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, model size, or dataset size. We implement an empirical experiment to test whether this phenomenon can appear in Reinforcement Learning (RL). To this end, we train DQN and 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 test 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 (DD) is a well-studied phenomenon in supervised learning (SL). 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 or training time 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, which can be seen on the left of Figure 1.1. However, work in the past years has shown that this traditional regime can be escaped. 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; Loog et al., 2020), as can be seen on the right of Figure 1.1. This phenomenon has been coined "Double Descent" (DD) and has been observed across various architectures and datasets in SL (Nakkiran et al., 2019).

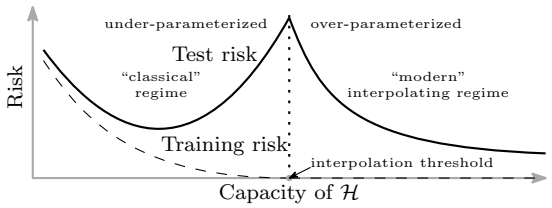


Figure 1.1: DD risk curve. After the initial overfitting phase, increasing model capacity, training time, or dataset size further leads to a second descent in test risk. Figure taken from Belkin et al. (2019).

The underlying mechanisms behind DD are still debated, but several complementary hypotheses have emerged in the SL 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

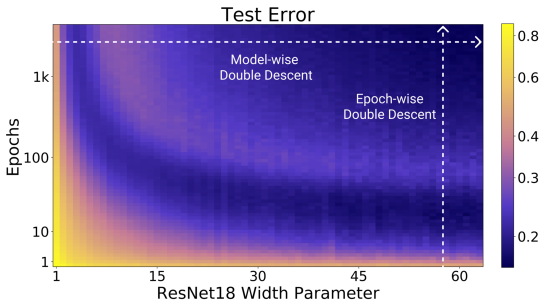


Figure 1.2: DD observed in practice in SL on CIFAR-10 with ResNets, varying training epochs (vertical axis), model capacity (horizontal axis), and test risk (color coding). The figure shows Double Descent occurs under both regimes. Figure taken from Nakkiran et al. (2019).

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 Literature Review: DD in SL

In SL, non-monotonic generalization behavior as a function of model complexity has been discussed in various forms for decades, with renewed attention in recent years (Loog et al., 2020). For instance, Boes & Opper (1996) reported that when model size is close to the number of training samples, optimization can slow substantially when training single layer perceptrons to a fixed test-risk threshold. The recent wave of interest was strongly influenced by Belkin et al. (2019), who provide a clear empirical demonstration of DD across various architectures and datasets in the capacity regime. They conduct an empirical study of DD using neural networks and decision trees on the MNIST dataset (Deng, 2012) as well as other real and synthetic datasets, which show that increasing the capacity of the tested models beyond that at which zero training risk is achieved leads to an ascent and then descent in terms of the test risk, as shown in Figure 1.1. They coin the term "Double Descent" to refer to this phenomenon, and posit that it occurs at the "interpolation threshold", which is the point at which model capacity (number of parameters) is equal to the size of the training set. Expanding upon this work, Nakkiran et al. (2019) show that DD can occur in both the capacity and the episodic regimes. They conduct a large-scale empirical study of DD across various architectures to show that DD is a widespread phenomenon in SL, and also show that it applies to not only the capacity regime but also the episodic regime, as well as showing that test risk as a function of dataset size can also exhibit non-monotonic behavior. They define a measure of "Effective Model Capacity" (EMC), the number of training samples a model can learn to zero train risk under a given model and training configuration, and posit that the interpolation threshold is at this EMC, determined not only by model parameter count. Figure 1.2 from their paper shows test risk as a function of both model capacity and training epochs, using convolutional neural networks of varying widths trained on an image classification task. DD is visible under both the capacity and the episodic regimes. They

also emphasize the role of label noise in amplifying the effect, showing that DD is more pronounced when there is label noise in the training data. Later, Nakkiran (2019) shows more clearly that DD can occur in the dataset size regime using linear regression models. d'Ascoli et al. (2021) further investigates DD and finds that peaks in the test risk can occur at two distinct points. They define the number of training samples N, the number of model parameters P, and the model's input dimension D. They find that the test risk can peak at both the interpolation threshold $N \approx P$ and at the "classical" threshold $N \approx D$, leading to a "Triple Descent" curve when plotting test risk against N. The curve exhibits its first descent in the regular underparameterized regime ($N < D$), followed by a peak at $N \approx D$, then a second descent in the overparameterized but non-interpolating regime ($D < N < P$), and finally a third peak at the interpolation threshold $N \approx P$, before descending again in the highly overparameterized regime ($N > P$). However, recent work has questioned the ubiquity of DD in SL. Buschjäger & Morik (2021) replicates the random forest experiments from Belkin et al. (2019) and finds that, when controlling for the number of trees and considering total node count, the test risk decreases monotonically with model size, showing no evidence of DD in this setting. Furthermore, Curth et al. (2023) argue that much of the reported "DD" in classical (non-deep) models arises because the plotted parameter count implicitly follows multiple distinct complexity directions, and the "second descent" appears when experiments switch from one way of adding parameters to another, so its location is not inherently tied to $p = n$. They further propose an effective-parameter view, under which the apparent DD curves can be reconciled with more traditional U-shaped behavior.

1.2 Prior Work on DD in RL

While DD is documented in SL, its status in RL 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, similar to the contributions of our work. 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 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 DD within RL, we need to define analogous concepts for test and train splits as well as risk. Since DD is defined as an emergence and then 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 tarin 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 computed as the empirical risk of the model, calculated on the train set using a loss function, while the test risk is computed as expected risk, approximated by using the same loss function on the test set. Common loss functions include mean squared error for regression tasks and cross-entropy loss for classification tasks. In RL, however, the concepts of train and test splits and calculating risk 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 RL, 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 its layout. By training an agent on a subset of these environments (the train 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 replace the concept or risk, we use mean return as a proxy for risk, where higher performance indicates lower risk. Additionally, we consider the Fisher Information Matrix (FIM) to indicate overfitting or memorizing in the model.

2.1 RL Preliminaries

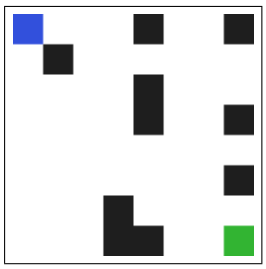
In RL, an agent interacts with an environment modeled as a Markov Decision Process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$. $\mathcal{S}$ represents the state space of the environment, which is the set of states that the agent can be in. $\mathcal{A}$ is the action space, which is the set of actions that the agent can take. The transition probability function $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1] = p(s_{t+1}|a_t, s_t)$ defines the probability of transitioning to state s_{t+1} given that the agent is currently in state s_t and takes action a_t. The reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R} = r(s_t, a_t)$ assigns a scalar reward to each state-action pair, indicating the immediate reward the agent receives for taking action a in state s. We can shape the reward using a potential function $\phi : \mathcal{S} \to \mathbb{R}$, which allows us to modify the reward structure to encourage certain behaviors. When using a potential function to shape the reward, the reward function becomes $r' : \mathcal{S} \times \mathcal{A} \to \mathbb{R} = r(s_t, a_t) + \gamma\phi(s_{t+1}) - \phi(s_t)$, where γ is the discount factor. This formulation encourages the agent to move towards states with higher potential, which can guide the learning process. The discount factor $\gamma \in [0, 1]$ determines the importance of future rewards compared to immediate rewards. When calculating future rewards, the agent discounts a reward t time steps in the future by γ^t, meaning that rewards received further in the future are worth less than immediate rewards, and decreasing γ places more emphasis on immediate rewards. The agent's behavior is determined by a policy $\pi_\theta : \mathcal{A} \times \mathcal{S} \to [0, 1] = \pi_\theta(a|s)$, which is a function defined by a parameter vector θ that yields the probability of selecting a specific action in a specific state.

One episode of interaction with the environment is defined as a sequence of states, actions, and rewards: $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$, where s_t is the state at time step t, a_t is the action taken at time step t, and r_t is the immediate reward received at time step t. The return G_t at time step t is defined as the discounted sum of future rewards: $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$. The agent's objective is to learn a policy that maximizes the expected return from each state, which is achieved by varying the parameter vector θ. We can express the expected return of following the policy π_θ from state s taking actino a as the action-value function $q_{\pi_\theta}(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R} = \mathbb{E}_{\pi_\theta}[G_t|s_t = s, a_t = a]$. The state-value function $v_\pi(s) : \mathcal{S} \to \mathbb{R} = \mathbb{E}_{\pi_\theta}[G_t|s_t = s]$ is the expected return from state s when following policy π. The advantage function $a_{\pi_\theta}(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R} = q_{\pi_\theta}(s, a) - v_{\pi_\theta}(s)$ represents how much better taking action a and following the policy π_θ in state s is compared to the expected return in that state under policy π. The agent learns by updating its policy parameters θ based on the observed returns and the estimated value functions, using algorithms such as policy gradients or value iteration.

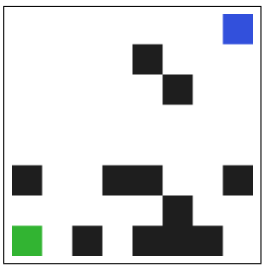
2.2 Environment

The environments used for this study are mazes, created as a subclass of the `gymnasium` (Towers et al., 2024) `Env` class. 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 regenerated to ensure solvability. The position of the goal and starting position of the agent can be randomized or set manually. 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 position and the goal position are randomized to one of the corners of the map at the start of each episode. Two example maps can be seen in Figure 2.1.

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



(a) Standard configuration.



(b) Randomized configuration.

Figure 2.1: Example environments. Walls are shown in black, the agent in blue, and the goal in green.

to indicate whether it is empty, a wall, the agent's position, or the goal's position. 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 state space for this Markov Decision Process (MDP) is thus

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

The action space is discrete and allows for moving in the four cardinal directions:

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

$p(s_{t+1}|s_t, a_t)$ is deterministic and always results in moving in the specified direction if it is accessible. If it is a wall or out of bounds, the agent does not move and $s_{t+1} = s_t$. The agent receives a reward of $+1$ upon reaching the goal. In all other timesteps, a small time penalty is applied together with potential based reward shaping based on the euclidean distance to the goal:

$$r'(s_t, a_t) = \begin{cases} 1, & \text{if goal reached} \\ \phi(s_t) - \phi(s_{t-1}) - 0.01, & \text{otherwise} \end{cases} \tag{2.3}$$

with the potential function defined as:

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

In the randomized configuration the minimum return is approximately -0.65899, while in the standard configuration it is -0.64. The maximum return in the randomized configuration is 1, while in the standard configuration it is approximately 0.96210. Calculations for these values can be found in the Appendix A. We compute the mean optimal return for the seeds 0–9,999 under the standard configuration as 0.958585. 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 only 1.75% among these seeds. In the randomized configuration, the mean optimal return over the seeds 0–9,999 is 0.969240, and 20.58% of the maps are optimal with a maximum return of 1. Considering the environment is truncated at 64 timesteps, we can set the discount factor $\gamma = 1$. This makes returns more interpretable.

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.

2.3 Metrics

For a y-axis representing train and test risk, we use mean return over our train and test maps as a proxy for risk. Higher return indicates lower risk. 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 regularly once every 1,000 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 FIM regularly. We use the state-action pair FIM, which is defined as:

$$F = \mathbb{E}_{(s,a)\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a \mid s)\, \nabla_\theta \log \pi_\theta(a \mid s)^T\right] \tag{2.4}$$

where (s, a) are state–action pairs drawn from trajectories generated by 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 $g = \nabla \log \pi_\theta(a \mid s)$. 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 sampled from trajectories.

The trace of the FIM, $\mathrm{Tr}(F)$, provides a measure of the magnitude of the FIM and thus 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, 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 (2.4), 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, using Petersen & Pedersen (2008):

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

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 trajectories generated by 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

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

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 $|\theta| \times |\theta|$ where $|\theta|$ is the number of parameters in the model.

Since the trace of the FIM scales with $|\theta|$, 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 also 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.

2.4 Models

We run experiments both using Deep Q-Networks (DQNs) (Mnih et al., 2013) and using a Policy Gradient method, specifically Trust Region Policy Optimization (TRPO) (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. Training hyperparameters can be found in the Appendix E.

2.4.1 DQNs

A DQN is a value-based RL 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:

$$q^*(s_t, a_t) = \mathbb{E}_{s_{t+1}} \left[r_t + \gamma \max_{a_{t+1}} q^*(s_{t+1}, a_{t+1}) \mid s_t, a_t \right] \tag{2.5}$$

where r_t is the immediate reward received after taking action a_t in state s_t, γ is the discount factor, and s_{t+1} is the next state. The DQN approximates $q^*(s, a)$ using a neural network parameterized by a parameter vector θ, denoted as $q_\theta(s, a)$. 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:

$$\mathbb{E}_{(s,a,r,s')} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - q_\theta(s, a) \right)^2 \right] \tag{2.6}$$

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.

2.4.2 TRPO

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:

$$L(\theta) = \mathbb{E}_{s,a \sim \pi_{\theta_{\text{old}}}} \left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} A^{\pi_{\theta_{\text{old}}}}(s, a) \right] \tag{2.7}$$

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:

$$\mathbb{E}_{s \sim \pi_{\theta_{\text{old}}}} \left[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot \mid s) \parallel \pi_\theta(\cdot \mid s)) \right] \leq \delta \tag{2.8}$$

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 for both the policy and the value networks. 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.

2.5 Training Runs

To search for DD, 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	Environment Configuration
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

Table 2.1: Capacity Regime Configurations

On the following configurations, we trained agents of depth 3 and widths 4, 16, 64, 256, and 1024 for unlimited time (in practice for 1.6–1.8M episodes, until our high performance cluster ended the jobs):

Type	Train Maps	Environment Configuration
TRPO	50	Standard
TRPO	100	Standard

Table 2.2: Episodic Regime Configurations

3 Results

In this section, we present the results of our experiments searching for DD in RL. Unfortunately, across all configurations tested, we did not observe instances of DD in either the episodic or capacity regime.

3.1 DQN Fails to Learn

First, we examined performance using DQNs on our environment. Unfortunately, this architecture was unable to solve the environment altogether. We observe frequent catastrophic forgetting, with spikes in mean return being present but short-lived. Examples of a DQN failing to learn the environment reliably can be found in Appendix C, along with plots showing DQN mean return as a function of model capacity. These show that the failure to learn occurs over all tested model capacities. No DD is observed, as neither the training risk nor the test risk exhibit any significant learning. Similarly, the average Fisher information does not exhibit significant change and stays extremely close to zero across our tested model capacities. These results are uninformative for studying DD, as the model fails to learn the environment. Without a rise in train performance, a subsequent ascent in test performance cannot be assessed. Given the poor performance of DQNs on this environment, we therefore turned to an alternative architecture that we expected to perform more effectively.

3.2 TRPO: No DD

We next examined performance using TRPO. This architecture was able to learn the environment reliably across a range of model capacities. However, we were again unable to observe DD in either the episodic 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 2.1. To plot the mean returns, we normalize by using the mean minimal and optimal returns across the seeds used for the specific configuration to scale the returns between 0 and 1. This makes the results more interpretable between different environment configurations. First, we used 10 train maps and 10,000 training episodes per model, with the standard environment configuration. We can observe that both training and test performance are both low for the smallest model capacities. As we increase model capacity, training performance increases rapidly until it reaches the mean maximum return. Test performance increases slightly but stabilizes quickly at about 10% of mean maximum return. No second ascent is observed in the test performance when increasing model capacity, and thus we cannot observe DD. At the same time, we observe a spike in the average Fisher information as the gap in mean return on the train and test set emerges. This is expected behaviour, as the model is likely to become more sensitive to parameter changes when it is overfitting on the train 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. Figure 3.1 shows the results of this experiment. Considering the minimal improvement in test performance, we increased the training set size as well as the training episodes. We tested a training set size of 50 seeds and 20,000 training episodes, a training set size of 100 train maps and 50,000 episodes, and a training set size of 1,000 maps and 100,000 episodes. Here, we can observe similar results, with slightly improved test performance after the first ascent to about 0.2 of mean maximum return for 50 train maps and about 0.4 for 100 train maps. The training performance decreases with larger amounts of train maps, with mean return keeping around 0.8 for 50 train maps and 0.7 for 100 train maps. In these two configurations, no DD

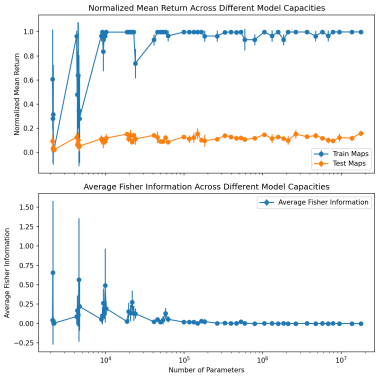


Figure 3.1: Trained on 10 maps for 10,000 episodes.

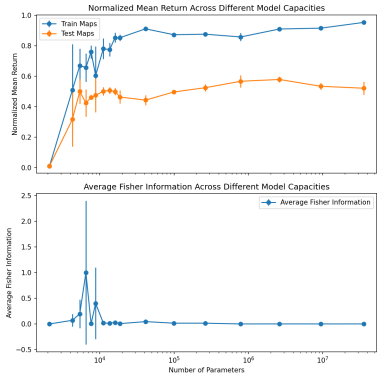


Figure 3.2: Trained on 500 maps for 1,000,000 episodes.

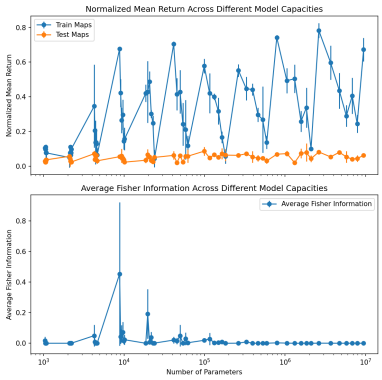


Figure 3.3: Trained on 50 maps for 20,000 episodes with randomized start and goal positions.

can be observed as the generalization gap does not close when increasing model capacity. As before, the average Fisher information again spikes as the train and test return diverges, but again decreases as model capacity increases further. When using 1,000 train maps, the train and test remain approximately equal and no generalization gap emerges at all, with both train and test return stabilizing at about 0.4 of mean maximum return. Considering there is no generalization gap, we cannot observe DD. In this configuration we also cannot observe the spike in the average Fisher information that emerges with the generation gap. There is one high outlier value, but no sustained increase in the average Fisher information as in the other configurations. Plots showing these results can be found in Appendix B.

At this stage, there is no generalization gap, and thus further studying such configurations is not useful to our study of DD. To continue with large train sets, we need to allow the model to learn longer and reach higher training performance. We thus increased the amount of training episodes per model to 1,000,000. For this increased training time, we tested train set sizes of 100, 500, 750, and 1,000 maps. This longer training time shows the models reaching much closer to the optimal performance. However, even with this increased training time, we were again unable to observe DD 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 all possible maps and consistent with our results using fewer training episodes. 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 decreases as model capacity increases further. An example of these results is shown in Figure 3.2, with other configurations showing similar results in Appendix B.

Finally, on the configurations of 50 train maps and 20,000 training episodes per model as well as 100 train maps 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 use the basic rule of down and to the right to reach the goal. Here, we observe that mean return varies strongly with model capacity, specifically with the depth of the model, with deeper models performing worse than shallower models when keeping the width fixed. 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). The test performance remains very low throughout all model capacities. This indicates that we would need to train on a larger set of train maps to see any generalization capacity whatsoever. The average Fisher information spikes approximately when the model reaches its highest performance on the training set, but then comes back down again.

This is similar to our earlier findings of the average Fisher information spiking as the generalization gap emerges. The results are visualized in Figure 3.3 and in Appendix B.

3.3 Episodic Regime

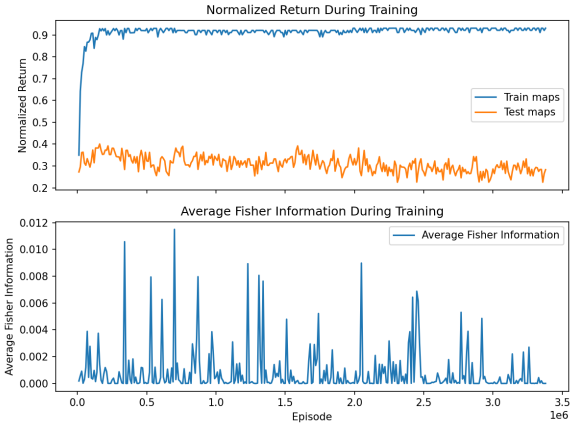


Figure 3.4: TRPO training and test performance over time for a model of width 1024 and depth 3, using 100 train maps and unlimited training episodes.

DD 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 DD under our conditions, we trained models of depth 3 and widths 4, 16, 64, 256, and 1024 on two configurations: 50 train maps and 100 train maps. Figure 3.4 shows an example of the learning behavior during training. We trained without a limit on the number of training episodes, but in practice were limited to about 1.6M to 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 D.

Again, the results do not exhibit signs of DD. We can observe a quick increase in train return 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 stabilizes at a low level between 0.2 and 0.4 for the tested configurations. 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 DD phenomenon.

4 Discussion

In general, we are unable to observe DD in our experiments. We tested for both forms of DD: the episodic and capacity regime. 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. However, the generalization gap does not close after emerging under any of our tested configurations. This suggests that DD does not occur under the conditions that we tested.

With regard to the average Fisher information, 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 DD may not be a universal phenomenon in RL, especially under the conditions we tested. It is impossible to determine a precise reason for this, considering the complexity of RL. However, we can speculate on a range of possible, potentially complementary explanations.

First, 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 train 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 remains in the partial fitting regime without transitioning into a qualitatively different overparameterized regime.

Secondly, unlike SL, RL data is 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 multi-layer perceptrons (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 train 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 DD 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 FIM for a policy $\pi_\theta(a \mid s)$ is an expectation of score outer products, as defined in Equation (2.4). 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 train 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.

First, we are constrained by the amount of computational resources 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 type 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 more suitable tasks. Future studies could investigate DD 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 DD 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. Considering DD often relies on the presence of noise in SL, adding stochasticity to the environment could create conditions more conducive to observing DD phenomena. The same effect could possibly be achieved by using a partially observable MDP (POMDP) setup, where the agent only receives limited observations of the environment state rather than the full grid. This would introduce uncertainty and require the agent to generalize from incomplete information, potentially influencing the emergence of DD behavior.

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 results 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 DD in RL 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 DD as documented in SL contexts. Notably, the absence of a clear interpolation threshold and the nonstationary nature of RL data can play significant roles in shaping generalization behavior differ-

ently 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 DD in RL, but rather motivate further investigation of DD within RL. Future research, as described in Section 5, is essential to clarify the conditions under which DD might manifest in RL.

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

Boes, S., & Opper, M. (1996, 01). Dynamics of training. In (p. 141-147).

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

Buschjäger, S., & Morik, K. (2021). There is no double-descent in random forests. *ArXiv*, *abs/2111.04409*. Retrieved from https://api.semanticscholar.org/CorpusID:243848097

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

Curth, A., Jeffares, A., & van der Schaar, M. (2023). A u-turn on double descent: rethinking parameter counting in statistical learning. In *Proceedings of the 37th international conference on neural information processing systems.* Red Hook, NY, USA: Curran Associates Inc.

d'Ascoli, S., Sagun, L., & Biroli, G. (2021). Triple descent and the two kinds of overfitting: where and why do they appear? *Journal of Statistical Mechanics: Theory and Experiment*, *2021*(12), 124002. doi: 10.1088/1742-5468/ac3909

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, *29*(6), 141–142.

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

Loog, M., Viering, T., Mey, A., Krijthe, J. H., & Tax, D. M. J. (2020). A brief prehistory of double descent. *Proceedings of the National Academy of Sciences*, *117*(20), 10625-10626. Retrieved from https://www.pnas.org/doi/abs/10.1073/pnas.2001875117 doi: 10.1073/pnas.2001875117

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. (2019). *More data can hurt for linear regression: Sample-wise double descent.* Retrieved from https://arxiv.org/abs/1912.07242

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

Petersen, K. B., & Pedersen, M. S. (2008, October). *The matrix cookbook.* Technical University of Denmark. Retrieved from http://www2.imm.dtu.dk/pubdb/p.php?3274 (Version 20081110)

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

Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., ... others (2024). Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*.

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 Minimum and Maximum Environment Returns

We use d_{max} to represent the maximum distance the agent can be from the goal, and d_{start} to represent the distance from the agent's starting position to the goal, and s_{start} and s_{end} to represent the starting and ending states of the agent. To calculate the minimum reward in the randomized configuration, we assume the agent starts in a corner adjacent to the corner of the goal tile, so that the agent can move farther away. In this case, the reward can be calculated as follows:

$$G_{min} = (-0.01 \times 64) - \frac{1}{100} \times (d_{\max} - d_{\text{start}}) \tag{A.1}$$

$$= -0.64 - \frac{1}{100} \times (\sqrt{7^2 + 7^2} - 8) \tag{A.2}$$

$$\approx -0.64 - \frac{1}{100} \times (9.899 - 8) \tag{A.3}$$

$$= -0.64 - \frac{1}{100} \times 1.899 \tag{A.4}$$

$$= -0.64 - 0.01899 \tag{A.5}$$

$$= -0.65899 \tag{A.6}$$

To calculate the minimum reward on the standard configuration, we assume the agent does not move any closer to the goal during the episode. Since it starts directly opposite from the goal, it cannot move any farther either. Therefore, the minimum reward in the standard configuration is simply the reward for taking 64 steps without reaching the goal:

$$G_{min} = -0.01 \times 64 \tag{A.7}$$

$$= -0.64 \tag{A.8}$$

. The maximum reward in the randomized configuration is achieved if the agent starts in the corner adjacent to the corner of the goal tile and reaches the goal by moving directly to the goal without having to move around any obstacles. In this case, the agent needs 7 timesteps to reach the goal. The time penalty and reward shaping are not applied in the timestep in which the agent reaches the goal, so the reward can be calculated as follows:

$$G_{max} = (-0.01 \times 6) - \frac{1}{100} \times (1 - d_{\text{start}}) + 1 \tag{A.9}$$

$$G_{max} = -0.06 - \frac{1}{100} \times (1 - 7) + 1 \tag{A.10}$$

$$G_{max} = -0.06 - \frac{-6}{100} + 1 \tag{A.11}$$

$$G_{max} = -0.06 + 0.06 + 1 \tag{A.12}$$

$$G_{max} = 1 \tag{A.13}$$

The maximum reward in the standard configuration is achieved if the agent reaches the goal in the minimum number of steps, which is 14 since the agent must move 7 tiles downwards and 7 tiles to the right. The last step does not include the time penalty or reward shaping, so the distance-based reward applies only until one tile next to the goal and the time penalty applies to only 13 steps. Thus the maximum reward is:

$$G_{max} = 1 - (0.01 \times 13) - \phi(s_{end}) + \phi(s_{start}) \tag{A.14}$$

$$= 1 - 0.13 - (\frac{1}{100} \times 0) + (\frac{1}{100} \times \sqrt{7^2 + 6^2}) \tag{A.15}$$

$$\approx 0.87 + (\frac{1}{100} \times 9.21954446) \tag{A.16}$$

$$= 0.87 + 0.09210 \tag{A.17}$$

$$= 0.96210 \tag{A.18}$$

$$\tag{A.19}$$

B Additional TRPO Plots

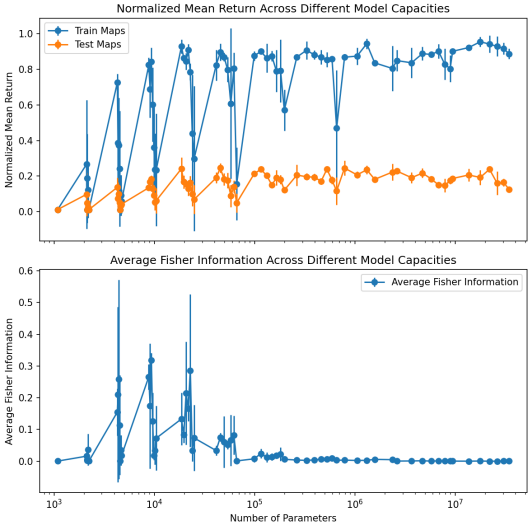


Figure B.1: TRPO training and test performance, and average Fisher information, as a function of model capacity, using 50 train maps and 20,000 training episodes per model.

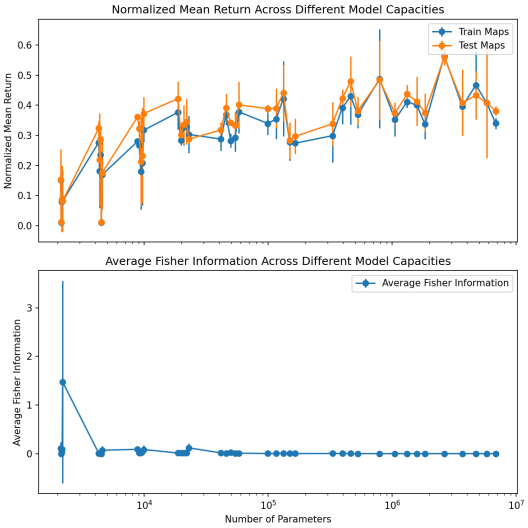


Figure B.3: TRPO training and test performance, and average Fisher information, as a function of model capacity, using 1,000 train maps and 100,000 training episodes per model.

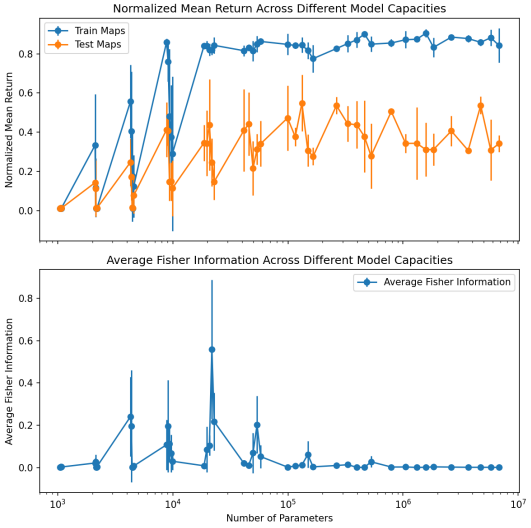


Figure B.2: TRPO training and test performance, and average Fisher information, as a function of model capacity, using 100 train maps and 50,000 training episodes per model.

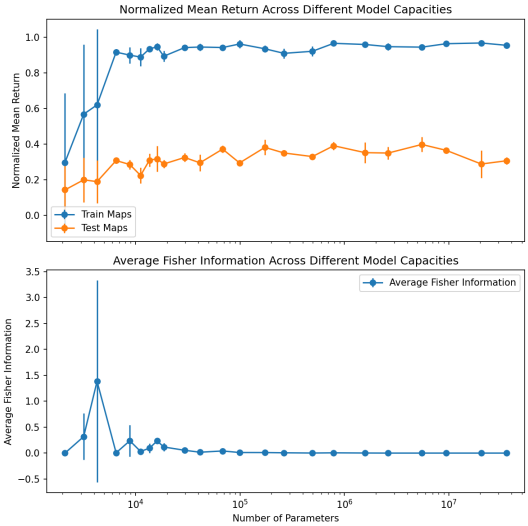


Figure B.4: TRPO training and test performance, and average Fisher information, as a function of model capacity, using 100 train maps and 1,000,000 training episodes per model.

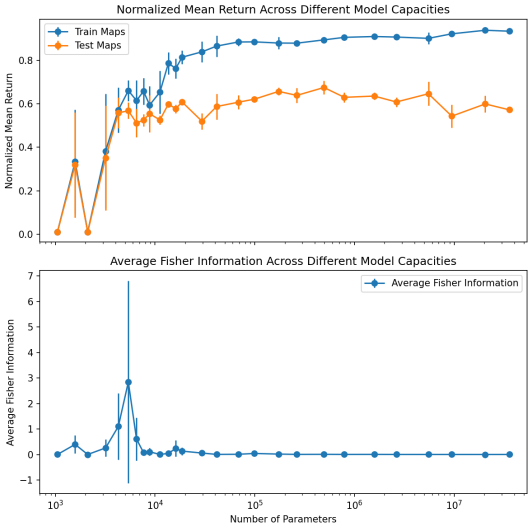


Figure B.5: TRPO training and test performance, and average Fisher information, as a function of model capacity, using 750 train maps and 1,000,000 training episodes per model.

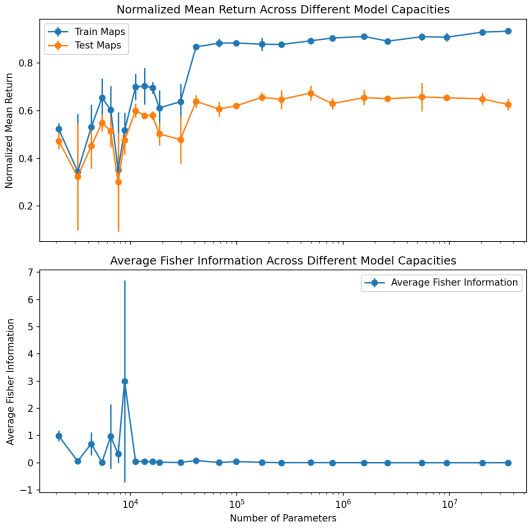


Figure B.6: TRPO training and test performance, and average Fisher information, as a function of model capacity, using 1,000 train maps and 1,000,000 training episodes per model.

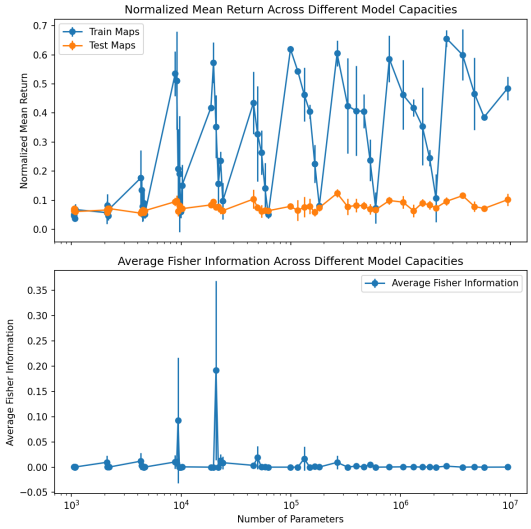
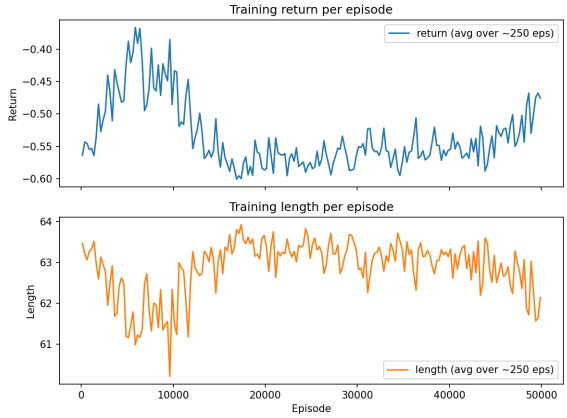


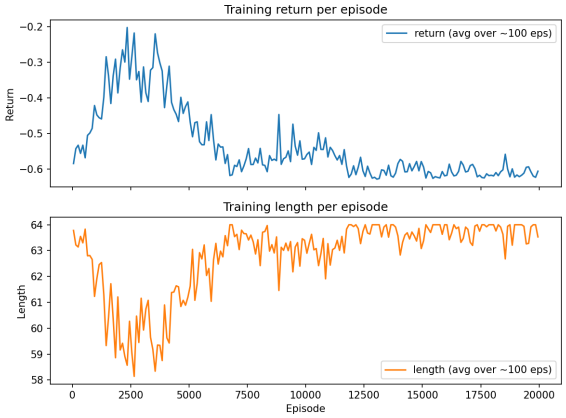
Figure B.7: TRPO training and test performance, and average Fisher information, as a function of model capacity, using 100 train maps and 50,000 training episodes per model and randomized start and goal positions.

C DQNs Failing To Learn

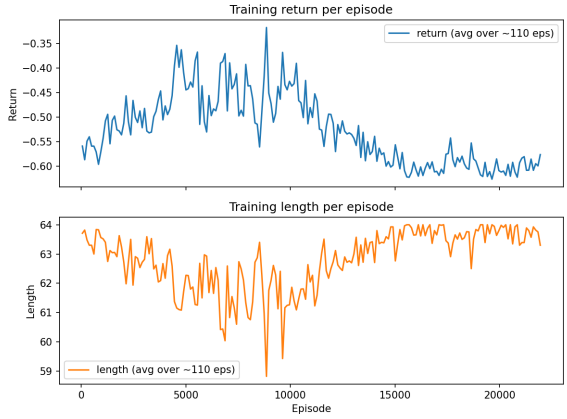
Throughout runs, DQNs failed to learn a good policy on the environment. Examples of DQN mean return and episode length during training are shown in Figure C.1. In some cases we see some initial learning, but the model seems to forget and revert to minimal return. Plotting our DQN results by capacity, we see no significant learning happening across all tested capacities, as shown in Figure C.2.



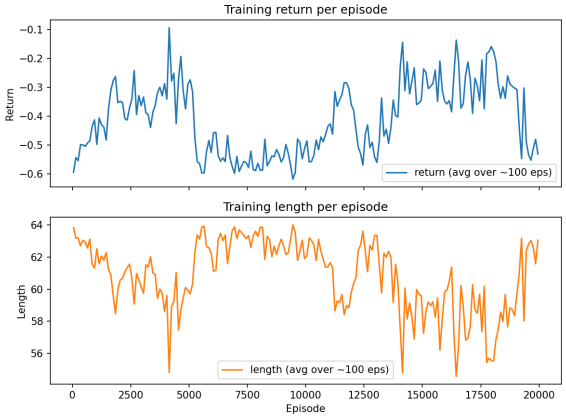
(a) A DQN with 3 hidden layers and 512 neurons per hidden layer, trained on 100 seeds for 50,000 episodes.



(b) A DQN with 3 hidden layers and 512 neurons per hidden layer, trained on 50 seeds for 20,000 episodes.

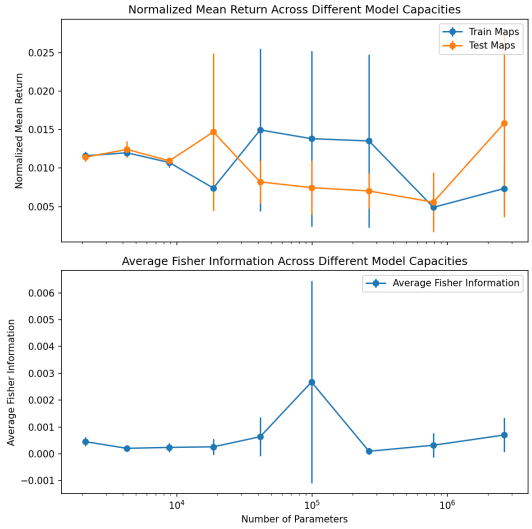


(c) A DQN with 3 hidden layers and 1024 neurons per hidden layer, trained on 100 seeds for 50,000 episodes.

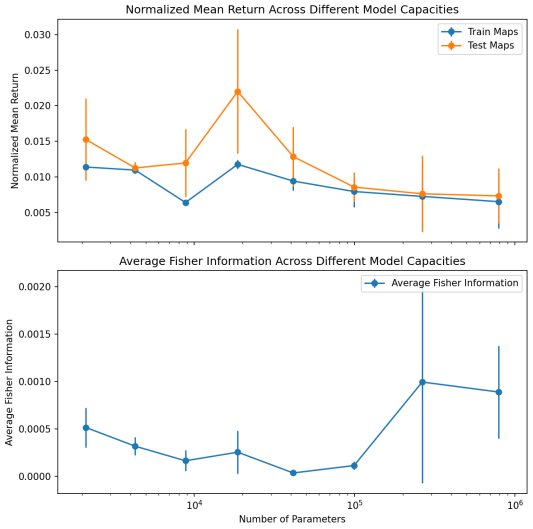


(d) A DQN with 3 hidden layers and 1024 neurons per hidden layer, trained on 50 seeds for 20,000 episodes.

Figure C.1: Mean return and episode length of selected DQNs during training.



(a) Metrics of DQNs using 50 train maps and 20,000 training episodes per model.



(b) Metrics of DQNs using 100 train maps and 50,000 training episodes per model.

Figure C.2: DQN training and test risk, and average Fisher information, as a function of model capacity. Neither curve exhibits significant learning.

D Longer Training Runs

We ran long training runs for the episodic regime under TRPO for 50 and 100 train maps 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.

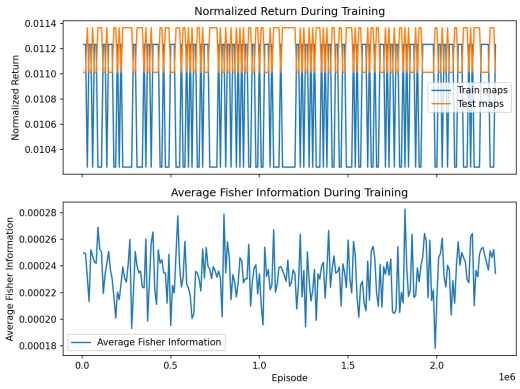


Figure D.1: Model width 4, training set size 50. The model is too small and fails to learn.

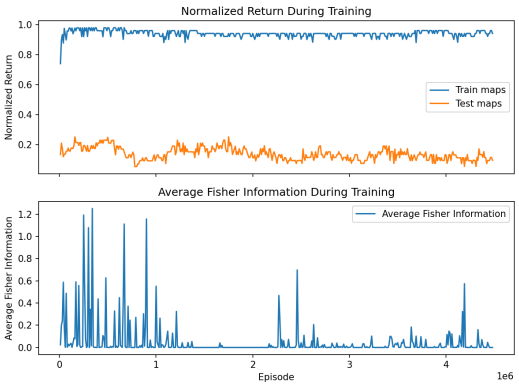


Figure D.2: Model width 16, training set size 50.

E Training Hyperparameters

E.1 DQN

We use a replay buffer size of 50,000 transitions and a target network update frequency of 5,000 training steps. The DQN is trained using the Adam optimizer (Kingma & Ba, 2017) with a learning rate of 10^{-3} and a batch size of 64. The exploration-exploitation trade-off is managed using an epsilon-greedy strategy, where the exploration rate ϵ decays

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

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

linearly from 1.0 to 0.05 over the first 30% of the episodes.

E.2 TRPO

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}.

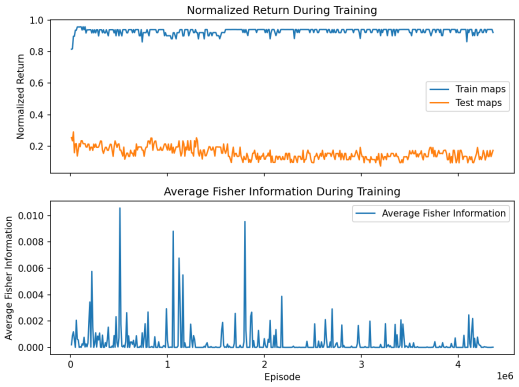


Figure D.5: Model width 1024, training set size 50.

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

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

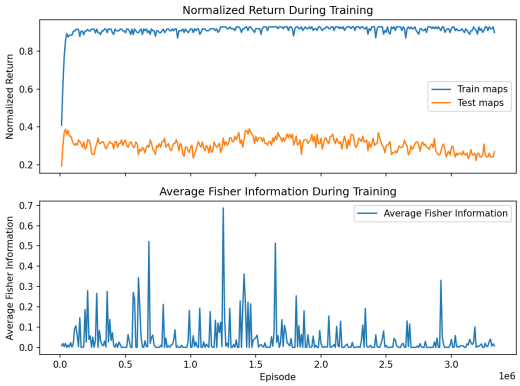


Figure D.8: Model width 64, training set size 100.

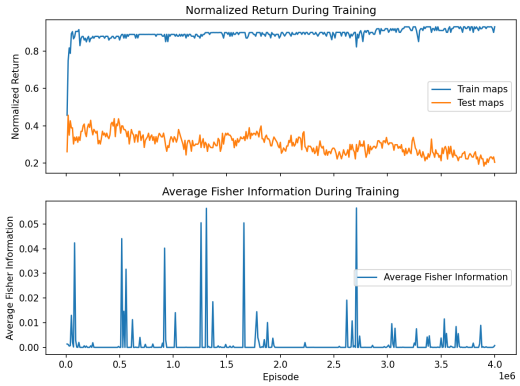


Figure D.9: Model width 256, training set size 100.

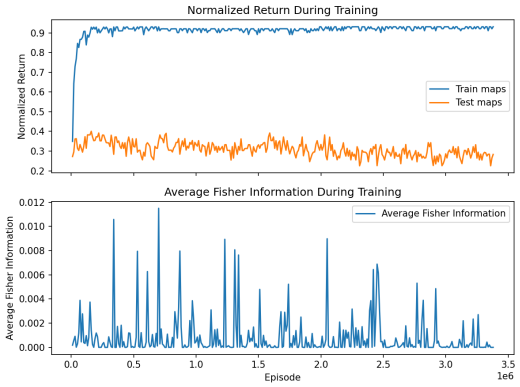


Figure D.10: Model width 1024, training set size 100.