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 6
# Automatic Differentiation

In this exercise, you will develop an intuition for the gradient flow through a coupled a system consisting of a physics solver and a neural network. In this exercise, you will implement the automatic differentiation operation of a numerical solver for Burger's equation. The exercise can be split into three parts. First, a numerical solver is implemented based on explicit first-order Euler time-differencing and a first-order spatial upwind scheme. Secondly, we target the backpropagation of a solver step

**Important Note**: You are **NOT** allowed to use $\Phi_{\text{Flow}}$, PyTorch, TensorFlow, JAX or other framework supporting auto-differentiation in the current exercise. You will only allowed to use NumPy for the calculation and Matplotlib for plot.

## (1) Numerical solver for Burger's equation

We are interested in the one-dimensional Burger's equation written in conservative form as

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\frac{u^2}{2}\right) = 0. \tag{1}$$

Using the explicit Euler's scheme for time-differencing and first-order spatial upwinding we arrive at the discretized equation

$$u_i^{(n+1)} = u_i^{(n)} - \frac{\Delta t}{2}\frac{\partial ({u^{(n)}}^2)}{\partial x}\Big|_i \qquad \text{where} \quad \frac{\partial ({u^{(n)}}^2)}{\partial x}\Big|_i = \begin{cases} \dfrac{{u^{(n)}}^2_{i+1} - {u^{(n)}}^2_i}{\Delta x} & \text{if } \bar{u}_i < 0, \\[2ex] \dfrac{{u^{(n)}}^2_{i} - {u^{(n)}}^2_{i-1}}{\Delta x} & \text{otherwise,} \end{cases} \tag{2}$$

where $n$ and $i$ indicate a temporal and spatial discretisation point respectively, and $\bar{u}_i$ describes a local average over two neighbouring cells. The physical domain is periodic in spatial direction.

**Implement the scheme above.** Your numerical results across 40 numerical steps should match figure 1. The simulation setup is: Domain length $L_x = 2\pi$, discretisation points $N_x = 32$, for $\frac{\Delta t}{\Delta x} = 1$ and the following initial condition:

$$u^{(0)} = \begin{cases} \sin(3x) & \text{if } \frac{\pi}{2} < x < \pi, \\ 0 & \text{else.} \end{cases}$$

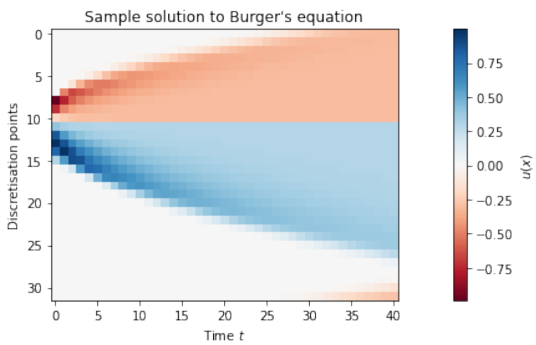


Figure 1: Simulation sample

## (2) Backpropagation

(a) What is the analytical solution of $\frac{\partial u_i^{(n+1)}}{\partial u_i^{(n)}}$, $\frac{\partial u_i^{(n+1)}}{\partial u_{i-1}^{(n)}}$ and $\frac{\partial u_i^{(n+1)}}{\partial u_{i+1}^{(n)}}$ for the above explicit Euler's scheme?

(b) Given a scalar loss function $L$, we know the gradient of $\partial L/\partial u^{(n+1)}$ and the value of the previous state $u^{(n)}$. Implement a backpropagation function which returns the value of $\partial L/\partial u^{(n)}$ and briefly explain the implementation.

## (3) Reconstructing inital conditions

You are given the simulation state shown in Figure 2, which was obtained from running above solver for 15 timesteps. The simulation domain is $L_x = 2\pi$, $N_x = 32$ and this time we chose $\frac{\Delta t}{\Delta x} = 0.1$. The state is stored in the file `burgers_target_state.npy` accompanying this exercise. **Use your backpropagation implementation to find an approximation to the initial condition that leads to the observed simulation state. Plot your initial condition.** You can use any optimizer and loss function of your choice. Also show the loss over optimizer steps of your optimizer run.

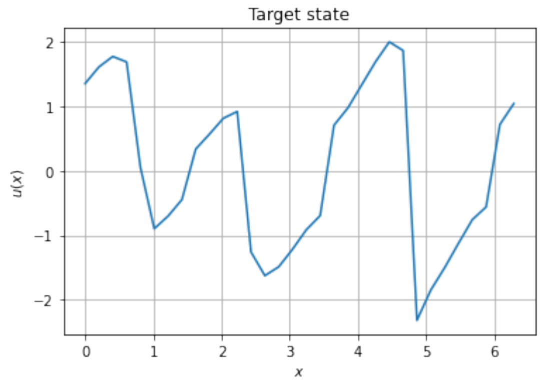


Figure 2: Simulation state after 15 steps

*Tips*: Numpy functions that you may use in this exercise:

- `numpy.pad()`;
- `numpy.logical_not()`;
- `numpy.roll()`.

## 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).