Lecture 4 - Unsupervised learning

Utrecht University - Seminar Series Neural Networks in Finance

Kristoffer Andersson

March 14, 2024

Table of contents

Take a step back from previous lectures and view neural networks on a conceptual level, *i.e.*, we view a neural network as a function approximator or as a parametrized mapping to approximate some other function.

Today, we:

- do not specify structure such as layers, weights, biases, activation functions etc.
- assume that optimization of a given loss function can be carried out efficiently, *i.e.*, given a loss function, we assume access to a numerical scheme to find NN parameters such that the loss function is (approximately) minimized;

The emphasize is on how to find a good loss function.

Supervised learning

Supervised learning:
Data are labeled, meaning that for each input in the training data you have access to, and can compare with the true output.

Typical problems:

- **Classification** predicts discrete values, *e.g.*, classify a given image as either a cat or a dog. Input data would be value at each pixel and label would be "cat"/"dog".
- **Regression** deals with continuous data. For instance, given a set of parameters, predict the value of a house.

Learning without labels

Unsupervised learning:
Data is not labeled and learning is self-organized with the aim to find underlying patterns. More difficult to set up an algorithm and requires more knowledge of the structure. Typical example is k-means clustering.

Reinforcement learning:
Algorithm learns to react in it's environment. An agent with a set of possible actions travels from one state to another and gets rewarded for success. Follows a trial and error strategy. This is also similar to how animal learns. For example, reward good behaviour with food and penalize bad behaviour by appropriate punishment.

Supervised learning

Want to approximate a function $f : X \to Y$, where $X \subset \mathbb{R}^d$ and $Y \subset \mathbb{R}$ (scalar output for simplicity). Use a neural network, $\Phi(\cdot \,|\, \theta) : X \to Y$, where θ is the parameters of the network (weights and biases for instance).

Supervised learning:
Data - For $i = 1, 2, \ldots M$, we have pairs (x_i, y_i). Here $x_i \in X$ and $y_i \in Y$ (labels), such that $f(x_i) = y_i$.

Loss function - Since we want $\Phi(\cdot \,|\, \theta) \approx f$, we set *e.g.*,

$$\frac{1}{M} \sum_{i=1}^{M} \|\Phi(x_i \,|\, \theta) - y_i\|_Y^2,$$

for some appropriate norm $\|\cdot\|_Y$.

Supervised to unsupervised learning

What if we do not have access to labels y_i?

Then we:

- cannot compare $\Phi(x_i \mid \theta)$ with y_i and loss function on previous slide is no longer feasible;
- instead, need to come up with a loss function such that when minimized, it holds that $\Phi(\cdot \mid \theta^*) \approx f$. (Here, θ^* refers to an optimized parameter).

To come up with such a loss function, we employ known structure/properties of f.
For instance:

- Physical laws if f describes a physical system;
- Properties from the financial markets if f describes for instance a financial asset or a trading strategy etc..

Physics informed neural networks - Paper by M. Raissi et. al.

The next section of this lecture is based on the paper:

M. Raissi, P. Perdikaris and G.E. Karniadakis *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*
Journal of computational physics (2019).

Problem formulation

Want to solve the following PDE

$$\begin{cases} \dfrac{\partial u}{\partial t}(t,x) + \mathcal{N}(t,x,u,\mathrm{D}_x u, \mathrm{D}_{xx} u) = 0, & t \in [0,T],\ x \in \Omega, \\ u(0,x) = g(x), & x \in \Omega, \\ u(t,x) = v(t,x), & t \in [0,T],\ x \in \partial\Omega. \end{cases} \tag{1}$$

Here $\Omega \subset \mathbb{R}^d$ is the spatial domain of the PDE and u is the sought solution (assumed to be sufficiently regular) and $\mathcal{N}$ is a (possibly) nonlinear operator. In the above, we have Dirichlet boundary condition, but could also include derivatives.

Could be for instance Black-Scholes equation in which case $u(0,x)$ is the value of an option assuming that the underlying follows Geometric Brownian motion.

Solving PDE with supervised learning (Alternative approach)

Straight forward approach to solve (1) using supervised learning would be:

1. Use an expensive method, *e.g.*, finite elements or finite difference method to solve (1) on a grid in time and space. Denote approximative solution by $\bar{u}(t_i, x_j)$, with $0 = t_0 \leq t_1 \leq \cdots \leq t_{N_{\text{time}}} = T$ and x_j being spacial grid point j for $j = 1, \ldots N_{\text{space}}$.
2. Construct the loss function

$$\frac{1}{N_{\text{time}} \times N_{\text{space}}} \sum_{i=0}^{N_{\text{time}}} \sum_{j=1}^{N_{\text{space}}} |\Phi(t_i, x_j \,|\, \theta) - \bar{u}(t_i, x_j)|^2.$$

The hope would then be that if an optimal parameter configuration θ^* could be found that $\Phi(\cdot, \cdot \,|\, \theta^*)$ is an accurate approximation of u on the entire domain $[0, T] \times \Omega$. Basically using a neural network for interpolation.

Solving PDE with supervised learning - Cont.

Drawbacks of proposed method:

- The approach requires another method for generation of reference solutions (labels). Difficult in high dimensions;
- For non-regular solutions or few grid points not at all sure that interpolation between is accurate. Then more grid points are required;
- If we can easily generate solutions on a finer grid, then not clear why a neural network interpolation is even needed?

The aim for Physics-informed neural networks (PINNs) is to be able to approximate (1) in cases where it is not possible to generate reference solutions.

Introduction PINN

The strategy is to use structure given by the problem formulation to construct a loss function which does not require labels.
What is given by (1)?

$$\textbf{Interior dynamics:} \quad \frac{\partial u}{\partial t}(t,x) + \mathcal{N}(t,x,u,\mathrm{D}_x u, \mathrm{D}_{xx} u) = 0,$$

$$\textbf{Initial condition:} \quad u(0,x) = g(x),$$

$$\textbf{Boundary condition:} \quad u(t,x) = v(t,x).$$

Construct loss function to enforce these to be satisfied.

Overview methodology

The main idea is to construct a loss function with three parts:

1. for the interior of the PDE;
2. for the initial conditions;
3. for the boundary condition.

In principle, this is done by replacing the PDE solution u with the neural network $\Phi(\cdot\,,\cdot\,|\,\theta)$. If our approximation is accurate, we should have:

$$\textbf{Interior dynamics:} \quad \frac{\partial \Phi}{\partial t}(t,x\,|\,\theta) + \mathcal{N}(t,x,\Phi,\mathrm{D}_x\Phi,\mathrm{D}_{xx}\Phi) \approx 0,$$

$$\textbf{Initial condition:} \quad \Phi(0,x\,|\,\theta) \approx g(x),$$

$$\textbf{Boundary condition:} \quad \Phi(t,x\,|\,\theta) \approx v(t,x).$$

Automatic differentiation

To obtain $\frac{\partial \Phi}{\partial t}$, $D_x\Phi$ and $D_{xx}\Phi$ we employ automatic differentiation.

In short: Every program is built from a sequence of elementary operations (addition, subtraction, multiplication and division) and elementary functions (*e.g.*, cos, exp, log, etc.). Therefore, by the chain rule, it is possible to perform analytic differentiation on these representations.

```
def u(t, x):
    u = neural_net(tf.concat([t,x],1), weights, biases)
    return u
```

Correspondingly, the *physics-informed neural network* $f(t, x)$ takes the form

```
def f(t, x):
    u = u(t, x)
    u_t = tf.gradients(u, t)[0]
    u_x = tf.gradients(u, x)[0]
    u_xx = tf.gradients(u_x, x)[0]
    f = u_t + u*u_x - (0.01/tf.pi)*u_xx
    return f
```

Continuous or discrete in time

Previous slide gives a conceptual view of the method, but still not clear how to implement. First choice is if we want to consider a **time discrete** or a **time continuous** method.

- **Continuous:** No discretization in time. Implies that temporal derivative has to be approximated by automatic differentiation of the neural network;
- **Discrete:** Construct a temporal mesh $\Pi_{N_{\text{time}}} := \{0 = t_0, t_1, \ldots, t_{N_{\text{time}}} = T\}$. For simplicity equidistant, *i.e.*, $h = t_{i+1} - t_i$ for all $t_{i+1}, t_i \in \Pi_{N_{\text{time}}}$. Temporal derivative is approximated by the Runge–Kutta method.

In the next slides, each method is described in more details.

Time continuous method

As described before, the solution to (1), u, is approximated with a neural network $\Phi(\cdot, \cdot \,|\, \theta)$.

Interior dynamics:

$$\frac{\partial \Phi}{\partial t}(t, x \,|\, \theta) + \mathcal{N}(t, x, \Phi, \mathrm{D}_x\Phi, \mathrm{D}_{xx}\Phi) \approx 0, \quad (t, x) \in [0, T] \times \Omega.$$

Here $\frac{\partial \Phi}{\partial t}$, $\mathrm{D}_x\Phi$, and $\mathrm{D}_{xx}\Phi$ are approximated by automatic differentiation. Can be done easiliy in Pytorch, Tensorflow, Keras etc.

Initial condition:

$$\Phi(0, x \,|\, \theta) \approx g(x), \quad x \in \Omega.$$

Boundary condition:

$$\Phi(t, x \,|\, \theta) \approx v(t, x), \quad (t, x) \in [0, T] \times \partial\Omega.$$

As stated before, the boundary condition could also involve derivatives.

Time continuous method - cont

Since the expression on the previous slide should hold uniformly on the entire domain, we integrate each term over the relevant domain.

Interior dynamics:

$$\begin{aligned}&\mathrm{MSE}_{\mathrm{int}}(\theta)\\ :=&\frac{1}{|\Omega| \times T}\int_{\Omega}\int_{0}^{T}\left|\frac{\partial \Phi}{\partial t}(t,x\,|\,\theta)+\mathcal{N}(t,x,\Phi,\mathrm{D}_x\Phi,\mathrm{D}_{xx}\Phi)\right|^2 \mathrm{d}\mu(x)\mathrm{d}t.\end{aligned}$$

Here $|\Omega|$ is the size of the spatial domain under the measure μ. Usually a uniform measure on Ω is sufficient and then $|\Omega|$ is just the Euclidean size of the domain and $\mathrm{d}\mu(x) = \mathrm{d}x$.

Time continuous method - cont

...and for the initial and boundary conditions.

Initial condition:

$$\mathrm{MSE}_{\mathrm{IC}}(\theta) := \frac{1}{|\Omega|} \int_{\Omega} \left|\Phi(0, x \,|\, \theta) - g(x)\right|^2 \mathrm{d}\mu(x).$$

Boundary condition:

$$\mathrm{MSE}_{\mathrm{BC}}(\theta) := \frac{1}{|\partial\Omega| \times T} \int_{\partial\Omega} \int_{0}^{T} \left|\Phi(t, x \,|\, \theta) - v(t, x)\right|^2 \mathrm{d}\mu(x)\mathrm{d}t.$$

Similarly as for $|\Omega|$, $|\partial\Omega|$ is the size of the boundary of Ω under the measure μ.

Time continuous method - Combining terms

The final loss function is then obtained by combining the terms:

$$\mathcal{L}(\theta) := \text{MSE}_{\text{int}}(\theta) + \text{MSE}_{\text{IC}}(\theta) + \text{MSE}_{\text{BC}}(\theta).$$

In the definitions of the terms we have used a scaling based on the size of respective domain. In the exact case, the loss function should be zero regardless the scaling but in practice, this choice matters. Therefore, an alternative formulation could be

$$\mathcal{L}(\theta) := c_{\text{int}}\text{MSE}_{\text{int}}(\theta) + c_{\text{IC}}\text{MSE}_{\text{IC}}(\theta) + c_{\text{BC}}\text{MSE}_{\text{BC}}(\theta).$$

In the context of a MSc thesis, supervised by C. W. Oosterlee, optimal weights were derived. Resulted in a very successful paper:

van der Meer, Remco, Cornelis W. Oosterlee, and Anastasia Borovykh. "Optimally weighted loss functions for solving PDEs with neural networks." Journal of Computational and Applied Mathematics 405 (2022): 113887.

Time continuous method - cont

Even though this method is continuous both in time and space, a computer can only handle finite number of evaluation points. Equivalent to saying that the integrals in $\text{MSE}_{\text{int}}(\theta)$, $\text{MSE}_{\text{IC}}(\theta)$, and $\text{MSE}_{\text{BC}}(\theta)$ need to be approximated. This can be done with a Monte–Carlo simulation.

The most straight forward procedure is to sample evaluation points uniformly both in time and space. Using $M = M_{\text{int}} + M_{\text{IC}} + M_{\text{BC}}$ samples in the Monte–Carlo run (size of training data) we would generate evaluation points according to:

$$\begin{aligned}
&\textbf{Interior:} \quad (t_i, x_i) \sim \mathcal{U}([0, T] \times \Omega), \quad \text{for } i = 1, 2, \ldots, M_{\text{int}}, \\
&\textbf{Initial condition:} \quad x_i \sim \mathcal{U}(\Omega), \quad \text{for } i = 1, 2, \ldots, M_{\text{IC}}, \\
&\textbf{BC:} \quad (t_i, x_i) \sim \mathcal{U}([0, T] \times \partial\Omega), \quad \text{for } i = 1, 2, \ldots, M_{\text{BC}}.
\end{aligned}$$

Time continuous method - Implementable scheme

The empirical versions of the loss function is given by:

Interior dynamics:

$$\begin{aligned}&\overline{\mathrm{MSE}}_{\mathrm{int}}(\theta; X_M)\\ &:= \frac{1}{M_{\mathrm{int}}} \sum_{i=1}^{M_{\mathrm{int}}} \left| \frac{\partial \Phi}{\partial t}(t_i, x_i \,|\, \theta) + \mathcal{N}(t_i, x_i, \Phi, \mathrm{D}_x\Phi, \mathrm{D}_{xx}\Phi) \right|^2,\end{aligned}$$

Initial condition:

$$\overline{\mathrm{MSE}}_{\mathrm{IC}}(\theta; X_M) := \frac{1}{M_{\mathrm{IC}}} \sum_{i=1}^{M_{\mathrm{IC}}} \left| \Phi(0, x_i \,|\, \theta) - g(x_i) \right|^2,$$

Boundary condition:

$$\overline{\mathrm{MSE}}_{\mathrm{BC}}(\theta; X_M) := \frac{1}{M_{\mathrm{BC}}} \sum_{i=1}^{M_{\mathrm{BC}}} \left| \Phi(t_i, x_i \,|\, \theta) - v(t_i, x_i) \right|^2.$$

Time continuous method - Implementable scheme

Combining the terms above, we have the fully implementable loss function:

$$\overline{\mathcal{L}}(\theta\,;\,X_M) := \overline{\mathrm{MSE}}_{\mathrm{int}}(\theta; X_M) + \overline{\mathrm{MSE}}_{\mathrm{IC}}(\theta; X_M) + \overline{\mathrm{MSE}}_{\mathrm{BC}}(\theta; X_M).$$

Inclusion of X_M is to emphasize that the loss function is a random number depending on the specific sample. It should, however, hold that

$$\lim_{M\to\infty} \overline{\mathcal{L}}(\theta\,;\,X_M) = \mathcal{L}(\theta),$$

provided that each of M_{int}, M_{IC} and M_{BC} approaches infinity.

Time continuous method - full error

In addition, according to the universal approximation theorem we could state that:

Assuming (1) *has a unique classical solution* u*, then for each* $\epsilon > 0$*, there exists a neural network structure (sufficiently wide and deep), such that*

$$\|u - \Phi(\cdot, \cdot \,|\, \theta^*)\| \leq \epsilon.$$

Here $\|\cdot\|$ *is some appropriate norm and* θ^* *is the minimizer of the loss function* $\mathcal{L}(\theta)$*.*

As stated in a previous lecture, the above gives no insight into how to find θ^*, only that it exists.

Time continuous method - full error

We have $\theta^* = \arg\min_\theta \mathcal{L}(\theta)$ and from before $\lim_{M\to\infty} \overline{\mathcal{L}}(\theta; X_M) = \mathcal{L}(\theta)$.

Under some additional conditions we finally have

$$\|u - \lim_{M\to\infty} \Phi(\cdot, \cdot \,|\, \bar{\theta}^*_{X_M})\| \leq \epsilon,$$

where $\bar{\theta}^*_{X_M} = \arg\min_\theta \overline{\mathcal{L}}(\theta; X_M)$.

In words: *Assuming that we have enough data so that the empirical loss function has converged there is a neural network structure (sufficiently wide and deep etc.) such that the classical solution to* (1) *can be approximated arbitrarily well.*

Again: This gives no insight into how to find $\bar{\theta}^*_{X_M}$

Time continuous method - Schrödinger equation

Consider the Schrödinger equation with $T = \pi/2$ and $\Omega := [-5, 5]$:

$$\begin{cases} i\dfrac{\partial h}{\partial t}(t,x) + 0.5\mathrm{D}_{xx}h(t,x) + h|h|^2 = 0, \quad t \in [0,T],\ x \in \Omega, \\ h(0,x) = 2\mathrm{sech}(x), \quad x \in \Omega, \\ h(t,-5) = h(t,5),\ \mathrm{D}_x h(t,-5) = \mathrm{D}_x h(t,5), \quad t \in [0,T]. \end{cases} \tag{2}$$

Classical field equation used to study quantum mechanical systems. Chosen to show that with small adjustments the method can handle complex valued equations and boundary conditions periodic both in h and $\mathrm{D}_x h$. The solution to (4) is the approximated by

To deal with the complex valued solution, we let $\Phi(t,x\,|\,\theta) = [\Phi_1(t,x\,|\,\theta), \Phi_2(t,x\,|\,\theta)]$, *i.e.,* let the output of the network take on values in $\mathbb{R}^2$.

$$\Phi_1(t,x\,|\,\theta) + i\Phi_2(t,x\,|\,\theta) \tag{3}$$

Time continuous method - Numerical results for Schrödinger equation

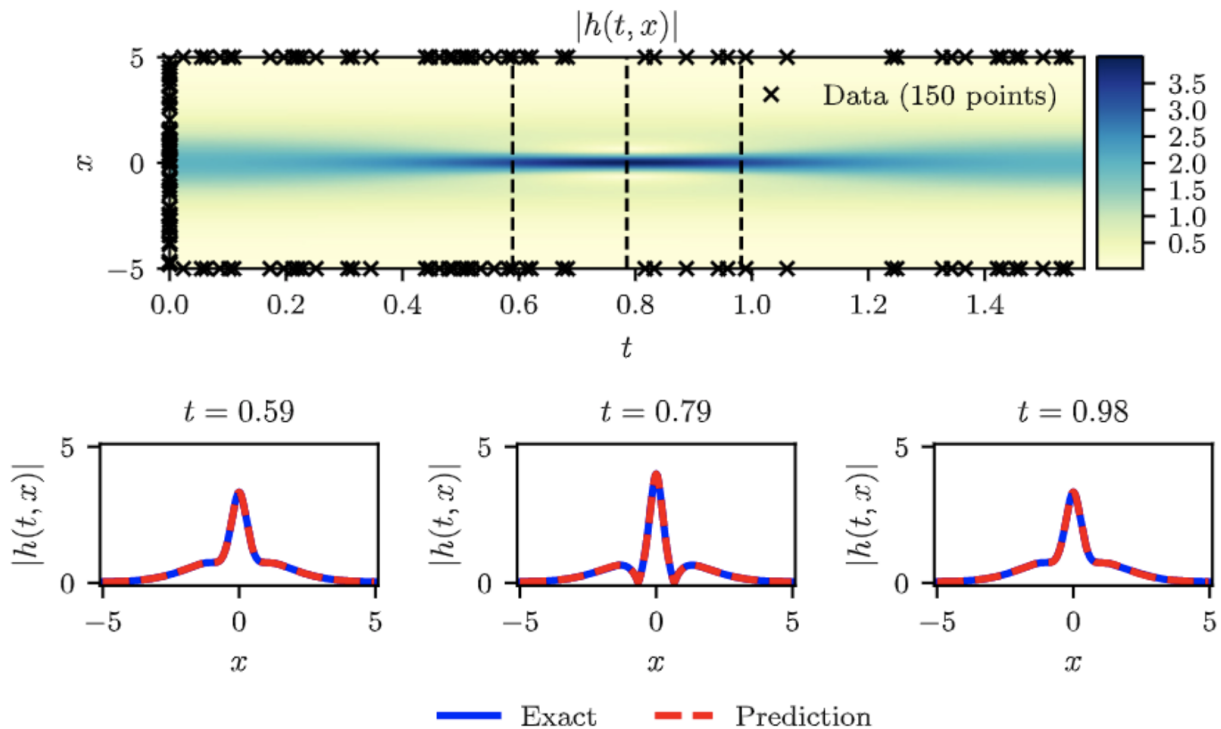


Fig. 1. *Schrodinger equation: Top:* Predicted solution $|h(t,x)|$ along with the initial and boundary training data. In addition we are using 20,000 collocation points generated using a Latin Hypercube Sampling strategy. *Bottom:* Comparison of the predicted and exact solutions corresponding to the three temporal snapshots depicted by the dashed vertical lines in the top panel. The relative $\mathbb{L}_2$ error for this case is $1.97 \cdot 10^{-3}$.

Time discrete method

As already mentioned, another approach is to consider a discretization in time. Recall the time mesh $\Pi_{N_{\text{time}}} := \{0 = t_0, t_1, \ldots, t_{N_{\text{time}}} = T\}$. At some mesh point t_k, the temporal derivative of u can be approximated as

$$\frac{\partial u}{\partial t}(t_k, x) \approx \frac{u(t_{k+1}, x) - u(t_k, x)}{h}.$$

Inserting this approximation, and rearanging the PDE (1) yields

$$u(t_{k+1}, x) \approx u(t_k, x) + h\mathcal{N}(t_k, x, u, \mathrm{D}_x u, \mathrm{D}_{xx} u).$$

Note that u, $\mathrm{D}_x u$, and $\mathrm{D}_{xx} u$ are all evaluated at (t_k, x). This is an explicit Euler scheme, but implicit schemes could also be considered.

Time discrete method - A sequence of networks

Recall that for the continuous method we had one neural network $\Phi(\cdot, \cdot \,|\, \theta) \colon [0, T] \times \Omega \to \mathbb{R}$. For the time discrete method, there is no need to evaluate between temporal mesh points and we therefore instead consider a sequence of networks

$$\left\{\Phi_0(\cdot \,|\, \theta_0), \Phi_1(\cdot \,|\, \theta_1), \ldots, \Phi_{N_{\text{time}}}(\cdot \,|\, \theta_{N_{\text{time}}})\right\},$$

where for each k, $\Phi_k(\cdot \,|\, \theta_k) \colon \Omega \to \mathbb{R}$. By using a sequence of networks, the complexity of each network decreases, since it only approximates one slice of time. On the other hand, the complexity of the optimization procedure may increase (often not a problem for up to at least 100 stacked networks in practice).

Time discrete method - cont

Skipping some steps which are similar in the continuous method we obtain the following terms for the loss function:

Interior dynamics:

$$\mathrm{MSE}_{\mathrm{int}}(\theta) := \frac{1}{|\Omega|} \sum_{k=0}^{N_{\mathrm{time}}-1} \int_{\Omega} \Big| \Phi_{k+1}(x \,|\, \theta_{k+1}) - \Phi_k(x \,|\, \theta_k) - h\mathcal{N}(t_k, x, \Phi_k, \mathrm{D}_x\Phi_k, \mathrm{D}_{xx}\Phi_k) \Big|^2 h \mathrm{d}\mu(x).$$

Here, $\Phi_k, \mathrm{D}_x\Phi_k$ and $\mathrm{D}_{xx}\Phi_k$ are evaluated at x and parametrized by θ_k.

Time discrete method - cont

...and for the initial and boundary conditions.

Initial condition:

$$\mathrm{MSE}_{\mathrm{IC}}(\theta) := \frac{1}{|\Omega|}\int_{\Omega} \left|\Phi_0(x\,|\,\theta_0) - g(x)\right|^2 \mathrm{d}\mu(x).$$

Boundary condition:

$$\mathrm{MSE}_{\mathrm{BC}}(\theta) := \frac{1}{|\partial\Omega|}\sum_{k=0}^{N_{\mathrm{time}}-1}\int_{\partial\Omega} \left|\Phi_k(x\,|\,\theta_k) - v(t_k, x)\right|^2 \mathrm{d}\mu(x)h.$$

Since we have access to the boundary and initial conditions, these two terms could be considered to be supervised learning.

Time discrete method - cont

Since we already have a time discrete formulation, we only need to sample from the spatial domain. Let $M = M\text{int} + M_{\text{IC}} + M_{\text{BC}}$ be the number of samples in the Monte–Carlo run (size of training data) we would generate evaluation points according to:

Interior: $x_{i,j} \sim \mathcal{U}(\Omega)$, for $i = 0, 1, \ldots, N_{\text{time}}$, and $j = 1, 2, \ldots, M_{\text{int}}$,

Initial condition: $x_{0,j} \sim \mathcal{U}(\Omega)$, for $j = 1, 2, \ldots, M_{\text{IC}}$,

BC: $x_{i,j} \sim \mathcal{U}(\partial\Omega)$, for $i = 0, 1, \ldots, N_{\text{time}}$, and $j = 1, 2, \ldots, M_{\text{BC}}$.

Time discrete method - Implementable scheme

The empirical versions of the loss function is given by:

Interior dynamics:

$$\overline{\mathrm{MSE}}_{\mathrm{int}}(\theta; X_M) := \frac{1}{M_{\mathrm{int}} \times (N_{\mathrm{time}} - 1)} \sum_{j=1}^{M_{\mathrm{int}}} \sum_{i=0}^{N_{\mathrm{time}}-1} \Big| \Phi_{i+1}(x_{i+1,j} \,|\, \theta_{i+1}) - \Phi_i(x_{i,j} \,|\, \theta_i) - h\mathcal{N}(t_i, x_{i,j}, \Phi_i, \mathrm{D}_x\Phi_i, \mathrm{D}_{xx}\Phi_i) \Big|^2 h,$$

Initial condition:

$$\overline{\mathrm{MSE}}_{\mathrm{IC}}(\theta; X_M) := \frac{1}{M_{\mathrm{IC}}} \sum_{i=1}^{M_{\mathrm{IC}}} \left| \Phi_0(x_{0,j} \,|\, \theta_0) - g(x_{0,j}) \right|^2,$$

Boundary condition:

$$\overline{\mathrm{MSE}}_{\mathrm{BC}}(\theta; X_M) := \frac{1}{M_{\mathrm{BC}} \times (N_{\mathrm{time}} - 1)} \sum_{j=1}^{M_{\mathrm{BC}}} \sum_{i=0}^{N_{\mathrm{time}}-1} \left| \Phi_i(x_{i,j} \,|\, \theta) - v(t_i, x_{i,j}) \right|^2.$$

Time discrete method - Implementable scheme

Combining the terms above, we have the fully implementable loss function:

$$\overline{\mathcal{L}}(\theta\,;\,X_M) := \overline{\mathrm{MSE}}_{\mathrm{int}}(\theta; X_M) + \overline{\mathrm{MSE}}_{\mathrm{IC}}(\theta; X_M) + \overline{\mathrm{MSE}}_{\mathrm{BC}}(\theta; X_M).$$

Again, it holds that

$$\lim_{M\to\infty} \overline{\mathcal{L}}(\theta\,;\,X_M) = \mathcal{L}(\theta),$$

provided that each of M_{int}, M_{IC} and M_{BC} approaches infinity.

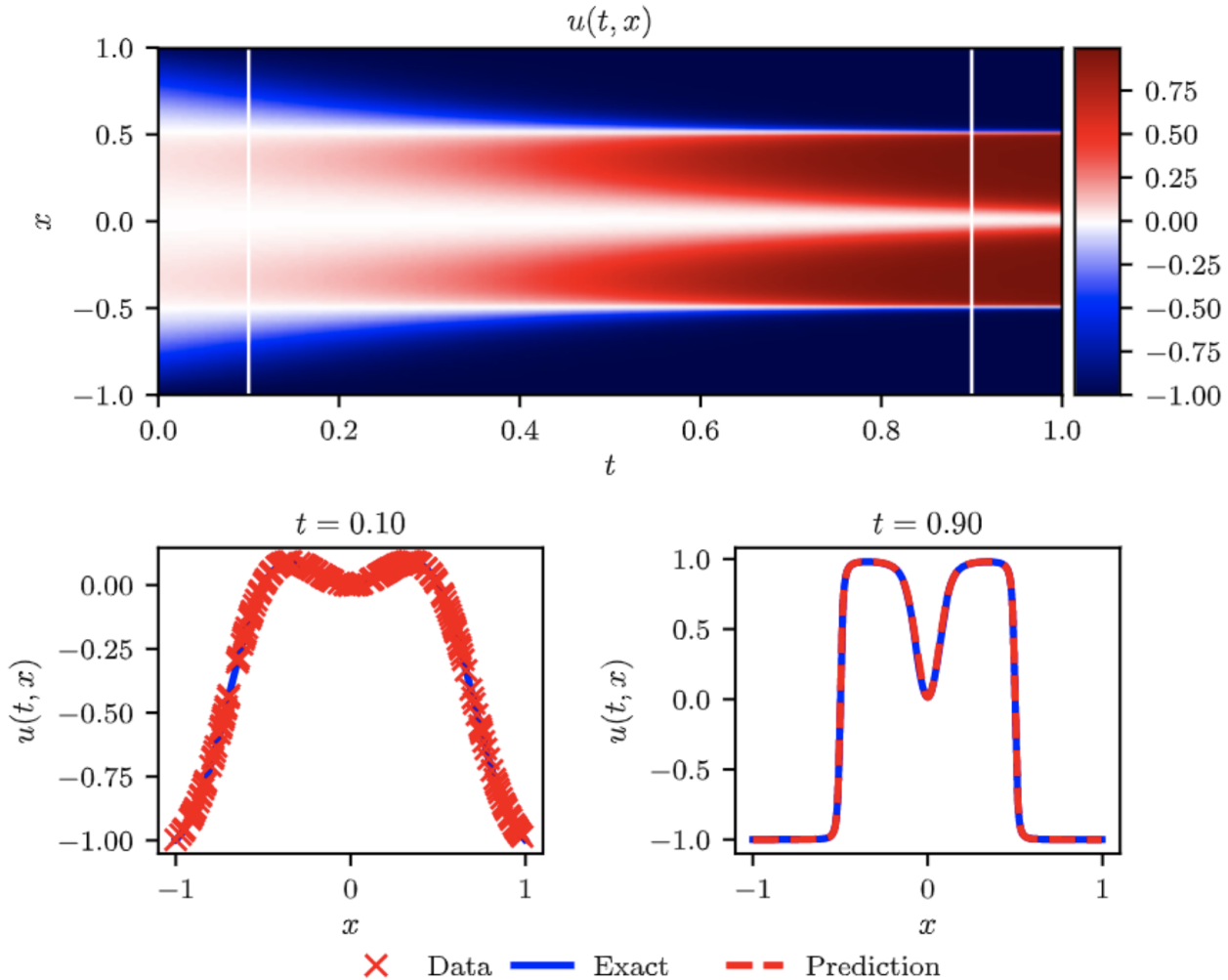
Time discrete method - Allan Cahn equation

Consider the Allan Cahn equation with $T = 1$ and $\Omega := [-1, 1]$:

$$\begin{cases} \dfrac{\partial u}{\partial t}(t,x) - 0.0001 \mathrm{D}_{xx} u(t,x) + 5u^3 - 5u = 0, \quad t \in [0,T],\ x \in \Omega, \\ h(0,x) = x^2 \cos(\pi x), \quad x \in \Omega, \\ h(t,-1) = h(t,1),\ \mathrm{D}_x h(t,-1) = \mathrm{D}_x h(t,1), \quad t \in [0,T]. \end{cases} \tag{4}$$

The Allen–Cahn equation is a well-known equation from the area of reaction–diffusion systems. It describes the process of phase separation in multi-component alloy systems, including order-disorder transitions.

Time continuous method - Numerical results for Allan Cahn equation



Optimal control as a reinforcement problem

A typical optimal control problem is given by a controlled dynamical system and a cost functional:

$$\begin{cases} \dfrac{\mathrm{d}x(t)}{\mathrm{d}t} = f\big(t, x(t), u(t)\big), \quad x(0) = x_0; \ t \in [0, T], \\ J\big(t, x(t)|u\big) = \displaystyle\int_t^T R\big(t, x(t), u(t)\big)\mathrm{d}t + G\big(x(T)\big). \end{cases}$$

Aim is to find a strategy $u = \big(u(t)\big)_{t\in[0,T]}$, such that $J(0, x_0|u)$ is minimized, *i.e.*, find

$$u^* = \arg\min_u J(0, x_0|u).$$

Use a neural network for the control signal

It can be shown that the optimal control is of feedback type, *i.e.*, $u(t) = u\big(t, x(t)\big)$. We represent this feedback-control with a neural network and try to find a parametrization θ^* such that for all $t \in [0, T]$

$$\Phi\big(t, x(t) \,|\, \theta\big) \approx u^*\big(t, x(t)\big).$$

Straight-forward by using the cost functional as loss function.

$$\mathcal{L}(\theta) = J(0, x_0 \,|\, \Phi(\cdot, \cdot \,|\, \theta))$$

Instead of optimizing over the complicated space of control processes, we optimize over the space of neural network parameters. Compare

$$u^* = \arg\min_u J(0, x_0 | u) \quad \textbf{Original control problem},$$
$$\theta^* = \arg\min_\theta J\big(0, x_0 \,|\, \Phi(\cdot, \cdot \,|\, \theta)\big) \quad \textbf{Prametrized problem}$$

Setup a portfolio

Assume we have one (stochastic) stock $\{S_n\}_{n\in\{0,1,\ldots,N\}}$ and one (deterministic) bond $\{B_n\}_{n\in\{0,1,\ldots,N\}}$, where $\{0,1,\ldots,N-1\}$ are the possible trading dates.

Portfolio value at time n

$$\begin{cases} \Pi_0 = \pi_0 \quad \text{(given)}, \\ \Pi_{n+1} = \alpha_n S_{n+1} + \alpha_n^B B_{n+1}. \end{cases}$$

α_n^S and α_n^B refers the number of stocks and bonds at time n, respectively.

What rules apply for $\boldsymbol{\alpha} = \{\alpha_0, \ldots, \alpha_{N-1}\}$ and $\boldsymbol{\alpha}^B = \{\alpha_0^B, \ldots, \alpha_{N-1}^B\}$?

Portfolio optimization problem

Require a **self financing** portfolio *i.e.*,

$$\alpha_n^B = \frac{1}{B_n}(\alpha_n S_n - \Pi_n),$$

Objective is to maximize some functional of the terminal wealth, *e.g.*, a mean-variance criterion

$$U(\boldsymbol{\alpha}) = \mathbb{E}[\Pi_N] - \lambda \mathrm{Var}[\Pi_N].$$

and in turn, the optimization problem is to find

$$\boldsymbol{\alpha}^* \in \arg\max_{\alpha \in \mathcal{A}} U(\boldsymbol{\alpha}).$$

Here $\mathcal{A}$ represents the space of admissible (allowed) trading strategies taking on values in $A \subset \mathbb{R}^N$.

Use neural networks to approximate $\boldsymbol{\alpha}^*$

Assume the form $\alpha_n^* = \alpha_n^*(S_n)$ (Markovian trading strategy).

Aim: Represent $\alpha_n^*(\cdot)$ with neural networks. Use N neural networks

$$\Phi(\cdot \,|\, \theta_n) \colon \mathbb{R} \to A_n, \quad \text{which (hopefully) approximates } \alpha_n^* : \mathbb{R} \to A_n,$$

well.

Final problem formulation

Portfolio dynamics:

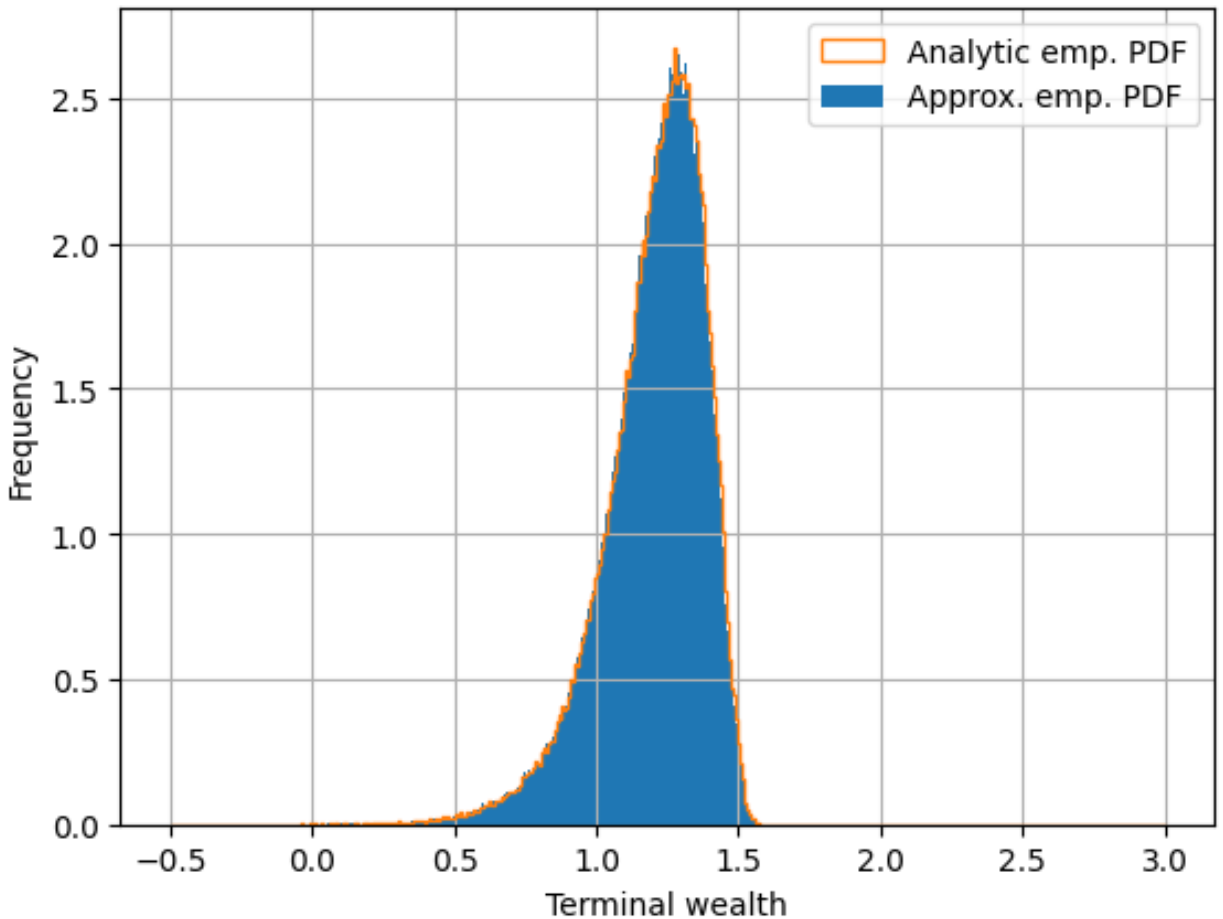
$$\begin{cases} \Pi_0 = \pi_0 \quad \text{(given)}, \\ \Pi_{n+1} = \left(S_{n+1} + \frac{B_{n+1}}{B_n} S_n\right) \Phi(S_n \,|\, \theta_n) + \frac{B_{n+1}}{B_n} \Pi_n. \end{cases}$$

optimization problem:

$$\text{maximize}_{\theta_0,\ldots,\theta_{N-1}} \mathbb{E}_M[\Pi_N] - \lambda \text{Var}_M[\Pi_N],$$

where $\mathbb{E}_M[\cdot]$ and $\text{Var}_M[\cdot]$ represent the $M-$sample empirical mean and variance, respectively.

Mean-variance portfolio optimization problem - Neural network solution vs. analytic counterpart



Mean-variance portfolio optimization problem - Neural network solution vs. analytic counterpart