Advanced Deep Learning and Kernel Methods Challenge 2

Gabriele Pintus
gabrielegavino.pintus@studenti.units.it

ABSTRACT

In this notebook we investigates the learnability of functions by neural networks through empirical analysis in two distinct steps. The first explores the effects of under and over parameterization in a teacher-student setup, where student networks learn from input-output pairs generated by a frozen teacher network. The second challenge examines the ability of deep residual networks to learn hierarchical versus non-hierarchical polynomial functions. the sixth-order multivariate complete Bell polynomial B_6 is compared to a scrambled counterpart $\tilde{B}_6$. All the code is written in Python and it can be found in the GitHub repository at the following link: Challenge 2

1 EXERCISE1

This exercise investigates how the parameterization of a student neural network relative to a teacher network affects its ability to learn.

1.1 Setup

We instantiate the teacher model T as a fully connected neural network, mapping a 100-dimensional input to a single output scalar, with three hidden layers of 75, 50, 10 neurons respectively. We then instantiate three student models S_1, S_2, S_3 as fully connected neural networks with the following architectures:

$$S_1 : 100\text{-}10\text{-}1$$
$$S_2 : 100\text{-}75\text{-}50\text{-}10\text{-}1$$
$$S_3 : 100\text{-}200\text{-}200\text{-}200\text{-}100\text{-}1$$

After doing so with proceed by generating the testing data by sampling from the teacher model.

$$D_{test} = \{(x_i, T(x_i))\}_{i=1}^{N}$$
$$x_i \sim \mathcal{U}(0,2)^{100}$$
$$N = 6 \cdot 10^4$$

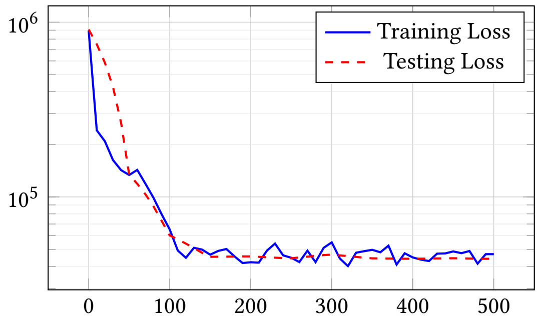
Conversely, the training data is generated lazily by sampling from the teacher during training.

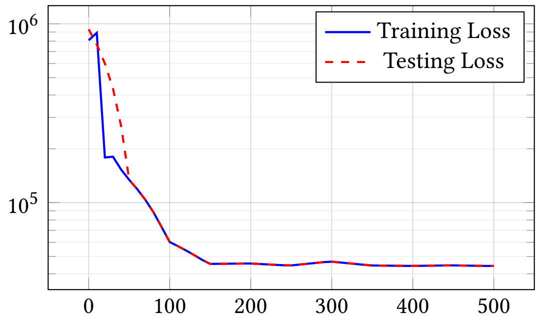
1.2 Training

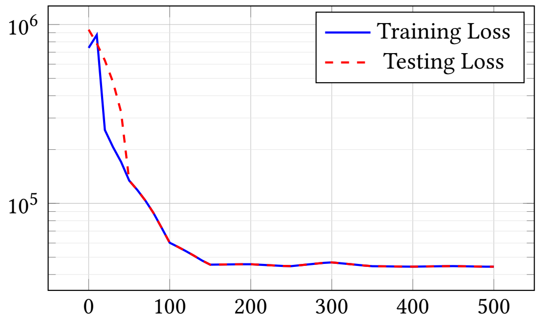
We train each model with the Adam optimizer for 1000 steps with a batch size of 128. We use the mean squared error loss function and the learning rate is tuned by empirical validation as suggested in class. The learning rate for each model is as follows:

$$S_1 : 2\text{e-}1$$
$$S_2 : 3.5\text{e-}2$$
$$S_3 : 9\text{e-}3$$

The loss evolution plots are reported below.







The x-axis has been truncated to 500 steps because the loss evolution of the student models was not evolving much after that point. Hereafter we report the final loss achieved by each model.

Model	Train Loss	Test Loss
S_1	47755	44895
S_2	**42176**	54148
S_3	47430	**42891**

The best on-sample performance is achieved by the S_2 model, which shares the same architecture as the teacher model. However, when evaluating out-of-sample performance, the S_3 model, the over-parameterized one, exhibits the lowest estimated generalization error. Interestingly, the worst-performing model among the three is S_2, the equally parameterized version, while even the under-parameterized model, S_1, outperforms it.

This behavior can be explained by the double descent phenomenon, where model performance initially improves as complexity increases, deteriorates at the interpolation threshold, and then improves again with further complexity. The three models, in this case, perfectly reproduce the three regime described in literature [2] [3] [4] , which we briefly summarize below.

- **Under-parameterized regime**: the model is too simple to capture the complexity of the data.
- **Interpolation threshold**: the model is complex enough to interpolate the data, but not to generalize.
- **Over-parameterized regime**: the model is complex enough to generalize.

1.3 Weights distribution

We now analyze the distribution of the weights of the student models, both network-wide and layer-wise.

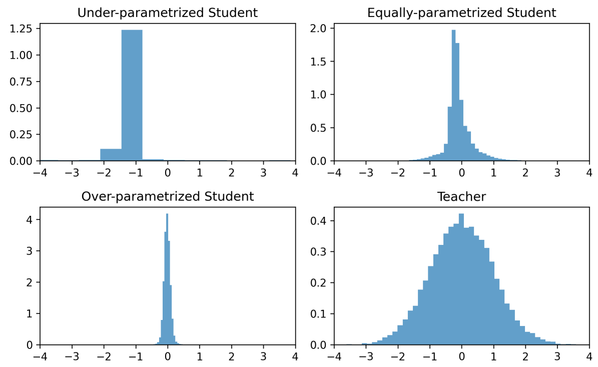


Figure 1: Weights distribution for the student models.

Model	Mean	Std
S_1	-0.98	2.39
S_2	-0.10	0.42
S_3	-0.01	0.11
T	0.00	0.99

Figure 2: Summary statistics for students and teacher models.

As we can see, the model closest in mean to the teacher is the over-parameterized one, which is also the one with the lowest variance. The under-parameterized model, instead, has the highest variance and the lowest mean.
Means and standard deviations for each layer are reported in the table below.

Layer	μ_1	σ_1	μ_2	σ_2	μ_3	σ_3	μ_T	σ_T
w_1	-0.98	2.39	-0.13	0.50	-0.03	0.14	0.00	0.99
b_1	-0.51	2.11	-0.04	0.31	-0.01	0.05	-0.12	0.82
w_2	-1.32	2.34	-0.06	0.27	-0.01	0.10	0.01	0.99
b_2	-7.27	0.00	0.02	0.40	0.02	0.11	0.00	0.93
w_3	-	-	-0.06	0.25	-0.01	0.09	-0.02	1.02
b_3	-	-	0.14	0.46	0.04	0.11	-0.11	0.99
w_4	-	-	-0.02	0.34	-0.01	0.10	-0.12	0.91
b_4	-	-	-0.61	0.00	0.00	0.14	0.78	0.00
w_5	-	-	-	-	0.00	0.14	-	-
b_5	-	-	-	-	-0.11	0.14	-	-

As we can see, the over-parameterized model has the weights more similarly distributed across layers, with the under-parameterized model having the most different distributions. Moreover if we compute the network-wide norm and the layer-wise norms we can see that the over-parameterized model has the lowest norm on average.

Layer	S_1	S_2	S_3	T
w_1	81.67	44.82	20.22	85.37
b_1	6.88	2.71	0.79	7.14
w_2	8.49	16.89	20.37	60.67
b_2	7.27	2.84	1.56	6.55
w_3	-	9.10	18.66	36.10
b_3	-	2.39	1.63	5.00
w_4	-	1.68	14.56	4.61
b_4	-	0.61	0.93	0.78
w_5	-	-	1.39	-
b_5	-	-	0.11	-

2 EXERCISE 2

In this exercise, we will train a deep residual network on examples generated by two specific functions. The first function is the 6-dimensional multivariate complete Bell polynomial B_6, while the second is the scrambled version of the first function $\tilde{B}_6$, as described in the assignment.

$$\begin{aligned} B_6(x_1, x_2, x_3, x_4, x_5, x_6) = x_1^6 + 15x_2x_1^4 + 20x_3x_1^3 + 45x_2^2x_1^2 + 15x_2^3 \\ + 60x_3x_2x_1 + 15x_4x_1^2 + 10x_3^2 + 15x_4x_2 \\ + 6x_5x_1 + x_6 \end{aligned}$$

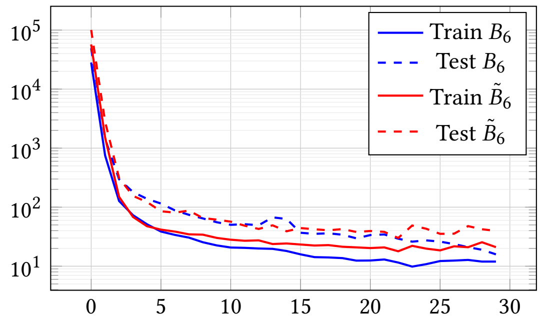
2.1 Setup

We begin by generating both the training and testing data for the two functions by sampling from a uniform distribution and pass it through the functions.

$$\begin{aligned} D_{train} &= \{(x_i, B_6(x_i))\}_{i=1}^N \\ D_{test} &= \{(x_i, \tilde{B}_6(x_i))\}_{i=1}^M \\ x_i &\sim \mathcal{U}(0,1)^6 \\ N &= 10^5 \\ M &= 6 \cdot 10^4 \end{aligned}$$

2.2 Training

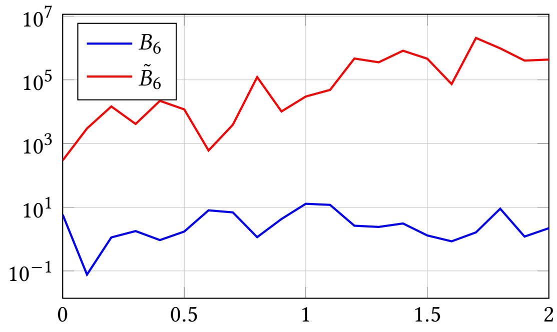
We use a nine-layers fully connected residual neural network and we train it for 30 epochs with a batch size of 20. As usual, we use the Adam optimizer and the mean squared error loss function. The learning rate is set to 10^{-3}. Hereafter we report the loss evolution plots for the training and testing data.



Function	Train Loss	Test Loss
B_6	**11.97**	**15.87**
$\tilde{B}_6$	21.02	39.59

As we can see from the plots and the table, the network is able to learn more easily the first function B_6 than the second $\tilde{B}_6$. This provides empirical evidence in favor of the hypothesis that the neural networks better approximate hierarchical compositional functions.

If we then fix five of the six input variables, make the free variable vary in the range $[0, 2]$ and plot the loss function, we obtain the following plot.



Which shows that the network is able to generalize well to perturbations of the input data when considering the function B_6, but not when considering the function $\tilde{B}_6$.

3 CONCLUSIONS

In this report, we explored the double descent phenomenon observed when training neural networks with varying numbers of parameters. Additionally, we showed that neural networks can more effectively approximate target functions that possess a hierarchical and compositional structure. This highlight the significance of the target function's structure in influencing the network's performance.

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, Cambridge, MA, 2016.
[2] M. Belkin, D. Hsu, S. Ma, and S. Mandal, "Reconciling modern machine-learning practice and the classical bias–variance trade-off", *Proceedings of the National Academy of Sciences*, vol. 116, no. 32, pp. 15849–15854, July 2019. [Online]. Available: arXiv:1812.11118
[3] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever, "Deep Double Descent: Where Bigger Models and More Data Hurt", preprint, arXiv:1912.02292 [cs.LG], 2019. [Online]. Available: arXiv:1912.02292.
[4] M. Lafon and A. Thomas, "Understanding the Double Descent Phenomenon in Deep Learning", preprint, arXiv:2403.10459 [cs.LG], 2024. [Online]. Available: arXiv:2403.10459.
[5] A. Deza, Q. Liao, A. Banburski, and T. Poggio, "Hierarchically Compositional Tasks and Deep Convolutional Networks", preprint, arXiv:2006.13915 [cs.LG], 2021. [Online]. Available: arXiv:2006.13915.
[6] T. Poggio and M. Fraser, "Compositional Sparsity of Learnable Functions", *Bulletin of the American Mathematical Society*, vol. 61, no. 3, pp. 438–456, 2024. [Online]. Available: https://www.ams.org/journals/bull/2024-61-03/S0273-0979-2024-01820-5/.