Hands-on session: Physics-constrained data-driven methods for chaotic flows

Luca Magri & Nguyen Anh Khoa Doan

December 3, 2019

Contents

1 Physics-Informed Neural Network for PDE solution approximation

In this part of the exercise session, we will implement a physics informed artificial neural network to approximate the solution of the Burgers' equation:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \tag{1}$$

The Burgers equation can be seen as a one-dimensional version of the Navier-Stokes equation where the pressure term is neglected.
The list of tasks for this session is presented here:

1. Familiarize yourself with the python script/jupyter notebook. A small description of the main script `PINN_Burgers_run` is provided hereunder.

   - Section 2.1: Import of various libraries
   - Section 2.2: Import of the reference dataset. The reference solution for this particular case is provided in `burgers_shock.h5` in the `Data` folder.

     This dataset contains the solution of the Burgers' equation, computed over the domain $x \in [-1, 1]$ and for $t \in [0, 1]$ with Dirichlet boundary conditions, $u(-1, t) = u(1, t) = 0$, and the initial condition, $u(x, 0) = -\sin(\pi x)$. The kinematic viscosity is $\nu = 0.01/\pi$. This is the solution of an initial sinusoid evolving into a shock front.
   - Section 2.3: Definition of the neural network architecture (number of neurons and layers)
   - Section 2.4: Training of the neural network. Different cases are considered depending on the location of the training data.
     - Section 2.4.1: The "IBC-case" (Initial and Boundary Condition) - for this case, we assume to only have knowledge of the solution at the initial time ($t = 0$) and on the boundary condition $x = -1$ and $x = 1$.
     - Section 2.4.2: The "data-case" (data onle) - for this case, we assume to only have the knowledge of the solution at randomly distributed points in the space-time domain.
   - Section 2.5: Comparison of the predictions of the NN
   - Section 2.6: Save the prediction from the neural networks.

   Have a look at the `Burgers_PINN.py` file. It is the implementation of a conventional feedforward neural network. Familiarize yourself with the code structure.

   Run the python script/jupyter notebook `PINN_Burgers_run` and observe the resulting solution from data-only training.

2. Implement the physical constraints in the `Burgers_PINN.py`. To do so, follow the tasks hereunder:

   - in section 1.1, fill in the commented section (after line 45). In this section you will need to define:
     - placeholders for the x and t values for the collocation points for the physical constraints
     - define a variable which receives the values of the physical residual at those points
     - define a new loss variable
     - define a new optimizer which minimize this new loss variable
   - In section 1.4, fill in the `net_f` function. This function receives arrays `x` and `t` where the physical residual $\mathcal{F}$ has to be estimated. The physical residual is given by the Burgers' equation:

$$\mathcal{F}(x, t) = \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} \tag{2}$$

To compute the gradients, make use of the `tf.gradients` function of `tensorflow`. This allows to automate the gradients computation of a function with regards to a given variable. An example to use it is given hereunder:

```
u = func(x,t) # u is a function of x and t
u_t = tf.gradients(u,t)[0] # gradient of u with regards to input t
u_x = tf.gradients(u,x)[0] # gradient of u with regards to input x
```

- In section 1.4, fill in the `predict_f` function which returns the physical residual for a given run of a `tensorflow` session
- In section 1.4, fill in the `train_phys` function: define an optimizer with the physical loss as its objective function.

3. Modify `PINN_Burgers_run` to train and run the NN with the physical loss (and with some data-knowledge). To do so, follow the tasks hereunder:
   - Add a section 2.4.3, and taking inspiration from sections 2.4.2, create a set of randomly distributed points for the physical constraints.
   - Train your physics-informed neural network using the `train_phys` function you have just created

4. Analyse the effect of the number and location of the collocation points. In particular you can try the following combination:
   - Use only data points on the initial condition and boundary conditions and collocation points inside the domain.
   - Use data points randomly distributed in the domain and collocation points everywhere.

5. Repeat the exercise using the dataset `burgers_expWave.h5` (also available in the `Data` folder). The initial condition for that case is $u(x, 0) = \exp(-x^2)\sin(10\pi x)$ with the same boundary conditions. The kinematic viscosity is $\nu = 0.01/\pi$.

   To obtain a good approximation, you will need to also change the size and depth of the neural network.

6. (Optional) Repeat the exercise using the dataset `burgers_PeriodicWave.h5` (also available in the `Data` folder). The initial condition for that case is $u(x, 0) = \exp(-2(x-\pi/2)^2)\sin(x)$ (over a domain $[\pi, \pi]$). The viscosity is changed to $\nu = 0.1/\pi$. Note that for this, you will also have to change the `Burgers_PINN.py` to implement a physical loss to enforce the periodic boundary conditions.

7. Repeat the exercise using the dataset `KS_data_L6_simple.h5`. This dataset is for the Kuramoto-Sivashinsky equation:

$$\frac{\partial u}{\partial t} + \frac{\partial^2 u}{\partial x^2} + \frac{\partial u}{\partial x^4} + u\frac{\partial u}{\partial x} = 0 \tag{3}$$

   It is solved over the domain with $x \in [0, 2\pi L]$ periodic boundary conditions. The initial condition is $u(x, 0) = -\sin(x/(2\pi L))$ and $L = 6$. The Kuramoto-Sivashinsky equation is a model for diffusive instabilities in a laminar flame front.

   Work in the `PINN_KS` folder. Note that you will have to now implement *periodic boundary conditions* in addition to the physical loss.
   - Make the appropriate modifications in the `KS_PINN` file - follow a similar procedure as for the Burgers case.
   - Make the appropriate modifications in the `KS_case` file in section 2.4.2 - follow a similar procedure as for the Burgers case.

2 Echo State Network for Chaotic System Modelling

In this part of the exercise session, we will implement an Echo State Network to model the behaviour of the Lorenz system. The Lorenz system is a prototypical chaotic system governed by:

$$\frac{dx}{dt} = \sigma(y - x), \frac{dy}{dt} = x(\rho - z) - y, \frac{dz}{dt} = xy - \beta z \tag{4}$$

The dataset considered here has parameters $\sigma = 10$, $\rho = 28$, $\beta = 8/3$.
The list of tasks for this section is:

1. Familiarize yourself with the code. The main script `PIESN_Lorenz_run` is organised as follow:
   - Section 2.2 imports the ESN class. Familiarize yourself with the ESN class.
   - Section 2.3 reads the dataset from the Lorenz system and split the dataset into training, validation and prediction part.
   - Sections 2.4 and 2.5 define the ESN and the tensorflow graph for the training and prediction of the ESN.
   - Section 2.6 performs the "training" of the ESN
2. Run the `PIESN_Lorenz_run` script and observe the difference between the teacher-forced prediction and the natural prediction of the ESN. Where does this major difference come from?
3. Analyse the effect of the hyperparameters by modifying them in Section 2.4. In particular:
   - Study the effect of `decay` (also called the leakage rate). Try using a small value of `decay` and observe its effect on the teacher-forced prediction and the natural response.
   - Study the effect of `rho_spectral` (the spectral radius of $\boldsymbol{W}$). Try using a value close to 1 and observe its effect on the natural response.
   - Study the effect of `sigma_in` (the input scaling).
   - Play in combination with the number of units in the reservoir and the Ridge regression factor `lmb`.
4. Implement the physics-constraint in the ESN. To do this, follow the tasks hereunder:
   - In section 2.7, fill in the `Lorenz_step_tf` function. This function receives a tensor `U_in` and computes the "one-step" forward in time prediction using an explicit Euler scheme.
   - Build a new graph extension which makes a natural prediction of the ESN for a given initial state of the reservoir `stateL` for a duration of `valid_hor=1000`.
   - Compute the physical loss on that natural prediction of the ESN.
   - Define a new `LOSS_TF` which combines the loss on the data and the loss on the physical residual.
   - Run the optimizer for sufficient steps to decrease the total loss.
5. Reset the ESN hyperparameters to their original values and train the Physics-Informed ESN (PI-ESN) and observe the improvements in the prediction horizon.
6. Switch from the Lorenz system dataset to the model of shear turbulence called the MFE model (see *Moehlis, J., Faisst, H., & Eckhardt, B. (2004). A low-dimensional model for turbulent shear flows. New Journal of Physics, 6.*). This model is based on a modal decomposition of a turbulent shear flow between two walls under sinusoidal volume forcing. The model keeps the 9 major modes of this problem which allows it to possess the main feature of shear turbulence such as the self-sustaining process with the formation and subsequent breaking down of streaks.

For convenience, work in the `PIESN_MFE` folder and transfer the appropriate modifications from your Burgers equation code.

The dataset is in `MFE_Re600_T30000.h5` and the equations and useful routines are provided in the `MFE_eq` file which has the equivalent one-step prediction from the equation as the one coded for the Burgers' equation. A function which reconstructs the velocity field from the values of the modes is also provided.

The objective here is for you to try to devise an ESN which can model that system:

- First, use data-only ESN and try to modify the hyperparameters
- Second, also use the physics-informed ESN.