Advanced Deep Learning and Kernel Methods

Challenge 2:

An empirical study on the learnability of functions by NNs

Emanuele Ruoppolo
Università degli Studi Trieste
emanuele.ruoppolo@studenti.units.it

1 Introduction

Artificial Neural Networks (ANNs) are known to be universal approximators, meaning that they can approximate any function to any desired degree of accuracy. However, the learnability of a function by a neural network depends on many factors as the network's architecture, the learning parameters and the original function structure. The goal of this challenge is to empirically study the learnability of functions by ANNs, focusing on two particular cases:

1. The study of the effect of **under-** and **over-**parameterisation;
2. The study of learnability of **hierarchical structures** by Recurrent Neural Networks (RNNs).

In the first part we aim to approximate a *Teacher* function, represented by a fully-connected ANN with three *students* networks, each with a different fully-connected architecture, being them respectively under-, equally- and over-parametrized with respect to the Teacher network. Thus, once tuned the parameters and trained the networks we compared the results in terms of test loss, and by analyzing the weights and biases distributions of the networks, both layer-wise and all pooled.

In the second part we studied how RNNs are able to efficiently learn hierarchical structures. To do so we used a RNN to approximate a **sixth-order multivariate complete Bell polynomial** and a *scrambled* polynomial, with a similar structure although not hierarchical, then comparing the results.

2 Under- and Over-Parameterisation Study

2.1 Setup and Training Procedure

To investigate how parameterisation affects the learnability of functions, we instantiated a *Teacher* network $\mathcal{T}$ as a fully connected feedforward neural network mapping a 100-dimensional input to a single output scalar. The network consists of:

- 3 hidden layers of sizes 75, 50, and 10, respectively;
- ReLU activation functions after each hidden layer (but not the output);
- Weights and biases initialised i.i.d. from a standard normal distribution, which remain fixed thereafter.

We next generated a test set of $N = 60\,000$ samples $\{(x_i, \mathcal{T}(x_i))\}$ by drawing each x_i from the uniform distribution $\mathcal{U}([0, 2]^{100})$ and computing the corresponding label $y_i = \mathcal{T}(x_i)$.

Subsequently, three *Student* networks were created with the following architectures:

- S_u (**Under-parameterised**): 1 hidden layer of size 10;
- S_e (**Equally-parameterised**): 3 hidden layers of sizes 75, 50, and 10 (same as $\mathcal{T}$);
- S_o (**Over-parameterised**): 4 hidden layers of sizes 200, 200, 200, and 100.

To compare their capacities, recall that for an input–output layer pair of sizes n_I and n_O, the total number of parameters (weights and biases) is $n_I \cdot n_O + n_O$. Hence, the parameter counts for each network are:

- S_u: 1021
  1. $100 \times 10 + 10 = 1010$
  2. $10 \times 1 + 1 = 11$
- S_e: 11 896
  1. $100 \times 75 + 75 = 7575$
  2. $75 \times 50 + 50 = 3800$
  3. $50 \times 10 + 10 = 510$
  4. $10 \times 1 + 1 = 11$
- S_o: 120 801
  1. $100 \times 200 + 200 = 20\,200$
  2. $200 \times 200 + 200 = 40\,200$
  3. $200 \times 200 + 200 = 40\,200$
  4. $200 \times 100 + 100 = 20\,100$
  5. $100 \times 1 + 1 = 101$

All student models were trained on an MSE loss using the Adam optimizer. At each iteration, we sampled fresh inputs x_{train} from $\mathcal{U}([0, 2]^{100})$ and generated the corresponding labels via the teacher network $\mathcal{T}$. The fixed test set was used to monitor generalization error.

Before training, we performed a grid search for learning rates in $\{1\times10^{-4}, 3.5\times10^{-4}, 5\times10^{-4}, 9\times10^{-4}\}$. We chose the rate that produced the lowest test-set loss after a reduced $\sim$ 10 000-iteration run. This procedure yielded 9×10^{-4} for S_u and S_e, and 1×10^{-4} for S_o.

We trained S_u and S_e for 15 000 iterations each, while S_o was trained for 25 000 iterations, until both training and test losses reached a quasi-stationary regime. All networks used a batch size of 256, logging both training and test losses every 50 iterations. Figure 1 shows the training curves, and Table 1 summarizes the final generalization errors.

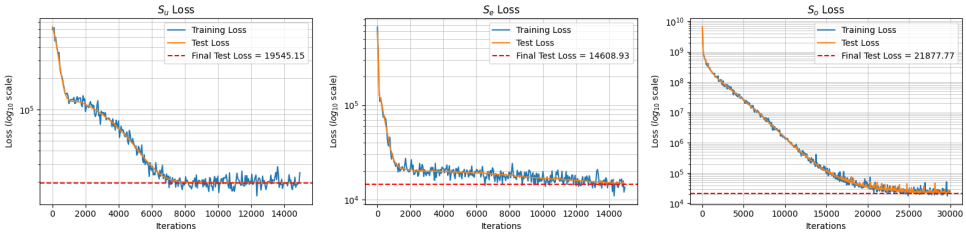

Figure 1: Training and test losses for S_u, S_e, and S_o.

Model	Generalization Error
S_u	19 545
S_e	14 609
S_o	21 877

Table 1: Final generalization errors for the three student networks.

From these results, we can highlight two main observations:

1. **Convergence speed:**
   - S_e converged quickest (within $\sim$ 2 000 iterations).
   - S_u required more than 8 000 iterations to reach its plateau.
   - S_o, given its large parameter count, converged more slowly (over 25 000 iterations).

2. **Generalization:**
    - S_e achieved the lowest final test loss (14 609).
    - S_u showed limited capacity (19 545).
    - S_o, despite having many more parameters, did not generalize as well (21 877) and required significantly more computational effort.

Overall, these findings illustrate the classic under- and over-parameterization trade-off: while an under-parameterized network struggles to learn the target function effectively, an over-parameterized network can learn slowly and may generalize poorly without careful regularization. We can conclude that the "just-right" (S_e) architecture outperformed both extremes in terms of convergence speed and out-of-sample performance.

2.2 Distribution of Learned Parameters

Figure 2–3 and Tables 2–3 compare the learned parameter distributions of the *Student* models to those of the fixed *Teacher* network. The Teacher's weights and biases were initialized (and kept) at i.i.d. standard normal values. Below, we summarize the most notable observations.

Layer-by-layer trends.

- **Layer 1 (common to all models).** As shown in Table 2, the *Under*-parameterized network (S_u) has a substantially larger mean and standard deviation for its first-layer weights (0.48 and 1.74, respectively) compared to the Teacher's near-zero mean and unit standard deviation. This suggests that S_u must stretch or distort its limited parameter space to approximate the target function. In contrast, both *Equallyly-* and *Over*-parameterized networks (S_e and S_o) exhibit weight means and standard deviations much closer to the Teacher's, indicating they can more flexibly match the original function without requiring extreme parameter values.
- **Subsequent hidden layers.** For layers 2 and 3 (not present in S_u), both S_e and S_o maintain parameter statistics (means and standard deviations) very close to those of the Teacher. Some slight deviations appear in the biases (e.g., the bias mean in layer 2 for S_e is 0.12, whereas the Teacher's is -0.23), but overall the learned distributions remain near the standard-normal baseline.
- **Layer 4 (S_o only).** Since the Over-parameterized model has one extra hidden layer, its layer-4 distribution cannot be directly compared to the Teacher's. However, the weights in this layer maintain a mean and standard deviation that are again close to zero and one, respectively. This consistency suggests that, although S_o has many more parameters, the optimizer still steers them toward a distribution loosely resembling the Teacher's initialization scale.
- **Output layer.** The largest discrepancies arise in the Under-parameterized model's output layer, where the mean of the weights (0.97) and their standard deviation (4.13) deviate markedly from the Teacher's values (-0.26 and 0.91, respectively). This result again indicates that S_u must use "extreme" parameters to compensate for its simpler architecture. In contrast, S_o and S_e have final-layer weight distributions that remain closer to the Teacher's, with smaller absolute means and narrower spreads.

Overall weight and bias distributions. When aggregating all the layers (see Table 3), we observe:

- *Under*-parameterized weights show a larger mean (0.49) and significantly higher standard deviation (1.78) than the Teacher's. Its aggregated bias distribution also has a notably higher mean (0.68).
- *Equallyly-* and *Over*-parameterized models end up with global weight means near zero (0.04 and -0.01, respectively) and standard deviations very close to the Teacher's reference value ≈ 1.0. Their overall biases also stay closer to zero, with standard deviations just under 1.0.

Interpretation. These distributional differences align well with the training results:

- The *Under*-parameterized network is forced to take on more extreme parameter values to approximate the Teacher's function, which is reflected in broader distributions and shifted means.

- The *Equallyly*- and *Over*-parameterized models generally remain close to the Teacher's parameter scale, suggesting they can more readily *mimic* the Teacher without inflating or skewing parameters excessively.
- Having too few parameters (S_u) yields suboptimal generalization, but having many more (S_o) can also hinder final performance unless training is carefully tuned or regularized. The parameter distributions confirm these observations: although S_o remains near the Teacher's scale, it requires significantly more training iterations and still does not match S_e in final test performance.

Overall, these distributional analyses help clarify how each model's capacity shapes the path it takes through parameter space. The *Under*-parameterized model must "stretch" its weights and biases, while the *Equallyly*- and *Over*-parameterized models more closely preserve the Teacher's standard-normal initialization scale, albeit with subtle shifts reflecting the different network depths and training dynamics.

Layer	S_u		S_e		S_o		**Teacher**	
	μ_u	σ_u	μ_e	σ_e	μ_o	σ_o	μ_T	σ_T
w_1	0.48	1.74	0.06	1.01	-0.03	0.99	0.00	0.99
b_1	0.67	0.70	-0.16	0.97	-0.01	0.05	0.15	0.78
w_2	–	–	0.01	1.01	-0.01	0.99	0.00	1.02
b_2	–	–	0.12	1.09	-0.10	1.01	-0.23	1.06
w_3	–	–	-0.04	0.98	0.00	1.00	-0.02	1.03
b_3	–	–	-0.12	0.49	-0.02	0.99	0.19	1.19
w_4	–	–	–	–	-0.01	0.99	–	–
b_4	–	–	–	–	-0.26	0.90	–	–
w_{out}	0.97	4.13	0.17	0.77	-0.03	1.05	-0.26	0.91
b_{out}	0.83	–	-0.40	–	-0.76	–	-0.91	–

Table 2: Layer-wise parameter statistics, comparing the three student models (S_u, S_e, S_0) to the Teacher.

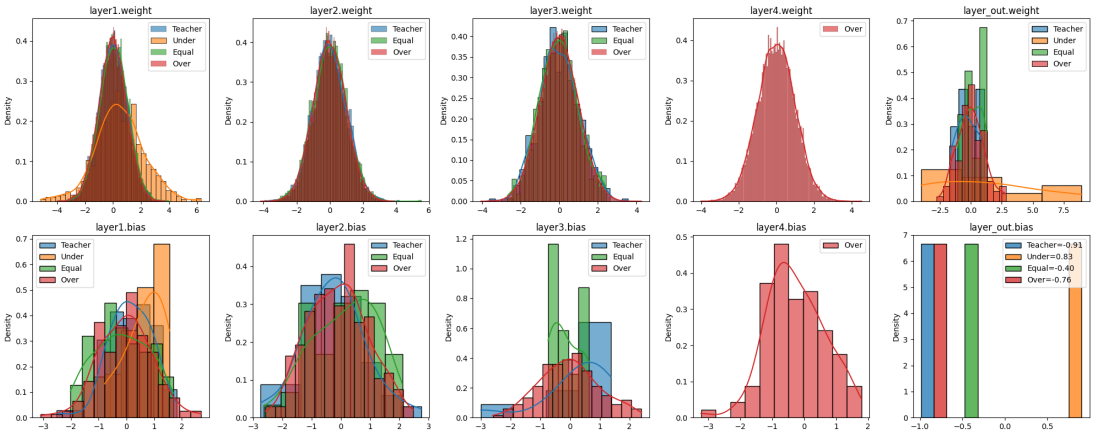

Figure 2: Distributions of weights and biases layer-wise.

Layer	S_u		S_e		S_o		**Teacher**	
	μ_u	σ_u	μ_e	σ_e	μ_o	σ_o	μ_T	σ_T
w_T	0.49	1.78	0.04	1.01	-0.01	1.00	0.00	1.00
b_T	0.68	0.67	-0.06	1.00	-0.08	0.98	0.00	0.94

Table 3: Aggregated parameter statistics, comparing the three student models (S_u, S_e, S_0) to the Teacher.

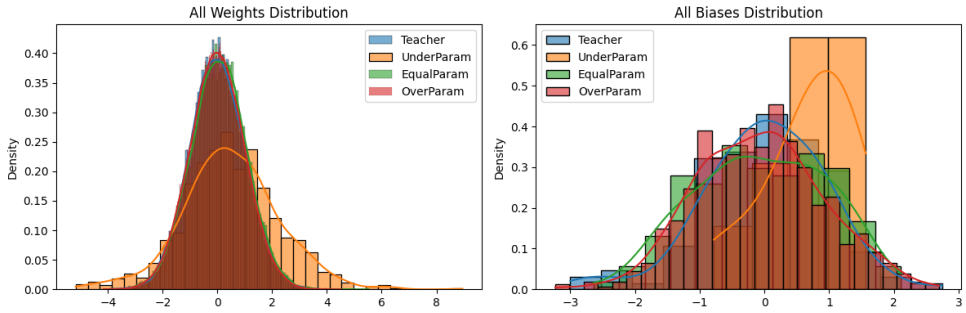

Figure 3: Distributions of aggregated weights and biases.

Conclusions. Overall, these results demonstrate that an under-parameterized model must drastically adjust its parameter distributions—often producing outlying weights or biases—to make up for insufficient capacity, leading to weaker generalization. Conversely, over-parameterized networks converge more slowly and still can fail to outperform a suitably matched (Equallyly-parameterized) architecture in terms of final test performance. Notably, the Equallyly-parameterized student not only reaches its target faster but also aligns its parameters more closely with the teacher's distribution, confirming that model capacity commensurate with task complexity typically yields the most efficient and accurate approximation.

3 RNNs and hierarchical structures

In this second experiment, we investigate how hierarchical structure in a target function affects a deep model's ability to learn and generalize. We consider two sixth-order polynomials on $\mathbb{R}^6$: the *hierarchical* Bell polynomial B_6 and a *scrambled* version $\tilde{B}_6$. The latter preserves the same monomials and coefficients but permutes their input variables in a way that removes the strictly hierarchical pattern of B_6. Here we report B_6 and the scrambled polynomial $\tilde{B}_6$:

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

$$\tilde{B}_6(x_1,x_2,x_3,x_4,x_5,x_6) = x_6^6+15x_4^4x_5+20x_1^3x_2+45x_3^2x_4^2+15x_5^3+60x_6x_1x_4+15x_2^2x_3+10x_1^2+15x_3x_6+6x_2x_5+x_2$$

3.1 Setup and Training Procedure

Datasets. We sampled input vectors $\boldsymbol{x} \in \mathcal{U}[0,2]^6$, independently for each train/test set, from a uniform distribution, generating:

- 10^5 samples for each polynomial's *training* set.
- 6×10^4 samples for each polynomial's *test* set.

Labels for each dataset were computed as $y = B_6(\boldsymbol{x})$ or $y = \tilde{B}_6(\boldsymbol{x})$, respectively.

Model Architecture. We employed a *fully-connected residual* deep network with 9 total layers:

- 1 input layer,
- 7 hidden layers (each of width 50, with ReLU activations),
- 1 output layer (no activation).

In each hidden layer, a skip connection is added only if the layer width remains 50. For layers of different dimensionality (such as the input or output), the skip connection is omitted.

Training Details. All models were trained using the Adam optimizer with a batch size of $\mathsf{B} = 256$. Although the assignment suggested a batch size of $B = 20$, training results were highly unstable, resulting in both training and test curves with high variability across successive epochs. This instability is reasonable given the high number of training points. Therefore, we decided to increase the batch size, which led to more stable curves.

We performed a grid search over learning rates $\{5 \times 10^{-5}, 1 \times 10^{-4}, 3 \times 10^{-4}\}$, ultimately selecting $\eta = 1 \times 10^{-4}$ for both B_6 and $\tilde{B}_6$ based on preliminary runs (~ 30 epochs). We logged the training and test MSE losses after each epoch, with 50 epochs for the final training.

3.2 Results and Discussion

Figure 4 shows the training curves for both polynomials. While both models rapidly reduce their training loss, the hierarchical polynomial converges to a considerably lower test loss. Quantitatively, the final test loss for B_6 is approximately 12, whereas for $\tilde{B}_6$ it is over 55. This discrepancy suggests that the network more readily exploits the explicit hierarchical patterns in B_6 than the scattered interactions of $\tilde{B}_6$.

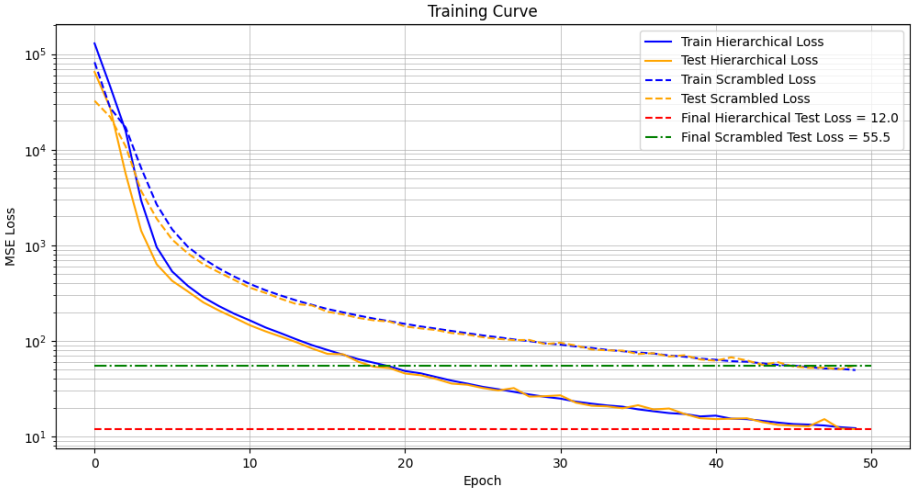

Figure 4: Training (blue) and test (yellow) losses for the hierarchical polynomial (B_6, in solid lines) and scrambled polynomial ($\tilde{B}_6$, in dashed lines). Horizontal dashed lines denote the final test losses.

Input Sweeps. To better understand each network's learned mapping, we performed one-dimensional "sweeps" for each input x_i, keeping other components fixed to random values sampled from $\mathcal{U}[0, 2]$. We then plotted in figure 5 the average (on 100 independent trials) polynomial outputs (blue) and model predictions (red) as x_i varied from 0 to 2.

- **Hierarchical Polynomial Sweeps (top row).** For the Bell polynomial B_6, we observe a clear pattern of decreasing model accuracy as we progress from x_1 to x_6. For variables x_1 through x_4, the model predictions closely match the true function values. However, for x_5 and particularly x_6, noticeable deviations appear between the model and the polynomial. This pattern reflects the hierarchical structure of B_6, where x_1 appears in multiple high-powered terms (including x_1^6), making it the most influential variable, while x_6 appears only once with power 1, making it the least significant contributor to the output.
- **Scrambled Polynomial Sweeps (bottom row).** In contrast, for the non-hierarchical polynomial $\tilde{B}_6$, we observe remarkably consistent accuracy across all six input variables. The model predictions align almost perfectly with the true polynomial values for each variable sweep. This uniform performance stands in stark contrast to the hierarchical case and demonstrates that the network can learn all variable relationships with similar precision when no single variable dominates the polynomial's behavior.

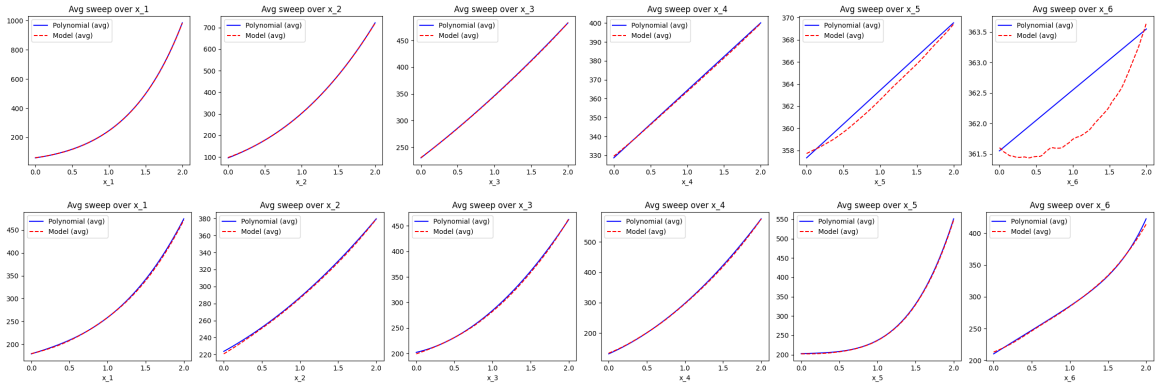

Figure 5: Comparison of polynomial (blue) vs. model (red) outputs in one-dimensional input sweeps. Top row: B_6 (hierarchical) polynomials; bottom row: $\tilde{B}_6$ (scrambled). Each sub-figure shows a different input dimension swept from 0 to 2, with all other components held fixed.

Interpretation. The differential performance observed between the hierarchical and non-hierarchical polynomials shows fundamental aspects of neural network learning dynamics. Learning the relationships that have the biggest impact on lowering overall error is given priority by the residual network in the hierarchical Bell polynomial B_6. Because x_1 has a greater impact on the output than x_6, the network concentrates more representational capacity to effectively modelling the relationship with dominating variables, at the expense of less significant ones.

On the contrary there is no discernible hierarchy among the inputs in the scrambled polynomial $\tilde{B}_6$, where we have rearranged the variables' relative importance (for example, making x_6 appear in a 6th-power term). The network is then able to optimize its weights more consistently across all variable relationships as a result of the more balanced error distribution that results from training. The network does not have to compromise accuracy on any one input dimension because there is no dominant variable, although its generalisation error is higher, since its scrambled structure is not as learnable as an hierarchical one.

These findings demonstrate how the neural network's learning and representational resource allocation are strongly influenced by the target function's underlying mathematical structure. Because it learns hierarchical relationships in phases, giving priority to the most influential components first, the residual architecture seems to be especially sensitive to them.

3.3 Conclusions

Our tests show that when deep residual networks are given hierarchical versus non-hierarchical multivariate functions, they learn differently. The network clearly emphasizes recording the associations of dominating variables when learning the hierarchical Bell polynomial B_6, exhibiting lower accuracy for variables that contribute less significantly to the output. In contrast, the network exhibits consistent accuracy across all input dimensions while learning the non-hierarchical scrambled polynomial $\tilde{B}_6$.

The model trained on B_6 achieves a final test loss that is roughly 4.6 times lower than the model trained on $\tilde{B}_6$ (12.0 vs. 55.5), which is more convincing evidence of the benefit of hierarchical structure. This substantial performance gap, despite both polynomials containing the same number of terms with identical powers and coefficients, strongly suggests that the hierarchical organization of B_6 makes it inherently more learnable by neural networks with residual connections.

Particularly when modeling complex systems with several levels of hierarchical structure, these findings have significant ramifications for deep learning applications. They propose that the relative relevance of input features in the goal function determines how residual networks spontaneously distribute their representational capacity. This characteristic makes them ideal for domains where hierarchical relationships are inherent, but it may necessitate careful design considerations when all input dimensions—regardless of their relative contribution to the output—need to be described with comparable precision. These results also shed light on how neural networks may infer the fundamental structure of complex functions by giving priority to the most important elements first. This learning strategy is similar to how people typically approach challenging problem-solving activities. Despite having similar representational complexity, the significantly better generalization performance on the hierarchical polynomial suggests that neural network architectures have an inductive bias that favors hierarchically structured functions, which may account for their success in a variety of real-world domains where hierarchical relationships are prevalent.