Advanced Deep Learning and Kernel Methods
Challenge 2

Valentinis Alessio
Università degli Studi di Trieste

1 Introduction

The goal of this challenge was to conduct an empirical analysis of the learnability of Neural Netwoks, in particular exploring the effect of the number of parameters and on the capability of learning hierarchical functions.

2 Source Code

The complete code, including all experiments and implementation discussed in the following sections is available at the following GitHub repository.

3 Exercise A: The effect of *under*- and *over*-parametrization in the Teacher/Student setup

The objective of this exercise was to train a Neural Network supervisedly on input/output pairs generated from a Neural Network with frozen archietecture and weights, called Teacher. In particular we were interested in the effect of the number of parameters of the Student on the learning process.

3.1 Experimental Setup

The Teacher was a simple feedforward Neural Network with 3 hidden layers of dimension 74, 50, 10 with ReLU activation functions. Te network takes as input 100-dimensional vetors and outputs a scalar value. Weights and biases were initialized as *i.i.d.* samples from a Standard Normal distribution.

The Students on the other end, were of three kinds:

- **Underparametrized**: a feedforward Neural Network with 1 hidden layers of dimension 10, with ReLU activation functions.
- **Equally parametrized**: identical to the Teacher.
- **Overparametrized**: a feedforward Neural Network with 4 hidden layers of dimension 200, 200, 200, 100 with ReLU activation functions.

The training was done using the following procedure:

1. Generate a test set of 6×10^4 samples from the Teacher, taking as input 100-dimensional vectors of i.i.d. samples from a Multivariate Uniform distribution in $[0, 2]^{100}$.
2. Train each network on MSE loss on 1000 epochs, harvesting for each sample a fresh batch of 128 samples from the Multivariate Uniform distribution and obtaining the target by passing them through the Teacher. Adam optimizer was used, with a learning rate selected on a grid search $[5 \times 10^{-4}, 5 \times 10^{-3}, 3.5 \times 10^{-2}, 1 \times 10^{-2}, 2 \times 10^{-1}]$ and the best one was selected based on the validation loss.
3. Evaluate the performance of the Student on the test set and harvest the weights and biases.

3.2 Results

3.2.1 Performance evaluation

After performing some hyperparameter tuning, the best learning rate found were:

- **Underparametrized**: 1
- **Equally parametrized**: 0.035
- **Overparametrized**: 0.035

The results of the training process are reported in the following figure:

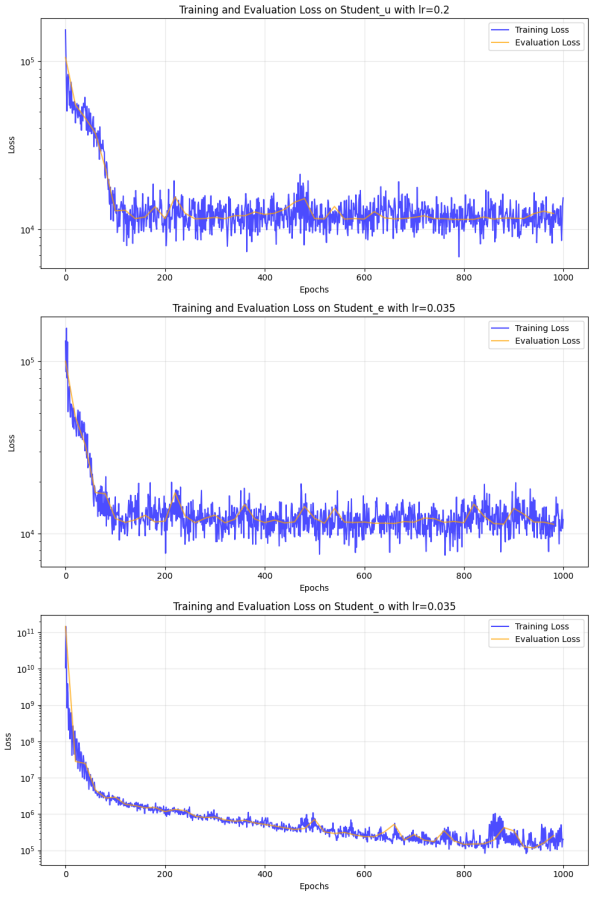

Figura 1: Training curves of the Students

After training process, the performance of the Students on the test set was evaluated and the results are reported in the table below:

Student	Training Loss	Test Loss
Underparametrized	13496.06	11752.7
Equally parametrized	**11804.85**	14566.2
Overparametrized	12928.31	**10965.8**

From this table we can observe how the best *in-sample* performance was obtained by the equally parametrized Student, while the best *out-of-sample* performance was obtained by the overparametrized Student. The underparametrized Student, on the other hand, performed poorly in both cases.

This behavior can be interpreted as an empyrical evidence of the *double descent* phenomenon:

- **Underparametrization regime**: the underparametrized student is not able to learn the target function, as the capacity of the model is not enough to capture the complexity of the function.

- **Interpolation regime**: the equally parametrized student is able to learn the target function in the training set, but it generalizes worse than the underparametrized student, as it is more prone to overfitting.
- **Overparametrization regime**: the overparametrized student is able to learn the target function in the test set, as it has enough capacity to both capture the complexity of the function and generalize well.

3.2.2 Parameters distributions

The next step was to analyze the distribution of the weights and biases of all the models and compare them, in order to understand how the number of parameters affects the parameter values. This analysis was performed both globally and layer-wise.

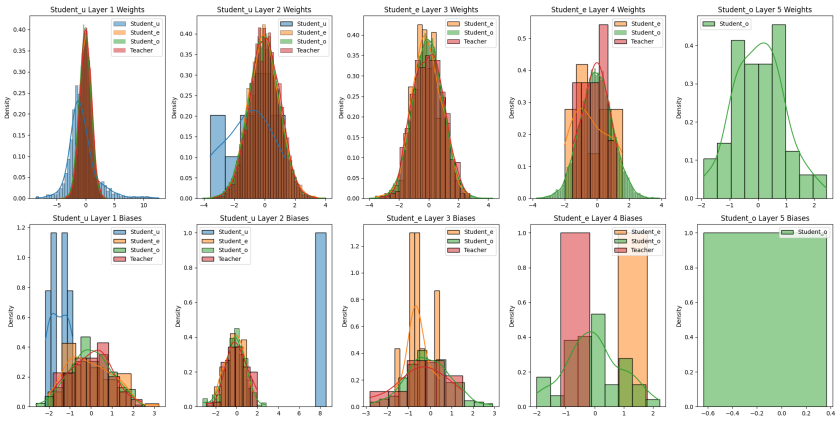

Figura 2: Parameters distribution of the Students

From the histograms in 2, we can see how the underparametrized Student has the worst-reflecting distribution compared to the Teacher, again supporting the hypothesis that the model is not able to learn properly the target function. The equally parametrized Student and the overparametrized one have a more similar distribution, with the overparametrized one having a slightly more peaked distribution.

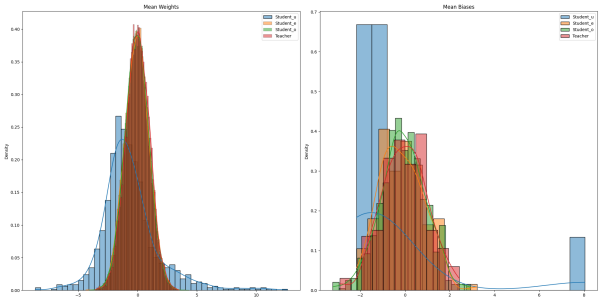

Figura 3: Parameters distribution of the Students

From the histograms in 3 we can observe how globally the distribution of the parameters of the underparametrized Student are the most different from the Teacher, while concerning the other two models the distribution is more similar.

4 Exercise B: Function learning and hierarchical functions

The objective of this exercise was to train a deep residual neural network supervisedly on examples generated from specific polynomials. Although looking at the monomials of these two polynomials they may seem very similar, the functions are actually very different, as the first one is the Bell polynomial of degree 6, a hierarchical function, while the second one is obtained by randomly permute the indexes of each monomial independently. The formulation of the two polynomials is the following:

$$
\begin{aligned}
B_6(x_1, x_2, x_3, x_4, x_5, x_6) &= x_1^6 + 15x_2x_1^4 + 20x_3x_1^3 + 45x_2^2x_1^2 + 15x_2^3 + 60x_3x_2x_1 + 15x_4x_1^2 + \\
&\quad + 10x_3^2 + 15x_4x_2 + 6x_5x_1 + x_6 \\
&= \sum_{i=0}^{5} \left(\binom{5}{i} \right) B_{5-i}(x_1, \ldots, x_{5-i}) x_6 \\
\tilde{B}_6(x_1, x_2, x_3, x_4, x_5, x_6) &= x_3^6 + 15x_3x_2^4 + 20x_4x_2^3 + 45x_4^2x_3^2 + 15x_5^3 + 60x_4x_2x_5 + 15x_6x_5^2 + \\
&\quad + 10x_2^2 + 15x_6x_4 + 6x_2x_5 + x_1
\end{aligned}
$$

4.1 Experimental Setup

The main part of the challenge consisted in training a deep ResNet, but first, we had to generate the training and test set. They were simply harvested by taking as input 6-dimensional vectors of i.i.d. samples from a Multivariate Uniform distribution in $[0, 2]^6$ and the corresponging targets were obtained by passing them through the two polynomials. The training set was composed of 10^5 samples, while the test set was composed of 6×10^4 samples. A total of two training sets and two test sets were generated, one for each polynomial.

The ResNet was composed of 9 hidden layers, each of dimension 50, with ReLU activation functions. The network takes as input 6-dimensional vectors and outputs a scalar value.

The training was done on the following parameters:

1. Adam optimizer with a learning rate of 1×10^{-3} (selected from a grid search with the aim of minimizing the train loss).
2. MSE loss on 50 epochs, with batches of 20 samples.

After the training was completed, a final evaluation was done on the test set, and corresponding loss values were harvested.

After this process, a last process had to be performed, i.e.:

- Sample a brand new test vector from the Multivariate Uniform distribution.
- For each of the components of the vector, evaluate both teh polynomials and the trained ResNets on a fine grid centered in that component.
- Keep track of the results and the errors.

4.2 Results

The results of the training process are reported in the following figure:

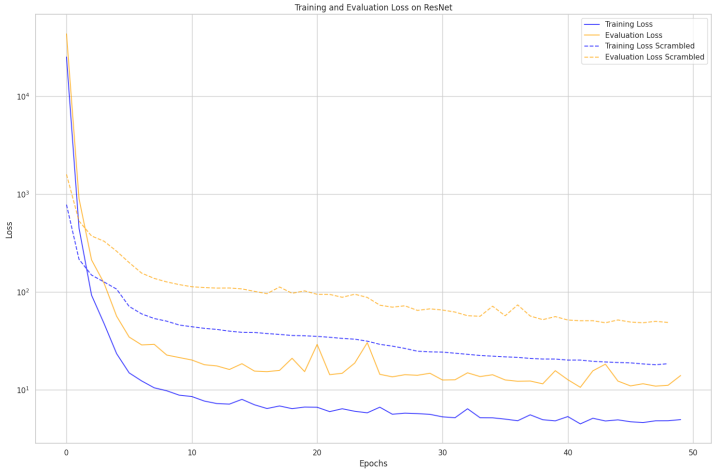

Figura 4: Training curves of the ResNets

After the training process, the performance of the ResNets on the test set was evaluated and the results are reported in the table below:

Polynomial	**Test Loss**
B_6	1.74656
$\tilde{B}_6$	16.9151

Looking both at the training curves and the test loss values, we can see how the same archietecture is more capable of learning the hierarchical function B_6, as its recursive formulation better resembples the architecture of the ResNet. This is not only reflected in the test loss, which is by far lower in the net trained on B_6, but also in the training curves, where the net trained on B_6 converges faster and to a lower value.

The last part of the challenge was to evaluate the ResNets on a fine grid centered in a new test vector. The results are reported in the following figure:

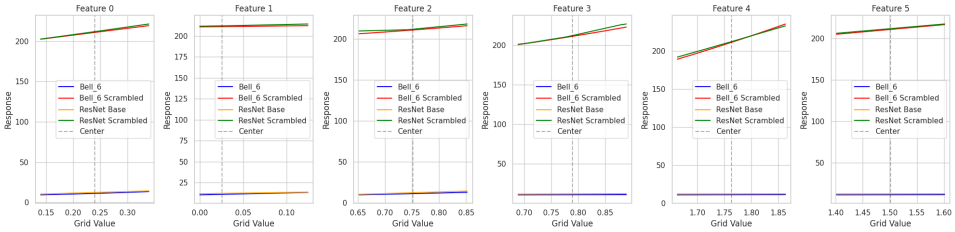

Figura 5: Fine grid evaluation of the ResNets

From this plot only not many information can be extracted, as the results yielded from the two polynomials and the corresponding ResNets are on differentscales, making interpretation not that straightforward. However, we can see how the ResNet trained on B_6 is able to better approximate the target function, as the solutions are better overlapped.

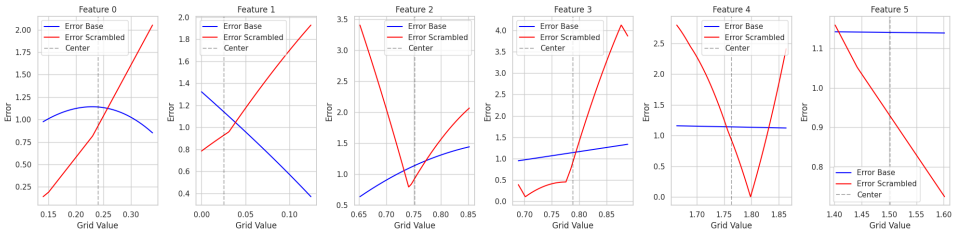

Figura 6: Errors of the ResNets on the fine grid

If we instead look at the errors, we can see how the ResNet trained on B_6 has a lower and more constant error on the evaluation grid, for almost all the components of the input vector, while the ResNet trained on $\tilde{B}_6$ has a higher and highly variable error. This is another empirical evidence that the ResNet trained on the original hierarchical function is able to not only learn the target function better, but also generalize better, while the other ResNet looks like it is overfitting the training set.

5 Conclusions

In this challenge we have explored the learnability capabilities of Neural Networks, in particular focusing on the effect of the number of parameters and their ability to learn hierarchical functions. In particular we have seen empirically the well-known *double descent* phenomenon, where the best generalization performance is obtained by the overparametrized model, and the importance of the architecture of the model in learning hierarchical functions.

Furthermore, we have seen how the same architecture can perform very differently on two similar functions, depending on the structure of the functions themselves. Moreover we have seen how a resnet can learn properly the bell polynomial, being able to generalize well on a fine grid evaluation, while it is not the case for a randomly permuted version of the same polynomial.