An empirical study on the learnability of functions by NNs

Luis Fernando Palacios Flores *[1]

[1] *University of Trieste*

This report outlines my approach to addressing the challenge's objective, as detailed in the assignment: to empirically investigate the learnability of specific functions by deep neural networks (DNNs) under different conditions. The importance of this challenge lies in understanding the interplay of model architecture and expressivity, training dynamics, and target function structure, as these highlight key concepts of implicit biases in deep learning. As stated in the challenge assignment, the report is structured in two sections. The first section tackles the Teacher/Student setup, examining how well student networks (with varying capacities) learn to mimic a teacher network—fixed with a specific architecture and weight initialization—and how their learning dynamics change under conditions such as different learning rates. The second section studies how the intrinsic structure of a target function affects the learnability efficiency of a deep residual network (DRN) in terms of generalization error. My complete solution is available on my GitHub page. Finally, I acknowledge using ChatGPT to debug and enhance my code and to better understand some of the functions employed in the challenge, as well as several related concepts.

1 The effect of under- and over-parameterization in the Teacher-Student setup

According to the challenge assignment, the teacher model $\mathcal{T}$ was a fully-connected feedforward neural network that maps a 100-dimensional input to a single output scalar. It had 3 hidden layers of sizes 75, 50, and 10. The ReLU activation function was used after all neurons, except for the output. The student models $\mathcal{S}$ were also fully-connected feedforward neural networks that mapped a 100-dimensional input to a single output scalar. These student models had different architectures: S_u had one hidden layer of size 10, S_e had the same architecture as the teacher, and S_o had 4 hidden layers of sizes 200, 200, 200, and 100.

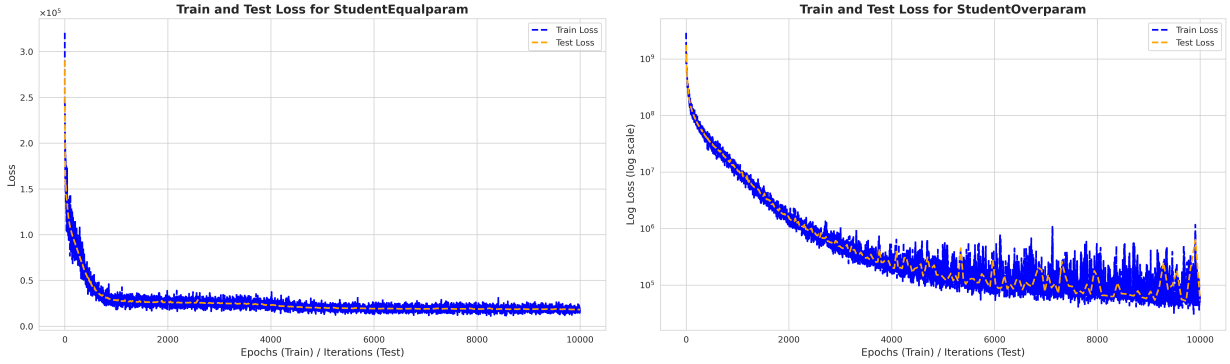

Figure 1: Training and "Test" (Validation) losses for the student models with standard normal initialization and learning rate $\times 10^{-3}$.

The goal of this task was to examine how well student networks can learn to mimic a teacher network and to identify the influence of a model's expressivity under a common architecture and learning process. The learning dynamics of the student networks were studied under various conditions, such as different learning rates. The student models were trained over 10000 iterations with learning rates ranging from

*luisfernando.palaciosflores@studenti.units.it

10^0 to 10^{-3}. They were trained using Mean Squared Error (MSE) loss with the Adam optimizer. Both training and validation data were generated by querying the teacher model with a fixed batch size of 128. The inputs $\boldsymbol{x}_i \in \mathbb{R}^{100}$ were sampled from the Uniform distribution on $[0, 2]^{100}$, and the outputs were $y_i = \mathcal{T}(\boldsymbol{x}_i)$. The validation set contained 6×10^4 data points. I logged training losses at every iteration and validation losses at selected intervals. I employed a logarithmically spaced scheduler to evaluate validation loss, covering up to 10% of the total iterations, since the learning dynamics change rapidly early in training. To ensure reproducibility for both sections of the challenge, I fixed the random seed of the libraries used in the experiments.

The teacher model's parameters were initialized with a standard normal distribution. Ideally, the student models' parameters should converge to the teacher's parameter distribution, as this process should primarily depend on learning dynamics and the expressivity of the student models. To test this, the students were initialized with (1) a standard normal distribution and (2) PyTorch's default initialization (essentially Kaiming for all parameters). The former is more likely to converge to the teacher's parameter distribution.

In my experiments, the loss function generally decreased more smoothly when using the standard normal initialization, compared to the default PyTorch initialization (see Figures 1 and 7). However, in both cases the loss could begin at high values (particularly with standard normal, ranging from 10^5 to 10^{12}), then quickly drop to around 10^5 or 10^4, and finally decrease more slowly. Indeed, final validation losses ended in the order of 10^4 for both initializations. Moreover, training and validation loss curves seemed to decrease at a similar rate, suggesting the student models mimicked the teacher regardless of initialization.

I expect the losses might have decreased further given more iterations, but my computational resources were limited. I also tried utilizing a learning rate scheduler, the ReduceLROnPlateau scheduler to be specific, but tuning its hyperparameters wasn't a trivial task; the learning rates often became too small, preventing the models from adequately learning the teacher behavior not its parameters distribution.

Figures 2 and 3 show the final parameter distributions of the student models. At lower learning rates, students more closely approximated the teacher's parameter distribution, while higher learning rates led to distributions with heavier tails, particularly under the default initialization (Figure 3), likely due to instability in convergence. Notably, with the smallest learning rate, the default initialization's final parameter distribution begins to resemble a bell-shaped curve, suggesting that additional training steps could further align it with a normal distribution.

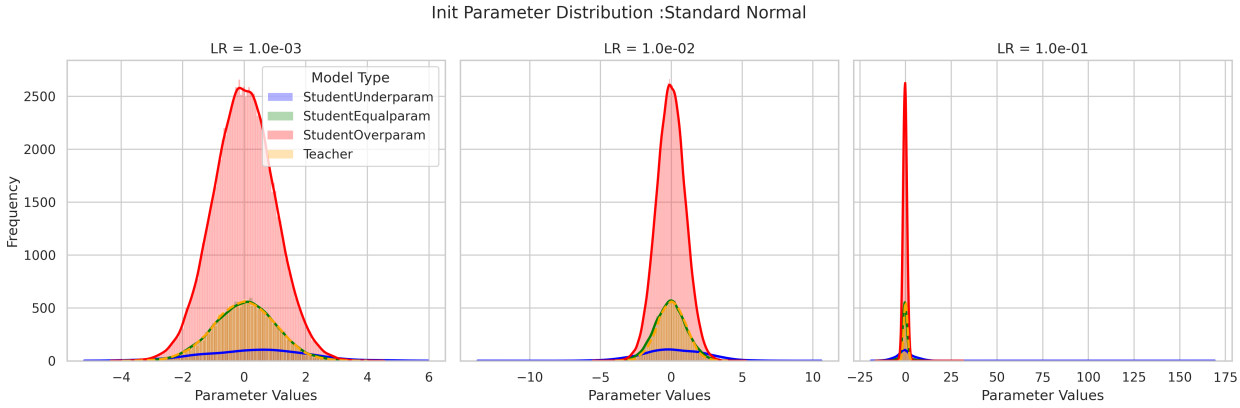

Figure 2: Histogram of the final parameters distribution of the student models with standard normal initialization.

In the case of the standard normal initialization at learning rate $\times 10^{-3}$, the student model with the same number of parameters as the teacher (S_e) learned the teacher's parameter distribution better than the other students (Figure 2). Its final evaluation loss was also lower ($\sim 1.8 \times 10^4$) than that of S_u ($\sim 2.5 \times 10^4$) or S_o ($\sim 5.4 \times 10^4$). This ordering of final evaluation loss remained consistent across other learning rates, albeit with higher values. For the default initialization, the ordering was less consistent, except at a learning rate

of 10^{-1}, where a similar trend was observed.

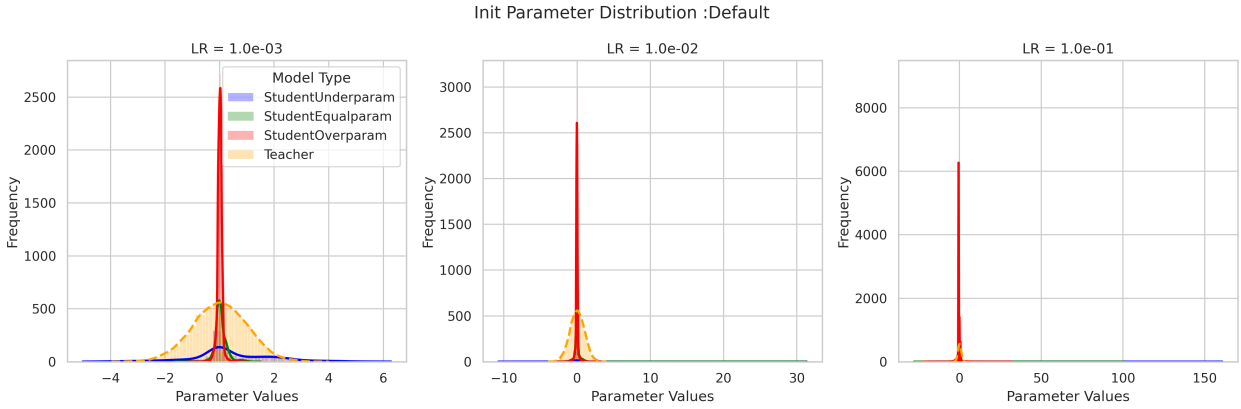

Figure 3: Histogram of the final parameters distribution of the student models with default initialization.

Learning Rate	Model	Mean	Std	Statistic	p-value
0.01	StudentUnderparam	0.060	7.26	0.281	2.35×10^{-66}
	StudentEqualparam	−0.402	1.89	0.175	8.13×10^{-160}
	StudentOverparam	−0.298	1.13	0.128	4.76×10^{-154}
0.01	StudentUnderparam	0.168	2.27	0.222	4.99×10^{-41}
	StudentEqualparam	−0.058	1.10	0.048	1.79×10^{-12}
	StudentOverparam	−0.033	1.00	0.023	2.07×10^{-5}
0.001	StudentUnderparam	0.322	1.67	0.189	8.51×10^{-30}
	StudentEqualparam	0.027	1.01	0.012	3.97×10^{-1}
	StudentOverparam	−0.007	0.99	0.013	5.0×10^{-2}

Table 1: Summary statistics for student models with standard normal initialization across different learning rates. The mean, standard deviation, KS statistic, and p-value are shown.

Contrary to the expectation that learning rates minimizing the time to reach a quasi-stationary state would improve generalization error and help student models learn the teacher's parameter distribution, smaller learning rates did not yield lower final evaluation losses than larger ones and required more iterations to stabilize due to slower learning dynamics. However, they resulted in a better approximation of the teacher's parameter distribution. Additionally, in my experiments, higher model expressivity increased the time needed to reach a quasi-stationary state, likely because more complex models required additional iterations to converge. Moreover, the student model with more parameters than the teacher exhibited greater loss fluctuations.

A Kolmogorov-Smirnov (KS) test (Table 1) further supported the previous observations, showing that students tended to align better with the teacher's distribution at smaller learning rates. The null hypothesis is that the two samples are drawn from the same distribution. Only S_e at learning rate $\times 10^{-3}$ achieved a significantly higher p-value, indicating stronger alignment with the teacher's parameters. For models with default initialization, the null hypothesis was not rejected in all cases (Table 3).

A more detailed layer-wise analysis of the student models' parameter distributions is presented in Table 2. Only layers directly comparable to those in the teacher model were analyzed. Hypothesis tests suggest that the student models' parameter distributions resemble the teacher's in the final layers, though this observation is limited by the small sample size (10 parameters in the final layer), the number of iterations, and the use of

a single seed. A similar but weaker trend was observed for student models with default initialization (Table 4). This suggests that a global analysis of parameter distributions is more informative than a layer-wise analysis for assessing whether student models successfully learned the teacher's parameter distribution.

Learning Rate	Model	Layer	Mean	Std	Statistic	p-value
0.1	StudentUnderparam	0	−0.086	4.98	0.282	4.77×10^{-63}
		-1	13.475	49.26	0.545	7.47×10^{-2}
	StudentEqualparam	0	−0.386	2.12	0.197	2.11×10^{-129}
		1	−0.443	1.42	0.146	9.46×10^{-36}
		2	−0.342	1.38	0.159	4.96×10^{-6}
	StudentOverparam	-1	−0.088	1.47	0.182	9.971×10^{-1}
		0	−0.446	1.07	0.189	5.92×10^{-173}
		-1	−0.105	0.75	0.328	1.869×10^{-1}
0.01	StudentUnderparam	0	0.201	2.16	0.222	6.93×10^{-39}
		-1	−2.837	6.41	0.636	2.07×10^{-2}
	StudentEqualparam	0	−0.033	1.12	0.048	3.77×10^{-8}
		1	−0.112	1.06	0.055	1.82×10^{-5}
		2	−0.036	1.07	0.071	1.575×10^{-1}
	StudentOverparam	-1	0.237	0.62	0.455	2.115×10^{-1}
		0	−0.049	1.00	0.032	2.35×10^{-5}
		-1	−0.076	0.87	0.278	3.529×10^{-1}
0.001	StudentUnderparam	0	0.335	1.64	0.191	8.33×10^{-29}
		-1	−0.865	3.62	0.636	2.07×10^{-2}
	StudentEqualparam	0	0.061	1.02	0.027	7.8×10^{-3}
		1	−0.051	0.99	0.035	1.90×10^{-2}
		2	0.085	1.05	0.078	8.68×10^{-2}
	StudentOverparam	-1	0.327	0.89	0.364	4.792×10^{-1}
		0	−0.040	1.00	0.029	1.52×10^{-4}
		-1	−0.099	0.88	0.239	5.392×10^{-1}

Table 2: Layer-wise summary statistics for Student Models initialized with a standard normal distribution across different learning rates. The table presents the mean, standard deviation (std), KS statistic, and p-value for each layer of the model. The model column is collapsed to group layers under each student model. The layer index follows a convention where 0 represents the input layer, positive indices refer to hidden layers, and −1 represents the output layer.

In summary, under my computational constraints, student models with equal or fewer parameters than the teacher learned faster and more stably. Their learning dynamics were primarily influenced by the learning rate and network expressivity. While smaller learning rates led to a closer match with the teacher's parameter distribution, they did not necessarily yield the lowest validation loss within a limited number of iterations. Additionally, student models with standard normal initialization generally approximated the teacher's distribution more effectively but often trained more slowly and exhibited greater loss fluctuations. My results suggest that, with additional training steps, even models using default initialization could have improved their alignment with the teacher's parameters, provided they avoided convergence to poor local minima.

2 Function learning and hierarchical structure

In this section, the goal was to investigate the learning efficiency of a DRN—measured by its generalization error—when learning a hierarchical polynomial versus a non-hierarchical polynomial. The hierarchical polynomial is the complete Bell polynomial B_6:

$$B_6(x_1 \dots x_6) = x_1^6 + 15x_1^4x_2 + 20x_1^3x_3 + 45x_1^2x_2^2 + 15x_1^2x_4 + 15x_2^3 + 10x_3^2 + 60x_1x_2x_3 + 15x_2x_4 + 6x_1x_5 + x_6 \quad (1)$$

The non-hierarchical polynomial is a scrambled version of the hierarchical polynomial. I generated all possible permutations of six variables and randomly assigned unique permutations to the variables of the hierarchical polynomial, ensuring the final polynomial still depended non-trivially on all inputs, without any two monomials reducible by commutativity. Fixing the same random seed as in the previous section, I obtained:

$$\tilde{B}_6(x_1 \dots x_6) = x_3^6 + 15x_3^4x_4 + 20x_1^3x_3 + 45x_2^2x_5^2 + 15x_1^2x_3 + 15x_5^3 + 10x_1^2 + 60x_1x_4x_5 + 15x_4x_6 + 6x_4x_5 + x_3 \quad (2)$$

I created 10^5 training data points and 6×10^4 validation points by querying these two polynomials. The inputs $\boldsymbol{x}_i \in \mathbb{R}^6$ were sampled from the Uniform distribution on $[0, 2]^6$. The DRN used here was fully connected, with 6 input neurons, hidden layers of sizes 100, 100, and 50, and a single output neuron. It employs the same activation function, optimizer, and loss as in the previous section. The network was trained for 30 epochs with batch size 20 and various learning rates (10^0 to 10^{-3}). I logged both training and evaluation losses every epoch.

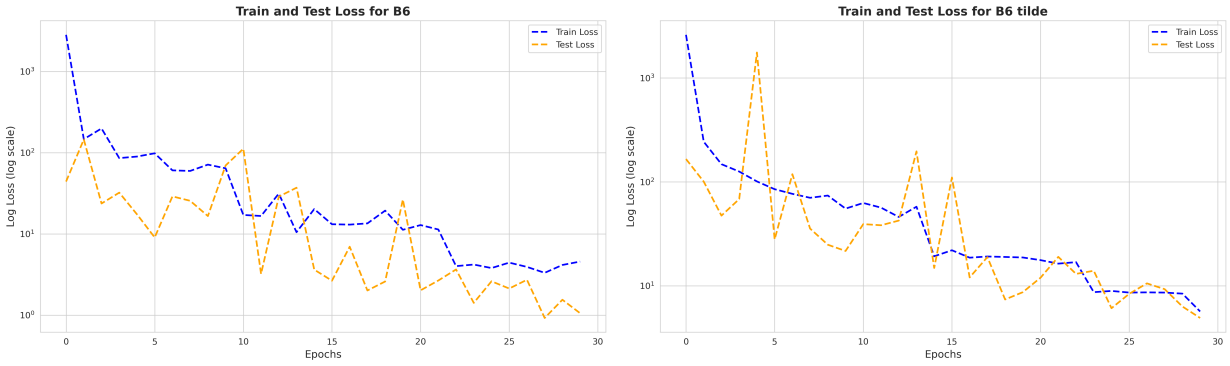

Figure 4: Training and "Test" (Validation) losses for the hierarchical polynomial B_6 and the non-hierarchical polynomial $\tilde{B}_6$ with learning rate $\times 10^{-3}$.

After training, I evaluated the models over 50 trials, each time randomly sampling from $[0, 2]^6$ and then sweeping one variable at a time over 200 points in $[0, 2]$. This yielded a batch of 200 samples per variable per trial. I then computed the mean and standard deviation of the outputs for each input variable.

Figure 4 shows that at learning rates of 10^{-2} and 10^{-3}, both B_6 and $\tilde{B}_6$ exhibited similar training and validation loss trends, with losses of the same order of magnitude. The DRN learned both polynomials with almost no difference in performance, as further confirmed by the component-wise input evaluations in Figures 5 and 6. The only notable difference was that the non-hierarchical polynomial had a slightly higher final validation loss than the hierarchical polynomial (4.91 vs. 1.05). In both cases, training and validation losses followed a generally decreasing trend, though the validation loss exhibited some fluctuations. The loss curves did not indicate that one polynomial was inherently easier to learn than the other. I generated and trained models for 4 additional seeds, and the results were consistent across all seeds. Even with 500 training iterations, losses continued to decrease without signs of overfitting.

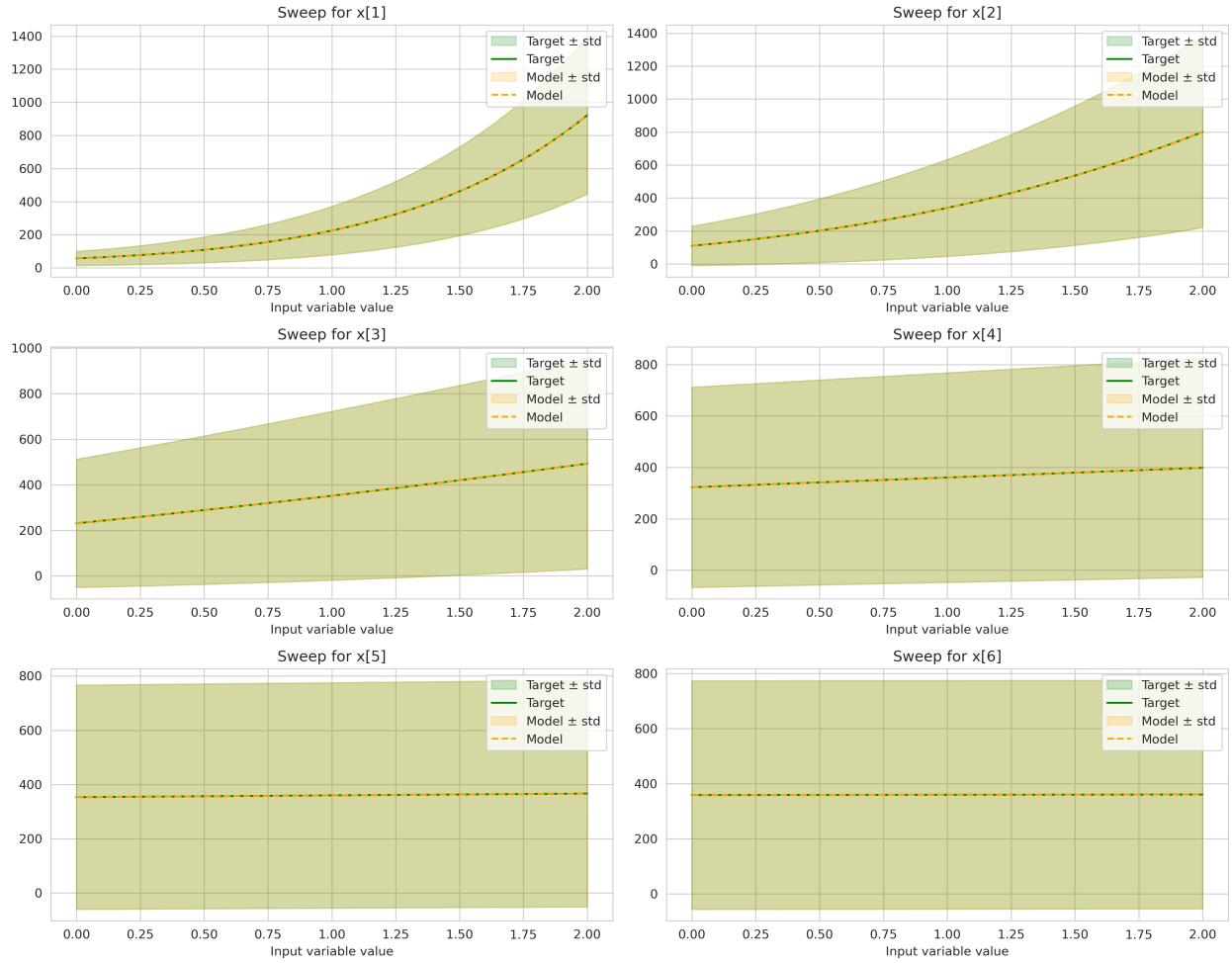

Figure 5: Component-wise mean and standard deviation of the outputs of the polynomials B_6 its corresponding DRN model trained with learning rate $\times 10^{-3}$.

At a learning rate of 10^{-1}, the DRN learned a good approximation of $\tilde{B}_6$ but struggled with B_6 (Figures 8 and 9). While both showed similar trends in training vs. validation loss, the magnitude for B_6 was around 10^5, whereas for $\tilde{B}_6$ it was around 10^3. The final validation losses were $\sim 1.1 \times 10^5$ for B_6 and $\sim 6.5 \times 10^3$ for $\tilde{B}_6$. At the highest rate of 10^0, the DRN failed to learn either polynomial, producing large outputs and very high, non-monotonic losses.

In conclusion, while my experiments were not extensive enough to definitively determine whether the hierarchical polynomial B_6 or its non-hierarchical counterpart is inherently easier or more efficient to learn with the given DRN architecture, they suggest no consistent advantage for either. However, in one specific case, the non-hierarchical polynomial was learned more efficiently than the hierarchical one. More comprehensive tests—with additional seeds and training epochs—could clarify whether non-hierarchical versions of B_6 possess intrinsic structures or properties that make them as learnable as B_6 or even more efficient with the fully connected DRN architecture, or if certain families of derived functions vary in learnability relative to B_6. In contrast to the findings of Abbe et al. [1] for the hierarchical staircase function versus its non-hierarchical counterpart, I did not observe that B_6 was generally easier to learn than $\tilde{B}_6$ with this DRN

architecture. However, since B_6 and $\tilde{B}_6$ differ in complexity from those studied by Abbe et al., it remains possible that, on average, across all valid permutations of B_6, no significant learning-efficiency gap emerges for this particular network. Further investigation is needed to confirm or refute this hypothesis.

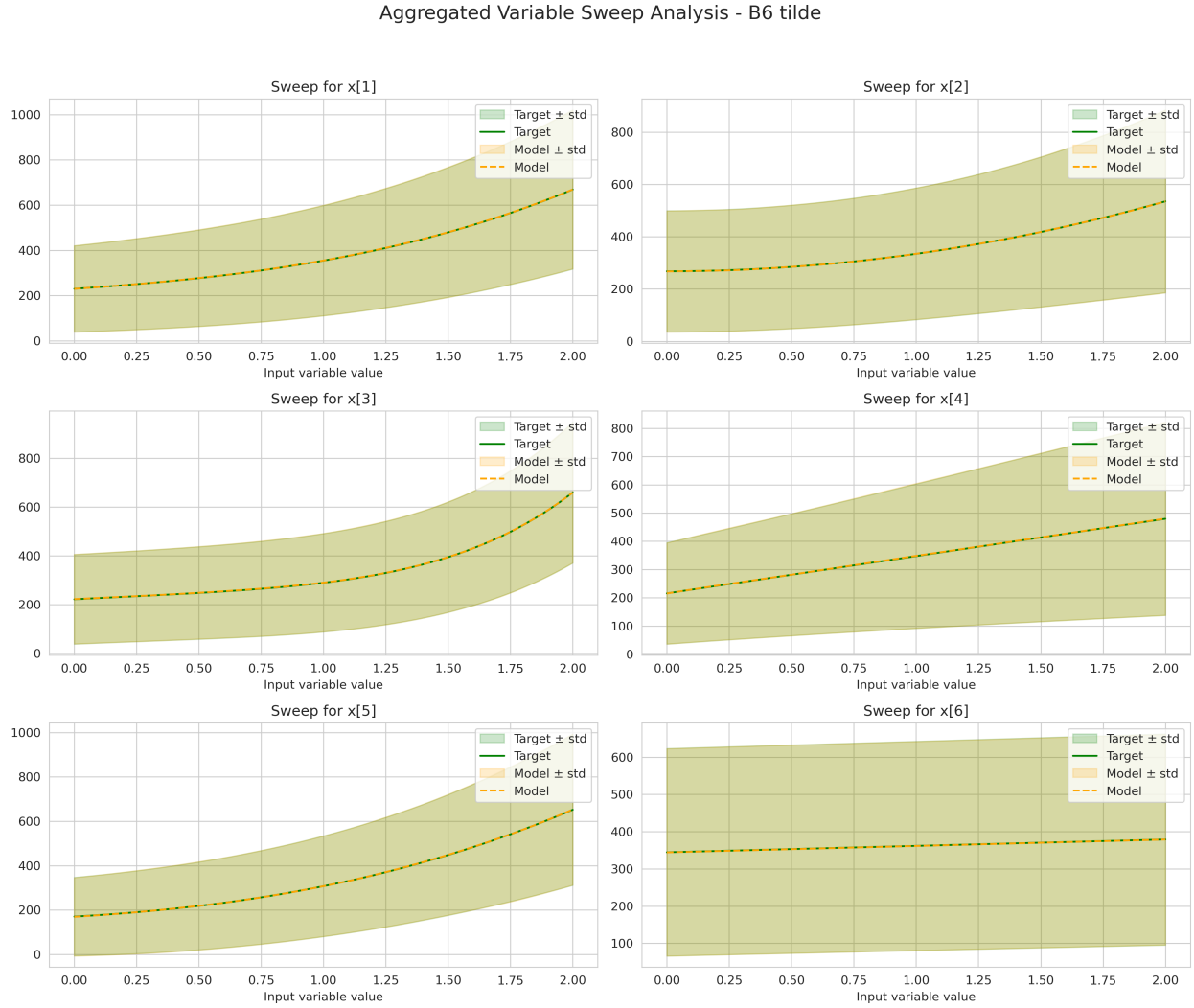

Figure 6: Same as Figure 5 but for the non-hierarchical polynomial $\tilde{B}_6$.

References

[1] E. Abbe, E. Boix-Adserà, M. S. Brennan, G. Bresler, and D. Nagaraj, "The staircase property: How hierarchical structure can guide deep learning," *CoRR*, vol. abs/2108.10573, 2021. arXiv: 2108.10573. [Online]. Available: https://arxiv.org/abs/2108.10573.

A Student models results with default parameters distribution initialization

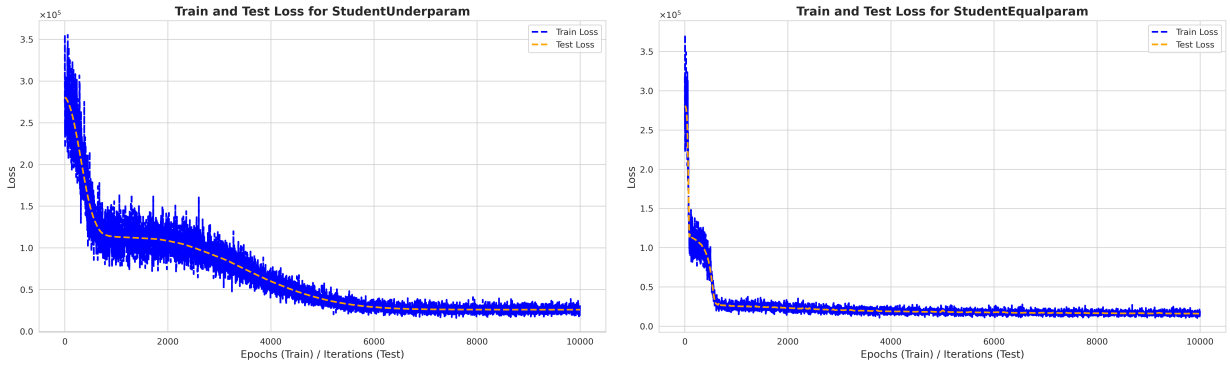

Figure 7: Same as Figure 1 but for the default parameters distribution.

Learning Rate	Model	Mean	Std	Statistic	p-value
0.01	StudentUnderparam	−0.183	6.31	0.514	1.82×10^{-231}
	StudentEqualparam	−0.384	2.30	0.314	0.0
	StudentOverparam	−0.346	0.71	0.354	0.0
0.01	StudentUnderparam	0.017	2.34	0.331	2.31×10^{-92}
	StudentEqualparam	−0.010	0.51	0.261	0.0
	StudentOverparam	−0.032	0.18	0.403	0.0
0.001	StudentUnderparam	0.460	1.54	0.222	5.02×10^{-41}
	StudentEqualparam	0.041	0.23	0.344	0.0
	StudentOverparam	0.007	0.09	0.417	0.0

Table 3: Same as Table 1 but for the default parameters distribution.

Learning Rate	Model	Layer	Mean	Std	Statistic	p-value
0.1	StudentUnderparam	0	−0.328	3.81	0.525	3.17×10^{-227}
		-1	13.156	46.72	0.273	8.326×10^{-1}
	StudentEqualparam	0	−0.412	2.34	0.402	0.0
		1	−0.381	1.61	0.239	9.08×10^{-96}
		2	−0.051	4.61	0.176	2.37×10^{-7}
		-1	2.521	6.18	0.545	7.47×10^{-2}
	StudentOverparam	0	−0.541	0.56	0.629	0.0
		-1	0.272	1.10	0.395	6.50×10^{-2}
0.01	StudentUnderparam	0	0.009	2.09	0.336	9.08×10^{-90}
		-1	0.793	10.31	0.273	8.326×10^{-1}
	StudentEqualparam	0	−0.028	0.44	0.262	4.84×10^{-228}
		1	0.022	0.36	0.337	4.68×10^{-191}
		2	−0.042	0.53	0.294	7.49×10^{-20}
		-1	2.618	8.98	0.455	2.115×10^{-1}
	StudentOverparam	0	−0.042	0.25	0.352	0.0
		-1	−0.013	0.11	0.545	2.7×10^{-3}
0.001	StudentUnderparam	0	0.459	1.50	0.222	4.29×10^{-39}
		-1	0.565	3.42	0.364	4.792×10^{-1}
	StudentEqualparam	0	0.016	0.19	0.338	0.0
		1	0.084	0.29	0.394	3.20×10^{-263}
		2	0.101	0.27	0.349	5.92×10^{-28}
		-1	0.018	0.67	0.455	2.115×10^{-1}
	StudentOverparam	0	−0.008	0.09	0.415	0.0
		-1	0.001	0.15	0.545	2.7×10^{-3}

Table 4: Same as Table 2 but for the default parameters distribution.

B Component-wise evaluation of the hierarchical and non-hierarchical polynomials with learning rate $\times 10^{-1}$

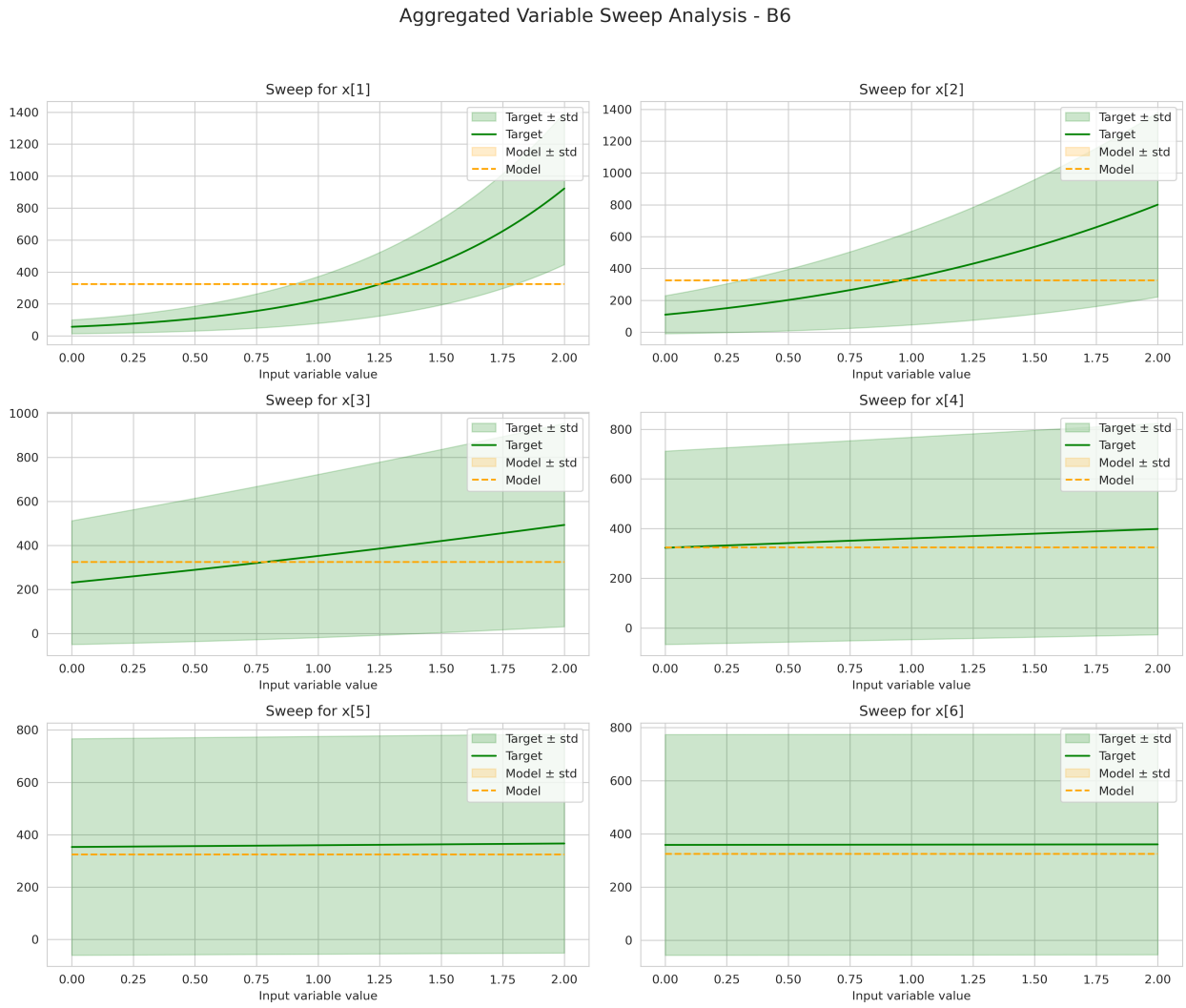

Figure 8: Same as Figure 5 but with learning rate $\times 10^{-1}$.

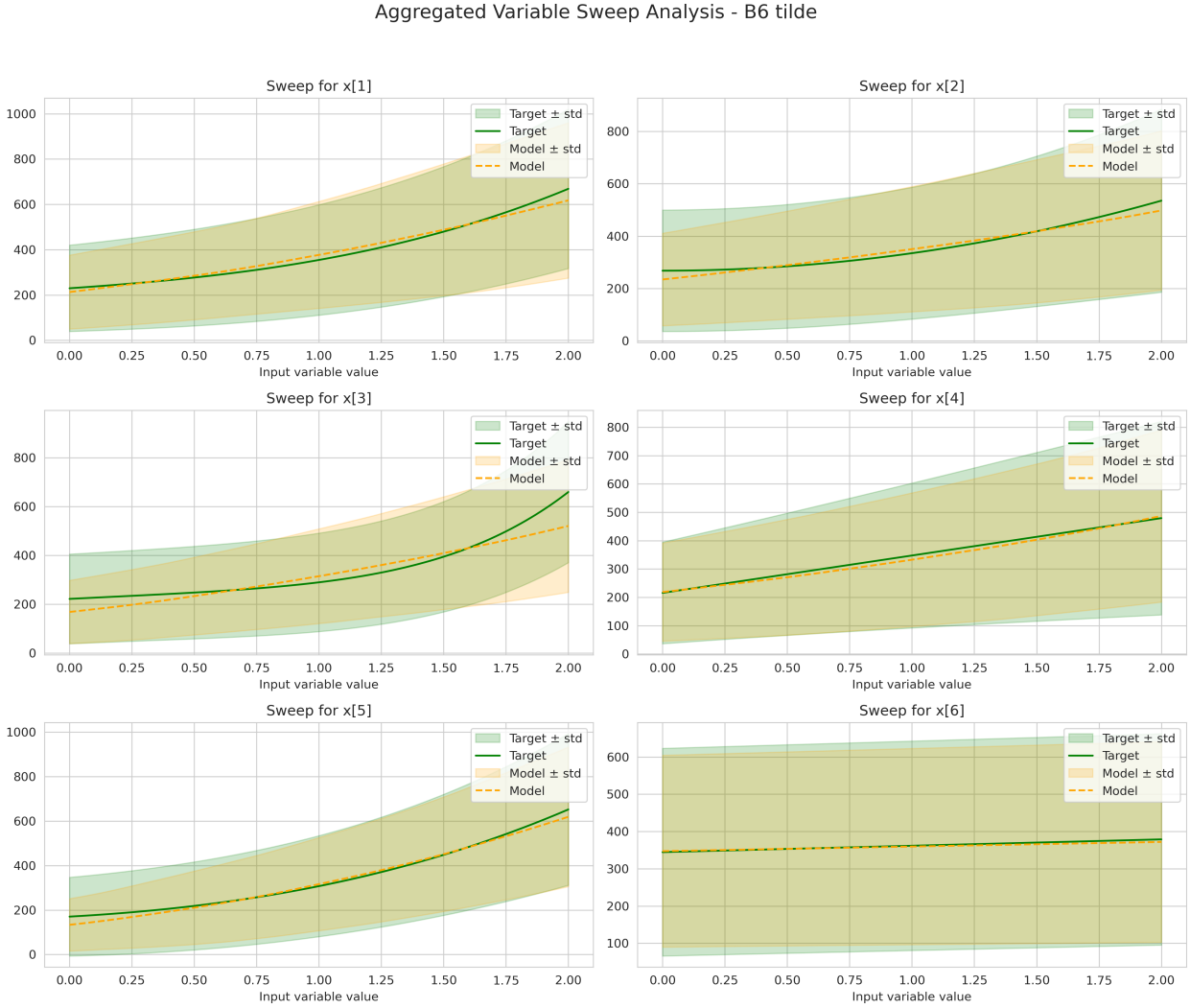

Figure 9: Same as Figure 6 but with learning rate $\times 10^{-1}$.