Investigating the effects of Overparameterization while solving the Pendulum with DeepONets

Anonymous Author(s)
Affiliation
Address
email

Abstract

Machine learning methods have made substantial advances in various aspects of physics. A particularly active area in this theme has been what is called "physics informed machine learning" [15] which broadly refers to the methods used for numerically solving differential equations using neural nets. DeepONets [22] are one of the most prominent ideas in this theme. In this work we study the training dynamics of DeepONets for solving the pendulum and demonstrate that contrary to usual expectations, test error here has its first local minima at the interpolation threshold i.e when model size $\approx$ training data size. Secondly, the best test error over all training iterations does not show any significant double-descent and particularly so when there is label noise in the training data and test data is clean. Lastly, we show evidence that triple-descent [1] is unlikely to occur for DeepONets.

1 Introduction

A particularly intriguing aspect in usual neural net training is that of population risk's dependence on the model size having one or two local maxima (called "double/triple descent" respectively) w.r.t the model size and it being monotonically decreasing for nets larger than a threshold value. In recent times the importance of Double-Descent was brought to light in [4]. They observed it for a number of models like decision trees, random features and two-layer neural networks. Evidence for double-descent for deep NNs like ResNets and Transformers was presented in [27], where they showed epoch-wise double-descent, and also that the double-descent curves at different training set sizes can intersect in a way s.t there can be over-parameterized nets where using more training data hurts. Further evidence of double-descent appeared in, [31], [8] and evidence for triple-descent has also appeared in [1].

From all these examples a few traits have emerged which are now believed to be universal across all neural experiments doing supervised learning:

- As the model size increases the minimum achievable test error can have multiple peaks but one local maxima always occurs in the vicinity the interpolation threshold i.e when the number of parameters is nearly the same as the training data-size.
- The above peak is exacerbated by the label noise.
- Asymptotically in the model size the minimum achievable test error keeps decreasing.

In light of these results we ask the following questions about training a DeepONet:

Does DeepONet training have a double-descent phenomenon at the interpolation threshold i.e when number of training data samples = the number of trainable parameters?

Does DeepONet training have a triple-descent phenomenon with the peaks of the test error vs number of parameters curve happening at the two thresholds identified in [1]?

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

How are any of these peaks in the above two phenomenon affected by injecting noise into the training data (and not the test data)?

We note that in the setup of the last question above, we are not adding the extra noise in the test-data. Hence this explores the phenomenon of double/triple descent in DeepONets while at the same time testing for out-of-distribution robustness i.e DeepONet's ability to generalize on true data while being trained on data of lower quality.

2 Experimental Setup

2.1 DeepONet Architecture

The DeepONet $\mathcal{U}$ we use maps as, $\mathcal{U} : \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}$. where m is the number of "sensor points". Given an input $(\mathbf{x}_{\text{func}}, x_{\text{loc}}) \in \mathbb{R}^m \times \mathbb{R}$ to $\mathcal{U}$, the DeepONet can be described as:

1. **A branch network:** This takes in as input $\mathbf{x}_{\text{func}}$, and has p feedforward, fully-connected layers with ReLU activations. The first layer has input dimension of m and output dimension w. For subsequent $p - 1$ layers input dimension = output dimension = w.
2. **A trunk network:** For the same p and w as above, this takes in an input x_{loc}, and has p feedforward, fully-connected layers with ReLU activations.The first layer has input dimension of 1 and output dimension w. For subsequent $p - 1$ layers input dimension = output dimension = w.
3. **Output:** This computes the inner product of the outputs of the branch net and the trunk net and adds a bias term to the result.

In our experimental study we choose $m = 100$ and $p = 4$ for all tests.

2.2 The training data and the loss function

For some "forcing function" $u : \mathbb{R} \to \mathbb{R}$, and "spring constant" k (set to 1 in the experiments), the O.D.E. to solve for the pendulum problem can be given as follows for a dynamical variable $\mathrm{s} \in \mathbb{R}^2$,

$$\frac{\mathrm{ds}}{\mathrm{d}t} = \mathbf{g}(\mathrm{s}, u, t) = (s_2, -k \cdot \sin(s_1) + u)$$

In light of the above, each training/test data can be seen as a $3-$tuple, given by $(\mathbf{x}_{\text{func}}, x_{\text{loc}}, y)$, where $\mathbf{x}_{\text{func}} = u(\mathbf{x}_{\text{sensors}}) \in \mathbb{R}^m$ for $\mathbf{x}_{\text{sensors}} = (x_1, x_2, ..., x_m) \in \mathbb{R}^m$, where $\mathbf{x}_{\text{sensors}}$ is an $m-$sized equi-spaced grid on the $[0, 1]$ interval. For our experiments, we sample u as a Chebyshev polynomial of degree atmost 20 by sampling the corresponding coefficients uniform randomly from $[-1, 1]$. $x_{\text{loc}} \in \mathbb{R}$ is a uniform random sample $\sim [0, 1]$. $y \in \mathbb{R}$ is the output for the above given training points. This is calculated as a standard O.D.E solver's approximate solution of the pendulum O.D.E. over an interval $[0, t_f]$ where $t_f = x_{\text{loc}}$. We refer the reader to [22] for more details about this type of data.

Hence for $\mathcal{S}$ training data samples, the ℓ_2 empirical loss is,

$$\hat{\mathcal{L}}_{\text{DON}}\,(\text{DeepONet}) \coloneqq \frac{1}{\mathcal{S}} \sum_{i=1}^{\mathcal{S}} \left(y_i - \text{DeepONet}(\mathbf{x}_{\text{func,i}}, x_{\text{loc,i}})\right)^2$$

2.3 Implementation

We train the DeepONet above on clean training data as well as when the true labels are distorted by a standard normal additive noise at standard deviations of $0.001, 0.01, 0.1, 1.0$. Note that we never add noise to the test-data and hence we are implicitly testing for robustness of the training.

Our code is built on top of the work in [22], covered under the Apache-2.0 license. Our implementation is available here. We ran all experiments on Google Colab [9] using the single GPU provided therein (NVidia Tesla T4 or K80). The training required for this paper took approximately 70 GPU-hours. Data used in this paper is included in the above implementation.

3 Results

In Figures 1 - 4, we show the training error attained at different model sizes (along the x-axis) on clean data and the test error (on clean data) attained at the same model sizes - for clean training data as

well as for different levels of noise contamination in the labels of the training data as described above. In Figures 5 - 6 we overlay the test-error curves for both datasets (1K and 5K training samples). The dashed lines indicate various interpolation thresholds as defined in [1], [27]. We train the models using G.D., S.G.D. (mini-batch size of 32 per iteration, using Adam optimizer [16] with learning rate 0.0001, $(\beta_1, \beta_2) = (0.9, 0.999)$) on two datasets of different sizes ((#Train, #Test) = $(1000, 200)$ and $(5000, 1000)$) measuring the best error attained during the whole training and the average end-point error, averaging over the last 10^4 iterations of the training.

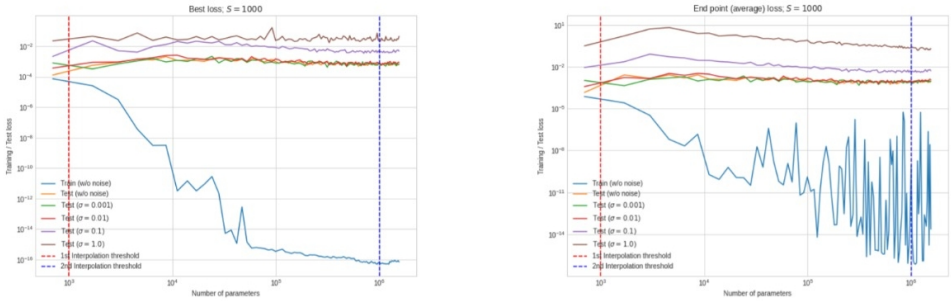

Figure 1: GD: (a) 1K samples, Best loss (b) 1K samples, Average end-point loss

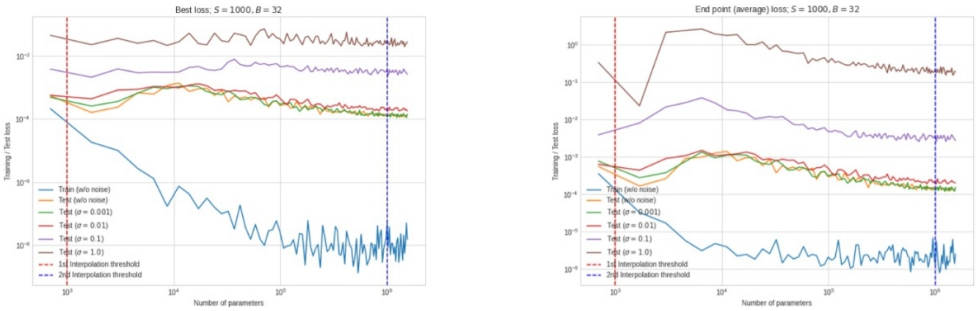

Figure 2: SGD: (a) 1K samples, Best loss (b) 1K samples, Average end-point loss

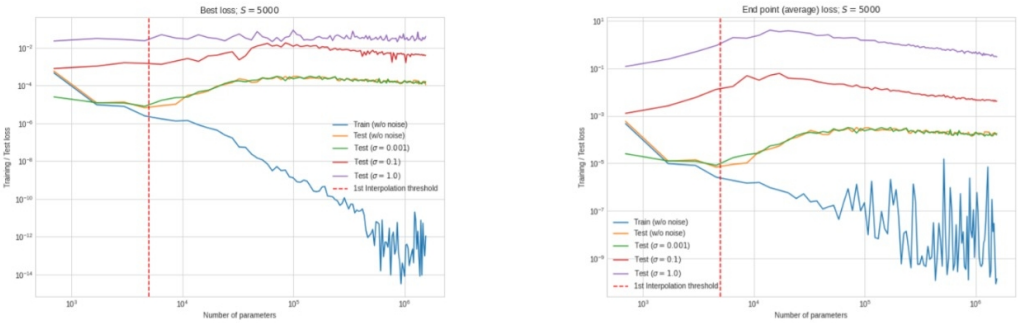

Figure 3: GD: (a) 5K samples, Best loss (b) 5K samples, Average end-point loss

4 Conclusion

The first local minima for the test error occurs for (number of parameters) equal to or slightly more than the training-data size (the interpolation threshold). We note that this is contrary to the observation

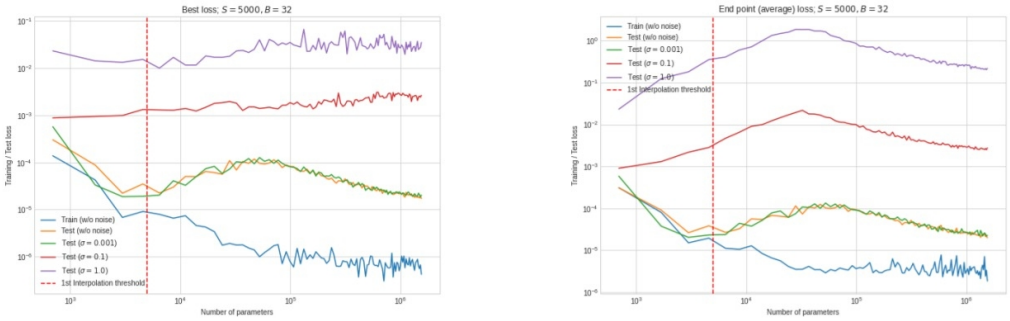

Figure 4: SGD: (a) 5K samples, Best loss (b) 5K samples, Average end-point loss

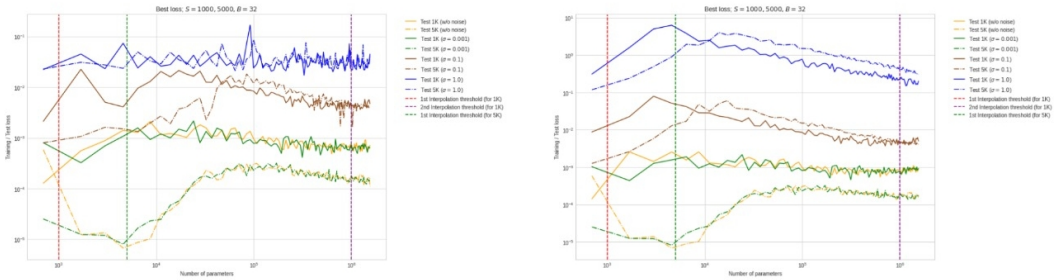

Figure 5: Comparison of a) best loss and b) average loss for GD

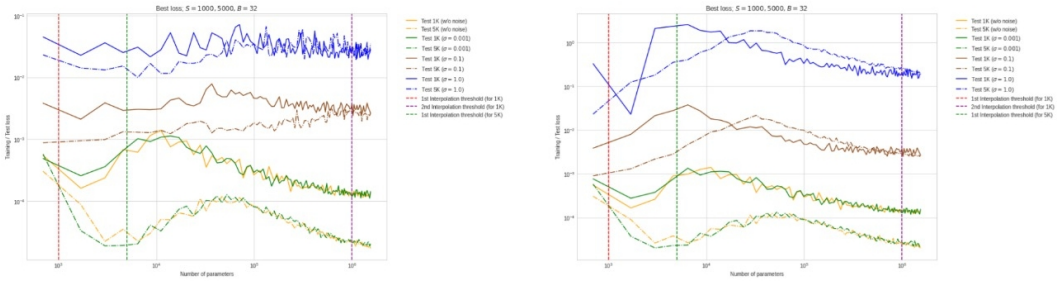

Figure 6: Comparison of a) best loss and b) average loss for SGD

in [27] who observed a local maximum of test error at the interpolation threshold. *Secondly,* The best loss setting has significant Double descent only for the SGD case, being more pronounced when the noise in training data is less severe (i.e., low σ). However, we always observe double descent in the average endpoint loss case, and it is more pronounced when trained using SGD. In all cases, the best loss setting exhibits much lesser double descent than the average loss. *Lastly*, we do not observe Triple descent at the locations specified in [1] in our experiments.

5 Future Work

In the experimental setup (Section 2) for the Branch and Trunk net, we choose both to be of the same depth, and their intermediate layers to have the same input and output dimensions. We leave the analysis of not using such 'symmetric' DeepONets to future work.

6 Societal Impact

To the best of our knowledge, we are not aware of any negative societal impact of this work.

References

[1] Ben Adlam and Jeffrey Pennington. *The Neural Tangent Kernel in High Dimensions: Triple Descent and a Multi-Scale Theory of Generalization*. 2020. arXiv: 2008.06786 [stat.ML].

[2] Ben Adlam and Jeffrey Pennington. *Understanding Double Descent Requires a Fine-Grained Bias-Variance Decomposition*. 2020. arXiv: 2011.03321 [stat.ML].

[3] Mikhail Belkin, Daniel Hsu, and Ji Xu. "Two Models of Double Descent for Weak Features". In: *SIAM Journal on Mathematics of Data Science* 2.4 (Jan. 2020), pp. 1167–1180. ISSN: 2577-0187. DOI: 10.1137/20m1336072. URL: http://dx.doi.org/10.1137/20M1336072.

[4] Mikhail Belkin et al. *Reconciling modern machine learning practice and the bias-variance trade-off*. 2019. arXiv: 1812.11118 [stat.ML].

[5] Mogens Dalgaard et al. "Global optimization of quantum dynamics with AlphaZero deep exploration". In: *npj Quantum Information* 6.1 (2020), pp. 1–9.

[6] Beichuan Deng et al. "Convergence rate of DeepONets for learning operators arising from advection-diffusion equations". In: *arXiv preprint arXiv:2102.10621* (2021).

[7] Weinan E and Bing Yu. *The Deep Ritz method: A deep learning-based numerical algorithm for solving variational problems*. 2017. arXiv: 1710.00211 [cs.LG].

[8] Mario Geiger et al. "Scaling description of generalization with number of parameters in deep learning". In: *Journal of Statistical Mechanics: Theory and Experiment* 2020.2 (Feb. 2020), p. 23401. DOI: 10.1088/1742-5468/ab633c. URL: https://doi.org/10.1088/1742-5468/ab633c.

[9] *Google Colaboratory*. URL: https://colab.research.google.com/.

[10] Sergei Gukov et al. "Learning to unknot". In: *Machine Learning: Science and Technology* 2.2 (Apr. 2021), p. 25035. DOI: 10.1088/2632-2153/abe91f. URL: https://doi.org/10.1088/2632-2153/abe91f.

[11] Ouns El Harzli, Guillermo Valle-Pérez, and Ard A. Louis. *Double-descent curves in neural networks: a new perspective using Gaussian processes*. 2021. arXiv: 2102.07238 [stat.ML].

[12] Trevor Hastie et al. *Surprises in High-Dimensional Ridgeless Least Squares Interpolation*. 2020. arXiv: 1903.08560 [math.ST].

[13] Vishnu Jejjala, Damian Kaloni Mayorga Pena, and Challenger Mishra. *Neural Network Approximations for Calabi-Yau Metrics*. 2021. arXiv: 2012.15821 [hep-th].

[14] Xiaowei Jin et al. "NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations". In: *Journal of Computational Physics* 426 (Feb. 2021), p. 109951. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2020.109951. URL: http://dx.doi.org/10.1016/j.jcp.2020.109951.

[15] George Em Karniadakis et al. "Physics-informed machine learning". In: *Nature Reviews Physics* 3.6 (2021), pp. 422–440.

[16] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[17] Ganesh Kini and Christos Thrampoulidis. *Analytic Study of Double Descent in Binary Classification: The Impact of Loss*. 2020. arXiv: 2001.11572 [stat.ML].

[18] Nikola Kovachki, Samuel Lanthaler, and Siddhartha Mishra. "On universal approximation and error bounds for Fourier Neural Operators". In: *arXiv preprint arXiv:2107.07562* (2021).

[19] Nikola Kovachki et al. "Neural operator: Learning maps between function spaces". In: *arXiv preprint arXiv:2108.08481* (2021).

[20] Samuel Lanthaler, Siddhartha Mishra, and George Em Karniadakis. "Error estimates for deeponets: A deep learning framework in infinite dimensions". In: *arXiv preprint arXiv:2102.09618* (2021).

[21] Zongyi Li et al. *Fourier Neural Operator for Parametric Partial Differential Equations*. 2021. arXiv: 2010.08895 [cs.LG].

[22] Lu Lu, Pengzhan Jin, and George Em Karniadakis. *DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators*. 2020. arXiv: 1910.03193 [cs.LG].

[23] Song Mei and Andrea Montanari. *The generalization error of random features regression: Precise asymptotics and double descent curve*. 2020. arXiv: 1908.05355 [math.ST].

[24] Andrea Montanari et al. *The generalization error of max-margin linear classifiers: High-dimensional asymptotics in the overparametrized regime*. 2020. arXiv: 1911.01544 [math.ST].

[25] Ben Moseley. *So, what is a physics-informed neural network?* Aug. 2021. URL: https://benmoseley.blog/my-research/so-what-is-a-physics-informed-neural-network/.

[26] Preetum Nakkiran. *More Data Can Hurt for Linear Regression: Sample-wise Double Descent*. 2019. arXiv: 1912.07242 [stat.ML].

[27] Preetum Nakkiran et al. *Deep Double Descent: Where Bigger Models and More Data Hurt*. 2019. arXiv: 1912.02292 [cs.LG].

[28] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (2019), pp. 686–707.

[29] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. "Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations". In: *arXiv preprint arXiv:1711.10561* (2017).

[30] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. "Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations". In: *arXiv preprint arXiv:1711.10566* (2017).

[31] S Spigler et al. "A jamming transition from under- to over-parametrization affects generalization in deep learning". In: *Journal of Physics A: Mathematical and Theoretical* 52.47 (Oct. 2019), p. 474001. DOI: 10.1088/1751-8121/ab4c8b. URL: https://doi.org/10.1088/1751-8121/ab4c8b.

[32] Sifan Wang, Hanwen Wang, and Paris Perdikaris. "Learning the solution operator of parametric partial differential equations with physics-informed DeepOnets". In: *arXiv preprint arXiv:2103.10974* (2021).

[33] E Weinan and Bing Yu. "The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems". In: *Communications in Mathematics and Statistics* 6.1 (2018).

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] (Please refer to Sec. 4)
   (b) Did you describe the limitations of your work? [Yes] (Please refer to Sec. 5)
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] (Please refer to Sec. 6)
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] (Please refer to Sec. 2.3)
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] (Please refer to Sec. 3)
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] (Please refer to Sec. 2.3)
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] (Please refer to Sec. 2.3)
   (b) Did you mention the license of the assets? [Yes] (Please refer to Sec. 2.3)
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] (Please refer to Sec. 2.3)
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]