Prof. Dr. Nils Thürey
Dr. Mario Lino, Benjamin Holzschuh, Patrick Schnell, Qiang Liu and Chengyun Wang
Technical University of Munich
School of Computation, Information and Technology
Summer Semester 2025

Advanced Deep Learning for Physics (IN2298)
Exercise 11
Kuramoto-Sivashinsky Learning

This exercise comprises the second of two parts of a comparative study between supervised and differentiable learning approaches based on the Kuramoto-Sivashinksy equation. In the previous exercise, a differentiable solver was implemented. In this exercise, we will use this solver to train and evaluate neural networks tasked with the prediction of the Kuramoto-Sivashinsky equation. We will use the dataset generated in the previous exercise with the physical domain size set to $X = 10$, resolved by $N_x = 50$ grid points. A differentiable solver is provided and thus this exercise can be solved independent from the previous one.

(1) Supervised network training

First, train a neural network in the classic supervised approach. We will utilise a one-dimensional convolutional residual network (ConvResNet) to predict the next solution sate, i.e. approximate the solver function for a fixed timestep such that $f_{\text{NN}}(u_n) = \mathcal{P}(u_n) = u_{n+1}$. For more information on ResNets, see [1, 2]. The ConvResNet has convolutional layers and skip connections. Note that the convolutions should be padded according to the periodic domain boundaries. Your network should be sufficiently deep (e.g. 10 hidden layers), and have a large enough hidden channel dimension. A total count of 15000 network parameters should be adequate for modelling the KS equation.
You can now setup a training script that trains the network to reduce the $\mathcal{L}_2$-loss between the dataset and network predictions. The simplest variant is to predict the next frame given an initial frame u_n such that the optimisation target becomes

$$\min_\theta \big(\mathcal{L}_2(u_{n+1}, f_{\text{NN}}(u_n))\big),$$

Assess the network performance after training! Run a long test-case (300+ consecutive prediction steps) to see how well the network can replace a numerical solver. You should plot the $\mathcal{L}_2$ over prediction-time w.r.t. a test-simulation $\tilde{u}$, i.e plot $\mathcal{L}_2(\tilde{u}_n, u_n)$. Additionally, we are interested in the physics loss between two consecutive predicted steps $\mathcal{L}_\mathcal{P}$ over time. Use the following definition

$$\mathcal{L}_\mathcal{P} = \mathcal{L}_2(f_{\text{NN}}(u_n), \mathcal{P}(u_n)).$$

Include these plots along with visualisations of prediction and numerical simulation in your report. Interpret your results briefly.

(2) Step unrolling

To enhance stability, we will use $S = 3$ recurrent applications of the network in the training setup. The training procedure is

$$\min_\theta \big(\sum_{s=1}^{S} \mathcal{L}_2(u_{n+s}, f^s_{\text{NN}}(u_n))\big),$$

where $f^s_{\text{NN}}(u_n)$ denotes recurrent predictions of s steps with the network.
Note: Training the network may take up to 1-2h in CPU-only mode, but can accelerated by using a GPU if available.

Assess the network performance after training! Use the same metrics as before and include them in your report. Interpret your results briefly.

(3) Differentiable physics network training

We now train the same network architecture with an unsupervised differentiable loss. The differentiable loss is a normalised variation of the physics loss

$$\mathcal{L}_\mathcal{D}(f_{\text{NN}}, \mathcal{P}) = \frac{\big|\big|f_{\text{NN}}(u) - \mathcal{P}(u)\big|\big|_2}{||f_{\text{NN}}(u)||}.$$

Using $S = 3$ once again, the differentiable physics training can be written as

$$\min_{\theta} \left(\sum_{s=1}^{S} \mathcal{L}_{\mathcal{D}}\left(f_{\text{NN}}^{s}(u_n), \mathcal{P}(f_{\text{NN}}^{s-1}(u_n))\right)\right),$$

with $f_{\text{NN}}^{0}(u_n) = u_n$. The normalisation in the loss function avoids that the network learns a trivial solution $f_{\text{NN}} = \mathbf{0}$. This trivial solution is a local minimum for the loss on the second prediction $\mathcal{L}_{\mathcal{D}}(f_{\text{NN}}(f_{\text{NN}}(u_n)), \mathcal{P}(f_{\text{NN}}(u_n)))$, and all following ones. By always setting $f_{\text{NN}} \approx \mathbf{0}$, the network could learn to accept a large loss for the first prediction in favour of achieving small losses in all succeeding predictions. The normalisation helps us avoid this behaviour by penalising zero-predictions.

Assess the network performance after training! Use the same metrics as before and include them in your report. Interpret your results briefly.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[2] C. Shorten, "Introduction to resnets," May 2019. [Online]. Available: https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4

Submission instruction

Please upload a single PDF file containing your results along with your code for implementation tasks or your derivation for non-implementation tasks (LaTex typesetting). The uploaded PDF should only include the final code, so please trim empty spaces and your intermediate work before submitting.

The easiest way to generate such a PDF is by using Jupyter notebooks and LaTex (we recommend MiKTeX for Windows users). With Jupyter and LaTeX installed, you can create a PDF from your notebook by running *jupyter nbconvert –to pdf your-notebook.ipynb*

Additional information

This is an individual assignment. Plagiarism will result in the loss of eligibility for the bonus this semester.

If you have any questions about the exercises, please contact us via the forum on Moodle. If you need further face-to-face discussion, please join our weekly online Q&A session (every Monday at 15:00 and 16:00 via BBB).