the error (e.g., classification error, mean squared error) on the training split, and the test risk is the error on the test split. In RL, however, the concepts of training and test splits are less clear-cut, since the agent learns from interactions with an environment rather than from a fixed dataset.

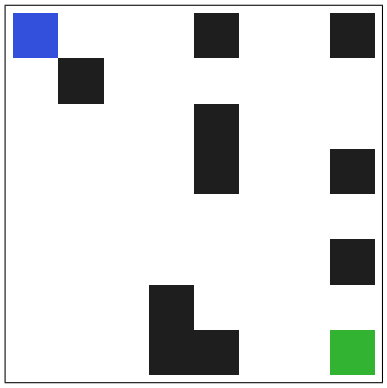
To create train and test splits in Reinforcement Learning, we create a family of environments randomly generated based on a seed. Each environment in this family shares the same underlying structure (e.g., state and action space, transition dynamics, reward structure) but differs in specific map makeup (e.g., layout and obstacle placement). By training the RL agent on a subset of these environments (the training split) and evaluating its performance on a separate subset of unseen environments (the test split), we can emulate the train-test paradigm from SL. This approach allows us to assess the agent's ability to generalize its learned policy to new, unseen environments.

To plot a Double Descent curve, we further need to define an analogue to risk. In Supervised Learning, risk is typically defined as the expected loss (e.g., classification error, mean squared error) on a dataset. For our purposes, we use mean performance (i.e., average return) as a proxy for risk, where higher performance indicates lower risk. Additionally, we consider the trace of the Fisher Information Matrix (FIM) to indicate overfitting or memorizing in the model.

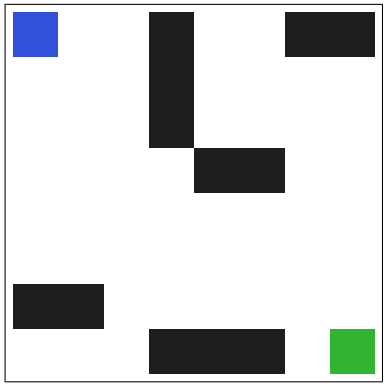
2.1 Environment

The environments used for this study are mazes. Each map is contained in an 8×8 grid, with the agent and the goal each taking up one tile. Tiles not occupied by the agent or the goal at game start are either empty or walls. Whether a tile is a wall is determined randomly during map generation, with a fixed probability of 0.2 for each tile to be a wall. This yields the possibility of generating maps that are unsolvable (i.e., there is no path from the agent to the goal). Such maps are discarded during generation and re-generated to ensure solvability. A selection of example maps can be seen in Figure 2.1.

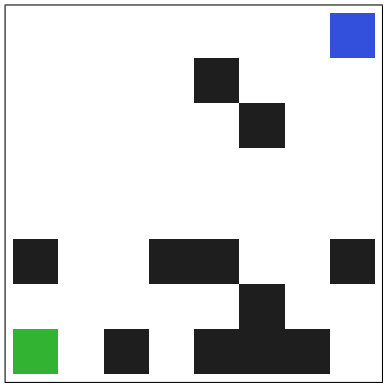
The position of the goal and starting position of the agent can be randomized or set manually. We study both settings. In the manual setting, the goal is always placed in the bottom-right corner of the map (coordinates (7,7)) and the agent always starts in the top-left corner (coordinates (0,0)). In the randomized setting, both the agent's starting



(a) Example environment in standard configuration. Walls are shown in black, the agent in blue, and the goal in green.



(b) Another example environment in standard configuration.



(c) Example environment in randomized configuration.

Figure 2.1: Example 8×8 maze environments used in this study.

position and the goal position are randomized to one of the corners of the map at the start of each episode.

The environment's observation space consists of a binary vector, representing a flattened 8×8 grid where each tile is encoded using one-hot encoding to indicate whether it is empty, a wall, the agent's position, or the goal's position. The state space for this Markov Decision Process (MDP) is thus

$$\mathcal{S} = \{0, 1\}^{512} \tag{2.1}$$

Each observation includes the current state of the grid as well as the previous state—frame-stacking. Each observation therefore encodes $8 \times 8 \times 4 \times 2 = 512$ bits. The action space is discrete and allows for moving in the four cardinal directions:

$$\mathcal{A} = \{\text{up}, \text{down}, \text{left}, \text{right}\} \tag{2.2}$$

The agent receives a reward of $+1$ upon reaching the goal. In all other timesteps, a potential-based reward is applied. The potential function $\phi(s)$ is defined as one hundredth of the negative Euclidean distance from the agent's current position to the goal position. The reward at each timestep is then given by $r_t = \phi(s_t) - \phi(s_{t-1})$, encouraging the agent to move closer to the goal. Finally, the agent also receives a small time-penalty of -0.01 at each timestep to disincentivize standing still or taking suboptimal paths. Thus, the reward function is defined as follows:

$$r_t = \begin{cases} 1, & \text{if goal reached} \\ \phi(s_t) - \phi(s_{t-1}) - 0.01, & \text{otherwise} \end{cases} \tag{2.3}$$

with the potential function defined as:

$$\phi(s) = -\frac{1}{100} \cdot d_{\text{euclidean}}(\text{agent\_pos}, \text{goal\_pos}) \tag{2.4}$$

For example, if the agent moves a distance of exactly 1 unit closer to the goal in a timestep, it receives a reward of $0.01 - 0.01 = 0$. In all other cases the reward is negative, except when the goal is reached. Considering the agent starts in the opposite corner in the standard configuration and can thus move no farther away, the minimum return in the standard configuration is $G_{min} = -0.01 \times 64 = -0.64$. The minimum return in the randomized configuration, occurring when the agent does not start

opposite of the goal, can be further lowered by the potential to increase distance to the goal of

$$G_{min} = \frac{1}{100} \times (\text{Max Distance} - \text{Starting Distance}) \tag{2.5}$$
$$= \frac{1}{100} \times (\sqrt{7^2 + 7^2} - 8) \tag{2.6}$$
$$\approx \frac{1}{100} \times (9.899 - 8) \tag{2.7}$$
$$= \frac{1}{100} \times 1.899 \tag{2.8}$$
$$= 0.01899 \tag{2.9}$$

Thus the minimum return becomes $-0.64 - 0.01899 \approx -0.65899$. The maximum return an agent can receive on a map depends on the map's configuration, as some can be solved in fewer steps than others. However, we can compute the maximum return in the optimal case. Here, in the standard configuration, the agent needs at least 14 steps to reach the goal (moving 7 steps down and 7 steps right). In this case, the maximum return is:

$$G_{max} = 1 - (0.01 \times 13) - \phi(s_{end}) + \phi(s_{start}) \tag{2.10}$$
$$= 1 - 0.13 - (\frac{1}{100} \times 0) + (\frac{1}{100} \times \sqrt{7^2 + 6^2}) \tag{2.11}$$
$$\approx 0.87 + (\frac{1}{100} \times 9.21954446) \tag{2.12}$$
$$= 0.87 + 0.09210 \tag{2.13}$$
$$= 0.96210 \tag{2.14}$$
$$\tag{2.15}$$

Indeed, when computing the mean optimal return for all maps using seeds 0–9,999, we get 0.958585, slightly below this. This is expected as the frequency of maps in which the agent is forced to take more than 14 steps to reach the goal is very low—1.75% among these seeds, to be precise. In the randomized configuration, the maximum return is 1.0, as the agent can be positioned so that it can move a full unit closer to the goal in every step and thus gain a reward of zero in every step until reaching the goal, when it gains a reward of 1.

Therefore, the MDP under this environment can be formally defined as the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where $\mathcal{S}$ is the state space as in Equation (2.1), $\mathcal{A}$ is the action space as in Equation (2.2), p is the state transition probability function defined by the environment dynamics, r is the reward function as in Equation (2.3), and γ is the discount factor set to 1.

2.2 Metrics

For a y-axis representing train and test risk, we use mean return (i.e., average cumulative reward per episode) as a proxy for risk. Higher return indicates lower risk. We measure mean return on both the training environments (train risk) and the test environments (test risk) after every training epoch. To do this, we run an inference rollout on the set of training environments and the set of test environments. The set of test environments is equal in size to the set of training environments, up to a maximum of 100 test environments. Each test and train evaluation consists of running one episode on each environment in the respective set and taking the mean return across all episodes. We record this performance once every 10,000 training episodes. This is our most straightforward measure of a generalization gap between training and test performance, and in this context we would look for a "Double Ascent" as the test performance initially lags behind training performance before catching up again.

To further study generalization performance, we also measure the trace of the Fisher Information Matrix (FIM) after every training epoch. Instead of using the trajectory-based FIM, we use the state-action pair FIM, which is defined as:

$$F = \mathbb{E}_{(s,a)\sim\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a \mid s) \nabla_\theta \log \pi_\theta(a \mid s)^T \right] \tag{2.16}$$

where (s, a) are state-action pairs sampled from behavior under the policy π_θ. In other words, the FIM is the expected outer product of the gradient with respect to the model parameters θ of the score function, which is the log-probability of taking action a in state s. The FIM captures how much information about the parameters is contained in the actions taken by the policy in different states. Because of this definition, we can estimate the FIM using samples of state-action pairs collected during rollouts.

Let $g = \nabla_\theta \log \pi_\theta(a \mid s)$ be the gradient of the

log-probability of action a given state s. Then the FIM is given by:

$$F = \mathbb{E}_{(s,a)\sim\pi_\theta}[gg^T] \tag{2.17}$$

The trace of the FIM, $\mathrm{Tr}(F)$, provides a measure of the sensitivity of the model's parameters to changes in the data distribution. A high trace value indicates that the model is highly sensitive to the training data, which can be a sign of overfitting or memorization. Conversely, a lower trace value suggests that the model is more robust and generalizes better to unseen data. By tracking the trace of the FIM on both training and test environments, we can gain insights into the model's generalization capabilities and its tendency to overfit as training progresses.

Using the definition of the FIM in Equation (2.17), we can derive an empirical estimator for the trace of the FIM without forming the full matrix using the fact that the trace of a matrix is defined as the sum of its diagonal elements (e.g. $\sum_i F_{ii}$). We can thus derive:

$$\begin{aligned}\mathrm{Tr}(F) &= \mathrm{Tr}(\mathbb{E}_{(s,a)\sim\pi_\theta}[gg^T]) && (2.18)\\ &= \mathbb{E}_{(s,a)\sim\pi_\theta}[\mathrm{Tr}(gg^T)] && (2.19)\\ &= \mathbb{E}_{(s,a)\sim\pi_\theta}[g^T g] && (2.20)\end{aligned}$$

Therefore, to form a Monte Carlo estimator for the trace of the FIM, we can sample N state-action pairs (s_i, a_i) from rollouts under the policy π_θ and compute the score function gradients $g_i = \nabla_\theta \log \pi_\theta(a_i \mid s_i)$ for each pair. The empirical estimator for the trace of the FIM is then given by:

$$\widehat{\mathrm{Tr}(F)} = \frac{1}{N}\sum_{i=1}^{N} g_i^T g_i \tag{2.21}$$

where $g_i = \nabla_\theta \log \pi_\theta(a_i \mid s_i)$. This estimator allows us to compute the trace of the FIM efficiently without explicitly constructing the full matrix. Forming the full matrix would be prohibitively expensive for models with a large number of parameters (in the order of millions), as the FIM is of size $|F| \times |F|$ where $|F|$ is the number of parameters in the model.

Considering the trace of the FIM naturally scales with the size of the FIM, we can compute a trace per parameter by dividing the trace by the number of parameters in the model. This allows us to compare FIM trace values across models of different sizes. Because the trace of a matrix is also the sum of its eigenvalues, the trace per parameter can be interpreted as the average eigenvalue of the FIM. In other words, this metric can be described as the average Fisher information.

Considering we would expect higher average Fisher information when the model is overfitting or memorizing the training data, we can use this metric as a complementary measure of generalization performance alongside mean return. To support findings of Double Ascent in the mean return curves, we would expect to see the mean Fisher information to correlate with the size of the generalization gap between training and test performance. As the gap emerges, we would expect the mean Fisher information of the model to increase, which would remain until the gap narrows or closes.

2.3 Models

To conduct our experiment, we need to specify a model type. We run experiments both using a Deep Q-Network (DQN) (Mnih et al., 2013) and using a Policy Gradient method, specifically using Trust Region Policy Optimization (TRPO) agent (Schulman et al., 2017). Both models use a feedforward neural network as function approximator. The architecture of the neural network consists of an input layer matching the size of the observation space (512 units), followed by varying numbers of hidden layers of varying width with ReLU activations, and an output layer matching the size of the action space.

2.3.1 Deep Q-Networks

A Deep Q-Network (DQN) is a value-based Reinforcement Learning algorithm that approximates the optimal action-value function $Q^*(s, a)$ using a deep neural network. The action-value function estimates the expected cumulative reward for taking action a in state s and following the optimal policy thereafter. The DQN uses the Bellman equation as the foundation for its learning process, which states that the optimal action-value function satisfies:

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s', a') \mid s, a\right] \tag{2.22}$$

where r is the immediate reward received after taking action a in state s, γ is the discount factor (in

our case, $\gamma = 1$), and s' is the next state. The DQN approximates $Q^*(s, a)$ using a neural network parameterized by θ, denoted as $Q(s, a; \theta)$. The network is trained to minimize the difference between the predicted Q-values and the target Q-values derived from the Bellman equation. The loss function $L(\theta)$ used for training the DQN is defined as:

$$\mathbb{E}_{(s,a,r,s')}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right] \tag{2.23}$$

where θ^- are the parameters of a target network that is periodically updated to stabilize training. The DQN employs experience replay, where transitions (s, a, r, s') are stored in a replay buffer and sampled randomly during training to break correlations between consecutive samples and improve learning stability.

We use a replay buffer size of 50,000 transitions and a target network update frequency of 5000 training steps. The DQN is trained using the Adam optimizer (Kingma & Ba, 2017) with a learning rate of 0.001 and a batch size of 64. The exploration-exploitation trade-off is managed using an epsilon-greedy strategy, where the exploration rate ϵ decays linearly from 1.0 to 0.05 over the first 30% of the episodes.

2.3.2 Trust Region Policy Optimizer

Trust Region Policy Optimization (TRPO) is a policy gradient method that aims to optimize the policy directly while ensuring stable and monotonic improvement. TRPO achieves this by constraining the step size of policy updates using a trust region approach, which prevents large, destabilizing updates to the policy. The core idea of TRPO is to maximize a surrogate objective function subject to a constraint on the Kullback-Leibler (KL) divergence between the old and new policies. The surrogate objective function is defined as:

$$L(\theta) = \mathbb{E}_{s,a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} A^{\pi_{\theta_{\text{old}}}}(s, a)\right] \tag{2.24}$$

where $\pi_{\theta_{\text{old}}}$ is the old policy, π_θ is the new policy parameterized by θ, and $A^{\pi_{\theta_{\text{old}}}}(s, a)$ is the advantage function estimating the relative value of action a in state s under the old policy. The KL divergence constraint is given by:

$$\mathbb{E}_{s\sim\pi_{\theta_{\text{old}}}}\left[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot \mid s) \parallel \pi_\theta(\cdot \mid s))\right] \leq \delta \tag{2.25}$$

where δ is a predefined threshold that limits the size of the policy update. TRPO uses a conjugate gradient algorithm to solve the constrained optimization problem efficiently. The policy is updated iteratively by computing the natural gradient of the surrogate objective and scaling it to satisfy the KL divergence constraint.

We implement TRPO using the same feedforward neural network architecture as described in Section 2.3.1. The policy network outputs logits for each action, over which we use a softmax activation function to get a distribution over the action space. The value function is approximated using a separate value network with the same architecture but with a single output unit representing the state value. The advantage function is estimated using Generalized Advantage Estimation (GAE) to reduce variance in the policy gradient estimates.

The TRPO hyperparameters are a maximum KL divergence of $\delta = 10^{-2}$, conjugate-gradient iterations set to 10 with damping 0.1, backtracking coefficient 0.5 for 10 line-search steps, GAE $\lambda = 0.95$, and a batch size of 20 episodes per policy update. The value function is optimized for 5 iterations per update using Adam with learning rate 10^{-3}.

2.4 Training Runs

To search for Double Descent, we ran the models described above on various configurations of training environments and training durations. On the following configurations, we trained agents of varying depths between 3 and 8 and widths between 4 and 1024:

Type	Train Maps	Episodes	Start & Goal Position
TRPO	10	10,000	Standard*
DQN	50	20,000	Standard
DQN	100	50,000	Standard
TRPO	50	20,000	Randomized
TRPO	100	50,000	Randomized
TRPO	500	1,000,000	Standard
TRPO	750	1,000,000	Standard
TRPO	1,000	1,000,000	Standard

*Agent starts in top left, goal in bottom right

On the following configurations, we trained agents of depth 3 and widths 4, 16, 64, 256, and 1024 for unlimited time (in practice for about 1.8M episodes, until our high performance cluster ended the jobs):

Type	Train Maps	Start & Goal Position
TRPO	50	Standard
TRPO	100	Standard

3 Results

In this section, we present the results of our experiments searching for Double Descent in Reinforcement Learning. Unfortunately, across all configurations tested, we did not observe instances of Double Descent in the episoding or capacity regimes.

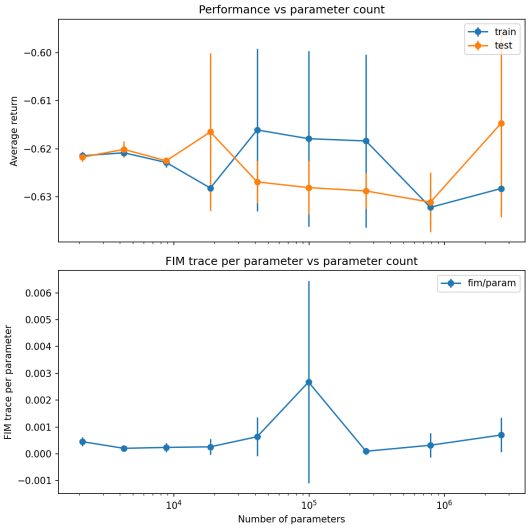
3.1 DQN Fails to Learn

Firstly, we examined performance using Deep Q-Networks (DQN) on our environment. Unfortunately, this architecture was unable to solve the environment altogether. We observe frequent catastrophic forgetting, with performance spikes being present but short-lived. Examples of a DQN failing to learn the environment reliably can be found in Appendix A. This occurs over a range of different model capacities in DQNs. Figure 3.1 shows metrics a range of model capacities, defined by varying width and depth of the neural network. No Double Descent is observed, as neither curve exhibits significant learning. Similarly, the FIM trace does not show interesting movement, staying extremely close to zero throughout different model capacities.

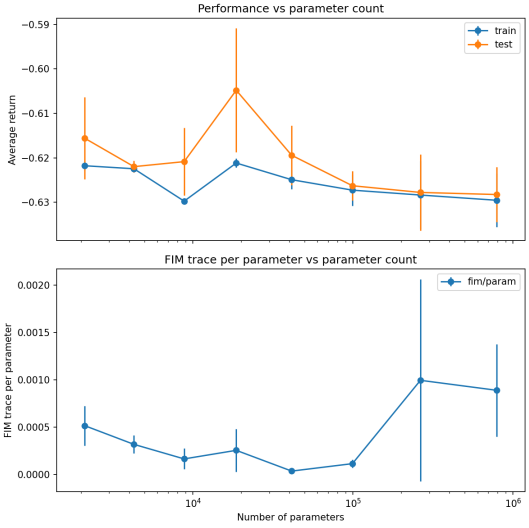
These results are not useful to our study of Double Descent, as the model fails to learn the environment in the first place. We cannot observe a second ascent of the test performance if there is no first ascent. Considering the poor performance of DQN on this environment, we decided to move to a different architecture that we hoped would perform better.

3.2 TRPO: No Double Descent

We next examined performance using Trust Region Policy Optimization (TRPO) on our environment. This architecture was able to learn the environment reliably across a range of model capacities.



(a) Performance of DQN using 50 training seeds and 20,000 training episodes per model.



(b) Performance of DQN using 100 training seeds and 50,000 training episodes per model.

Figure 3.1: DQN training and test performance, and average Fisher information, as a function of model capacity. Neither curve exhibits significant learning.

However, we were again unable to observe Double Descent in either the episoding or capacity regimes.

3.2.1 Capacity Regime

In the capacity regime, we conducted studies to heuristically search the available hyperparameter space. Our two hyperparameters to calibrate here are size of the training set (in seeds) and amount of training time (in episodes). We varied both hyperparameters across a range of values, as shown in Table 2.4.

First, we examined performance using 10 training seeds and 10,000 training episodes per model, with the standard environment configuration. Figure 3.2 shows the results of this experiment.

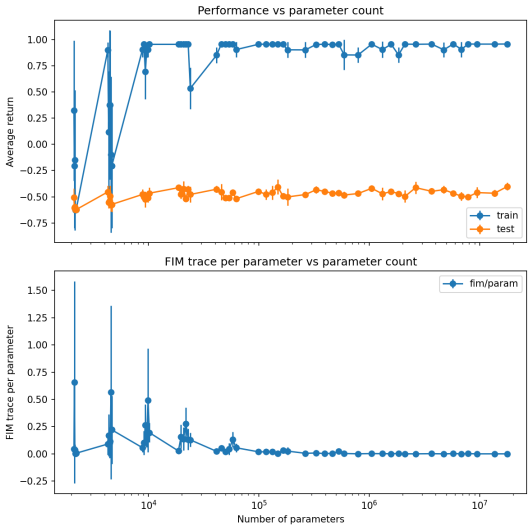


Figure 3.2: TRPO training and test performance, and average Fisher information, as a function of model capacity, using 10 training seeds and 10,000 training episodes per model.

We can observe that both training and test performance are low when the model is at its smallest. As we increase model capacity, training performance increases rapidly until it hits the maximum reward at approximately 0.96 mean return. Test performance increases slightly, stabilizing at approximately -0.5 mean return. No second ascent is observed in the test performance, and thus no Double Descent. At the same time, we observe a spike in the average Fisher information as the model reaches sufficient capacity to memorize the training set. This is expected behaviour, as the model is likely to become more sensitive to parameter changes when it has memorized the training set. However, we observe the average Fisher information decreasing again as model capacity increases further. This is unexpected, as we would expect the model to remain sensitive to parameter changes after memorizing the training set.

Considering the minimal improvement in test performance, we chose to test increasing the size of the training set to 50 training seeds and the training episodes to 20,000 episodes per model. Additionally, we tested a training set of 100 training maps and 50,000 episodes per training run. Figures 3.3 and 3.4 show the results of these experiments.

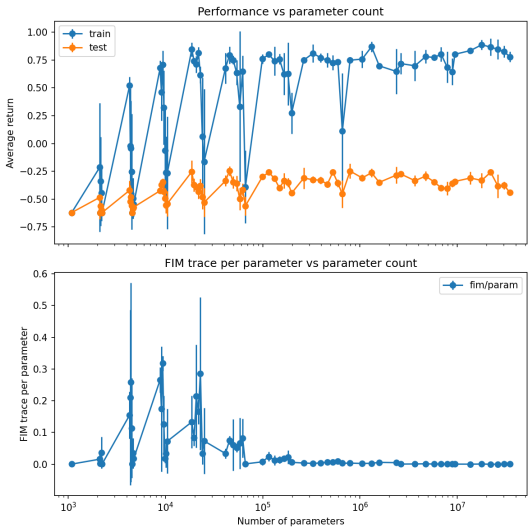


Figure 3.3: TRPO training and test performance, and average Fisher information, as a function of model capacity, using 50 training seeds and 20,000 training episodes per model.

Here, we can observe similar results to the previous experiment, but with slightly improved test performance after the first ascent to around -0.3 for 50 training maps and 0.0 for 100 training maps. The training performance decreases slightly with larger amounts of training maps, with mean return keeping around 0.8 for 50 training maps and 0.7 for 100 training maps. Again, no second ascent is observed in the test performance, and thus no Double Descent. The average Fisher information again

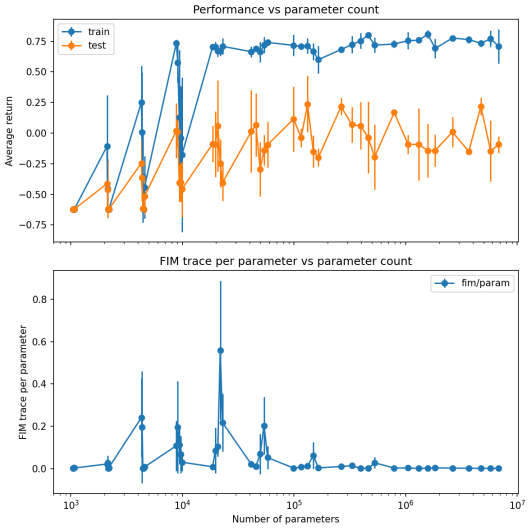


Figure 3.4: TRPO training and test performance, and average Fisher information, as a function of model capacity, using 100 training seeds and 50,000 training episodes per model.

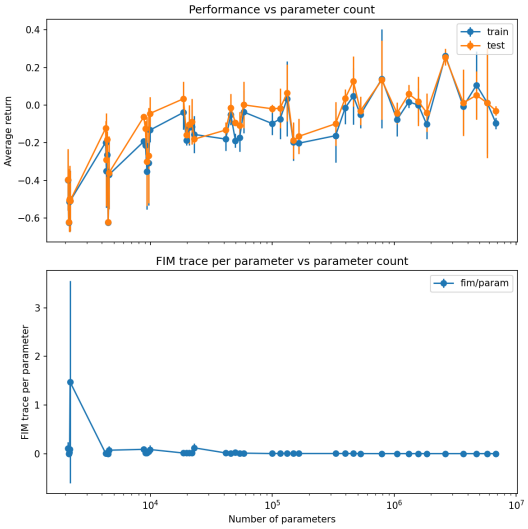


Figure 3.5: TRPO training and test performance, and average Fisher information, as a function of model capacity, using 1,000 training seeds and 100,000 training episodes per model.

spikes as the model reaches its highest performance on the training set but lags behind in test performance, and again decreases as model capacity increases further.

To test whether the model type was able to learn an unseeded (that is, without a difference in training and test maps) environment, we tested a a configuration with 1,000 training seeds when training for 100,000 episodes per mode. Figure 3.5 shows the results of this experiment.

Here, we can observe that both training and test performance are very similar, as expected when the training set is so large—we completely cover the distribution of possible maps. Performance increases slowly with model capacity, stabilizing at approximately 0 mean return for both training and test performance. Considering no generalization gap emerges at all, we have no Double Descent. The average Fisher information remains low, with one spike near the lowest model capacity, which is caused by an outlier in the data. This aligns with our expectation that the model does not memorize the training set, as it has effectively seen the full distribution of maps already.

At this stage, we noticed the training performance was not reaching the maximum possible return of approximately 0.96 mean return, even at very high model capacities. We hypothesized that increasing the amount of training time per model would allow the models to reach higher training performance, and thus potentially observe Double Descent in test performance. We thus increased the amount of training episodes per model to 1,000,000. For this newly increased training time, we tested training set sizes of 500, 750, and 1,000 maps. Figures 3.6, 3.7, and 3.8 show the results of these experiments.

This longer training time shows the models reaching much closer to the optimal performance, stabilizing around 0.8 average return. Unfortunately, even with this increased training time, we were again unable to observe Double Descent in test performance. We see the generalization gap closing slowly with increased training set size, which is expected as the model sees more of the distribution of maps. The average Fisher information continues to spike as the model reaches its highest performance on the training set but lags behind in test performance, and again decreases as model capacity increases further.

Finally, on our initial configurations of 50 training seeds and 20,000 training episodes per model

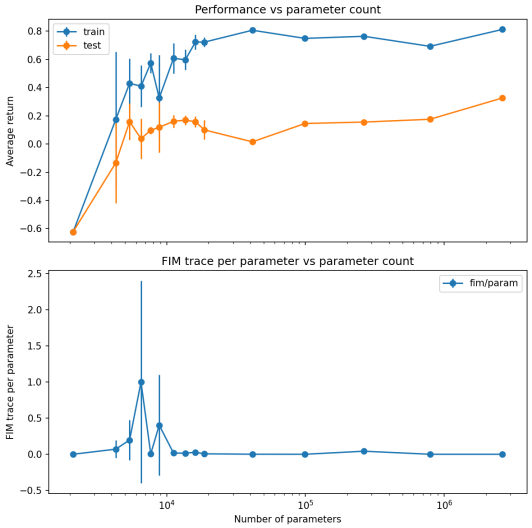


Figure 3.6: TRPO training and test performance, and average Fisher information, as a function of model capacity, using 500 training seeds and 1,000,000 training episodes per model.

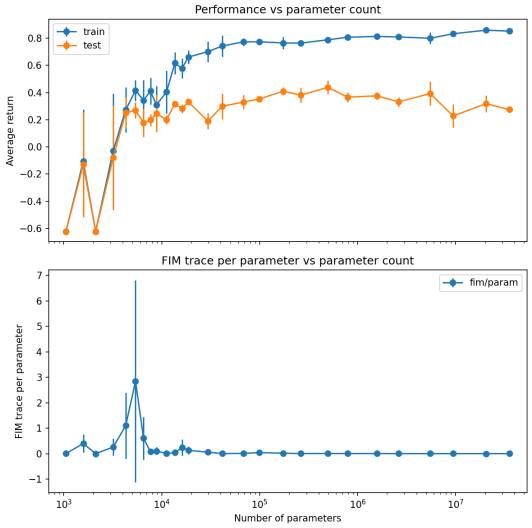


Figure 3.7: TRPO training and test performance, and average Fisher information, as a function of model capacity, using 750 training seeds and 1,000,000 training episodes per model.

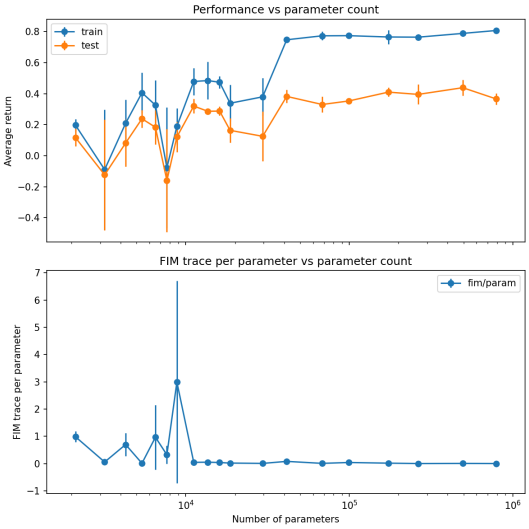


Figure 3.8: TRPO training and test performance, and average Fisher information, as a function of model capacity, using 1,000 training seeds and 1,000,000 training episodes per model.

as well as 100 training seeds and 50,000 training episodes per model, we tested the randomized environment configuration, with both agent starting and goal position being selected randomly from the four corners of the map. This way the agent would be unable to learn the basic rule to move down and to the right. Figures 3.9 and 3.10 show the results of this experiment.

Here, we observe different dynamics. The training performance varies strongly with model capacity. After investigating this we can note that this is related to the width and depth of the model. In general, the model performs better with increasing width, but performance decreases upon adding more layers to a specific depth. This may be related to optimization difficulties in deeper networks as they can be more difficult to train, as shown by He et al. (2015). It is interesting that this phenomenon is especially strong when the start and goal positions are randomized instead of fixed. The test performance remains almost minimal throughout all model capacities, with a very slight upward trend with model capacity. This indicates that we would need to train on a larger set of training maps to see any generalization capacity whatsoever. The average Fisher information spikes erratically around

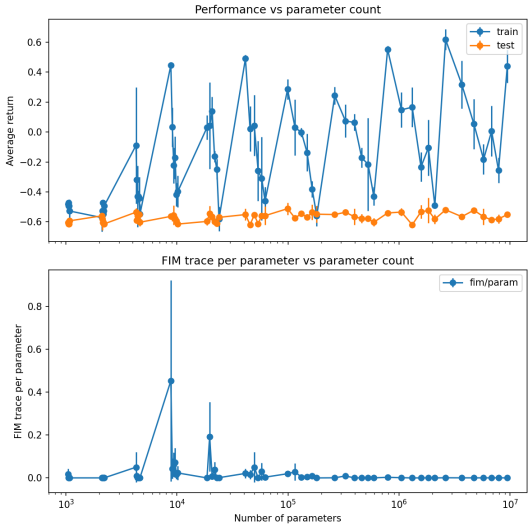


Figure 3.9: TRPO training and test performance, and average Fisher information, as a function of model capacity, using 50 training seeds and 20,000 training episodes per model and randomized start and goal positions.

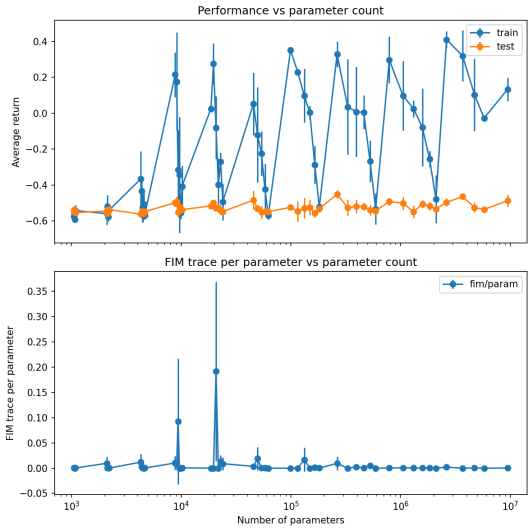


Figure 3.10: TRPO training and test performance, and average Fisher information, as a function of model capacity, using 100 training seeds and 50,000 training episodes per model and randomized start and goal positions.

where the model reaches its highest performance on the training set, but then comes back down again.

3.3 Episodic Regime

As discussed, Double Descent can also occur with respect to the amount of optimization steps taken during training, which we refer to as the episodic regime. To test whether this regime exhibits double descent under our conditions, we trained models of depth 3 and widths 4, 16, 64, 256, and 1024 on two configurations: 50 training maps and 100 training maps. Figures 3.11 and 3.12 show examples of the learning behavior during training. We trained without a limit on the number of training episodes, but in practice were limited to about 1.8M episodes until our high performance cluster ended the jobs. The remaining configurations exhibited similar behavior, except for the smallest models of width 4, which failed to learn the environment at all. These plots can be found in Appendix B.

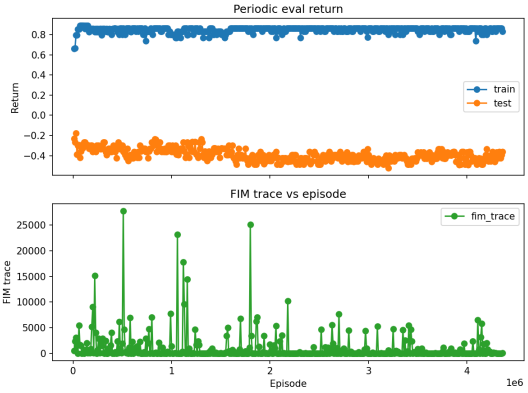


Figure 3.11: TRPO training and test performance over time for a model of width 1024 and depth 3, using 50 training seeds and unlimited training episodes.

Again, the results do not exhibit signs of Double Descent. For both training sets, we can observe a quick increase in training performance at the beginning, barely visible due to the scale of the graph. After this initial increase, training performance stays at a near-optimal level. The difference to the optimal level indicates that the model fails to solve a small number of maps or learned a suboptimal policy on maps it can solve. Test performance also increases quickly at the beginning and

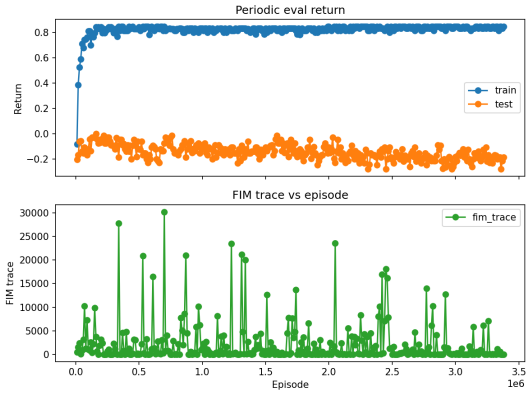


Figure 3.12: TRPO training and test performance over time for a model of width 1024 and depth 3, using 100 training seeds and unlimited training episodes.

stabilizes at a low level around -0.2 for 50 training maps and 0 for 100 training maps. However, we can observe a very slight downward trend in test performance over time, indicating that the model may be overfitting slightly to the training set. However, no second ascent is observed in the test performance within our training time, and thus no Double Descent phenomenon.

4 Discussion

In general, we are unable to observe Double Descent in our experiments. We tested for both forms of Double Descent: the episodic and capacity regime, but neither produced the a closing of the generalization gap between train and test performance. Initially, both performances increase with model capacity and training time, but after a certain point, the test performance stagnates while the train performance continues to increase, leading to a widening generalization gap. As we increase the training set size, we observe the generalization gap shrinks, but do not observe a change in the dynamics. This suggests that Double Descent does not occur under the conditions that we tested.

With regard to the average Fisher information (the mean eigenvalue, or the mean diagonal value of the Fisher information matrix), we observe an initial spike as the model begins to memorize the training data, after which it decreases again. This is an intriguing phenomenon, as it indicates that while the model initially focuses on fitting the training data closely (leading to high sensitivity to parameter changes), it subsequently adjusts its parameters in a way that reduces this sensitivity. This behavior could suggest that the model is finding a more robust representation of the data after the initial memorization phase. However, the decrease in average Fisher information does not correspond to an improvement in test performance, which remains stagnant after the initial increase. This decoupling of Fisher information dynamics and generalization performance is puzzling and warrants further investigation to understand the underlying mechanisms at play.

In the episodic regime, we observe a slight downward trend in test performance as we increase the number of training episodes, after an initial increase. This could be interpreted as overfitting manifesting itself in a degradation of test performance with more training. This trend emerges repeatedly across different training set sizes (50, 100) and different model capacities (3 depth and 16, 64, 256, 1024 widths). However, the effect is quite small and needs to be investigated further. We also fail to see consistent trends in the average Fisher information in the episodic regime, with values seemingly dominated by noise.

Overall, our results suggest that Double Descent may not be a universal phenomenon in reinforcement learning, especially under the conditions we tested. It is impossible to determine a precise reason for this, considering the complexity of Reinforcement Learning. However, we can speculate on a range of possible, potentially complementary explanations.

Firstly, there is no clear analogue of the interpolation threshold. A key ingredient for DD is traversing a regime where the model can (nearly) fit the training set. In our experiments, training performance for TRPO often saturates below the environment's near-optimal return, particularly as the number of training maps grows. If the policy does not reliably solve the training distribution, then we may never reach a true interpolation-like regime in which DD is expected to appear; instead, increasing capacity mainly improves partial fitting without transitioning into a qualitatively different overparameterized regime.

Secondly, unlike supervised learning, RL data

are policy-induced and nonstationary. As the policy improves, the state–action visitation distribution changes. This violates the fixed-distribution assumption that underlies many DD intuitions. As a result, the relationship between capacity and generalization may be dominated by exploration and visitation effects (what data are collected) rather than interpolation effects (how well a fixed dataset can be fit).

Lastly, our model takes a flattened grid as input, and both the policy and value networks are simple MLPs. For spatial navigation, MLPs struggle to capture structure such as locality or translation invariance, which convolutional or graph-based models handle better (see Cohen & Welling (2016)). As a result, making the MLP larger mainly improves memorization of the training maps rather than generalization to new maps. Test performance therefore plateaus even as training performance improves, so we do not observe the "second descent" that Double Descent predicts.

The dynamics of the average Fisher information also present an interesting avenue for further research, particularly in understanding its relationship with generalization performance. The initial increase of the average Fisher information during the memorization phase suggests that the model becomes highly sensitive to parameter changes as it fits the training data closely. This is expected as this represents a regime where the model is likely overfitting to the training data, which we can observe in the generalization gap between train and test performance. However, the subsequent decrease in average Fisher information, despite stagnant test performance, indicates that the model may be finding a more stable parameter configuration that is less sensitive to perturbations. This decoupling of Fisher information dynamics and generalization performance is puzzling and warrants further investigation to understand the underlying mechanisms at play.

One possible cause for this lowering of the average Fisher information despite the remaining stagnant test performance is that the Fisher information we estimate is fundamentally on-policy and therefore tightly coupled to the state–action visitation distribution induced by the current policy. Concretely, the empirical Fisher information matrix for a policy $\pi_\theta(a \mid s)$ is an expectation of score outer products, as defined in Equation (2.16). As training progresses, both the distributions of the states seen and the actions taken in any given state changes. This means that a decrease in the average Fisher information can reflect a change in what states are visited and how stochastic the policy is on those states, rather than an improvement in out-of-distribution generalization.

A plausible mechanism is increasing policy determinism. As TRPO improves on the training maps, the policy often becomes more confident, concentrating probability mass on a small set of actions. This can reduce the magnitude of $\nabla_\theta \log \pi_\theta(a \mid s)$ for on-policy actions, thereby reducing the empirical Fisher trace or mean diagonal. In other words, after an initial "memorization" phase with large, high-variance gradients (and thus a Fisher spike), the policy can enter a regime of saturated action preferences where the local sensitivity of the log-likelihood decreases even if the learned behavior remains brittle outside the training distribution. A similar dynamic can occur with a decreasing in the amount of states that are regularly visited. Importantly, this can coexist with poor performance on the test maps, as the policy can be locally stable and low-sensitivity on the training-induced state distribution while still failing catastrophically on unseen layouts that induce different state visitation.

5 Limitations and Future Work

Our study is subject to several limitations both in terms of scope and methodology, which point to avenues for future research.

Firstly, we are constrained by the amount of compute available. Because experiment runs take multiple days when taking into account limited availability of GPUs and wait times in job queues on our high-performance cluster, we were only able to run a limited number of configurations. This forces us to search the hyperparameter space in a heuristic manner, relying on researcher intuition to select promising configurations rather than performing a systematic grid or random search. Future work could leverage greater computational resources to explore a wider range of hyperparameters, architectures, and training regimes more exhaustively.

It would be especially interesting to explore the episodic regime into longer training runs, considering that we do observe a slight but consistent downward trend in test performance as we increase the number of training episodes.

Secondly, our experiments are limited to a single environment and TRPO and DQN as RL algorithms. While GridWorld-like environments are a common benchmark for RL research, they may not capture the full complexity of real-world tasks. Future studies could investigate Double Descent in a variety of environments, including continuous control tasks or more complex navigation scenarios. Additionally, exploring other RL algorithms, such as investing more time into DQN or other value-based methods, could provide further insights into whether Double Descent manifests differently across algorithmic paradigms. One straightforward extension would be to modify our environment to feature stochastic transitions or increase the frequency of obstacles, thereby increasing task complexity and potentially affecting generalization dynamics.

Thirdly, it would be interesting to test architectures with stronger spatial inductive bias. The flattened grid input paired with an MLP may encourage memorization of idiosyncratic layouts rather than learning reusable spatial features. A straightforward extension is to replace the MLP with a convolutional policy/value network operating on the 8×8 grid. If the generalization plateau is primarily an inductive-bias limitation, these models should raise test performance and may change the shape of generalization curves under capacity scaling.

Lastly, it could help our resulst to improve evaluation fidelity and report uncertainty. Our current evaluation reports only the mean return and average Fisher information. This can obscure subtler changes in the policy behavior. Future work should complement mean return with more diagnostic metrics such as success rate (fraction of episodes reaching the goal), steps-to-goal conditional on success, and the full return distribution over multiple rollouts with the stochastic policy. This would make it easier to find qualitatively different failure modes (e.g., policies that succeed rarely but very well versus policies that succeed often but only marginally). It would also be helpful to report return not as a raw number, but as a fraction of the optimal return (i.e., normalized between 0 and 1) on each tested map. This would make results more interpretable and comparable across different environments or reward scales.

6 Conclusion

Our investigation into the existence of Double Descent in Reinforcement Learning has yielded intriguing insights, albeit without definitive evidence of the phenomenon under the conditions tested. While we observed certain dynamics in model performance and Fisher information, these did not align with the expected patterns of Double Descent as documented in supervised learning contexts. Notably, the absence of a clear interpolation threshold and the nonstationary nature of RL data can play significant roles in shaping generalization behavior differently than in supervised settings. Furthermore, the architectural choices, particularly the use of MLPs for spatial tasks, may have limited the models' ability to generalize effectively. Our findings do not preclude the existence of Double Descent in RL, but rather motivate further investigation of Double Descent within Reinforcement Learning. Future research, as described in Section 5, is essential to clarify the conditions under which Double Descent might manifest in Reinforcement Learning.

References

Belkin, M., Hsu, D., Ma, S., & Mandal, S. (2019, July). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, *116*(32), 15849–15854. Retrieved from `http://dx.doi.org/10.1073/pnas.1903070116` doi: 10.1073/pnas.1903070116

Brellmann, D., Berthier, E., Filliat, D., & Frehse, G. (2024). *On double descent in reinforcement learning with lstd and random features.* Retrieved from `https://arxiv.org/abs/2310.05518`

Cohen, T. S., & Welling, M. (2016). *Group equivariant convolutional networks.* Retrieved from `https://arxiv.org/abs/1602.07576`

Hastie, T., Montanari, A., Rosset, S., & Tibshirani, R. J. (2020). *Surprises in high-dimensional*

ridgeless least squares interpolation. Retrieved from https://arxiv.org/abs/1903.08560

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep residual learning for image recognition.* Retrieved from https://arxiv.org/abs/1512.03385

Kingma, D. P., & Ba, J. (2017). *Adam: A method for stochastic optimization.* Retrieved from https://arxiv.org/abs/1412.6980

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*. Retrieved from https://arxiv.org/abs/1312.5602

Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., & Sutskever, I. (2019). *Deep double descent: Where bigger models and more data hurt.* Retrieved from https://arxiv.org/abs/1912.02292

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2017). Trust region policy optimization. *Proceedings of the 31st International Conference on Machine Learning*. Retrieved from https://arxiv.org/abs/1502.05477

Veselý, V., Todorov, A., & Sabatelli, M. (2025). *On the presence of double-descent in deep reinforcement learning.* Retrieved from https://arxiv.org/abs/2511.06895

A DQNs Failing To Learn

Throughout runs, DQNs failed to learn a good policy on the environment.

Figure A.1: A DQN's (width 512 and depth 3) performance over training episodes, trained on 100 seeds for 50,000 episodes each. We can see some initial learning, but the model seems to forget and revert to minimal return.

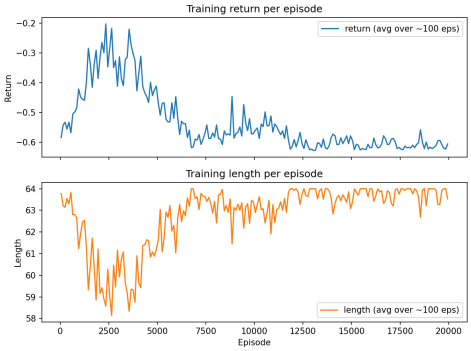


Figure A.2: A DQN's (width 512 and depth 3) performance over training episodes, trained on 50 seeds for 20,000 episodes each. We can see some initial learning, but the model seems to forget and revert to minimal return.

Figure A.3: A DQN's (width 1024 and depth 3) performance over training episodes, trained on 100 seeds for 50,000 episodes each. We can see some slow initial learning, but the model seems to forget and revert to minimal return.

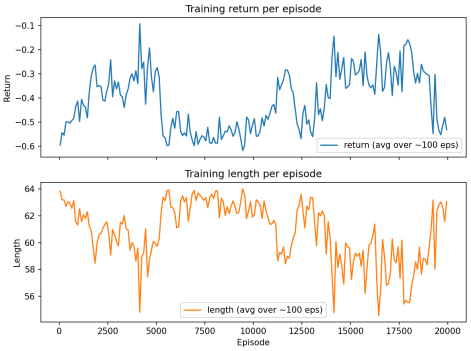


Figure A.4: A DQN's (width 1024 and depth 3) performance over training episodes, trained on 50 seeds for 20,000 episodes each. We can see fluctuations in performance, but the model does not reliably solve the environment.

B Longer Training Runs

We ran long training runs for the episodic regime under TRPO for 50 and 100 training seeds and model sizes with depth 3 and widths 4, 16, 64, 256, and 1024. Each model was trained for unlimited episodes. The results can be seen below.

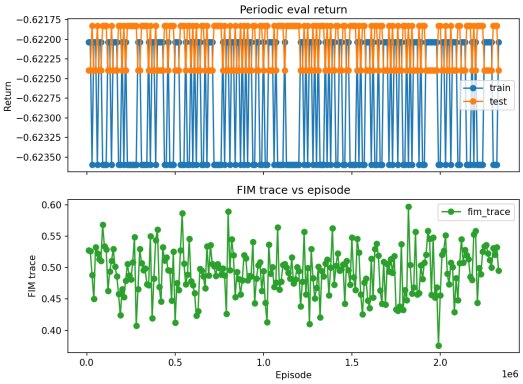


Figure B.1: Model width 4, training set size 50. The model is too small and fails to learn.

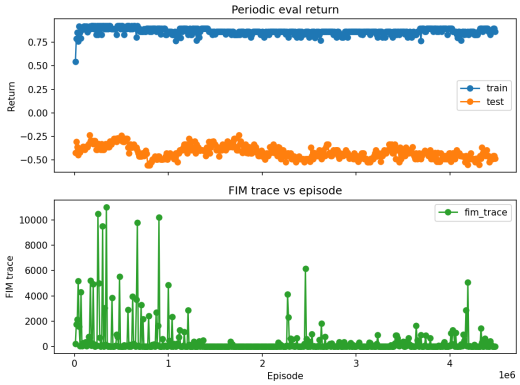


Figure B.2: Model width 16, training set size 50.

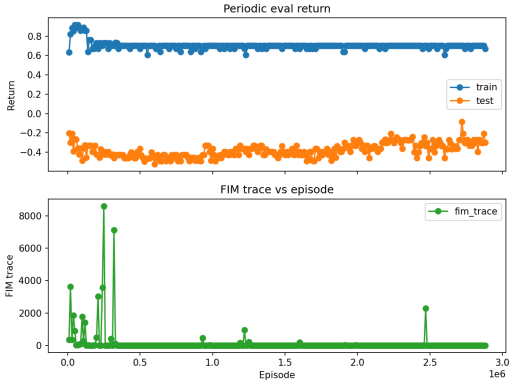


Figure B.3: Model width 64, training set size 50.

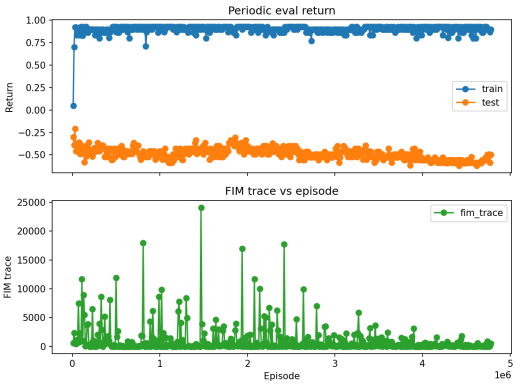


Figure B.4: Model width 256, training set size 50.

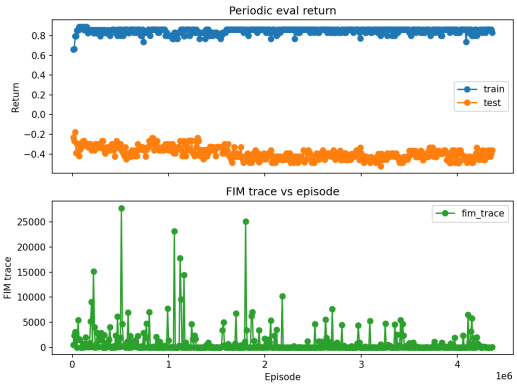


Figure B.5: Model width 1024, training set size 50.

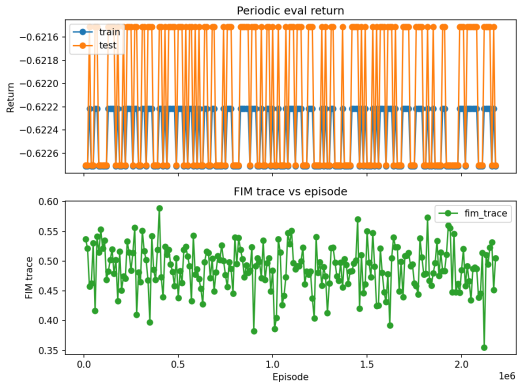


Figure B.6: Model width 4, training set size 100. The model is too small and fails to learn.

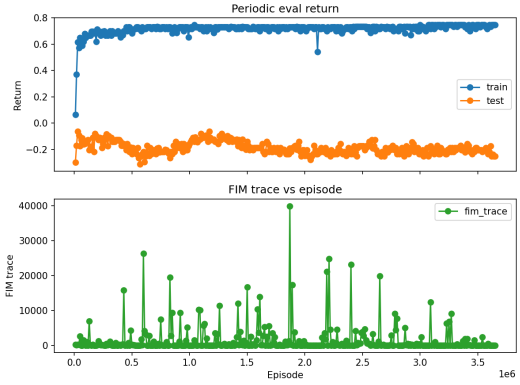


Figure B.7: Model width 16, training set size 100.

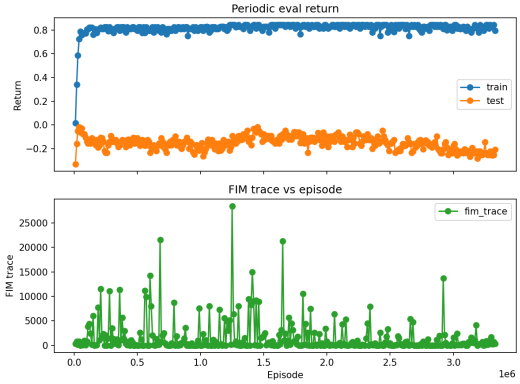


Figure B.8: Model width 64, training set size 100.

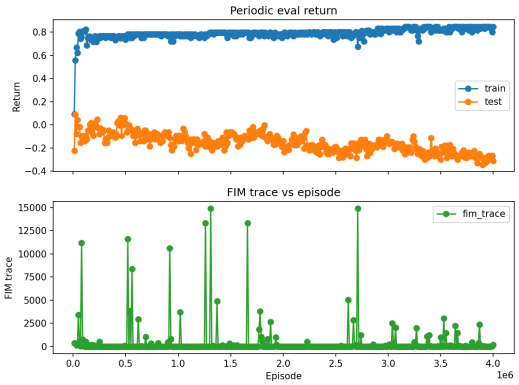


Figure B.9: Model width 256, training set size 100.

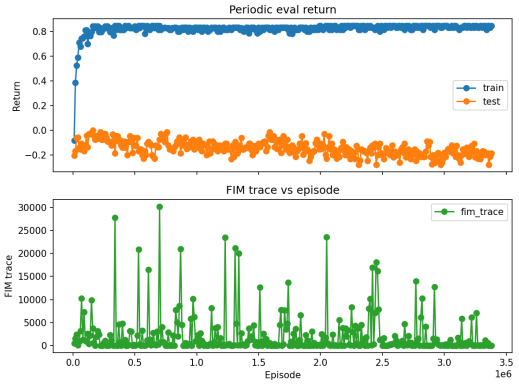


Figure B.10: Model width 1024, training set size 100.