PROBING OPTIMISATION IN PHYSICS-INFORMED NEURAL NETWORKS

Nayara Fonseca *, **Will Trojak** *
IBM Research Europe
Daresbury, WA44AD, United Kingdom
{nayara.fonseca,w.trojak}@ibm.com

Veronica Guidetti *
University of Modena and Reggio Emilia
Department of Physics, Informatics and Mathematics
Via G. Campi 213/a, 41125, Modena, Italy
veronica.guidetti@unimore.it

ABSTRACT

A novel comparison is presented of the effect of optimiser choice on the accuracy of physics-informed neural networks (PINNs). To give insight into why some optimisers are better, a new approach is proposed that tracks the training trajectory curvature and can be evaluated on the fly at a low computational cost. The linear advection equation is studied for several advective velocities, and we show that the optimiser choice substantially impacts PINNs model performance and accuracy. Furthermore, using the curvature measure, we found a negative correlation between the convergence error and the curvature in the optimiser local reference frame. It is concluded that, in this case, larger local curvature values result in better solutions. Consequently, optimisation of PINNs is made more difficult as minima are in highly curved regions.

1 INTRODUCTION

The idea of solving PDE problems using neural networks (NNs) was put forward by Lagaris et al. (1997; 1998); Lagaris et al. (2000) in the second half of the '90s and then revised in 2017 by Raissi et al. (2017a;b) who named the methodology Physics-Informed Neural Networks (PINNs). Relying on the universal approximation theorem of Cybenko (1989); Hornik (1991); Pinkus (1999), PINNs aim to deliver a universal regressor that can represent any bounded continuous function and solve any PDE/ODE problems, having input and output shape as the only limitation. Although the goal of PINNs was to produce a unifying method of solving PDE/ODEs, satisfactory results could not be achieved in a multitude of cases. The recent works of Karniadakis et al. (2021); Hao et al. (2022) provide an overview of the state-of-the-art; furthermore, Cuomo et al. (2022) focus on algorithms and applications and Beck et al. (2020) on theoretical results.

Several studies have analysed the effects of choosing different architectures, loss function formulations, and treatment of domain and collocation points. However, the effect of the optimiser choice on PINN performance needs more attention. Recently, new PINN-specific optimisation methods were developed or applied to improve poor convergence performance. For example, De Luca & Silverstein (2022) propose a *relativistic* optimisation algorithm that introduces chaotic jumps and Davi & Braga-Neto (2022) use a metaheuristic optimisation method, namely, particle swarm optimisation, to eliminate gradient-related problems.

This work aimed to improve the understanding of how PINNs performance is affected by optimiser choice. Specifically, we considered the following algorithms spanning different optimiser categories: gradient descent (GD) without momentum, LBFGS (Dennis & Moré, 1973; Goodfellow et al., 2016) (a second order quasi-Newton method), ADAM (Kingma & Ba, 2014) (an adaptive stochastic GD algorithm), and bouncing Born–Infeld (BBI) (De Luca & Silverstein, 2022). The first three optimisers are widely used in ML optimisation problems, whereas BBI is a recent realisation of a frictionless energy-conserving optimiser. See Appendix A for details.

*Equal contribution.

Moreover, we introduce a new method to study optimisation in neural networks, which provides training trajectory curvature data at low computational cost. Using this approach, we studied the evolution of different optimisers through the network parameter space during training. Further discussion is given in Section 2.1.

Specifically, we apply PINNs to solve the linear advection equation as described in Section 2.2. Linear advection is a simple PDE with a single parameter, i.e., the wave speed β, whose variation can tune the complexity of the PINN landscape (Krishnapriyan et al., 2021). Moreover, to understand how a network adjusts to different depths and widths, we considered two configurations of multi-layer perceptron architectures with different numbers of hidden layers and nodes per layer.

The main contributions of this work are:

- We found that the choice of optimisation algorithm significantly impacts the convergence of PINNs models. Specifically, second-order optimisers that do not directly follow gradient directions, such as LBFGS, performed better, as shown in Fig. 1a.
- We introduce a new low-cost method for studying the dynamics of optimisers. This relies on defining a local reference frame and evaluating the local curvature of the training trajectory in the NN parameter space.
- We found a negative correlation between PINNs convergence error and the newly introduced training trajectory curvature which is shown in Fig. 1b. This implies that good PINNs solutions lie in highly curved regions of the optimiser reference frame. Therefore, making PINNs converge is hard for those techniques designed to explore shallow landscapes that generalise well under perturbation.

2 BACKGROUND

2.1 TRACKING LOCAL CURVATURE

Solely observing the convergence performance of an optimiser will show which optimiser is the best for a given task but will give limited information as to why. To move a step towards explaining why one optimiser is better than another, we analyze the trajectory of every optimiser in their local reference frame.

To begin, let $\boldsymbol{\omega}$ be the neural network parameters, the weight update rule for update k is defined as $\Delta\boldsymbol{\omega}_k \equiv \boldsymbol{\omega}_{k+1} - \boldsymbol{\omega}_k = \mathbf{V}(\boldsymbol{\omega}_k, \boldsymbol{\eta})$, where $\boldsymbol{\eta}$ are the model hyperparameters. Assuming the continuum time limit, the trajectory in the parameter space during training can be described by:

$$\dot{\boldsymbol{\omega}} = \mathbf{V}(\boldsymbol{\omega}, \boldsymbol{\eta}), \tag{1}$$

where the overdot corresponds to a time derivative, i.e. $\dot{\boldsymbol{\omega}} = \mathrm{d}\boldsymbol{\omega}/\mathrm{d}t$. When using gradient descent, Eq. (1) can be interpreted as the equation of motion of a classical particle in a friction-dominated setup and $\mathbf{V}(\boldsymbol{\omega}, \lambda) = -\lambda \frac{1}{n} \sum_i \nabla_{\boldsymbol{\omega}}\ \mathcal{L}(\mathbf{x}^{(i)}, \boldsymbol{\omega})$. Here, λ is the learning rate, n is the number of samples in the training set $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(n)}\}$, and $\mathcal{L}$ is the loss function. For other optimisation algorithms, the definition of $\mathbf{V}$ changes and the ODE in Eq. (1) is no longer obtained via the Euler–Lagrange equations. Therefore, it does not have a dynamic interpretation. However, Eq. (1) can be efficiently used to understand the kinematics of the optimiser trajectories. This aspect is discussed further in Section 3.

To examine the motion in parameter space described by Eq. (1), we introduce the unit vector tangential to the training trajectory, defined as

$$\hat{\mathbf{T}} = \frac{\dot{\boldsymbol{\omega}}}{\|\dot{\boldsymbol{\omega}}\|_2}, \quad \text{with} \quad \|\dot{\boldsymbol{\omega}}\|_2 = \sqrt{\langle \dot{\boldsymbol{\omega}}, \dot{\boldsymbol{\omega}} \rangle}. \tag{2}$$

Here, $\hat{\mathbf{T}}$ is a time-dependent vector in a $N_{\boldsymbol{w}}$-dimensional space, for a $N_{\boldsymbol{w}}$ parameter NN. To track the local training trajectory curvature in this time-dependent reference frame, we can define the local (time-dependent) curvature, κ_t. This is the rate at which the tangent unit-vector changes with respect to time, and is given by:

$$\kappa_t = \left\| \frac{\mathrm{d}\hat{\mathbf{T}}}{\mathrm{d}t} \right\|_2 = \frac{1}{\|\dot{\boldsymbol{\omega}}\|_2} \left[\ddot{\boldsymbol{\omega}} \cdot \ddot{\boldsymbol{\omega}} - \left(\frac{\mathrm{d}\|\dot{\boldsymbol{\omega}}\|_2}{\mathrm{d}t} \right)^2 \right]^{1/2}. \tag{3}$$

Another parametrisation with a clearer geometric meaning is $\kappa_\omega = \left\| \frac{\mathrm{d}\hat{\mathbf{T}}}{\mathrm{d}\omega} \right\|_2 = \kappa_t / \|\dot{\boldsymbol{\omega}}\|_2$. This represents a local geometric quantity, i.e., the local curvature in the parameter space, removing the effects of time and trajectory speed[1]. To calculate the curvature at each training step a first-order approximation is used, details of which are given in Appendix C.

In comparison with a Hessian-based curvature calculation, the method presented here has a significantly lower computational cost as it does not require the Hessian of the loss function. Furthermore, in most cases, an optimiser does not solely follow the gradient loss and it is generally non-trivial to find the function relating the loss, its derivatives, and the trajectory followed by the optimiser. Therefore, although every optimiser is constructed with the aim of sharing its minima with the loss function, analyzing the loss Hessian may not characterise the optimiser trajectories and may not explain why a certain minimum is found. Moreover, different optimisers are based on different assumptions—such as energy conservation, friction-dominated motion, or others—further modifying the loss seen by the optimiser. Fig. 2 in Appendix C gives a pictorial representation of the loss function (or its distorted version seen by the optimiser) and its relationship to the trajectory in the parameter space.

2.2 Linear Advection

The PDE used for the tests in this work was the one-dimensional linear advection equation on the periodic spatial domain $\Omega = [0, 2\pi)$, given by

$$\frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} = 0, \quad \text{for} \quad \mathrm{u} : \mathrm{T} \times \Omega \mapsto \mathbb{R}, \quad \Omega = [0, 2\pi), \mathrm{T} \in [0, 1], \tag{4a}$$

$$u(x, 0) = u_0(x), \tag{4b}$$

$$u(0, t) = u(2\pi, t), \tag{4c}$$

where β is the wave speed and $u_0(x)$ is the initial condition. The exact solution of this system can be straightforwardly found using the superposition of Bloch waves. To find approximate solutions to this system we applied PINNs with loss functions similar to those used by Krishnapriyan et al. (2021), defined as

$$\mathcal{L}(\theta) = \frac{1}{N_u} \sum_{i=1}^{N_u} \left(\hat{u} - u_0^i \right)^2 + \frac{1}{N_f} \sum_{i=1}^{N_f} \lambda_i \left(\frac{\partial \hat{u}}{\partial t} + \beta \frac{\partial \hat{u}}{\partial x} \right)^2 + \frac{1}{N_b} \sum_{i=1}^{N_b} \left(\hat{u}(\theta, 0, t) - \hat{u}(\theta, 2\pi, t) \right)^2 . \tag{5a}$$

Here $\hat{u} = \hat{u}(\theta, x, t)$ is the neural network output, parameterised by θ. The initial condition used through out this work was $u(x, 0) = \sin(x)$ and $u(0, t) = u(2\pi, t)$. For all the experiments we fix $\lambda_i = 1$.

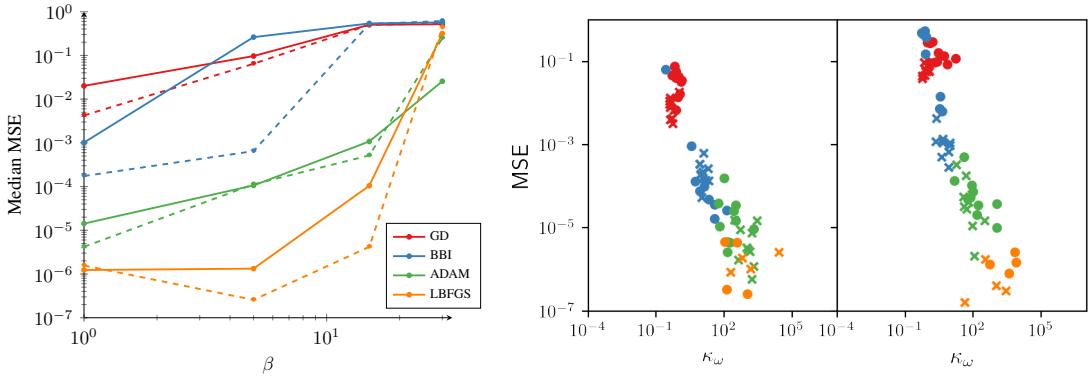
3 Results and Discussion

Fig. 1a shows the median over 10 samples of the mean squared error (MSE) evaluated on the entire domain between the $\hat{u}$ predicted by the PINN and the analytical solution. Here the samples were formed by using 10 different random network initialisation. The corresponding training and test losses are shown in Appendix D.1. During testing, it was observed that the learning rate significantly impacts the final performances; therefore, a grid search was performed over a range of learning rates to estimate the best configuration for each optimiser. See Appendix B for the details.

Two multi-layer perception architectures were considered, with layer structures $S = [2, 25, 25, 1]$ and $L = [2, 50, 50, 50, 50, 1]$ resulting in 751 and 7851 trainable parameters respectively[2]. The number of training points was around 2000, as described in Appendix B; therefore, the larger network is in the over-parameterised regime, i.e. it has more parameters than training data. Fig. 1a shows that the large network

[1] This is the local curvature of the training trajectory, not the curvature of the space of weights, which is flat.

[2] These values keep the perturbative behaviour of the NN constant, i.e. depth/width constant (Roberts et al., 2022).



(a) (b)

Figure 1: (a) Median MSE over 10 samples for various optimisers and β values. *Solid/dashed* lines refer to small/large (S/L) network architectures. (b) Relation between final values of convergence (MSE) and curvature (κ_ω) for $\beta = 1$ (left) and 5 (right). Dots and crosses refer to small (S) and large (L) networks, respectively.

generally has a lower error, in agreement with common knowledge in ML[3]. This effect is most noticeable for BBI [4]. A significant observation is that for all configurations of NN and optimiser, PINNs failed to produce good approximations for systems with large values of β.

In Fig. 1b, we show the inverse relation between the final values of MSE and the curvature κ_ω. Here, only $\beta = 1, 5$ were considered as they converged to reasonable errors for most of the optimisers, see Appendix D.2. Significantly, LBFGS achieves the highest values of κ_ω and the lowest error. Moreover, LBFGS has significant memory overhead but it typically converges in fewer epochs compared to the other optimisers. Note that such a negative correlation in Fig. 1b links generalisation (measured by the MSE on the entire domain) with the curvature κ_ω. Deriving their exact relation is non-trivial and we leave further investigations for future work. Nevertheless, we would like to stress that the high curvature values is not solely associated with orbiting the final local or global minima. In fact, this is due to κ_ω and MSE being negatively correlated throughout the whole trajectory, including its initial stages. We show examples of this behavior in Appendix D.2.

OUTLOOK. In contrast to traditional ML tasks such as image recognition and NLP, ML methods applied to science require more accurate models which, in general, are trained with high-quality data. This suggests that the training process on data from physical phenomena can be fundamentally different from common ML applications. A promising future research direction would be exploring the connection between model generalisation and the flatness of minima for problems requiring high accuracy. For example, the work of Dinh et al. (2017); Huang et al. (2020) discuss this in the context of traditional ML tasks. Finally, in the future, it would be insightful to compare Hessian-based methods such as that of Michaud et al. (2023), with our approach linking accuracy and local trajectory curvature.

[3]From classical statistical learning, one expects that over-parameterised models over-fit. However, there is overwhelming evidence that large models generalise well in several cases of interest, e.g. Szegedy et al. (2015); Huang et al. (2019).

[4]The total phase space volume for BBI can be analytically estimated and is parametrically large as the objective function is close to zero with an exponential dependence on the number of dimensions (see Appendix A.3 in De Luca & Silverstein (2022)).

ACKNOWLEDGMENTS

We thank Eloisa Bentivegna and Imran Nasim for discussions. NF and WT acknowledge the UKRI support through the grant MR/T041862/1. The authors acknowledge the IBM Research Cognitive Computing Cluster service for providing resources that have contributed to the research results reported within this work.

REFERENCES

Foundations of the new field theory. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 144(852):425–451, March 1934. doi: 10.1098/rspa.1934.0059.

Christian Beck, Martin Hutzenthaler, Arnulf Jentzen, and Benno Kuckuck. An overview on deep learning-based approximation methods for partial differential equations. *arXiv preprint arXiv:2012.12348*, 2020.

Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maizar Raissi, and Francesco Piccialli. Scientific machine learning through physics-informed neural networks: Where we are and what's next. *arXiv preprint arXiv:2201.05624*, 2022.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Caio Davi and Ulisses Braga-Neto. Pso-pinn: Physics-informed neural networks trained with particle swarm optimization. *arXiv preprint arXiv:2202.01943*, 2022.

Giuseppe Bruno De Luca and Eva Silverstein. Born-infeld (bi) for ai: energy-conserving descent (ecd) for optimization. In *International Conference on Machine Learning*, pp. 4918–4936. PMLR, 2022.

John E. Dennis and Jorge J. Moré. A characterization of superlinear convergence and its application to quasi-newton methods. *Mathematics of Computation*, 28:549–560, 1973.

Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In *International Conference on Machine Learning*, pp. 1019–1028. PMLR, 2017.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Zhongkai Hao, Songming Liu, Yichi Zhang, Chengyang Ying, Yao Feng, Hang Su, and Jun Zhu. Physics-informed machine learning: A survey on problems, methods and applications. *arXiv preprint arXiv:2211.08064*, 2022.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(91)90009-T. URL `https://www.sciencedirect.com/science/article/pii/089360809190009T`.

W Ronny Huang, Zeyad Ali Sami Emam, Micah Goldblum, Liam H Fowl, Justin K Terry, Furong Huang, and Tom Goldstein. Understanding generalization through visualizations. In *"I Can't Believe It's Not Better!" NeurIPS 2020 workshop*, 2020.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34:26548–26560, 2021.

I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural network methods in quantum mechanics. *Comput. Phys. Commun.*, 104:1–14, 1997. doi: 10.1016/S0010-4655(97)00054-4.

I. E. Lagaris, A. C. Likas, and D. G. Papageorgiou. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11(5):1041–1049, 2000. doi: 10.1109/72.870037.

I.E. Lagaris, A. Likas, and D.I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998. ISSN 1045-9227. doi: 10.1109/72.712178. URL `http://dx.doi.org/10.1109/72.712178`.

Eric J. Michaud, Ziming Liu, and Max Tegmark. Precision machine learning. *Entropy*, 25(1):175, January 2023. doi: 10.3390/e25010175.

Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143–195, 1999. doi: 10.1017/S0962492900002919.

Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, 2017a.

Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations, 2017b.

Daniel A. Roberts, Sho Yaida, and Boris Hanin. *The Principles of Deep Learning Theory*. Cambridge University Press, May 2022. doi: 10.1017/9781009023405.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2015. doi: 10.1109/cvpr.2015.7298594.

A BOUNCING BORN–INFELD (BBI) OPTIMISER

Usual machine learning optimisation algorithms can be naturally described in physics terms as a particle moving down an irregular hill. Stochastic gradient descent with momentum is a standard example, as it can be viewed as a noisy and discretised version of a particle motion. Crucially, this is a friction-based evolution, such that the particle stops when there is insufficient kinetic energy to escape a minimum in the potential. De Luca & Silverstein (2022) proposed an energy-conserving algorithm in which there is no friction and where the optimisation process slows down near the minima as this region dominates the phase space volume of the system. The algorithm of De Luca & Silverstein (2022) is based on the relativistic Born–Infeld dynamics (Bor, 1934), where the total potential energy (V) depends on the speed limit as $V = v_{\text{rel}}^2$, such that as $V \to 0$ the particle stops. For completeness, we summarise below BBI update rules (De Luca & Silverstein, 2022):

$$\mathbf{\Pi}_{i+1} = \mathbf{\Pi}_i - \frac{1}{2}\boldsymbol{\nabla} V_i \Delta t \left(\frac{V_i}{E} + \frac{E}{V_i} \right), \tag{6a}$$

$$\mathbf{\Theta}_{i+1} = \mathbf{\Theta}_i + \mathbf{\Pi}_{i+1} \Delta t \frac{V_i}{E}, \tag{6b}$$

with the nomenclature

$\mathbf{\Theta}_i$ parameter vector with components θ_i

$\boldsymbol{\Pi}_i$ momentum vector with components π_i

i optimisation step number

Δt optimisation step size (learning rate)

$V_i = V(\boldsymbol{\Theta}_i)$ the potential of the i-th step

$E = V_0 + \delta E$ constant dependent on the initialisation, and the additional initial energy parameter δE

$\boldsymbol{\Pi}_0$ the initial momentum, set as $-\frac{\boldsymbol{\nabla} V(\boldsymbol{\Theta}_0)}{|\boldsymbol{\nabla} V(\boldsymbol{\Theta}_0)|}\sqrt{\frac{E^2}{V_0} - V_0}$.

As E is constant, a particle can be trapped in long-lived orbits in motion. To avoid such stable orbits and boost chaotic mixing, random bounces are introduced by generating a new random momentum vectors with the same absolute momentum. There are three additional hyperparameters controlling bouncing: number of bounces (N_b), fixed timesteps for bounces (T_0), and progress-dependent timesteps for bounces (T_1). Additionally momentum can be re-scaled to conserve energy lost due to discretisation effects.

B Experimental Details

B.1 PINN training setup

Networks. We considered both 2-hidden-layer and 4-hidden-layer fully-connect networks with 25 and 50 nodes per layer, respectively. We used hyperbolic tangents as activation functions in order to have a proper comparison with Krishnapriyan et al. (2021).

Data. The training and test data are obtained by randomly sampling (x, t) points on the domain $\Omega = [0, 2\pi)$ and $T \in [0, 1]$ using a grid of side $n_x = 256$ and $n_t = 100$. We used $N_u = 100$ (initial conditions), $N_f = 2000$ (bulk) and $N_b = 80$ (periodic boundary conditions). Additionally, for the ADAM optimiser, we uniformly divide the total dataset into mini-batches of size $\mathcal{O}(400)$. For all cases, we split the data into training (80%) and test (20%) sets.

Hyperparameters. The learning rates and wave speeds (β) used for training are given in Table 1. For the specialised hyperparameters for ADAM ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$, weight-decay = 0), GD (no mini-batches and no momentum) and LBFGS (max-iter = 20, tolerance-grad = 10^{-7}, tolerance-change = 10^{-9}, history-size = 100), we used the Pytorch default values.[5] For BBI, we used $\Delta V = 0$ (objective function shift), $\delta E = 2$ (extra initial energy), $N_b = 4$ (number of bounces), $T_0 = 500$ (fixed timesteps for bounces), and $T_1 = 100$ (progress-dependent timesteps for bounces). We trained the models for 1000-5000 epochs depending on the convergence. Training dynamics (train and test losses) is shown in Appendix D.1.

B.2 Learning Rate search for the Linear Advection

Let us start by discussing the criteria used to compare the optimisation algorithms. Naturally, ADAM, BBI, GD and LBFGS have different hyperparameters. BBI, in particular, has specialised hyperparameters to boost chaotic mixing via random bounces (see Appendix A). We note empirically that for the linear advection equation, the learning rate is the hyperparameter that impacts the most in the performances. In Tab. 1 we show the results for the lowest average test losses using 5 trials with random initialised networks varying the learning rates in the range $[10^{-4}, 1]$. We trained the models for the first 300–1000 epochs depending on the convergence. We note that BBI and GD are numerically unstable for learning rates above ~ 0.1. By performing this search, we estimate the best scenario for each optimiser with respect to the learning rate. We acknowledge, however, that significant changes in the other hyperparameters may impact the performances.

[5] `https://pytorch.org/docs/stable/optim.html`

Table 1: **Learning rates for the lowest Test Loss.**

(a) $\begin{bmatrix}2 & 25 & 25 & 1\end{bmatrix}$

β	BBI	LBFGS	GD	Adam
1	0.1	0.1	0.01	0.001
5	0.01	0.1	0.01	0.001
15	0.01	0.01	0.0001	0.01
30	0.01	0.01	0.001	0.01

(b) $\begin{bmatrix}2 & 50 & 50 & 50 & 50 & 1\end{bmatrix}$

β	BBI	LBFGS	GD	Adam
1	0.01	0.1	0.01	0.0001
5	0.01	0.1	0.01	0.001
15	0.01	1	0.01	0.001
30	0.01	0.001	0.001	0.001

C Further details on the Curvature

C.1 Curvature discretisation

To derive a discretised version of the curvature with respect to the training steps, consider the first-order approximation of Eq. (1) given by

$$\dot{\boldsymbol{\omega}}_{k+1} \approx \boldsymbol{\omega}_{k+1} - \boldsymbol{\omega}_k = \mathbf{V}_k. \tag{7}$$

To obtain an approximation for the curvature in Eq. (3), the following steps can be taken

$$\ddot{\boldsymbol{\omega}}_{k+1} \approx \mathbf{V}_k - \mathbf{V}_{k-1}, \tag{8a}$$

$$|\dot{\boldsymbol{\omega}}_{k+1}| \approx \sqrt{\langle \mathbf{V}_k, \mathbf{V}_k \rangle}, \tag{8b}$$

$$\frac{d}{dt}|\dot{\boldsymbol{\omega}}_{k+1}| \approx \frac{\langle \mathbf{V}_k, \mathbf{V}_k \rangle - \langle \mathbf{V}_k, \mathbf{V}_{k-1} \rangle}{\sqrt{\langle \mathbf{V}_k, \mathbf{V}_k \rangle}}, \tag{8c}$$

$$\kappa_{t,k+1} \approx \frac{1}{\sqrt{\langle \mathbf{V}_k, \mathbf{V}_k \rangle}} \left[\langle \mathbf{V}_{k-1}, \mathbf{V}_{k-1} \rangle - \frac{\langle \mathbf{V}_{k-1}, \mathbf{V}_k \rangle^2}{\langle \mathbf{V}_k, \mathbf{V}_k \rangle} \right]^{1/2}. \tag{8d}$$

C.2 Local curvature and the loss function

Here we comment on the relation between κ_ω, i.e. the time-independent local curvature of the training trajectory in the parameter space, and the loss function. Following a physics intuition, given an objective function, $\mathcal{L}(\boldsymbol{\omega})$, we tend to see it as a potential and assume that our trajectory lies on the hypersurface described by $\mathcal{L}(\boldsymbol{\omega})$. This is not true, as different optimisers follow different theories that modify the loss landscape and the regime (e.g. energy-conserving and friction-dominated motion). Nevertheless, given the form of Eq. (1), we can reinterpret the optimiser as if it were a friction-dominated classical algorithm, such as GD. This gives us a hint about how large a gradient should be to induce that step in the GD algorithm, and it tells us how curved the distorted loss landscape, seen by the optimiser, is.

Fig. 2 shows a pictorial view of the relation between the loss function, or its distorted version seen by the optimiser, and the trajectory in the parameter space. The path followed on the loss hypersurface is depicted with a red curve, while its projection on the parameter space is in orange. The orange curve is precisely the focus of this study. Indeed, comparing trajectories living on different loss—or distorted loss—hypersurfaces is an ill-defined problem. On the other hand, monitoring the projected trajectories and their local curvature, κ_ω, tells us how curved is the fictitious loss landscape seen by each optimiser.

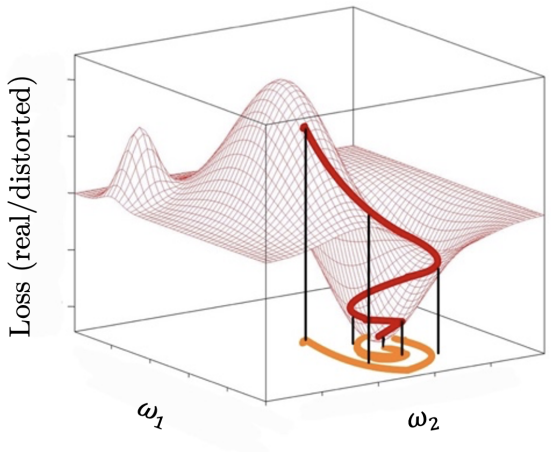


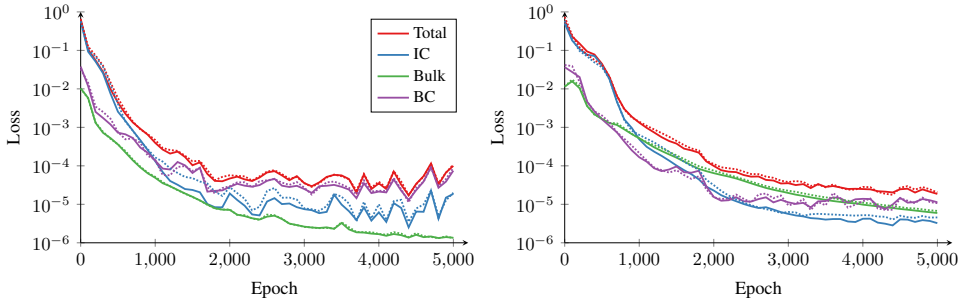
Figure 2: Pictorial view of the relation between the loss function (or its distorted version seen by the optimiser). The orange curve on the plane $\omega_1 - \omega_2$ represents the trajectory followed by the optimiser on the parameter space.

D Additional Plots

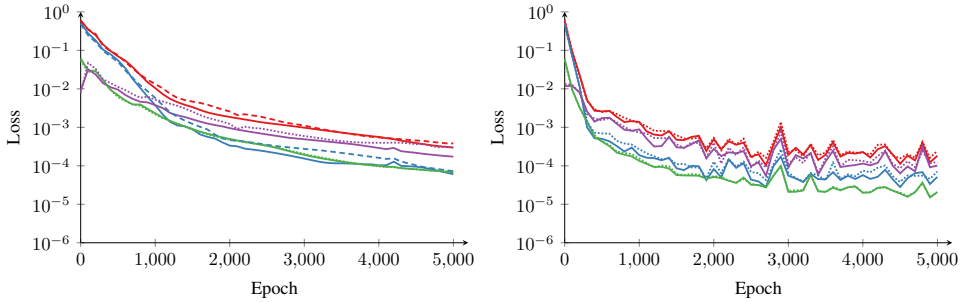
D.1 Training and Test Losses

Based on the performances in Fig. 1a in the main text, we can divide the optimisers into two groups:

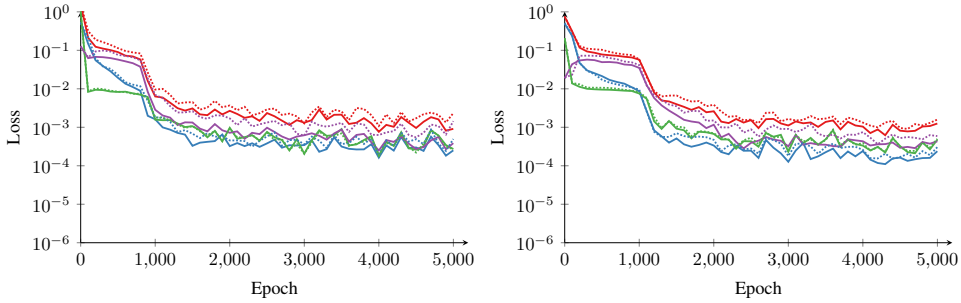
- *LBFGS and ADAM are the optimisers that obtain the best results*. Although LBFGS converges faster, ADAM has the smallest error for the highest β among all optimisers. Comparing train and test losses (Figs. 3 and 5), we note that in both cases the generalisation errors (difference between test and train errors) become larger as β increases. Also, for large β, their losses are dominated by the periodic boundary contribution (last term in Eq. 5a), and test losses tend to overfit.
- *The optimiser that performs worst is GD, followed by BBI*. In particular, there is no learning for $\beta = 15$ and 30. In these cases, the losses are dominated by the initial conditions (first term in Eq. 5a), see Figs. 4 and 6. The GD's poor performance is expected: it has no momentum to be able to overcome saddle points and no mini-batches as a source of stochasticity that helps in escaping from local minima. On the other hand, explaining BBI's low performance requires further investigation as BBI's specialised hyperparameters offer a broad range of tuning possibilities (De Luca & Silverstein, 2022). In particular, training BBI for a longer time and exploring different strategies to trigger the bounces are possibilities worth exploring.



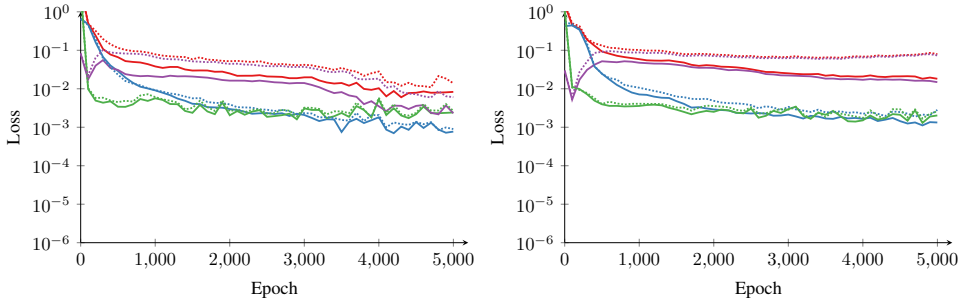
(a) $\beta = 1$, Network 1 (b) $\beta = 1$, Network 2



(c) $\beta = 5$, Network 1 (d) $\beta = 5$, Network 2

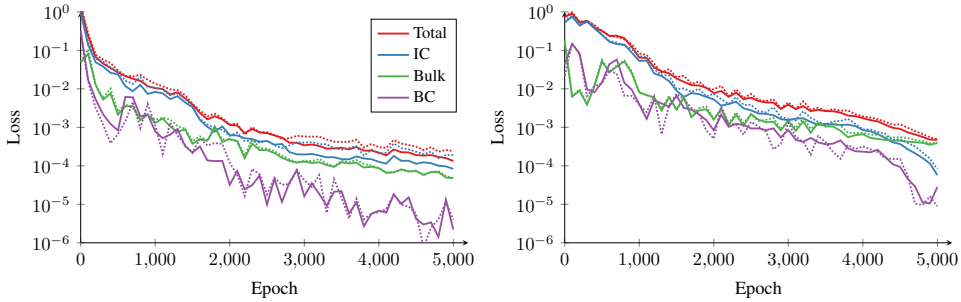


(e) $\beta = 15$, Network 1 (f) $\beta = 15$, Network 2



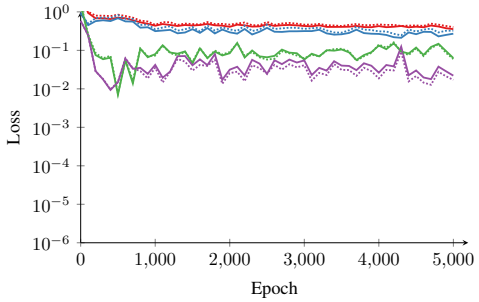
(g) $\beta = 30$, Network 1 (h) $\beta = 30$, Network 2

Figure 3: Median loss versus epoch over 10 samples when using ADAM. *Solid* line is the total, and the other lines show the separate contributions. *dashed* for initial condition, *dotted* for the bulk, and *dash-dotted* for the periodic boundary.

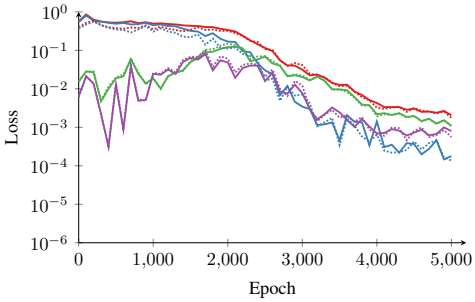


(a) $\beta = 1$, Network 1

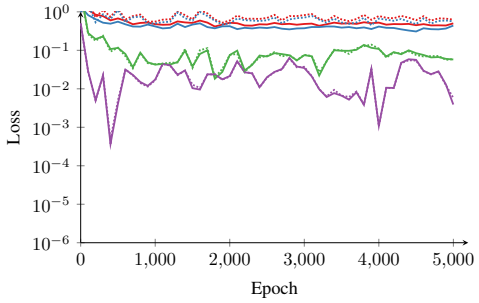
(b) $\beta = 1$, Network 1



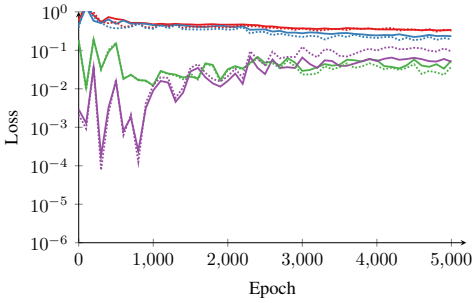
(c) $\beta = 5$, Network 1



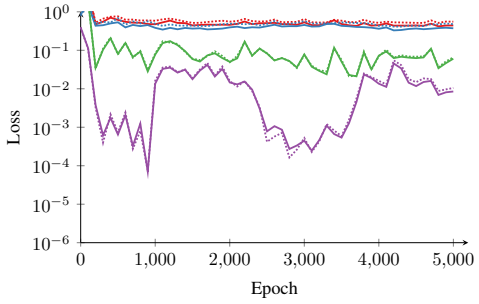
(d) $\beta = 5$, Network 1



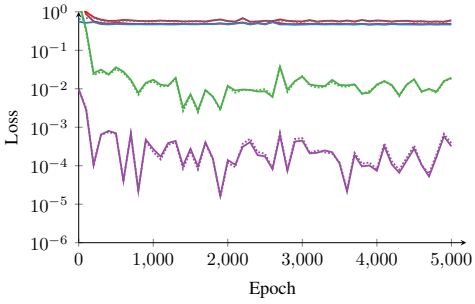
(e) $\beta = 15$, Network 2



(f) $\beta = 15$, Network 2

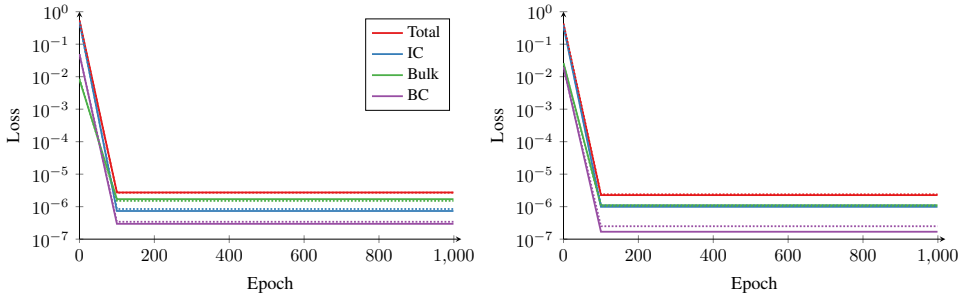


(g) $\beta = 30$, Network 2



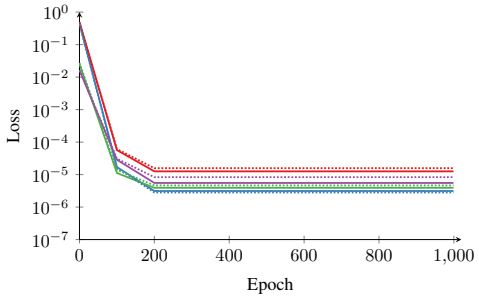
(h) $\beta = 30$, Network 2

Figure 4: Median loss versus epoch over 10 samples when using BBI. *Solid* line is the total, and the other lines show the separate contributions. *dashed* for initial condition, *dotted* for the bulk, and *dash-dotted* for the periodic boundary.

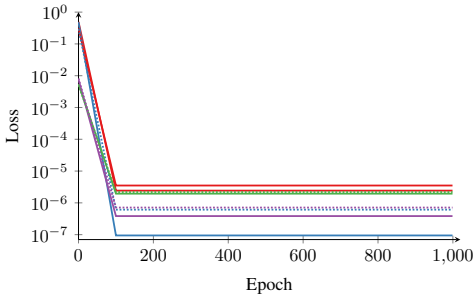


(a) $\beta = 1$, Network 1

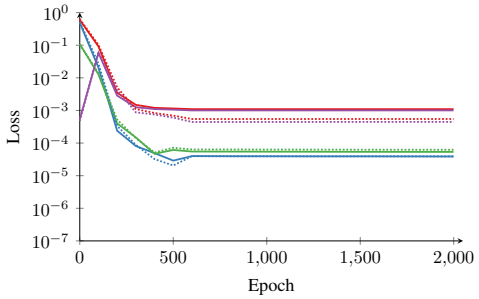
(b) $\beta = 1$, Network 2



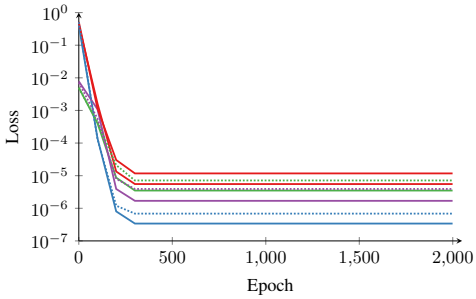
(c) $\beta = 5$, Network 1



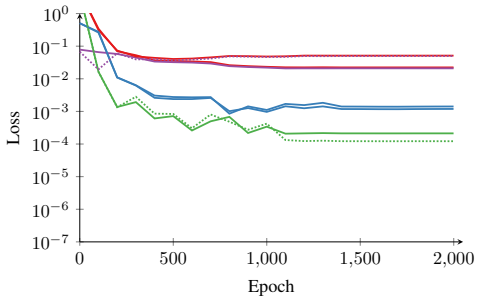
(d) $\beta = 5$, Network 2



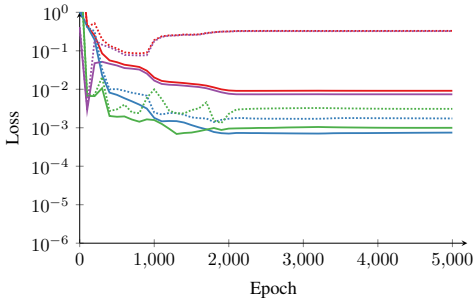
(e) $\beta = 15$, Network 1



(f) $\beta = 15$, Network 2

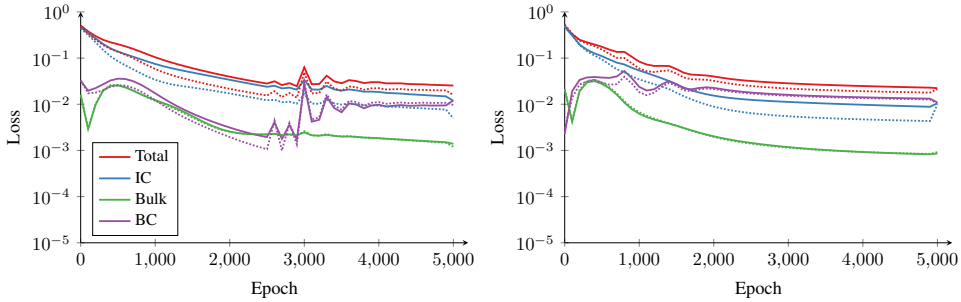


(g) $\beta = 30$, Network 1

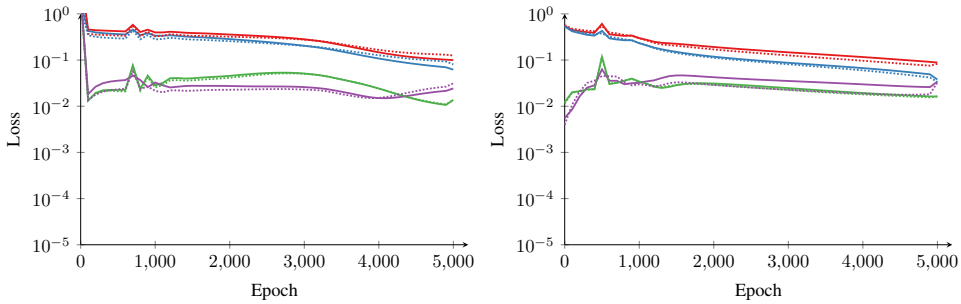


(h) $\beta = 30$, Network 2

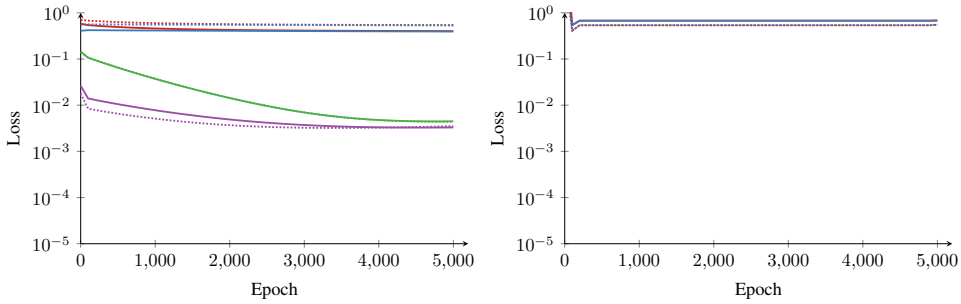
Figure 5: Median loss versus epoch over 10 samples when using LBFGS. *Solid* line is the total, and the other lines show the separate contributions. *dashed* for initial condition, *dotted* for the bulk, and *dash-dotted* for the periodic boundary. Note the changing number of epochs for convergence.



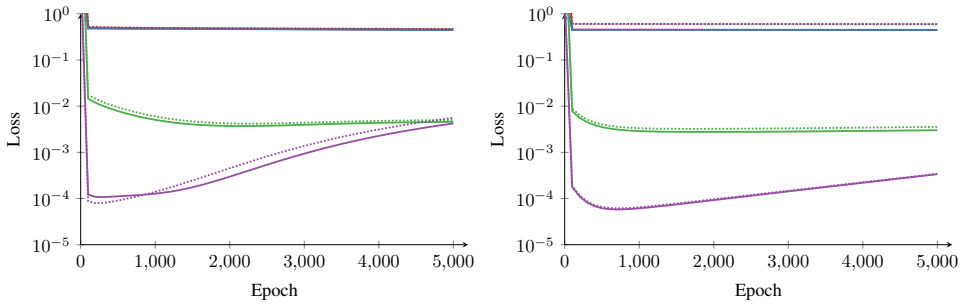
(a) $\beta = 1$, Network 1 (b) $\beta = 1$, Network 2



(c) $\beta = 5$, Network 1 (d) $\beta = 5$, Network 2



(e) $\beta = 15$, Network 1 (f) $\beta = 15$, Network 2



(g) $\beta = 30$, Network 1 (h) $\beta = 30$, Network 2

Figure 6: Median loss versus epoch over 10 when using GD. *Solid* line is the total, and the other lines show the separate contributions. *dashed* for initial condition, *dotted* for the bulk, and *dash-dotted* for the periodic boundary.

D.2 INTERPRETING AND VISUALIZING TRAINING DYNAMICS

This appendix aims to facilitate the understanding of the results shown in the main text, also providing a visualization when possible. Specifically, to better understand the behaviour of the trajectories, we calculate the Spearman correlation, ρ, between κ and the MSE evaluated on the grid points. Spearman's correlation allows us to evaluate whether these quantities move together without fixing a functional form between the two. Furthermore, to understand the directions taken by the trajectory, we calculate the cosine similarity between two consecutive speeds:

$$\cos(\theta_k) = \frac{\dot{\boldsymbol{\omega}}_k \cdot \dot{\boldsymbol{\omega}}_{k-1}}{\|\dot{\boldsymbol{\omega}}_k\| \|\dot{\boldsymbol{\omega}}_{k-1}\|} \qquad \text{for} \qquad k = 1, \ldots, n_{\text{epochs}} \,. \tag{9}$$

The graphs in Table 2 show that the correlation between MSE and ρ remains strongly negative (mainly < -0.5) in cases where the training reaches convergence (see, for example, the results where MSE < 0.01). The third column in Table 2 shows an example of the relation between κ_ω and MSE for $\beta = 1$ on the small (S) network. The colour of the dots varies from pale to intense following the number of evolution epochs. These examples show that the correlation between the curvature κ_ω and convergence remains negative even during the first stages of training. Therefore, the value of ρ cannot be attributed only to some final spiralling in the last minimum of convergence. The rightmost column of Table 2 shows an example (the same as the third column) of the relationship between the ratio in magnitude and the cosine similarity of consecutive speed vectors. We see that while some optimisers show a preferred behaviour (θ_{SGD} tends to π in the last stage of training, θ_{Adam} is nearly 0 before random dynamics kicks in, θ_{BBI} is always nearly 0), LBFGS does not show a preferred alignment with the gradient direction.

Finally, in Table 3, we show the relationship between the final values of curvature and MSE for each optimiser and β value. In general, especially for gradient-based methods, it is possible to see an increase in the final intrinsic curvature value as the convergence increases (as MSE decreases).

Table 2: Relation between convergence (MSE) and $\rho(\kappa_\omega, \text{MSE})$ or $\rho(\kappa_t, \text{MSE})$. Representative trajectory in κ_ω/κ_t and MSE space. Cosine similarity between consecutive speed vectors vs increase in magnitude in consecutive speeds. The colour of the dots varies from pale to intense following the number of evolution epochs.

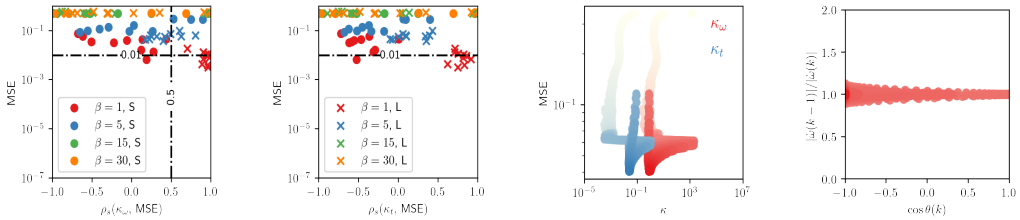
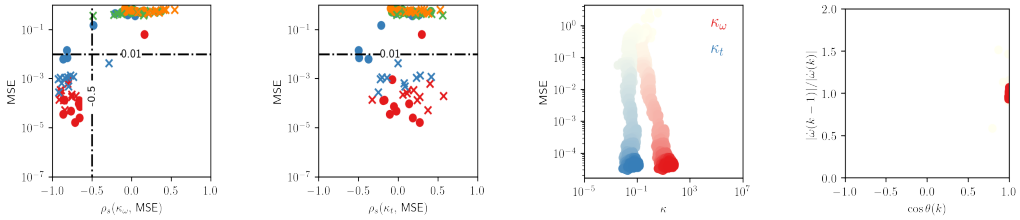
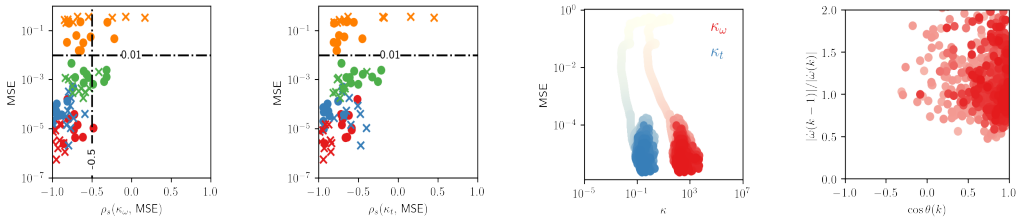
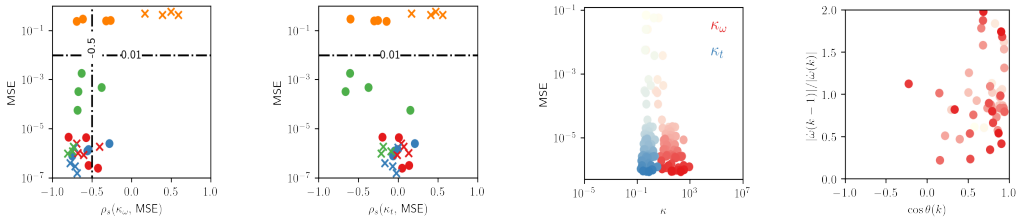
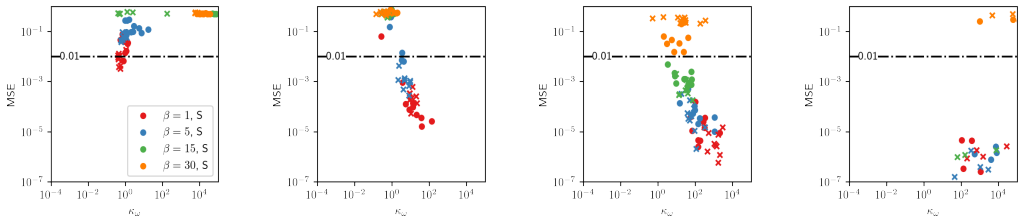
	κ_ω	κ_t	κ vs. MSE	Trajectory
GD				
BBI				
Adam				
LBFGS				

Table 3: Effect of optimiser on the relation between final values of MSE and curvatures κ_ω and κ_t

	GD	BBI	Adam	LBFGS
κ_ω				
κ_t