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

On The Existence of Double Descent in Reinforcement Learning

Bachelor's Project Thesis

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

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

1 Introduction

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

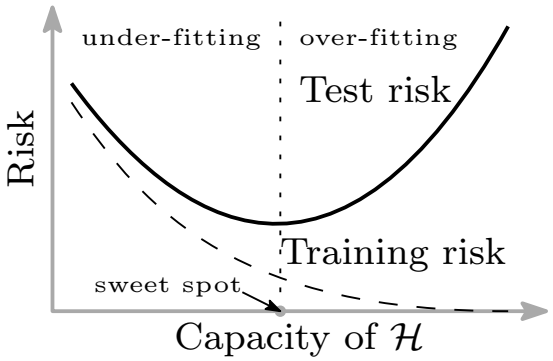


Figure 1.1: Traditional U-shaped risk curve. The training and test risk initially decrease together, until the model begins overfitting. Figure taken from Belkin et al. (2019).

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

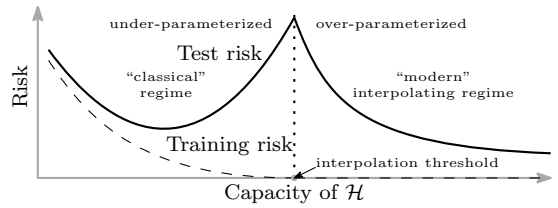


Figure 1.2: Double Descent risk curve. After the initial overfitting phase, increasing model capacity, training time, or dataset size further leads to a second descent in test risk. Figure taken from Belkin et al. (2019).

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

2 Methods

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

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

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

2.1 Environment

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

The 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 action space is discrete and allows for moving in the four cardinal directions (up, down, left, right). Actions are deterministic—the agent moves in the selected direction, unless this tile is inaccessible either through being a wall or being out of bounds. In this case the agent does not move at all. An episode terminates when the agent reaches the goal or is truncated after 64 timesteps, and there is no discounting (i.e., $\gamma = 1$).

The agent receives a reward of $+1$ upon reaching the goal. In all other timesteps, a potential-based reward is applied. The potential function $\phi(s)$ is defined as one hundredth of the negative Euclidean distance from the agent's current position to the goal position. The reward at each timestep is then given by $r_t = \phi(s_t) - \phi(s_{t-1})$, encouraging the agent

to move closer to the goal. Finally, the agent also receives a small time-penalty of -0.01 at each timestep to disincentivize standing still or taking suboptimal paths. Thus, the reward function is defined as follows:

$$r_t = \begin{cases} 1, & \text{if goal reached} \\ \phi(s_t) - \phi(s_{t-1}) - 0.01, & \text{otherwise} \end{cases} \tag{2.1}$$

with the potential function defined as:

$$\phi(s) = -\frac{1}{100} \cdot d_{\text{euclidean}}(\text{agent\_pos}, \text{goal\_pos}) \tag{2.2}$$

For example, if the agent moves a distance of exactly 1 unit closer to the goal in a timestep, it receives a reward of $0.01 - 0.01 = 0$. In all other cases the reward is negative, except when the goal is reached. The maximum reward an agent can receive on a map depends on the map's configuration, as some can be solved in fewer steps than others.

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

2.2 Metrics

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

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

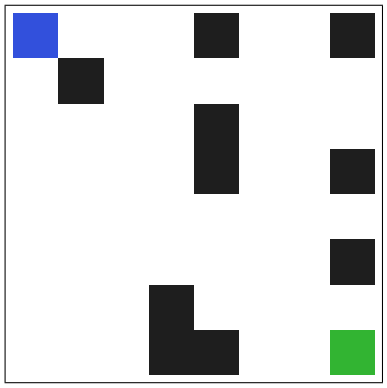
$$F = \mathbb{E}_{(s,a)\sim\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a \mid s) \nabla_\theta \log \pi_\theta(a \mid s)^T \right] \tag{2.3}$$

where (s, a) are state-action pairs sampled from behavior under the policy π_θ. The trace of the FIM, $\text{Tr}(F)$, provides a measure of the sensitivity of the model's parameters to changes in the data distribution. A high trace value indicates that the model is highly sensitive to the training data, which can be a sign of overfitting or memorization. Conversely, a lower trace value suggests that the model is more robust and generalizes better to unseen data. By tracking the trace of the FIM on both training and test environments, we can gain insights into the model's generalization capabilities and its tendency to overfit as training progresses. We would therefore expect the FIM trace to be roughly correlated with the size of the generalization gap, as higher FIM trace values should indicate more overfitting and thus a larger generalization gap.

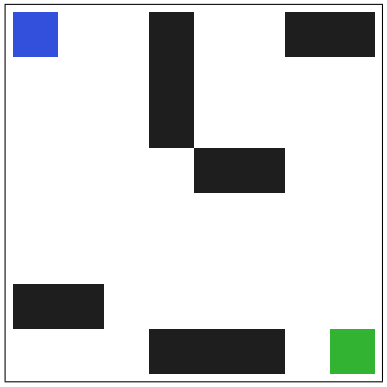
References

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

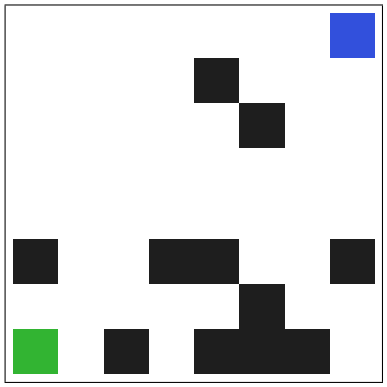
Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., & Sutskever, I. (2019). *Deep double descent: Where bigger models and more data hurt.* Retrieved from `https://arxiv.org/abs/1912.02292`



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



(b) Another example environment in standard configuration.



(c) Example environment in randomized configuration.

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