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 risk phenomenon in which a deep model's test risk worsens around an 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 Deep Q-Network (DQN) and Trust-Region Policy Optimization (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 returns on training and held-out test maps. However, upon increasing the size of the model as well as the amount of training episodes we do not observe a second descent regime in test risk. 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) has been observed in supervised learning (SL) (Belkin et al., 2019; Loog et al., 2020). When increasing model capacity, training steps, 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 (Hastie et al., 2001, p. 221). Past this point, increasing model capacity, training steps, 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, which can be seen on the left of Figure 1.1. However, recent work has shown that this classical regime does not always persist. After this initial overfitting phase, increasing model capacity, training steps, or dataset size further leads to a second descent in test risk, 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; Hastie et al., 2020).

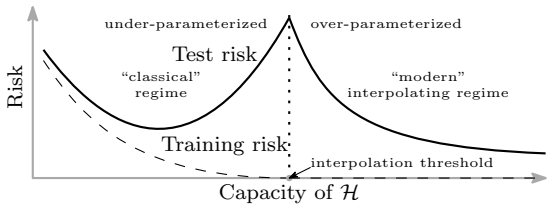


Figure 1.1: DD risk curve. After the initial overfitting phase, increasing model capacity, training steps, or dataset size further leads to a second descent in test risk. Figure taken from Belkin et al. (2019).

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 focused on DD 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

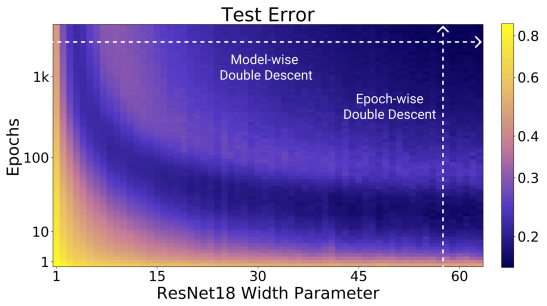


Figure 1.2: DD observed in practice in SL on CIFAR-10 with ResNets, varying training epochs (vertical axis), model capacity (horizontal axis), and test risk (color coding). The figure shows Double Descent occurs under both regimes. Figure taken from Nakkiran et al. (2019).

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 database (Lecun et al., 1998) as well as other real and synthetic datasets, showing that increasing the capacity of the tested models beyond that at which zero train risk is achieved leads to an ascent and then descent in terms of the test risk, as shown in Figure 1.1. The authors 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 train set. Expanding upon this work, Nakkiran et al. (2019) conduct a large-scale empirical study to provide further evidence that DD is a widespread in SL. The authors demonstrate that DD occurs not only with regard to the capacity regime but also with regard to the episodic regime (the amount of training steps), as well as 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 parameter count. Figure 1.2, taken 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 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. In the noiseless case, they often observe a 'plateau' rather than a peak in the test risk at the interpolation threshold. 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 investigate DD and find that peaks in the test risk can occur at two distinct points. Given the number of training samples N, the number of model parameters P, and the model's input dimension D, the authors 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 when the capacity matches the training data. They further propose an effective-parameter view, under which the apparent DD curves can be reconciled with more traditional U-shaped behavior.

The underlying mechanisms behind DD are still debated, but several complementary hypotheses have emerged in the SL literature. One explanation holds that the interpolation threshold creates a variance spike in the learned function (Belkin et al., 2019). As the model approaches the capacity to fit the training data exactly, it must contort its deci-

sion 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. 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 explicitly be shown to be due to the variance of the estimator diverging at the interpolation threshold (Hastie et al., 2020). Additionally, even when models are large enough to interpolate, training algorithms such as stochastic gradient descent do not pick arbitrarily from the set of parameter vectors that can achieve zero train risk. Instead, optimizers like stochastic gradient descent tend to select solutions with particular simplicity biases, such as small norms of the parameter vector (Gunasekar et al., 2017; Li et al., 2018). From this view, the second descent occurs because, with sufficient capacity, the optimizer can find interpolating solutions that also have smaller norms (Belkin et al., 2019). Nakkiran et al. (2019) also show DD is often most pronounced when there is irreducible noise in the targets (e.g., label noise). In that case, interpolating the train 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.

1.2 Prior Work on DD in RL

The existence of generalization gaps in RL is known for some time. Zhang et al. (2018) show that generalization gaps can emerge in RL when agents are trained on a set of train environments and evaluated on a held-out set of test environments, using gridworld maze environments randomly generated based on a seed. The authors find the generalization gap diminishes as the train set size increases but are unable to demonstrate a U-shaped curve in test performance as a function of training steps, with training return instead stagnating at a suboptimal level. Cobbe et al. (2019) also use procedurally generated environments to show that RL agents can overfit to a surprisingly large number of training environments, and that training performance can increase with training set size, which they attribute to an implicit curriculum effect. They use models of three capacities and find that larger models generalize better on their environments. Mediratta et al. (2024) test generalization gaps in offline RL and find that the generalization gap is smaller as the model's capacity increases. They also find overfitting in the episodic regime analogous to the "classical regime" in SL, noting that extended training yields further benefits in train performance but hurts test performance.

While generalization gaps are well documented, the status of DD in RL is less clear and appears to depend on the learning setup and on what is used as a proxy for generalization. Brellmann et al. (2024) provide a theoretical account of DD in an RL context by studying policy evaluation with the regularized Least-Square Temporal Difference algorithm 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. 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.

2 Methods

This section describes the methods we use to investigate the existence of DD in RL. We first outline mathematical preliminaries of RL concepts and notations. We then showcase the environment we use to create train and test splits in RL, and the metrics we use to measure generalization capacity. Finally, we describe the training setup and algorithms we use for our models.

2.1 RL Preliminaries

In RL, an agent interacts with an environment modeled as a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$ (Sutton & Barto, 2018). $\mathcal{S}$ represents the state space of the environment, the set of all possible states the environment can be in. $\mathcal{A}$ is the action space, the set of actions that the agent can take. The transition probability function $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1] = p(s'|a, s)$ defines the probability of transitioning to state s' given that the agent is in state s and takes action a. The reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R} = r(s, a)$ assigns a scalar reward to each state-action pair, indicating the immediate reward the agent receives for taking action a in state s. 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 valued less than immediate rewards, and decreasing γ increases the emphasis placed 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 a in state s. The environment is called "Markovian" if the transition probabilities and rewards depend only on the current state and action, and not on the history of past states and actions. The reward function can be shaped using a potential function $\phi : \mathcal{S} \to \mathbb{R}$, which is used 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. Ng et al. (1999) show that this formulation does not change the optimal policy in the environment. This MDP formulation $(\mathcal{S}, \mathcal{A}, p, r', \gamma)$ encourages the agent to move towards states with higher potential, which can be used to guide the learning process.

One episode of interaction with the environment is defined as a sequence of states, actions, and rewards: $\tau = (S_0, A_0, R_1, S_1, A_1, R_2, \ldots)$, 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: $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k,$, where T is the final time step of the episode. The agent's objective is to learn a policy that maximizes the expected return from each state, which is achieved by optimizing the parameter vector θ. We can express the expected return of following the policy π_θ from state s taking action 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 $v_{\pi_\theta}(s) : \mathcal{S} \to \mathbb{R} = \mathbb{E}_{\pi_\theta}[G_t|s_t = s]$ is the expected return from state s when following policy π_θ. 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_θ (Schulman et al., 2017). The agent learns by updating its policy parameters θ based on interaction with the environment. Markovian environments have at least one deterministic optimal policy, which is a policy that achieves the maximum expected return from each state. This policy is denoted by π^*. Similarly, the optimal action-value and state-value functions, that is the maximum expected return from each state-action pair and state respectively, are denoted by $q^*(s, a) = \max_\pi q_\pi(s, a)$ and $v^*(s) = \max_\pi v_\pi(s)$. The state-value and action-value functions can be expressed in terms of the Bellman equations. The Bellman equation for the state-value function is given by

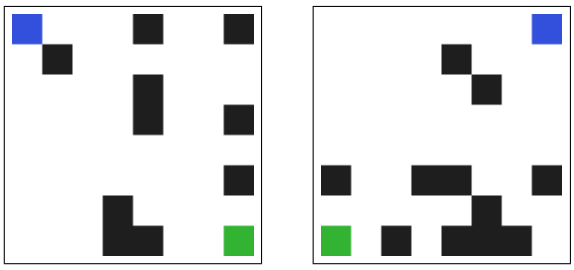
$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s')\right]$$

and the action-value function is given by

$$q_\pi(s, a) = \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s', a')\right].$$

2.2 Environment

As an analogue to train/test splits in SL, we create a family of gridworld-like maze environments randomly generated based on a seed, implemented as a subclass of the `gymnasium Env` class (Towers et al., 2024). 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



(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.

evaluating its returns on a separate subset of unseen environments (the test split), we can emulate the train-test paradigm from SL. 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. The position of the goal and starting position of the agent can be randomized or set deterministically. In the deterministic 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 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 time steps, a small time penalty is applied together with potential based reward shaping based on the Euclidean distance to the goal:

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

with the potential function defined as:

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

Considering the environment is truncated at 64 time steps, we can set the discount factor $\gamma = 1$, as this makes returns more interpretable. Therefore, the MDP for 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. In the randomized configuration the minimum return for this environment 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 Appendix A. We compute the mean maximum 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 maximum return over the seeds 0–9,999 is 0.969240, and 20.58% of the maps have a maximum return of 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 collect one episode from each map in the train and test sets. 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 regularly once every 1,000 training episodes and at the end of training. This is our most straightforward measure of a generalization gap between training and test returns, and in this context we would look for a generalization gap emerging between the train and test return curves as train return outpaces test return when we we increase model capacity or training episodes, which then narrows as we continue to increase capacity or training episodes, thus forming a DD curve.

To further study generalization capacity, we also measure the trace of the Fisher Information Matrix (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 $|\theta|^2$, 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. We call this metric 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 capacity alongside mean return. To support findings of DD 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 returns. 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.

The use of the FIM trace as a diagnostic signal for generalization capacity is motivated by recent work. Jastrzebski et al. (2020) show in the SL case that the trace of the FIM correlates with final generalization performance in deep networks trained with stochastic gradient descent, and that a large $\mathrm{Tr}(F)$ early in training, which they term catastrophic Fisher explosion, coincides with overfitting and poor test accuracy. They further demonstrate that explicitly regularizing $\mathrm{Tr}(F)$ recovers generalization, providing evidence that $\mathrm{Tr}(F)$ is not merely correlated with but causally linked to generalization behavior. Falzari & Sabatelli (2025) propose characterizing the primacy bias in deep RL through the evolution of the FIM trace, identifying a memorization phase marked by a sharp spike followed by a reorganization phase featuring a decline despite continued performance improvement. The authors interpret a high FIM trace magnitude as associated with poor generalization, and use its evolution to detect when early experiences disproportionately dominate learning. Todorov et al. (2025) similarly track the FIM trace as a plasticity indicator in multi-task RL, observing that agents converging to less sensitive parameter configurations exhibit lower and more stable trace values post-training, and that sparsification methods which improve generalization tend to reduce or stabilize the FIM trace.

2.4 Models

We run experiments both using DQNs (Mnih et al., 2015) and using a Policy Gradient method, specifically 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 B.

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:

$$q^*(s, a) = \mathbb{E}_{s'}\Big[\, r + \gamma \max_{a'} q^*(s', a') \;\Big|\; s, a \Big]$$

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

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)} \left[\Big(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \Big)^2 \right]$$

where θ^- are the parameters of a target network that is periodically updated, to stabilize training, and $(s, a, r, s') \sim U(D)$ are transitions sampled uniformly from a replay bufffer D, which contains such transitions collected from interactions with the environment. This stabilizes training by ensuring the target for the Q-value update is stable, and by breaking correlation between consecutive samples.

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) (Kullback & Leibler, 1951) divergence between the old and new policies. The surrogate objective function is defined as:

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

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:

$$\mathbb{E}_{s\sim\pi_{\theta_{\text{old}}}}\left[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot \mid s) \parallel \pi_\theta(\cdot \mid s))\right] \leq \delta \quad (2.5)$$

where δ is a predefined threshold that limits the size of the policy update. TRPO uses a conjugate gradient (CG) algorithm, an iterative method for solving large symmetric positive-definite linear systems without forming matrices explicitly, to compute the natural-gradient step by approximately solving $Fx = g$ for the search direction, where F is the FIM and g is the policy gradient. The constrained optimization problem is to maximize the surrogate objective $L(\theta)$ subject to the average KL-divergence constraint given by Equation 2.5, and the policy is updated by scaling the CG direction to satisfy this bound.

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 configurations shown in Table 2.1, we trained agents of varying depths between 3 and 8 and widths between 4 and 1024. On the configurations shown in table 2.2, 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).

3 Results

In this section, we present the results of our experiments searching for DD in RL. We were unable to observe DD across any tested configurations in both the episodic and capacity regimes.

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	100	1,000,000	Standard
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

Type	Train Maps	Environment Configuration
TRPO	50	Standard
TRPO	100	Standard

Table 2.2: Episodic Regime Configurations

3.1 DQN Fails to Learn

First, we examined generalization capacity using DQNs on our environment. This architecture was unable to solve the environment altogether. We observe frequent catastrophic forgetting, with increases 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 all tested model capacities fail to learn a meaningful policy. Similarly, the average Fisher information does not exhibit significant change and remains extremely close to zero across our tested model capacities. Given the poor performance of DQNs on this environment, the results are uninformative to our study of DD and we turned to an alternative architecture that we expected to perform more effectively.

3.2 TRPO: No DD

We next examined generalization capacity 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 are the size of the train set (in seeds) and the amount of training steps (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 minimum and maximum returns across the seeds used for the specific configuration to scale the returns between 0 and 1, enabling comparisons 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 returns are both low for the smallest model capacities. As we increase model capacity, training return increases rapidly until it reaches the mean maximum return. Test return increases slightly but stabilizes quickly at about 10% of mean maximum return. No second rise is observed in the test return 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, which subsides again as model capacity increases further. Figure 3.1 shows the results of this experiment. Considering the minimal improvement in test return, we increased the train set size as well as the training episodes. We tested a train set size of 50 seeds and 20,000 training episodes, a train set size of 100 train maps and 50,000 episodes, and a train set size of 1,000 maps and 100,000 episodes. Here, we can observe similar results, with slightly improved test return after the first rise to about 0.2 of mean maximum return for 50 train maps and about 0.4 for 100 train maps. The training return 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 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 D.

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 return. 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 maximum return. However, even with this increased training time, we were again unable to observe DD in test return. We see the generalization gap closing slowly with increased train 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 return on the train set but lags behind in test return, 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 D.

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).

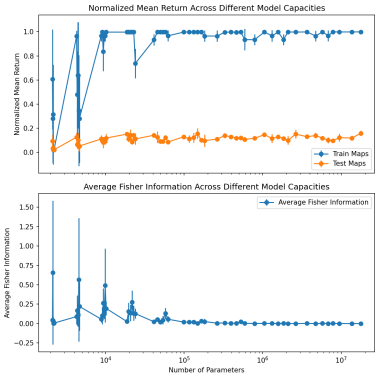


Figure 3.1: Trained on 10 maps for 10,000 episodes.

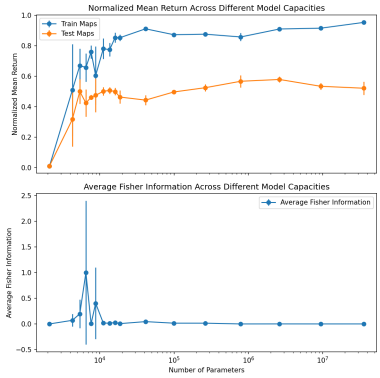


Figure 3.2: Trained on 500 maps for 1,000,000 episodes.

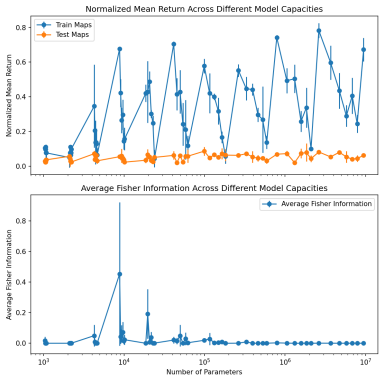


Figure 3.3: Trained on 50 maps for 20,000 episodes with randomized start and goal positions.

The test return 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 return on the train 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 D.

The initial increase of the average Fisher information 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 consistently 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 train set. One reason this could be occurring is because we normalize by dividing by parameter count of the model. We observe that the average Fisher information is highest at the capacity where the model starts to overfit on the train set, but as we increase capacity further, the added parameters may add dimensions of low or zero Fisher information, which would lower the average even if the total Fisher information without normalization remains high. However, after examining the absolute FIM trace, we see a very similar trend. This suggests that the decrease in average Fisher information could be strengthened by the normalization, but is not solely caused by it. A selection of plots including both average Fisher information and absolute FIM trace can be found in Appendix E. Another possible cause is that the Fisher information we estimate is 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). This means that a decrease in the average Fisher information can reflect a change in how stochastic the policy is, rather than an improvement in out-of-distribution generalization. Because the environment is Markovian and fully observable, there exists an optimal deterministic stationary policy (up to tie-breaking). Therefore, as we increase the capacity of the models, they may be able to learn a closer to optimal, and thus more deterministic policy on the train set, which would thus have lower Fisher information. This can coexist with poor returns 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. Additionally, if this is the case, the distribution of states visited by the model would also shrink as the capacity grows and the model becomes explores less and exploits more. If fewer states are regularly visited, then the average can decrease even if the policy remains sensitive to

parameter changes on those states.

If these hypotheses are accurate, this questions the usefulness of the average Fisher information as a measure of generalization capacity in RL, as it may be confounded by changes in the state–action visitation distribution and policy stochasticity rather than reflecting true improvements in out-of-distribution generalization.

3.3 Episodic Regime

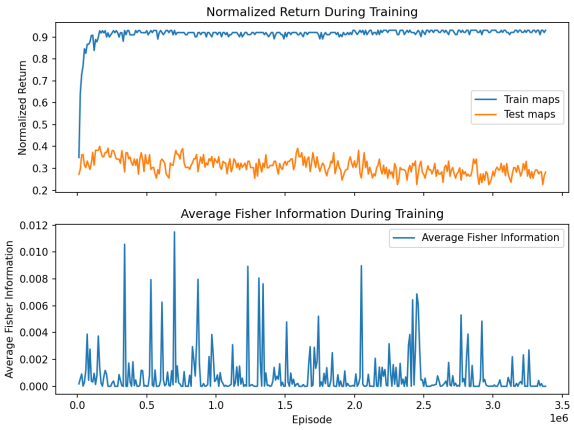


Figure 3.4: TRPO training and test return over time for a model of width 1024 and depth 3, using 100 train maps and unlimited training episodes.

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 F.

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 return stays at a near-maximum level. The difference to the maximum level indicates that the model fails to solve a small number of maps or learned a suboptimal policy on maps it can solve. Test return 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 return over time, indicating that the model may be overfitting slightly to the train set. However, no second rise is observed in the test return within our training time, and thus no DD phenomenon.

4 Discussion and Future Work

In general, we are unable to observe DD in our experiments after testing for both the capacity and episodic regimes. Initially, both train and test returns increase with both model capacity and training episodes. After a certain point, the test return stagnates while the train return continues to increase, leading to a widening generalization gap, which does not close under any of our tested configurations. The same occurs in the episodic regime. This suggests that DD does not occur under the conditions that we tested. We propose two 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 fit the train set. In our experiments, train return for TRPO often saturates below the environment's maximum return. 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. To attempt to reach perfect return on the training maps, we could make two main changes in the model. We could increase the number of parameters and training episodes further, which is currently limited by our computational resources, or we could change the architecture of our model from a MLP to a convolutional neural network (CNN) (Le Cun et al., 1989), which is better suited to our spatial task and observation space. In this case, we would process an 8×8 grid input with 8 channels (four object types and frame-stacking of two observations).

Secondly, unlike SL, RL data is policy-induced and nonstationary. As the policy improves, the state–action visitation distribution changes. This violates the usual fixed distribution that is an assumption in SL. 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). In future work, this could be addressed by training an off-policy method on a fixed dataset of trajectories collected on a set of training maps, and then plotting performance of the trained model on held-out test maps as a function of capacity. This would be more analogous to the SL setting and would allow us to test whether DD can appear in RL when the data distribution is fixed. It would be necessary to ensure that the chosen off-policy method can reliably fit the training data, which we struggled to achieve using a DQN.

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 would usually suggest that the model is finding a more generalizable policy after the initial memorization phase, corresponding to a flatter gradient landscape. However, the decrease in average Fisher information does not correspond to an improvement in test return, which remains stagnant after the initial increase. This decoupling of Fisher information dynamics and generalization capacity is puzzling and warrants further investigation to understand the underlying mechanisms at play. One way to further investigate this is by measuring the effective rank of the FIM along with its trace, which would give insight into how many dimensions of high sensitivity the model has and how this changes with capacity. Another possible analysis is to examine the state visitation distribution across training runs and model capacities to see if it shrinks as capacity increases, which would support the hypothesis that the model is becoming more deterministic and thus visiting fewer states. Another signal of the policy becoming more deterministic would be a decrease in the entropy of the policy distribution, which could be measured across model capacities.

In the episodic regime, we observe a slight downward trend in test return 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 return with more training. This trend emerges repeatedly across different train 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, such as by continuing training runs into the order of tens or hundreds of millions of episodes, both to observe for how long this trend continues and whether it eventually reverses into increasing average return in the episodic regime at extremely high episode counts. It is possible that a clear DD curve only emerges when plotting on a logarithmic scale with regard to episodes. Additionally, it would be interesting to also measure effective rank of the FIM in addition to its trace in the episodic regime, to see if it decreases over training.

Considering DD is strongest in SL when irreducible noise is present in the data, 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.

One main limitation we face is the availability of computational resources, which limited the number of configurations we could test and forced us to search the hyperparameter space in a heuristic manner. Future work could leverage greater computational resources to employ a grid search over the train set size, model capacity, and number of training episodes exhaustively. Concretely, it would be informative to create a similar plot for each train set size as Nakkiran et al. (2019) did in Figure 1.2.

In this work, we examined generalization behavior of RL agents across different model capacities and training regimes in an effort to show DD. Under both the episodic and capacity regimes, we were unable to observe a clear DD curve, with the generalization gap remaining mostly constant with increased capacity and training results. Overall, our results suggest that DD may not be a universal phenomenon 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

Cobbe, K., Klimov, O., Hesse, C., Kim, T., & Schulman, J. (2019, 09–15 Jun). Quantifying generalization in reinforcement learning. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th international conference on machine learning* (Vol. 97, pp. 1282–1289). PMLR. Retrieved from https://proceedings.mlr.press/v97/cobbe19a.html

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

Falzari, M., & Sabatelli, M. (2025). *Fisher-guided selective forgetting: Mitigating the primacy bias in deep reinforcement learning.* Retrieved from https://arxiv.org/abs/2502.00802

Gunasekar, S., Woodworth, B., Bhojanapalli, S., Neyshabur, B., & Srebro, N. (2017). Implicit regularization in matrix factorization. In *Proceedings of the 31st international conference on neural information processing systems* (p. 6152–6160). Red Hook, NY, USA: Curran Associates Inc.

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

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning.* New York, NY, USA: Springer New York Inc.

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

Jastrzebski, S., Arpit, D., Åstrand, O., Kerg, G., Wang, H., Xiong, C., ... Geras, K. J. (2020). Catastrophic fisher explosion: Early phase fisher matrix impacts generalization. In *International conference on machine learning.* Retrieved from https://api.semanticscholar.org/CorpusID:229680169

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

Kullback, S., & Leibler, R. A. (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics*, *22*(1), 79 – 86. Retrieved from https://doi.org/10.1214/aoms/1177729694 doi: 10.1214/aoms/1177729694

Le Cun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Handwritten digit recognition with a back-propagation network. In *Proceedings of the 3rd international conference on neural information processing systems* (p. 396–404). Cambridge, MA, USA: MIT Press.

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324. doi: 10.1109/5.726791

Li, Y., Ma, T., & Zhang, H. (2018, 06–09 Jul). Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations. In S. Bubeck,

V. Perchet, & P. Rigollet (Eds.), *Proceedings of the 31st conference on learning theory* (Vol. 75, pp. 2–47). PMLR. Retrieved from https://proceedings.mlr.press/v75/li18a.html

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

Mediratta, I., You, Q., Jiang, M., & Raileanu, R. (2024). The generalization gap in offline reinforcement learning. In *The twelfth international conference on learning representations.* Retrieved from https://openreview.net/forum?id=3w6xuXDOdY

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. Retrieved from https://doi.org/10.1038/nature14236 doi: 10.1038/nature14236

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

Ng, A. Y., Harada, D., & Russell, S. J. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the sixteenth international conference on machine learning* (p. 278–287). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

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

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (Second ed.). The MIT Press. Retrieved from http://incompleteideas.net/book/the-book-2nd.html

Todorov, A., Cardenas-Cartagena, J., Cunha, R. F., Zullich, M., & Sabatelli, M. (2025). Sparsity-driven plasticity in multi-task reinforcement learning. *Transactions on Machine Learning Research*. Retrieved from https://openreview.net/forum?id=9L4Z23EfE9

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

Zhang, C., Vinyals, O., Munos, R., & Bengio, S. (2018). *A study on overfitting in deep reinforcement learning.* Retrieved from https://arxiv.org/abs/1804.06893

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 time steps to reach the goal. The time penalty and reward shaping are not applied in the time step 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 Training Hyperparameters

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

B.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}.

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 Figures C.1–C.3. In some cases we see some initial learning, but the model seems to forget and revert to minimum return. Plotting our DQN results by capacity, we see no significant learning happening across all tested capacities, as shown in Figure C.4.

Figure C.1: 512 neurons per hidden layer, trained on 100 seeds for 50,000 episodes.

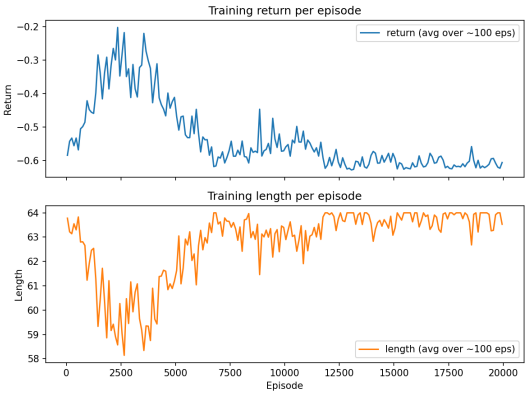


Figure C.2: 512 neurons per hidden layer, trained on 50 seeds for 20,000 episodes.

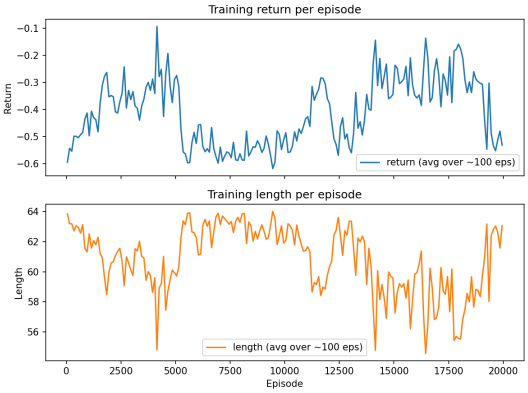
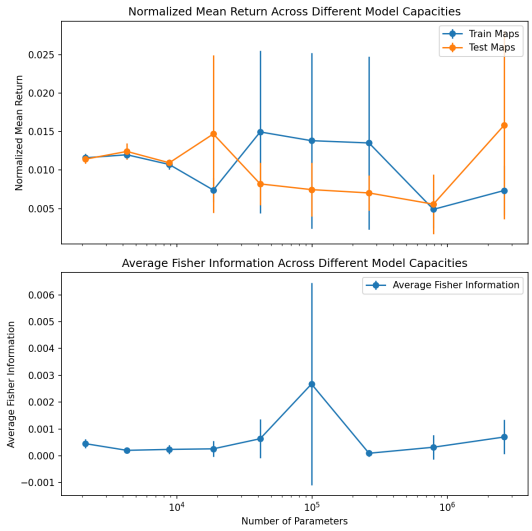
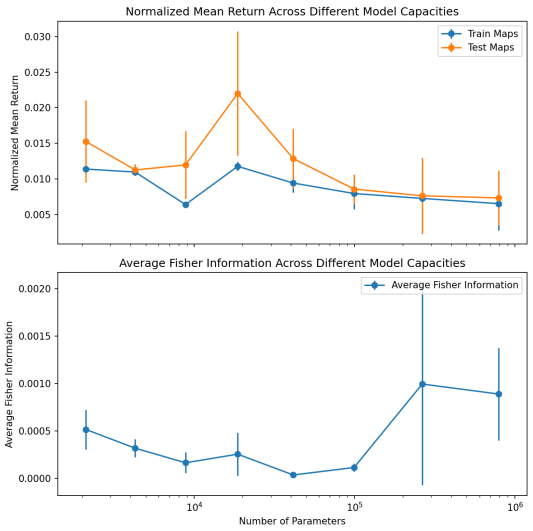


Figure C.3: 1024 neurons per hidden layer, trained on 50 seeds for 20,000 episodes.



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

D Additional TRPO Plots

Further plots showing training and test return as well as average Fisher information as a function of model capacity for TRPO models can be found in Figures D.1–D.7.

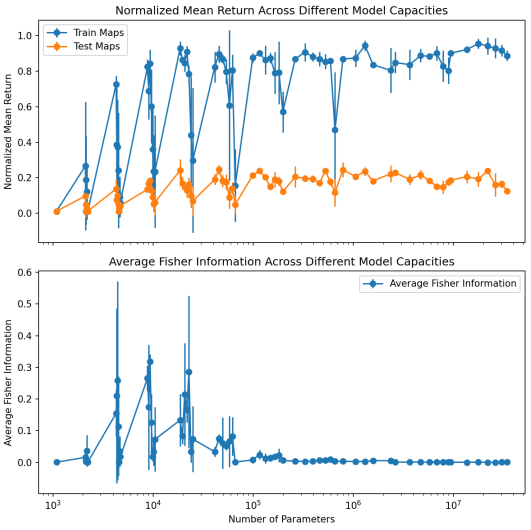


Figure D.1: TRPO training and test return, and average Fisher information, as a function of model capacity, using 50 train maps and 20,000 training episodes per model.

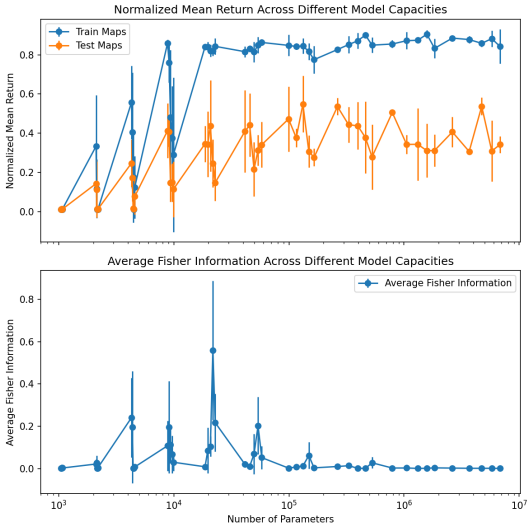


Figure D.2: TRPO training and test return, and average Fisher information, as a function of model capacity, using 100 train maps and 50,000 training episodes per model.

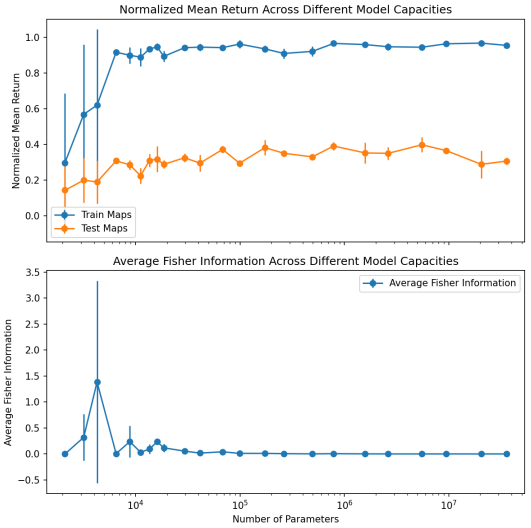


Figure D.4: TRPO training and test return, and average Fisher information, as a function of model capacity, using 100 train maps and 1,000,000 training episodes per model.

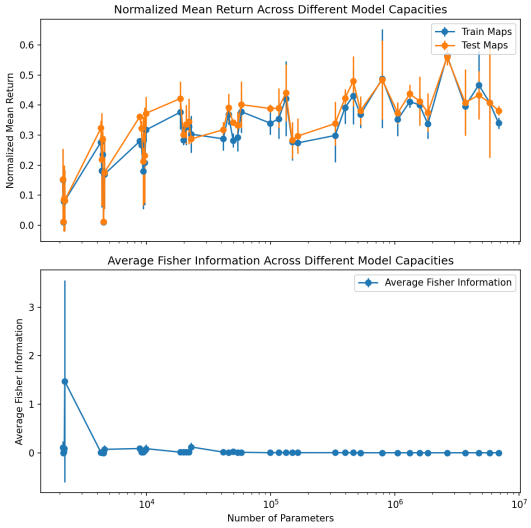


Figure D.3: TRPO training and test return, and average Fisher information, as a function of model capacity, using 1,000 train maps and 100,000 training episodes per model.

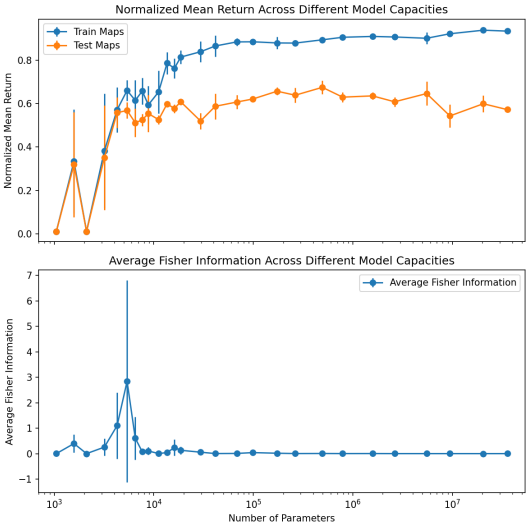


Figure D.5: TRPO training and test return, and average Fisher information, as a function of model capacity, using 750 train maps and 1,000,000 training episodes per model.

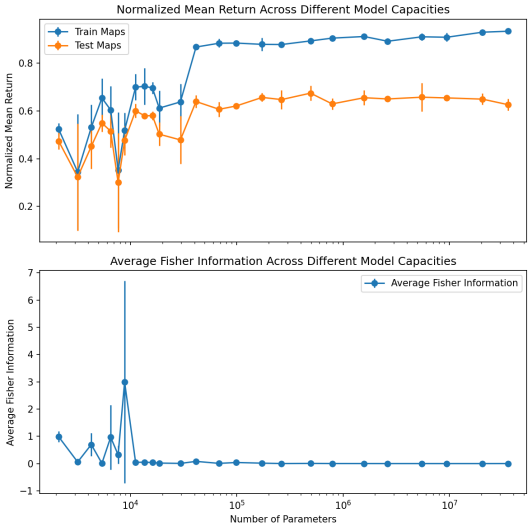


Figure D.6: TRPO training and test return, and average Fisher information, as a function of model capacity, using 1,000 train maps and 1,000,000 training episodes per model.

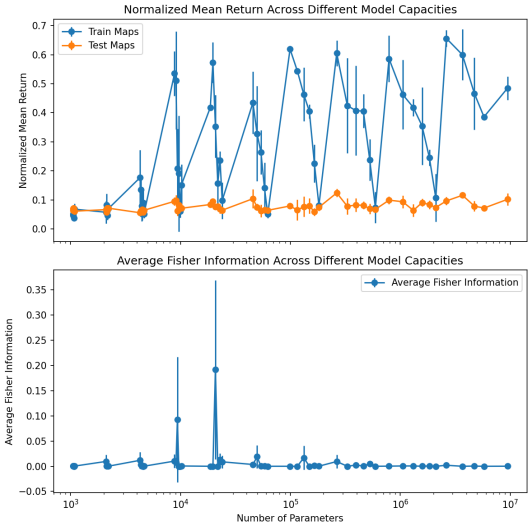
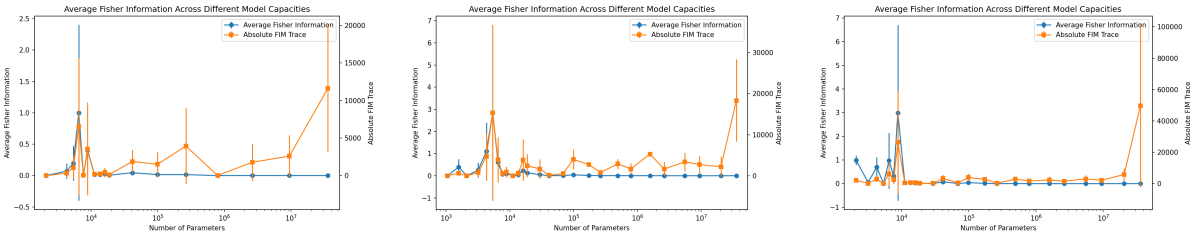


Figure D.7: TRPO training and test return, and average Fisher information, as a function of model capacity, using 100 train maps and 50,000 training episodes per model and randomized start and goal positions.

E Plotting the Absolute FIM Trace

To investigate why the average Fisher information trace exhibits a spike during training, we plot the absolute value of the trace across training steps for TRPO models of varying capacity trained for 1,000,000 episodes on 500, 750 and 1,000 train maps, shown in E.1. We observe the observed spike remains in the absolute FIM trace.



(a) 500 train maps. **(b) 750 train maps.** **(c) 1,000 train maps.**

Figure E.1

F Long 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 in Figures F.1–F.10.

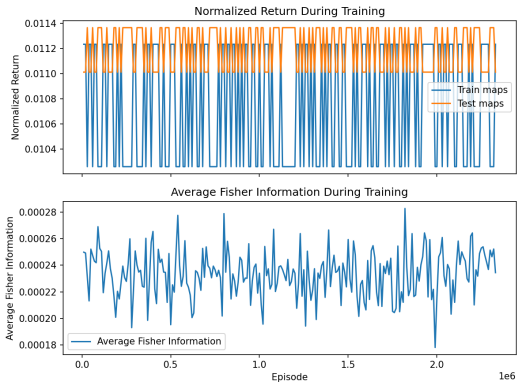


Figure F.1: Model width 4, train set size 50. The model is too small and fails to learn.

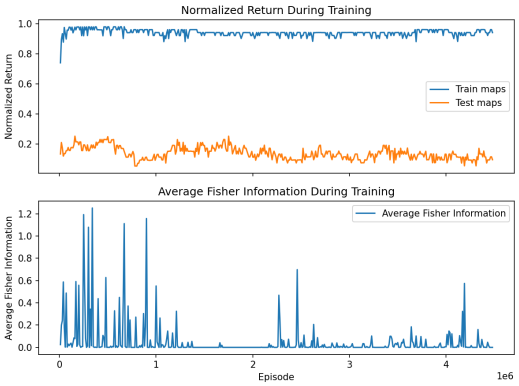


Figure F.2: Model width 16, train set size 50.

Figure F.3: Model width 64, train set size 50.

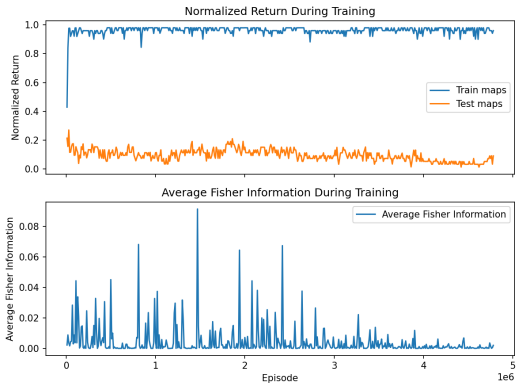


Figure F.4: Model width 256, train set size 50.

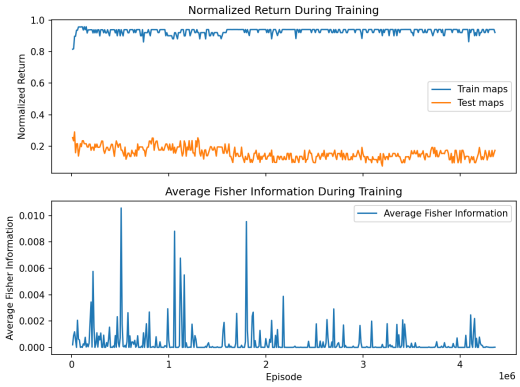


Figure F.5: Model width 1024, train set size 50.

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

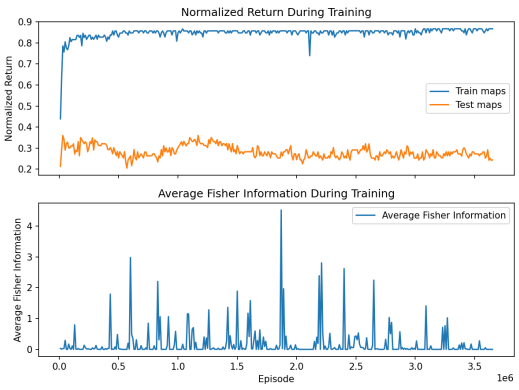


Figure F.7: Model width 16, train set size 100.

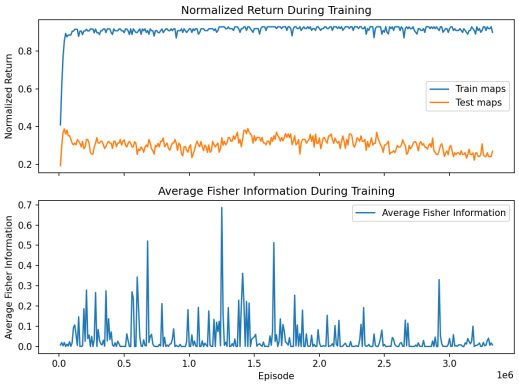


Figure F.8: Model width 64, train set size 100.

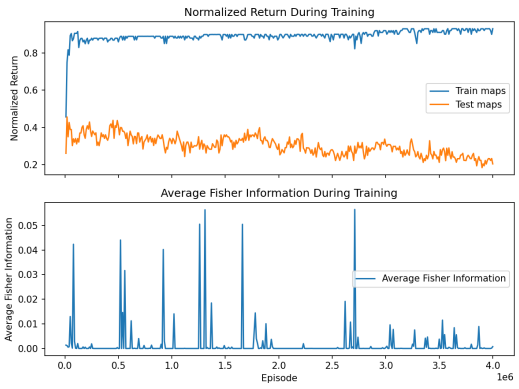


Figure F.9: Model width 256, train set size 100.

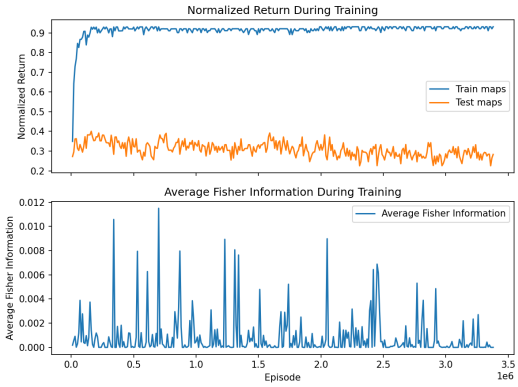


Figure F.10: Model width 1024, train set size 100.