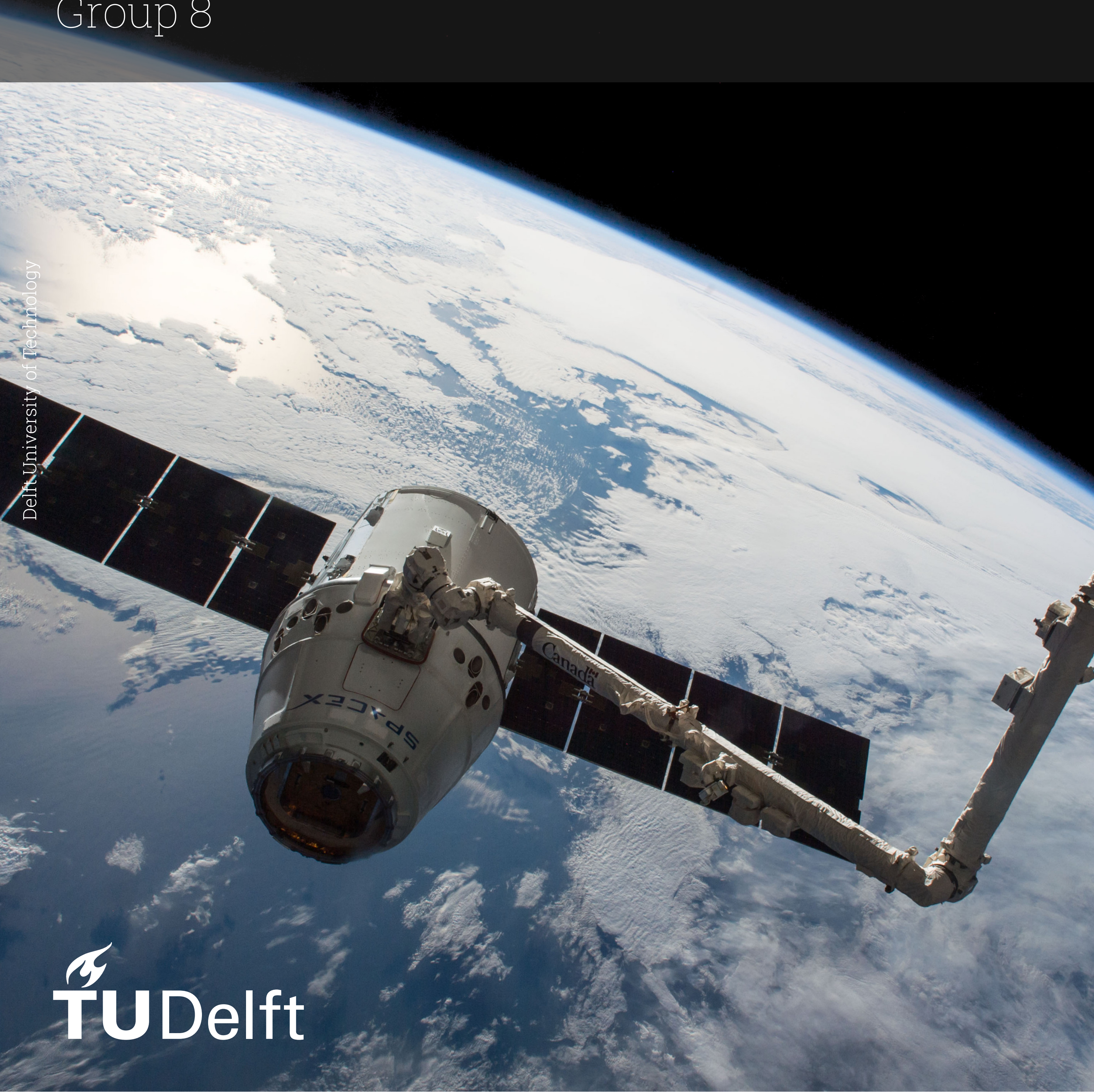
Sampling strategies for DeepONets

WI4450: Special Topics in Computational Sciences and Engineering

Group 8



Delft University of Technology

TUDelft

Sampling strategies for DeepONets

by

Group 8

Laurits Fog Balstrup - 6359329	
Markus Sandnes - 6358810	

Instructor:	A. Heinlein
Project Duration:	April, 2025 - June, 2025
Faculty:	Electrical Engineering, Mathematics and Computer Science, Delft

Cover:	Canadarm 2 Robotic Arm Grapples SpaceX Dragon by NASA under CC BY-NC 2.0 (Modified)
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

Contents

0.1. Introduction

Neural networks have been shown to have universal approximation capabilities. A practical problem when applying them on real world problems is however that it requires many resources to train. With a problem with variable input parameters such as a forcing term in a differential equation, classical physics informed neural networks (PINN's) may therefore not be feasible at all, since they would have to be trained for each new forcing term. This is where the DeepONet (DON) comes in. Instead of approximating the solution function directly, they approximate the operator that takes an input function, e.g. a forcing term, and gives the resulting solution. In this way, one only has to train it once on a variety of input functions, after which it can be applied on all input function of a similar variety. The question remains how well a DON generalized beyond the types of input functions it is trained on. How do we choose which input functions to train the DON with, depending on where we want the DON to be accurate? Does the DON actually learn the physics of the problem in such a way that it can produce a solution from an input function that is far from something it has ever seen before?

0.2. Litterature review

The first idea reminiscent of the modern DON comes from [4], who prove universal approximation capabilities of the one-layer DON.

The idea of the DON in it's form in this report stems from [2]. They propose the architecture with the branch and trunk net and show it's effectiveness over other similar architectures. They also investigate the effect of the complexity of the input function space and how this relates to the number of input sensor points. They show, among other things, that there seems to be a critical point depending on the complexity of the input function space, where adding more input sensor points does not increase performance.

In [3], the authors do a thorough investigation of a sampling strategy, where the training data is sampled in batches at each training iteration. They show that batch sampling decreases training time depending on the size of the batches and that one can achieve similar if not even better accuracy.

What remains to be seen is how these tings relate to the performance of the DON on input functions completely outside the domain in which it has been trained. This is what we refer to as the generalizability of the DON. In what way does the DON actually 'learn' the physics of the system?

0.3. Research question

Our overall question is the following:

How is the accuracy of the model on input functions within the training domain vs outside the training domain? I.e. how generalizable is the model?

To investigate this, we consider the following sub-questions:

1. How does input function distribution influence accuracy of models?
2. How does the accuracy depend of the number of sensor points? How does this relate to the complexity of the input functions used in training vs testing?
3. Can we improve the accuracy and/or generalizability of the model by sampling the training data at each iteration during the training process?

We firstly investigate the three sub-questions separately on a very simple 1D problem. Then we try to combine the results from the simple test problem on a more complex problem; a 1D time-dependent convection diffusion equation (CDE).

0.4. Methology

DeepOnet Architecture

Consider a general differential equation

$$\begin{aligned} Lu &= f && \text{in } \Omega \\ u &= g && \text{on } \overline{\Omega} \end{aligned} \tag{1}$$

for some differential operator $L : C^k(\Omega) \mapsto C(\Omega)$. With a DeepONet (DON), the aim is to estimate an operator $G : C(\Omega) \mapsto C^k(\Omega)$ which given f estimates u, i.e. $G(f) = u$. One might think of this as an inverse of L that also seeks to satisfy boundary conditions.

The DON has two components: A branch net and a trunk net. The branch net takes as input the forcing f. This can either be in the form of a parameter which determines the shape of f, or as is more commonly the case, f evaluated in some points $y = (y_1, y_2, ..., y_m) \in \Omega$. We call these points input sensor points and the forcing f an input function. The trunk net takes as input a set of points $x = (x_1, x_2, ..., x_n) \in \Omega$ in which we wish to estimate the value of the solution u, i.e. $G(f)(x)$. We call these the output points. Figure 1 illustrates the structure of the DON. In practice the DON only takes one output point x at a time and gives $G(f)(x)$. That is, the trunk net has the number of input neurons equal to the dimension of an output point x while the branch net has the number of input neurons equal to the number of input sensors m. In this way there is also total flexibility with the output points when training and using the DON; one can train the DON and estimate the resulting solutions $G(f) \approx u$ in any set of points.

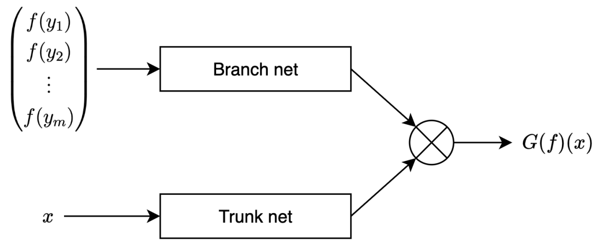


Figure 1: Illustration of the structure of a DeepONet. The output $G(f)(x)$ is an approximation of $u(x)$.

More formally, let $\theta = (\theta_B, \theta_T)$ denote the parameters of the DON. The output of the DON is then computed as an inner product between the branch network B_{θ_B} and the trunk network T_{θ_T}

$$G(f)(x) = \sum_{i=1}^{p} (B_{\theta_B}(f))_i \cdot (T_{\theta_T}(x))_i \tag{2}$$

where p denotes the number of output neurons in the neural networks that make up the branch and trunk nets respectively.

With the simple 1D problem, we have the fixed configuration [1,50,50,10] which means that we have 1 input neuron, $p = 10$ output neurons and 50 neurons in each of the two hidden layers. With the CDE, we have the fixed configuration [1,64,64,10]. We choose some slightly arbitrary learning rate schedules for each problem which are then kept fixed. These can be found in the supplied code.

To train the DON, we generate N triplets of data $(f, x, u(x))_i = (f_i, x_i, u_i(x_i))$ where u_i is the solution to (1) with the specific input function f_i. Multiple data points may be from the same f_i and resulting u_i evaluated in different output points x_i. With these triplets of data, we can construct a loss function as e.g.

$$\begin{aligned} \mathcal{L}(\theta) &= \alpha_{data}\mathcal{L}_{data}(\theta) + \alpha_{phy}\mathcal{L}_{phy}(\theta) \\ \mathcal{L}_{data}(\theta) &= \frac{1}{N}\sum_{i=1}^{N}(G(f_i)(x_i) - u_i(x_i))^2 \\ \mathcal{L}_{phy}(\theta) &= \frac{1}{N}\sum_{i=1}^{N}(L[G(f_i)](x_i) - f_i(x_i))^2 \end{aligned} \tag{3}$$

We use the fact that we can compute derivatives of the DON operating on some input function $G(f)$. The loss function can contain many other terms penalizing error on the boundary or at the initial condition, as we shall see later.

For training the DON's we use the ADAM optimizer presented in the lecture slides.

Data generation

1D test problem

For the 1D problem considered in the first part of the report

$$u_x = \omega\cos(\omega x) = f(x) \qquad \text{for } x \in [-1,1], \tag{4}$$

we have the explicit solution $u(x) = \sin(\omega x)$. In this way, the data points for the loss function

$$(f, x, u(x))_i = (\omega_i \cos(x_i), x_i, \sin(\omega_i x_i)) \tag{5}$$

can be directly computed at each training step.

1D time dependent convection diffusion equation

We want to solve the 1D time-dependent convection diffusion equation (CDE) with inital condition $u_0(x) = \sin(\pi x)$, homogenous boundary conditions and forcing term f

$$\frac{\partial u}{\partial t} + \mu\frac{\partial u}{\partial x} - \epsilon\frac{\partial^2 u}{\partial x^2} = f, \quad u(x,0) = \sin(\pi x), \quad u(0,t) = u(1,t) = 0, \quad x,t \in [0,1]. \tag{6}$$

with parameters $\mu = 1$ and $\epsilon = 0.1$ for all cases. The data used for training the DeepONet on the CDE is generated from an implicit finite element solver. We have implemented this solver ourselves which can be found in the Gitlab repository of Group 8. Specifically, we choose finite element discretizations of quadratic basis functions with continuous derivatives in space. For the time integration we use the Crank-Nicholson approach. This discretization approach is presented and discussed in [1]. For the right hand side we use a random linear combination of sinusoidal Fourier basis functions with coefficients $|a_n| \leq 1$ up to degree 4

$$f \in \Omega_4 := \{f(x) = \sum_{n=1}^{4} a_n \cdot \sin(n\pi x), \quad |a_n| \leq 1\}. \tag{7}$$

We solve equation 6 for N different right-hand sides f defined in equation 7. Each solution u is sampled at P spacio-temporal points, yielding P data triplets $(x, t, u(x,t))$ per solution. The spacio-temporal points in which we sample each u is different for each input function. In this way we obtain information about the problem all over the domain without having to sample an infeasible amount of points. We are thus left with a dataset of $N \times P$ data triplets. In this way, the same input function f appears in multiple data triplets

$$(f, (x,t), u(x,t))_{i,j} = (f_i, (x_{ij}, t_{ij}), u_i(x_{ij}, t_{ij})) \quad \text{for } i = 1, ..., N, j = 1, ..., P. \tag{8}$$

For our training set we choose $N, P = 100$ which means we are left with a dataset of 10.000 data triplets. The dataset generation for this problem is proposed in [2].

Batch sampling

Inspired by [3] we now consider sampling the training data at each iteration of the training process. When training, the loss function in (10) is computed form all the data points $(f, x, u(x))_i, i = 1, ..., N$ at each training step. We now instead sample a random batch of $M < N$ data points uniformly from the training data set without replacement such that

$$\{(f, x, u(x))_i\}_{i=1}^{N} \mapsto \{(f, x, u(x))_i\}_{i=1}^{M}. \tag{9}$$

Letting $0 < b_r < 1$ denote the batch ratio, we have $M = \lfloor b_r N \rfloor$.

The idea behind this batch sampling strategy is two-fold:

Firstly, the training process becomes faster, since the effective batch size becomes smaller. This depends on the size of the batches, the smaller they are, the bigger the reduction in resource consumption.

Secondly, the optimization process becomes stochastic. The hope is that this can improve the performance of the ADAM optimizer by decreasing the probability of the optimizer getting stuck in local minima and also increase the speed with which it descends in the loss landscape in general. It may also decrease the probability of the DON overfitting to the training data, thus increasing the generalizability to input functions outside the training space Ω.

Training 1D case

The equation (1) has the explicit solution $u(x) = \sin(\omega x)$. The loss function consists simply of a data loss term and a physics loss term weighted equally

$$\begin{aligned} \mathcal{L}(\theta) &= \mathcal{L}_{data}(\theta) + \mathcal{L}_{phy}(\theta) \\ \mathcal{L}_{data}(\theta) &= \frac{1}{N}\sum_{i=1}^{N}(G(f_i)(x_i) - \sin(\omega_i x_i))^2 \\ \mathcal{L}_{phy}(\theta) &= \frac{1}{N}\sum_{i=1}^{N}(\partial_x G(f_i)(x_i) - \omega_i \cos(\omega_i x_i))^2. \end{aligned} \tag{10}$$

Training 1D time dependent convention diffusion equation

For the CDE, we use a loss function with a balance between data loss, physics loss, boundary loss and initial loss. The data loss function $\mathcal{L}_{data}$ measures the error between the DON and finite element solver described earlier. The physics loss function $\mathcal{L}_{physics}$ measures the residual, i.e. deviance between the partial differential equation and forcing term. The boundary loss function $\mathcal{L}_{boundary}$ measures the error on the boundary. Lastly, the initial loss function $\mathcal{L}_{inital}$ measures the error for the initial condition, $u_0(x)$.

$$\begin{aligned} \mathcal{L}_{combined} &= \mathcal{L}_{data} + 10 \cdot (\mathcal{L}_{physics} + \mathcal{L}_{initial} + \mathcal{L}_{boundary}) \\ \mathcal{L}_{data}(\theta) &= \frac{1}{NP}\sum_{i,j=1}^{N,P}\left(G(f_i)(x_{ij}, t_{ij}) - u_i(x_{ij}, t_{ij})\right)^2 \\ \mathcal{L}_{physics}(\theta) &= \frac{1}{NP}\sum_{i,j=1}^{N,P}\left(\partial_t G(f_i)(x_{ij}, t_{ij}) + \mu\partial_x G(f_i)(x_{ij}, t_{ij}) - \epsilon\partial_{xx} G(f_i)(x_{ij}, t_{ij}) - f_i(x_{ij}, t_{ij})\right)^2 \\ \mathcal{L}_{boundary}(\theta) &= \frac{1}{NP}\sum_{i,j=1}^{N,P}\left((G(f_i)(0, t_{ij}) - 0)^2 + (G(f_i)(1, t_{ij}) - 0)^2\right) \\ \mathcal{L}_{initial}(\theta) &= \frac{1}{NP}\sum_{i,j=1}^{N,P}\left(G(f_i)(x_{ij}, 0) - u_0(x)\right)^2 \end{aligned}$$

The reasoning behind the weighting of the physics, initial and boundary loss is to force the model to trust the physics more than the data, which we hope will result in a more generalizable solver since the DON might learn the physics of the problem to a higher degree.

0.5. Numerical results

Parameter distribution 1D case

We want to investigate how the distribution of the input functions influence the accuracy of the DON. To this end, consider the equation

$$u_x = \omega\cos(\omega x) = f(x) \qquad \text{for } x \in [-1, 1] \tag{11}$$

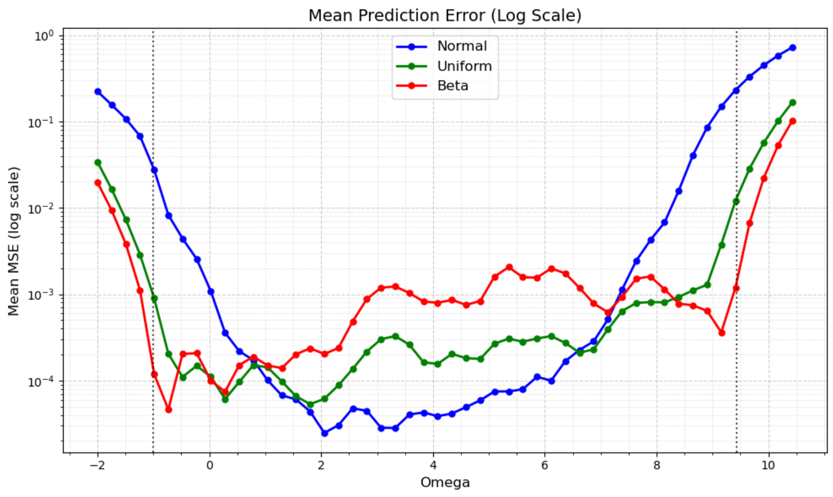
where $\omega \in [a, b]$. The input function is given to the branch net simply as ω and the spacial input to the trunk net is x. Define the input function space for training and test respectively as

$$\begin{aligned} \Omega &:= \{f(x) = \omega\cos(\omega x), \quad \omega \in [a, b]\}, \\ \Omega_{test} &:= \{f(x) = \omega\cos(\omega x), \quad \omega \in [a - \delta, b + \delta]\}. \end{aligned} \tag{12}$$

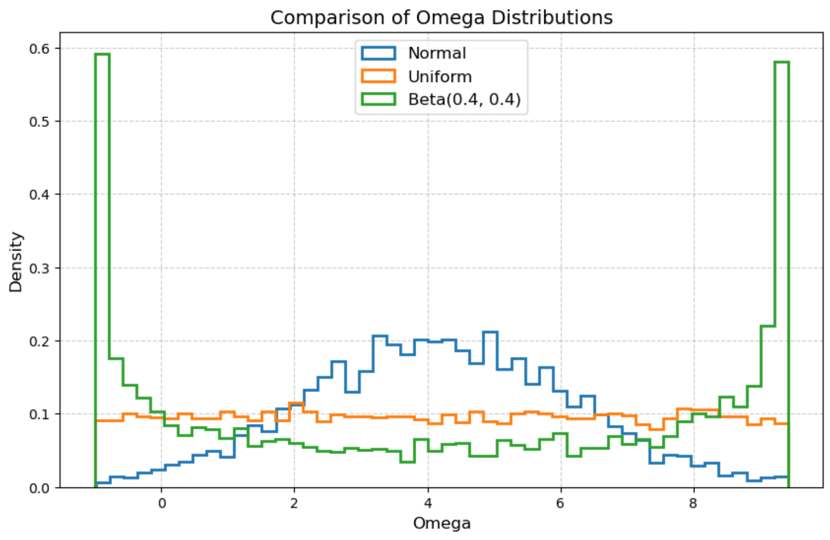
We test the accuracy of the DON on the test input function space Ω_{test} for different sampling distributions:

1. Uniform sampling of ω in $[a, b]$.
2. Normally distributed sampling restricted to ω in $[a, b]$ by cutting off tails.
3. Beta-distribution of ω in $[a, b]$ with $\alpha = \beta = 0.4$.

We compute the MSE for a number of input functions $f \in \Omega_{test}$ where $a = -1, b = 3\pi$ and $\delta = 1$. To reduce the noise in the results, we train and test 5 times and take to mean. Figure 2a and 2b illustrates the results. As expected, we observe the accuracy being somewhat inversely proportional to the input function distribution. I.e. the DON is more accurate for the types of input functions it has been more of. This effect also extends outside the input function training space $\Omega_{test} \setminus \Omega$. As such, the DON trained on beta distributed ω generalizes better outside the domain. If performs worse in the center of the domain compared to the other models however.



(a) Mean MSE of the DON trained on different ω distributions.



(b) Histogram of sampled ω values for each distribution.

Figure 2: Analysis of DON performance and input distributions.

Number of input sensors 1D case

We now investigate the effect of the number of input sensors for the problem (11). In this case, the input function is completely determined by a single variable, namely ω. But now we input the input function f to the branch net in the form of sensor point values, i.e. $(f(x_1), f(x_2), ..., f(x_m))$. We once again train the DON's on input functions $f \in \Omega$ where $a = -1, b = 3\pi$ and test on $f \in \Omega_{test}$ where $\delta = 1$. The training and testing is performed 5 times to estimate the mean MSE. The result is illustrated in figure 4.

We observe the improvement on the accuracy of the models leveling out already at $m = 2$. On input functions within the training space Ω there is as such no improvement with increasing the number of input sensor to more than 2. To explain this, consider the system of equations

$$\begin{aligned} f(x_1) &= \omega \cos(\omega x_1) \\ f(x_2) &= \omega \cos(\omega x_2). \end{aligned} \tag{13}$$

Given $f(x_1), f(x_2)$, can we estimate ω uniquely? With the chosen $[a, b] = [-1, 3\pi]$, we have not restricted ourselves to the case where $\omega \cos \omega x$ is e.g. monotonically increasing for $x \in [-1, 1]$. This would otherwise make an analytical proof easier. We therefore consider the residual

$$R(\omega) = (\omega_{true} \cos(\omega_{true} x_1) - \omega \cos(\omega x_1))^2 + (\omega_{true} \cos(\omega_{true} x_2) - \omega \cos(\omega x_2))^2 \tag{14}$$

and plot for different choices of x_1, x_2. Figure 3 shows the result.

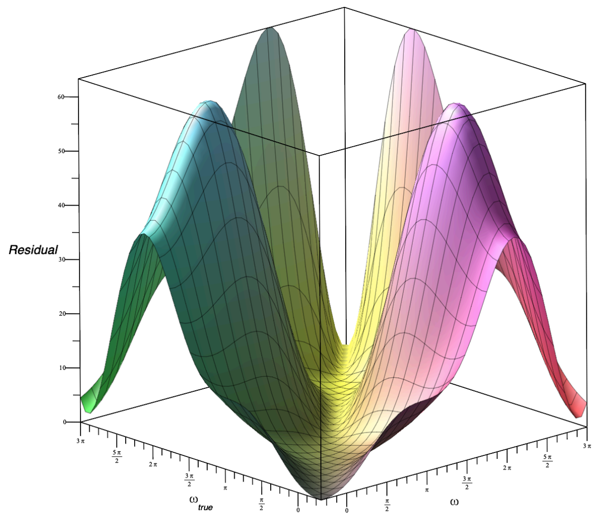


Figure 3: Plotting $R(\omega)$ for $x_1 = 0.5, x_2 = 0.2$. We see that $R(\omega) = 0$ for only one ω for each ω_{true}.

We observe that for almost all pairs x_1, x_2, the ω is in fact uniquely determined since $R(\omega) = 0 \Rightarrow \omega = \omega_{true}$. But if e.g. $x_1 = -x_2$, this is not the case. This is the reason for the choice $x_1 = -0.7, x_2 = 0.9$ in figure 4. If $x_1 = -x_2$, then the DON does not train properly at all. This explains why the DON should in theory be able to uniquely determine ω with a proper choice of sensor point locations x_1, x_2, and hence not show improvement with increasing the number of sensor points.

Interestingly however, there is a difference in accuracy when we look outside the training space $\Omega_{test} \backslash \Omega$. For ω outside $[a, b]$, we see a pretty clear improvement of having 3 or more sensor points. The difference between having 3 and 4 sensor points is practically zero however. Going above the threshold of 2 sensor points where the increase in performance within the input function training space levels out may as such actually increase the generalizability of the DON.

Batch sampling 1D case

We now try training the DON's using the batch sampling strategy as described in the Data generation section.

Table 1 shows the average training times with the different batch ratios. These times obviously depend on the implementation of the batch sampling. In this current setting, we actually see the batch sampling

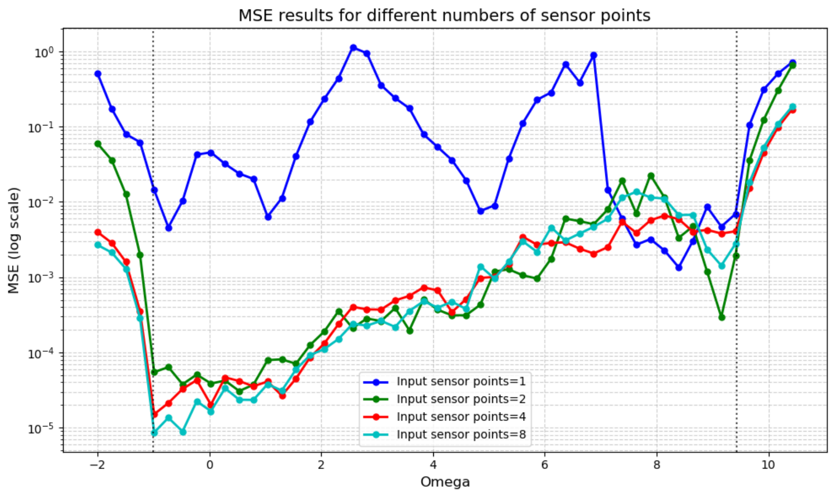


Figure 4: Mean MSE with different numbers of sensor points $f(x_i), i = 1, ..., m$. We chose here $x_1 = -0.7, x_2 = 0.9$.

performing worse wrt. time for $b_r \geq 0.5$. When using batch sampling in the 2D problem later in the report, the batch sampling does train faster for $b_r = 0.5$

Batch ratio	Baseline	0.01	0.1	0.2	0.5	0.8	0.9	0.95	0.99
Time (s)	36.74	24.34	27.26	29.66	43.18	47.63	54.93	54.62	55.88

Table 1: Training Time vs. Batch Ratio

Figure 5 shows the loss function of using the regular training vs using batch sampling. This is to show the increased stochasticity of the training process. One obviously has to reconsider the learning rate such that the training process does not become too stochastic.

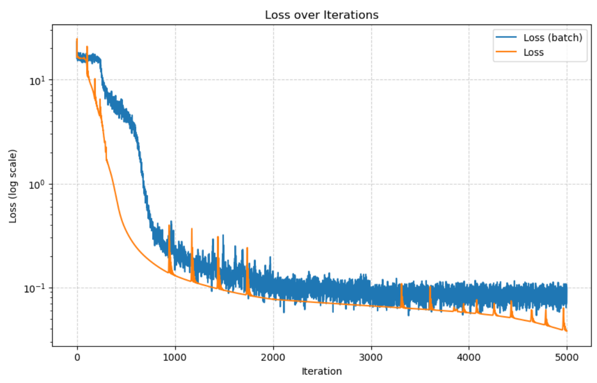


Figure 5: Loss function vs training iteration with regular training and batch sampling.

The choice of the batch ratio b_r is also crucial. Figure 6 shows the training and test MSE of DON's trained with and without batch sampling. For each batch ratio, the models are trained and tested 10 times and the means are then computed and plotted. We generally observe the batch sampling strategy performing worse both wrt. training and test MSE. We suspect that the current simple problem (11) with loss function (10) has such a simple loss-landscape that the regular training procedure easily finds a global minimum. Within the training space where $\omega \in [a, b]$, the batch sampling strategy generally improves the larger the batch ratio. The improvement is very steep from $b_r = 0.01$ to $b_r = 0.1$ whereafter it flattens out significantly. At $b_r = 0.5$ the improvement in training MSE with using larger batch ratio's is so small, that one $b_r = .5$ might be optimal. In the test space $\Omega_{test} \setminus \Omega$, the test MSE is almost constant from $b_r = 0.1$ compared to the baseline.

The benefit of the batch sampling may mostly be in terms of training time in this case. Later in the report we consider a more complicated problem where batch sampling may be more beneficial.

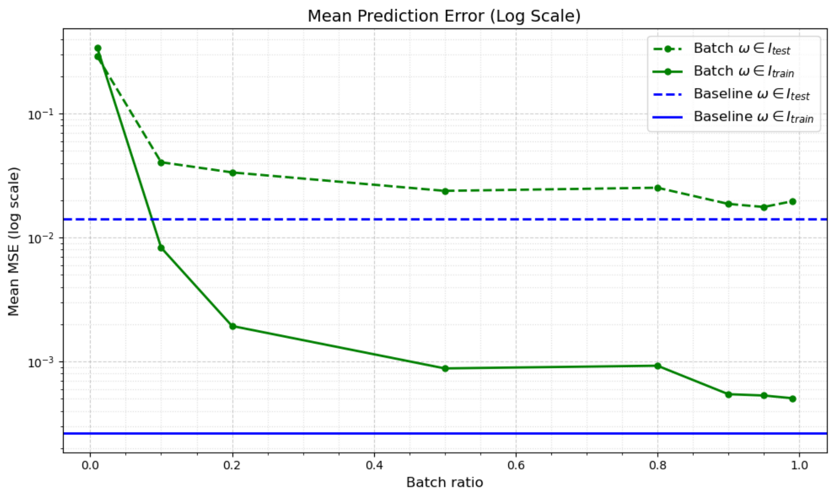


Figure 6: Train MSE of DON on input functions within the training domain Ω where $\omega \in [a, b] = I_{train}$ and of DON on input functions outside training domain $\Omega_{train} \setminus \Omega$ where $\omega \in [a - \delta, a] \cup [b, b + \delta] = I_{test}$.

Batch sampling 2D case

For the CDE we first investigate how batch sampling impacts accuracy. We test batch sizes of 20 %, 50% and 100% of the training data. The resulting DON's are tested on the input functions space used for training, recall (7), and also a larger one

$$\Omega_4 := \{f(x) = \sum_{n=1}^{4} a_n \cdot \sin(n\pi x), \quad |a_n| \leq 1\}$$
$$\Omega_{test} := \{f(x) = \sum_{n=1}^{4} a_n \cdot \sin(n\pi x), \quad |a_n| \leq 1.5\}$$

to measure their generalizability.

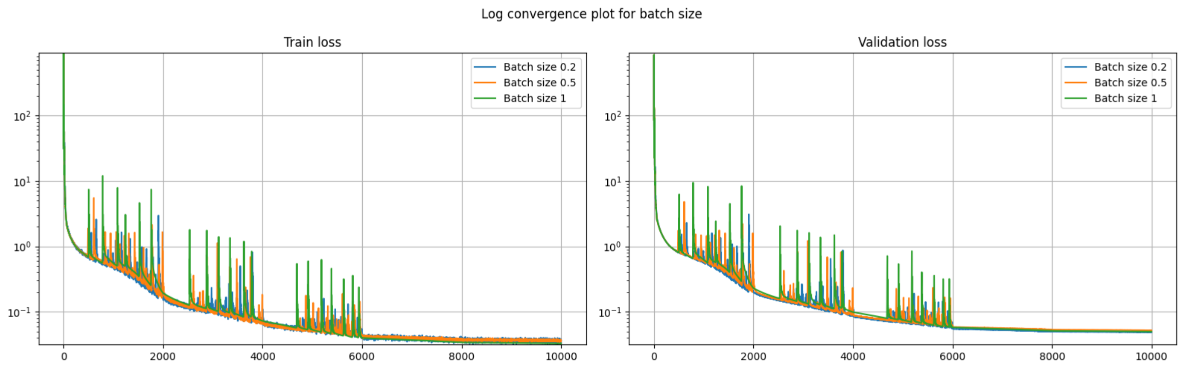


Figure 7: Train and validation plot for 20%, 50% and 100% batch sizes

Batch size	Training time	Test loss ($\lvert a_n \rvert \leq 1$)	Test loss ($\lvert a_n \rvert \leq 1.5$)
20%	$\approx 25min$	7.33 10^{-4}	5.21 10^{-3}
50%	$\approx 43min$	8.41 10^{-4}	5.02 10^{-3}
100%	$\approx 54min$	1.07 10^{-3}	5.19 10^{-3}

Table 2: Table containing training time and test loss for different batch sizes with forcing term $f(x) := \sum_{n=1}^{5} a_n \cdot \sin(n\pi x)$

Figure 7 shows the training loss and validation loss vs iterations with and without batch sampling. The hypothesis that batch sampling gives a more generalizable DON does not look to hold; they all have practically the same accuracy both tested on Ω_4 and Ω_{test}. One would at least have to perform more trainings and compute e.g. mean test and training looss, but due to time and computational limitations, we have not been able to do this. We do however see that the convergence is remarkably similar for all batches. From table 2 we see that the training time is significantly faster for smaller batch sizes. These results indicate that batch sampling does not produce a more generalizable DON, but the training time is drastically reduced. Therefore, going forward we will use a batch size of 20% for all training as it saves a lot of computational time and resources.

Number of input sensors 2D case

Now we want to investigate how the number of input sensors impact the accuracy of the DON for the CDE. Recall our forcing term expressed in equation 7. Since this function is a linear combination of 4 orthogonal basis functions we expect based on the results with the simple 1D problem that a sufficient number of input sensors is $m = 5$ and that any more sensors is redundant. The distribution of input sensors may of course be significant. For our experiments the input sensors are equally distributed along the x-axis. In a real world situation the sensor points along the domain of interest are fixed.

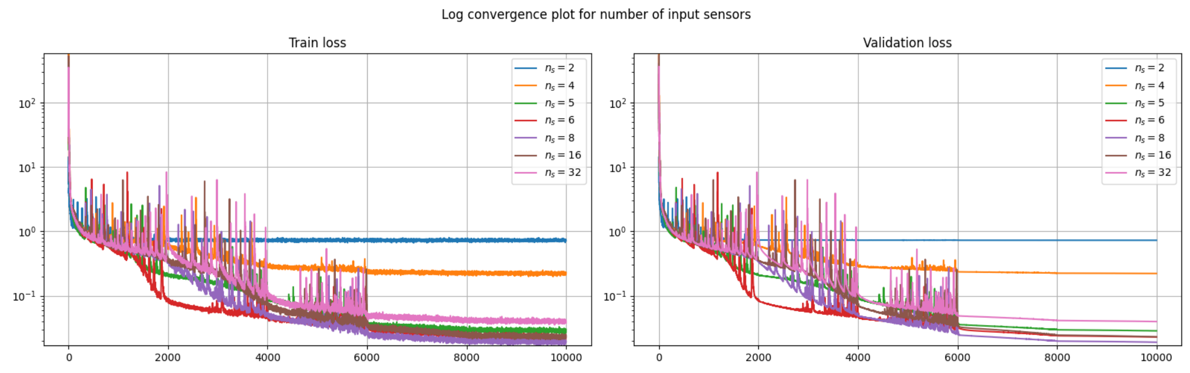


Figure 8: Log convergence plot for number of input sensors = {2,4,5,6,8,16,32}

Figure 8 shows that there is a clear validation loss threshold between $m = 4$ and $m = 5$ input sensors. When taking more than $m = 5$ input sensors, there may only be slight improvements in accuracy. Taking too many may even decrease accuracy, as is seen with $m = 32$ sensor points.

This is also reflected in the test loss in figure 9. In addition to Ω_4, Ω_{test} we now also test on

$$\Omega_5 := \{f(x) = \sum_{n=1}^{5} a_n \cdot \sin(n\pi x), \quad |a_n| \leq 1\}.$$

Here we see that on Ω_4 with test data with the same number of Fourier basis functions as the training data, i.e. 4 in this case, there is a clear threshold at $m = 5$ input sensors. These results strengthen the hypothesis that the number of input sensors over some threshold are redundant. Based on these results and the ones from the simple 1D problem, this threshold seems to be $m = k + 1$, where k is the number of degrees of freedom of the input functions. On Ω_5 where we increase the number of basis functions by one we still see a significant decrease in MSE when going from $m = 5$ to $m = 6$ input sensors. The decrease does not result in quite the same accuracy as testing on Ω_4 with $m = 5$ however. Moreover, if we increase number of input sensors to more than $m = 6$ the MSE significantly increases again. We do not have an explanation for why the accuracy increases briefly but then diminishes with more sensor points when the DON is applied on an input function space Ω_5 where functions may have higher frequency components.

0.6. Discussion and conclusion

It is generally difficult for DON's to generalize beyond the training input function space. In every case, the accuracy quickly diminishes when the DON is given an input function f that does not look similar

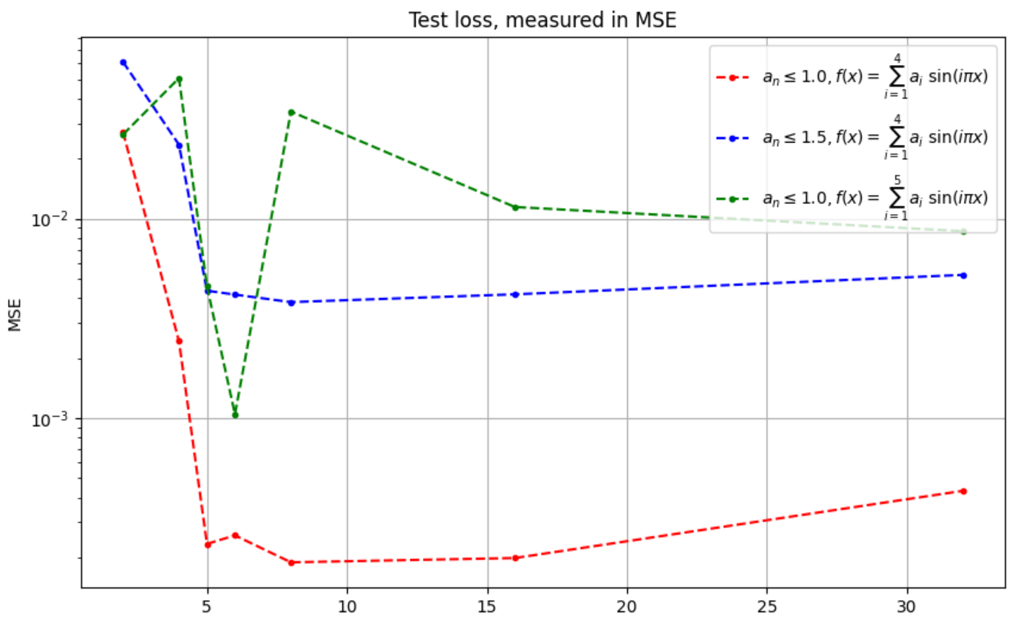


Figure 9: Comparison of test loss for different right hand side generations. All measured as Mean Squared Error

to something it has seen before. In this way, there is a direct relationship between the training input function space and the accuracy of the DON on different types of input function. This also holds within the training input function space; the DON is more accurate on the types of input functions it has more of. One thus has to choose beforehand where one whishes the DON to be most accurate.

There is a strong relationship between the number of input sensor points required for accurate results and the complexity of the input function space. When the number of input sensor points reach a threshold, adding more will not significantly increase accuracy further. This threshold might correspond to the point where an input function can be uniquely determined with the sensor data. There might be benefits of adding more sensor points beyond this threshold when applying the DON outside the training input function space. In the 1D case this is seen in figure 4 for ω outside the training domain whereas for the 2D case in figure 9 this is seen for input function with higher degrees of freedom in Ω_5.

Batch sampling does not seem to improve generalizability. In all cases the accuracy within and beyond the training input function space is similar with and without batch sampling. It does however improve computational efficiency significantly. With this in mind, one could allow the training process to run for more iterations in the same total time. This might give the batch sampling approach the possibility of yielding more accurate models. In our experiments, the models with and without batch sampling where run for the same number of iterations.

To sum up, it seems there are no free lunch. The DON almost only learns what it is given. The accuracy quickly diminishes when moving outside the training domain. The result in figure 9 where the accuracy is surprisingly high for input functions outside the training input function space for a specific number of input sensor points might be the biggest sign of a free lunch.

So to the question of whether the DON actually 'learns' the physics of the problem, the answer is: Only very slightly.

As for possible directions for future work:

In [3] they only sample the spacio-temporal data points whereas we sample the entire data triplets. The difference between these methods could be investigated. Since their approach supplies the DON with all the input functions in the training set at each iteration, their models might learn better. It would also be interesting to try to come up with other types of batch sampling. Since the motivation for the batch sampling partly is to avoid local minima of the loss function, the benefits may be very problem dependent. To properly assess the method, it therefore is crucial to apply it on more complex problems with a possibly more irregular loss-landscape.

The possible free lunch in figure 9 obviously needs an explanation.

The distribution of the input sensors are equally distanced in this report. One could investigate the effect of having different other distributions of input sensor. One must however keep in mind, that in a real world problem, the input sensor locations may be fixed due to other constraints.

References

[1] Abdelhaq Abouhafc. *Finite Element Modelling Of Thermal Processes With Phase Transitions*. 2006. URL: `https://homepage.tudelft.nl/d2b4e/numanal/abouhafc_scriptie.pdf` (visited on 06/06/2025).

[2] Pengzhan Jin Lu Lu and George Em Karniadakis. *DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators*. 2020. URL: `https://arxiv.org/pdf/1910.03193` (visited on 06/06/2025).

[3] Somdatta Goswami Sharmila Karumuria Lori Graham-Bradya. *Efficient Training of Deep Neural Operator Networks via Randomized Sampling*. 2024. URL: `https://arxiv.org/pdf/2409.13280` (visited on 06/10/2025).

[4] Hong Chen Tianping Chen. *Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems*. 1995. URL: `https://ieeexplore.ieee.org/document/392253` (visited on 06/10/2025).