Challenge 2
Annalisa Paladino

A. The effect of *under*- and *over*- parameterisation in the Teacher/Student setup

Overview

In this first exercise of the challenge[1], we train deep neural networks (*Students*) in a supervised manner using input/output pairs generated by a fixed deep neural network (*Teacher*) with frozen weights. The focus is on understanding how the parameterisation of the *Student* network influences its ability to learn from the *Teacher*, by evaluating performance across under-, exact-, and over-parameterised regimes.

Experimental design

Teacher

The *Teacher* model is a fully connected feedforward neural network with an input dimension of 100 and a single scalar output. There are 3 hidden layers of sizes $74, 50$ and 10, respectively, each followed by a ReLU activation function (except for the output layer). All weights and biases are initialized as i.i.d. samples from the standard normal distribution, and the parameters are frozen upon instantiation to serve as the ground truth throughout the experiment.

Training Data

The test set consists of $60\,000$ input/output pairs obtained by evaluating the *Teacher* model on 100-dimensional input vectors, where each component is independently sampled from the uniform distribution over the interval $[0, 2]^{100}$.

Students

The *Student* model is a fully connected feedforward neural network with an input dimension of 100 and a single scalar output. We consider three variants of the *Student* architecture, each differing in the number and size of the hidden layers, while maintaining the same overall structure as the *Teacher* model:

- **Under-parameterised**: 1 hidden layer of size 10
- **Exact-parameterised**: 3 hidden layers with sizes $74, 50$ and 10, identical to the *Teacher* model
- **Over-parameterised**: 4 hidden layers of sizes $200, 200, 200$ and 100

[1] https://github.com/annalisapaladino/AdvancedDeepLearning-KernelMethods/tree/main/Challenge%202

Number of weights and biases for each model:

	# weights	# biases	# parameters (weights + biases)
Teacher	11760	136	11896
Student_u	1010	11	1021
Student_e	11760	136	11896
Student_o	120100	701	120801

Training

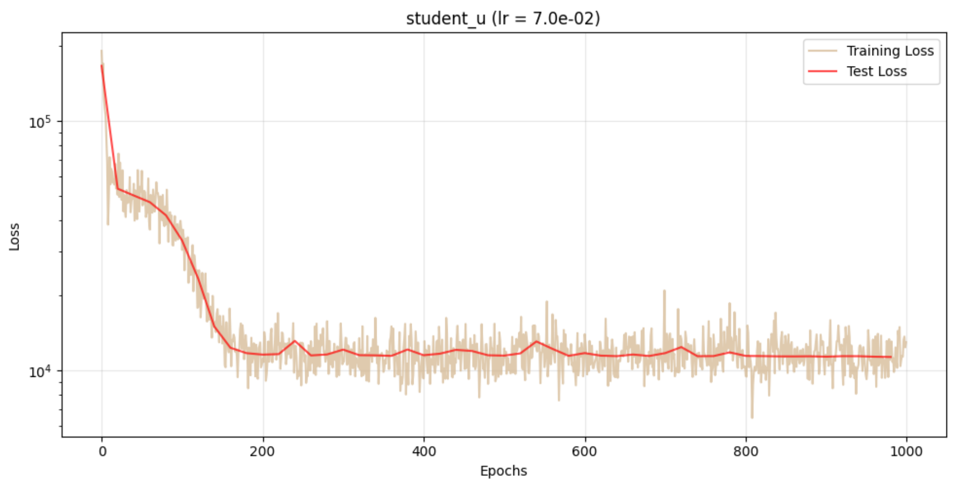
During the **training**, each *Student* model was optimised for 1000 epochs using the Mean Squared Error (MSE) loss. At each iteration, a new batch of 128 input samples was drawn from the same uniform distribution, and their corresponding outputs were computed by querying the *Teacher*. The Adam optimiser was used for gradient-based updates, with the learning rate selected via grid search over the values $\{5 \cdot 10^{-4}, 5 \cdot 10^{-3}, 1 \cdot 10^{-2},\ 3.5 \cdot 10^{-2}, 7 \cdot 10^{-2}, 1 \cdot 10^{-1}, 5 \cdot 10^{-1}\}$. The optimal learning rate was chosen based on validation loss performance.
After training, each model was evaluated on the fixed test set, and the final weights and biases were extracted for further analysis.

Results

In this section it's possible to find the results collected after the training process.

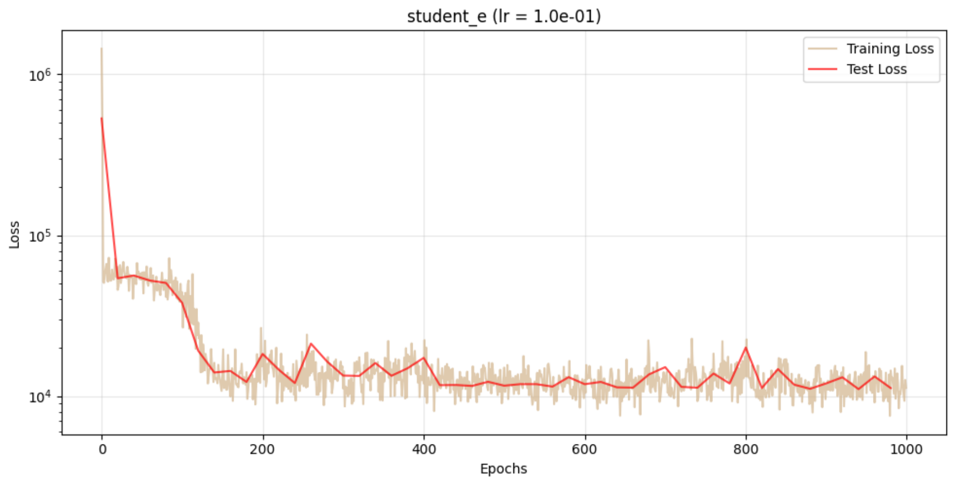
Under-parametrised Student:

- *Best Learning Rate*: $7.0 \cdot 10^{-2}$
- *Train Loss:* 12991.6211
- *Test Loss*: 11342.1270



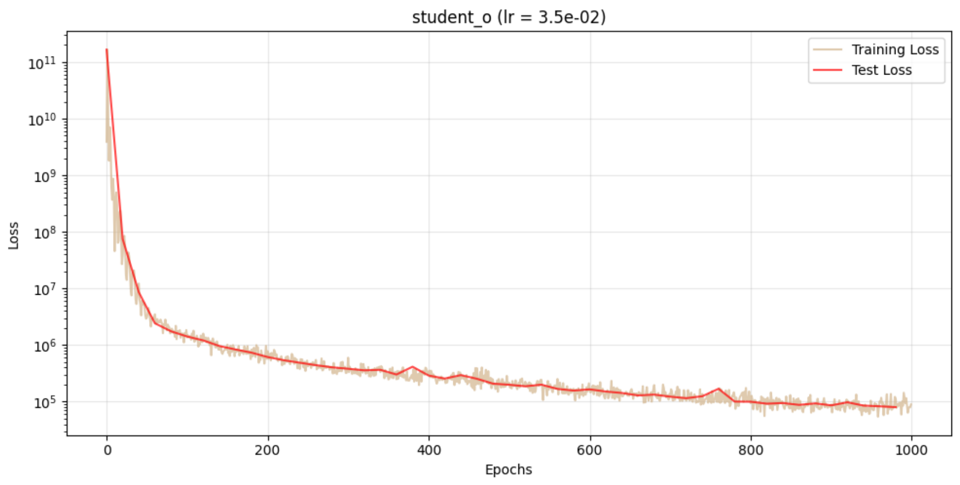
Exact-parametrised Student

- *Best Learning Rate*: $1.0 \cdot 10^{-1}$
- *Train Loss:* 11261.3838
- *Test Loss*: 11214.7285



Over-parametrised Student

- *Best Learning Rate*: $3.5 \cdot 10^{-2}$
- *Train Loss:* 89273.4531
- *Test loss*: 79931.9453



The results indicate that the best performance, in terms of both training and test loss, was achieved by the exactly-parameterized Student model. This model also converged the fastest, stabilizing after approximately 100 iterations. The under-parameterized Student followed, converging around 200 epochs with a slightly higher loss than the exactly-parameterized counterpart.
In contrast, the over-parameterized Student, due to its larger number of parameters, exhibited significantly slower convergence and poor generalization performance, ultimately reaching a substantially higher final loss.

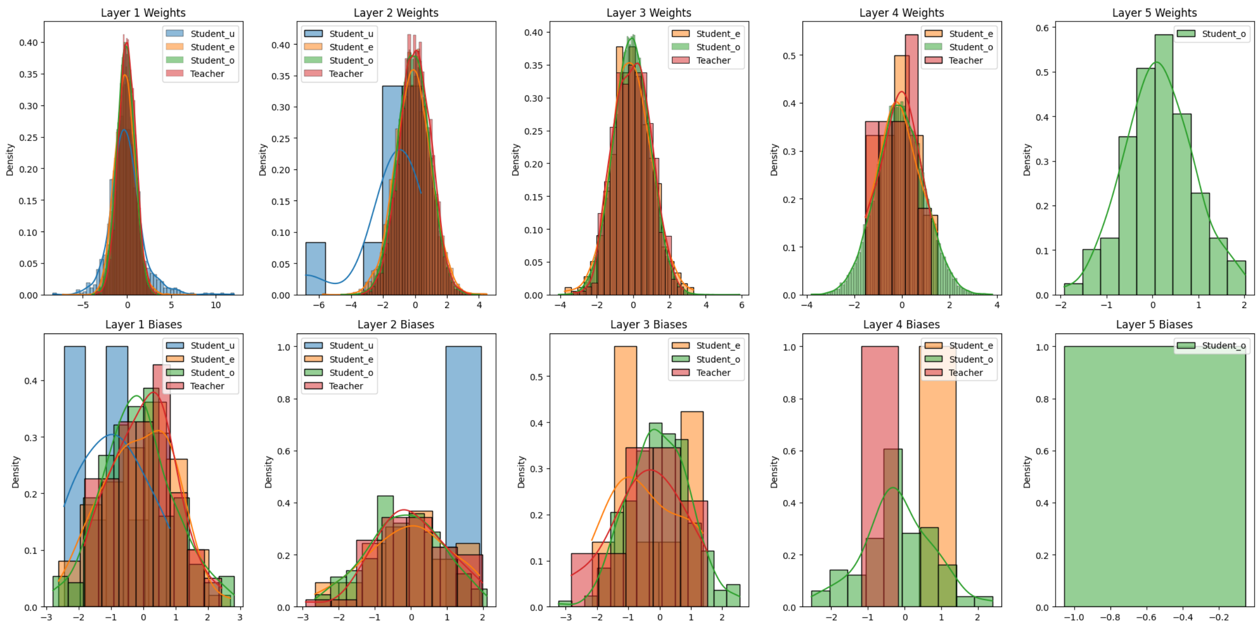
In summary, the architecture that matches the Teacher’s capacity (student_e) achieves the most effective trade-off between representational power and generalization.
While the under-parameterized Student demonstrates relatively good generalization, its limited capacity prevents its ability to fully represent the target function. On the other hand, over-parameterized models (student_o), despite potential benefits in trainability, are prone to overfitting and generalize poorly.

Parameters distribution

In this section we will collect (separately) weights and biases for each layer of the *Student* network, and compare their distribution to that of the *Teacher* network.

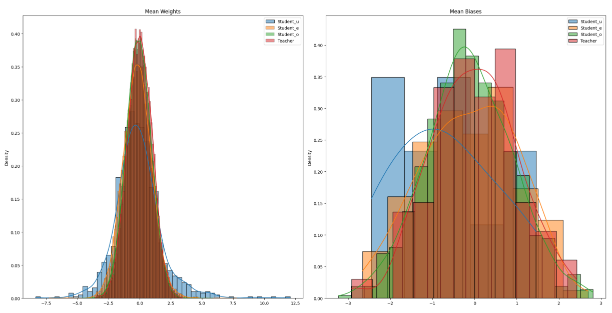
As we can see in the plots below, for the weights in the first 3 layers it's possible to notice that both the exact- and over-parametrised models show a good overlap with the teacher's weight distribution, suggesting that supervision effectively guided the learning of the initial weights. While the under-parametrised Student has the worst-reflecting distribution compared to the Teacher, it shows more pronounced tails, indicating higher variance and possible instability due to limited capacity.
While for the last two layers, where we only have the over-parametrised model, the fact that the distributions are well-centered and Gaussian suggests that the model handled the additional expressiveness well, although (as noted earlier) this did not translate into improved generalisation.

Biases exhibit greater variability across models, particularly in layers 2–4. The under-parametrised model displays more dispersed and less concentrated bias distributions, indicating noisier and more unstable learning. Exact- and over-parametrised models show distributions more similar to that of the teacher, especially student_e, which maintains a regular and symmetric profile across all layers.



Now we do the same for the collection of all weights and biases of the network:

- *Mean Weights*: the aggregated weight distributions show that student_e and student_o align with the teacher's distribution. In contrast, student_u exhibits significantly higher variance and heavier tails, suggesting it attempts to compensate for its limited expressive capacity through more extreme weights.
- *Mean Biases*: differences are more pronounced; student_u displays a highly irregular and shifted distribution, whereas the other two students closely approximate the teacher's Gaussian shape. Once again, student_e stands out for its balance and consistency



Conclusions

This experiment highlights how the degree of parameterisation in a *student* network impacts its ability to learn from a fixed *teacher*.
The exact-parameterised student achieves the best performance across all dimensions: it converges quickly, generalises well, and closely matches the teacher's parameter distributions. This confirms that architectural alignment with the teacher provides an optimal balance between expressivity and generalisation.
The under-parameterised student generalises surprisingly well despite its limited capacity. However, it fails to fully capture the target function and shows broader, more unstable parameter distributions.
The over-parameterised student, though expressive, suffers from inefficient training and poorer generalisation. Its additional layers introduce redundancy, reflected in drifted bias distributions and higher final losses.
Overall, the results confirm that *matching the teacher's architecture* enables the most effective learning in this setup, while both under- and over-parameterisation introduce specific limitations.

B. Function learning and hierarchical structure

Overview

In this second exercise of the challenge[2], we will explore how a deep residual networks learns two different sixth-degree polynomials: the **complete multivariate Bell polynomial** B_6, which has a **hierarchical** structure, and its **scrambled** counterpart $\tilde{B}_6$, which is obtained by randomly permute the indexes of each monomial independently. The formulation of B_6 and the scrambled polynomial $\tilde{B}_6$ is the following:

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

$$\tilde{B}_6(x_1,x_2,x_3,x_4,x_5,x_6) = x_2^6 + 15x_3x_2^4 + 20x_1x_4^3 + 45x_5^2x_3^2 + 15x_6^3 + 60x_4x_6x_5 + 15x_1x_5^2 + 10x_2^2 + 15x_3x_6 + 6x_4x_1 + x_5$$

Experimental design

Training and Test Sets

The **training** and **test** sets were both generated in association with the polynomials B_6 and $\tilde{B}_6$. Specifically, input vectors $\boldsymbol{x} = (x_1, \dots, x_6) \in \mathbb{R}^6$ were sample i.i.d. from the uniform distribution over the hypercube $[0,2]^6$ and corresponding output scalars $y \in \mathbb{R}$ were computed as either
$y = B_6(\boldsymbol{x})$ or $y = \tilde{B}_6(\boldsymbol{x})$, dependending on the target function.
The training set consisted of 10^5 samples, while the test set included $6 \cdot 10^4$ samples. Two distinct training and test sets were generated in total – one pair for each of the polynomials B_6 and $\tilde{B}_6$.

Model Architecture

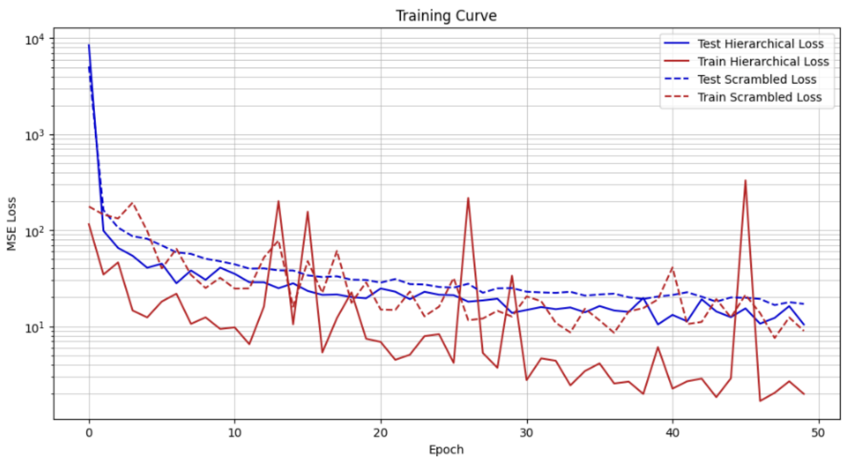
The network architecture used in this experiment consists of a **fully-connected residual** deep neural network with 9 layers in total, comprising an *input* layer, 8 *hidden* layers each with width 50, and a final *output* layer.

[2] https://github.com/annalisapaladino/AdvancedDeepLearning-KernelMethods/tree/main/Challenge%202

Residual *skip connections* are employed between layers that share the same dimensionality, allowing information to bypass intermediate transformations when beneficial. All hidden layers use the *ReLU* activation function, while the output layer remains linear.

Training

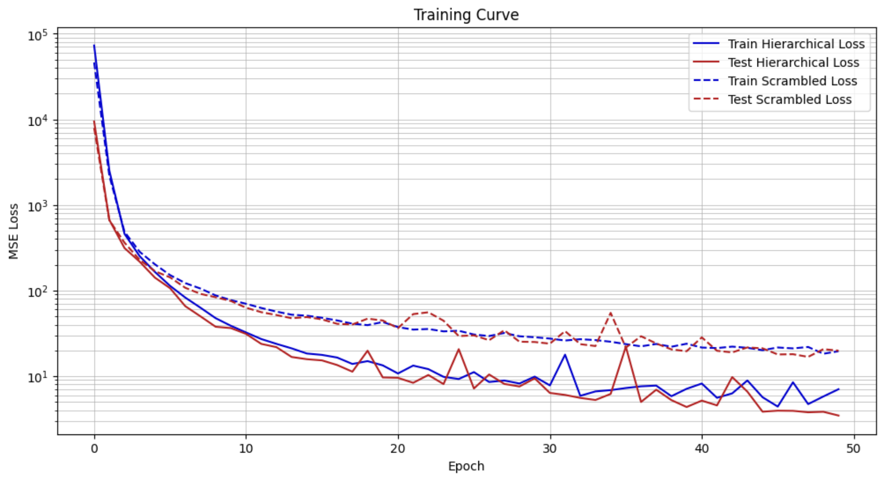
The **training** procedure was carried out for 50 epochs, adopts the *Adam* optimizer and minimizes the *Mean Squared Error* (MSE) loss using a batch size of 256, since a smaller batch size of 20 led to unstable training and less interpretable results, whereas the larger batch size provided more stable and interpretable outcomes. Below you can find the trining curve obtained with a batch size of 20.



A grid search over the learning rates $\{5 \cdot 10^{-5}, 1 \cdot 10^{-4}, 3 \cdot 10^{-4}, 5 \cdot 10^{-3}\}$ was conducted for both B_6 and $\tilde{B}_6$ in order to select an effective learning rate, for both polynomials the best value found was $3 \cdot 10^{-4}$.

Results

In this section it's possible to find the results collected after the training process.

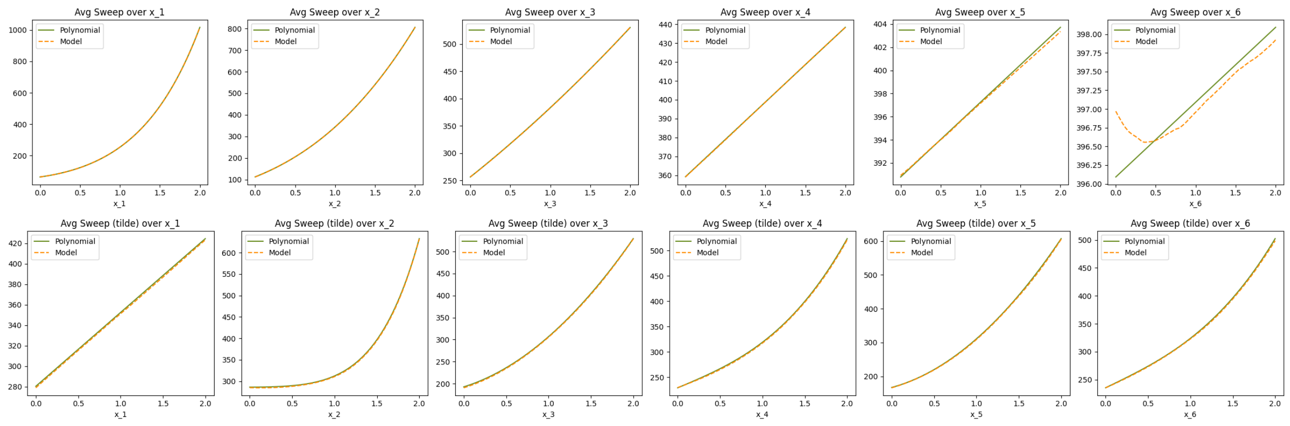


We can observe that both models achieve a rapid drop in loss during the initial epochs, with the hierarchical function showing a consistently lower training and test loss. This suggest that the model learns B_6 more efficiently, leveraging its compositional structure. In contrast, the scrambled polynomial shows an higher loss which indicates reduced generalisation and increased sensitivity to the lack of structure.

This is quantitavely confirmed by the final test losses summarised in the table below.

Polynomial	**Test Loss**
B_6	3.467
$\tilde{B}_6$	19.856

Lastly, to investigate how the trained network models the dependency of the output on individual input components, a sensitivity analysis was performed using one-dimensional sweeps. A single input vector $\boldsymbol{x} \in [0,2]^6$ was sampled, and each input variable x_i was independently varied over a fine grid within $[0,2]$, while the remaining components were kept fixed. The resulting inputs were evaluated by both the target polynomial and the corresponding trained model, and their outputs were plotted for comparison. These plots, averaged across multiple sampled inputs, are shown in two rows: the first corresponding to the hierarchical polynomial B_6, and the second to its scrambled counterpart $\tilde{B}_6$.



For B_6 we can see that the sweeps over x_1 to x_4 reveal a strong alignment between the polynomial outputs (green) and the model predictions (orange), indicating effective learning in these dimensions. However, as the sweep progresses to x_5 and especially x_6, the predictions begin to diverge from the ground truth, reflecting reduced accuracy. It's possible to explain this trend trough the hierarchical structure of B_6:

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

Notice how x_1 appears 7 times in the polynomial, x_2 appears 5 times, x_3 appears 3 times, x_4 appears 2 times, while x_5 and x_6 appear both only once. Moreover, terms such as x_1, x_2, x_3, not only appear more than once, but also with a high degree, while x_5 and x_6 don't, this makes them less significant on the overall. This means that the networks prioritize modeling more significant variables that have the greatest effect on reducing error, while less significant variables are modeled with lower precision.

On the contrary, for $\tilde{B}_6$ we can see a a strong alignment between the polynomial outputs (green) and the model predictions (orange) over all x_1 ... x_6.
It's possible to notice that, since there isn't a hierarchical structure there isn't a variable that is more significant than the others. Because of that the network distributes its learning effort more evenly across all variables, avoiding trade-offs but facing a higher generalization error due to the lack of an exploitable structure.

Conclusions

This experiment highlights how the underlying structure of a target function critically shapes the learning behavior of deep residual networks. When trained on the hierarchical Bell polynomial B_6, the network effectively exploits its compositional nature, prioritizing influential variables and achieving low test error. In contrast, the scrambled version $\tilde{B}_6$, which lacks such a hierarchy, leads to a more uniform allocation of learning effort but results in higher generalization error. These findings demonstrate that hierarchical structure facilitates more efficient representation and learning, while unstructured targets pose greater challenges for generalization in deep models.