From UDEs to PINNs

UDEs and Similar Approaches

UDEs provide a good starting point for two more approaches

If you keep the connection to physics, but your relax the ODE mechanism

- ...Then you get Physics Informed Neural Networks
  - Technically, UDEs can be considered PINNs
  - ...But the term refers typically to the approaches surveyed (e.g.) here

If you keep the ODE mechanism, but you drop the connection to physics

- ...Then you get Neural Ordinary Differential Equations
  - This was the first approach to integrate NNs and differential equations
  - The seminal paper is publicly available

We are going to briefly outline the former approach

From UDEs to PINNs

Let's start by recapping how UDEs work

At inference time, we (typically) integrate an initial value problem:

$$\begin{aligned}\dot{\hat{y}} &= f(\hat{y}, t, U(\hat{y}, t, \theta)) \\ \hat{y}(0) &= y_0\end{aligned}$$

At training time, we solve:

$$\begin{aligned}\text{argmin}_\theta \ & L(\hat{y}(t), y) \\ & \dot{\hat{y}} = f(\hat{y}, t, U(\hat{y}, t, \theta)) \\ & \hat{y}(t_0) = y_0\end{aligned}$$

- Which requires to embed ODE integration in gradient descent

From UDEs to PINNs

What if we tried to simplify the inference process?

For example, we could use a NN to approximate $y(t)$ itself

$$\hat{y}(t; \theta) \simeq y(t)$$

This approach has several immediate benefits:

- Inference becomes as efficient as evaluating $\hat{y}(t; \theta)$
  - No need to integrate anything, linear scalability w.r.t. the sampling points
- Handling PDEs also becomes pretty simple
  - We just need to use a multivariate t

But where is physics here?

Training in PINNs

The ODE is taken into account at training time

Superficially, the training problem is similar to the UDE one:

$$\begin{aligned}\text{argmin}_\theta \; & L(\hat{y}(t,\theta), y) \\ & \dot{\hat{y}}(t;\theta) = f(\hat{y}(t;\theta), t) \\ & \hat{y}(t_0;\theta) = y_0\end{aligned}$$

...But in fact, the situation is very different:

- Since both $\hat{y}(t;\theta)$ and $\dot{\hat{y}}(t;\theta)$ need to be learned
- ...Classical ODE integration methods are no longer viable

PINNs circumvent this issue by using NN training for ODE integration

Training in PINNs

In particular, we can apply a Lagrangian relaxation to the problem

We relax the constraints in the previous formulation so that we obtain:

$$\begin{aligned}\mathcal{L}(y, \hat{y}, t, \theta) = &\ L(\hat{y}(t; \theta), y) \\ &+ \lambda_{de}^T \|\dot{\hat{y}}(t; \theta) - f(\hat{y}(t; \theta), t)\|_2^2 \\ &+ \lambda_{bc}^T \|\hat{y}(t_0; \theta) - y_0\|_2^2\end{aligned}$$

In optimization, this is called a Lagrangian

- Besides the original loss L
- ...There is a penalty term linked to the ODE, with weights (multipliers) λ_{de}^T
- ...And a penalty term linked to the initial value, with multipliers λ_{bc}^T

Training in PINNs

The approach can be generalized

- In particular we can take into account both ODEs and PDEs
- ...And we can use different types of penalizers

We just need to abstract a bit the formulation:

$$\begin{aligned}\mathcal{L}(y, \hat{y}, t, \theta) = & \ L(\hat{y}(t; \theta), y) \\ & + \lambda_{de}^T L_{de}(F(\hat{y}, t; \theta)) \\ & + \lambda_{bc}^T L_{bc}(B(\hat{y}, t; \theta))\end{aligned}$$

- Where $F(y, t; \theta) = 0$ defines the original ordinary or partial DE
- ...And $B(y, t; \theta) = 0$ defines the original initial or boundary conditions
- The L_{de} and L_{bc} terms can be L2 norms, but also other types of penalizer

Training in PINNs

Then we train by:

- Sampling points $\{t_i\}_{i=1}^n$ in the input space
- Choosing θ so as to minimize the sum of Lagrangians

$$\text{argmin}_\theta \sum_{i=1}^n \mathcal{L}(y, \hat{y}, t, \theta)$$

We can employ gradient descent, as usual

Again, there is no need to use ODE/PDE integration at training time

...Because training is the integration process

- In fact, it is possible to drop the data-based loss L and the approach still works
- In such a case, PINNs can act as approximate ODE/PDE integrators

No Free Lunch

In the above description, it's easy to miss an important point

Let's consider again the DE-based components in the Lagrangian:

$$L_{de}(F(\hat{y}, t, \theta)) \qquad \text{which could be e.g.} \qquad \|\dot{\hat{y}}(t;\theta) - f(\hat{y}(t;\theta), t)\|_2^2$$

- The penalizer contains derivatives (possibly partial)
- ...And it should provide a contribution for gradient descent

This means that we need a way to compute the components of $\dot{y}$

...So that we obtain an expression that is again differentiable in θ

- This can be a bit tricky in practice!
- Viable approaches include symbolic differentiation (manual or automatic)
- ...Or partially numeric methods such as finite differences, etc.

No Free Lunch

Moreover, assigning a value to the multipliers is not trivial

These the "weight" vectors λ_{de} and λ_{bc}

$$\mathcal{L}(y, \hat{y}, t, \theta) = L(\hat{y}(t; \theta), y) + \lambda_{de}^T L_{de}(F(\hat{y}, t; \theta)) + \lambda_{bc}^T L_{bc}(B(\hat{y}, t; \theta))$$

Finding a good balance might be very tricky

- A good alternative might be using dual ascent

Finally, boundary conditions are incorporated at training time

...So, if they change, we need to repeat training

- In some contexts, this can be a major problem

No Free Lunch

Finally, one should be careful with the problem semantic

Let's consider for a given input vector t the constraint:

$$\|\dot{\hat{y}}(t;\theta) - f(\hat{y}(t;\theta), t)\|_2^2$$

The constraint is enforced in a soft fashion

...Meaning that it might be violated

- Proper weight calibration can help, but violations will still typically occur

Even if we manage exact satisfaction

...The constraint will hold only locally, for the specified t values

- When we move away from the t values considered at training time
- ...The NN may behave inconsistently with the underlying physics

Some Remarks

Let's conclude with some differences betweenn mainstream PINNs and UDEs

Unlike UDEs, PINNs need to learn the involved physics

- It might be necessary to use larger networks
- ...Because they will need to learn a more complex relation

The DE constraints are only approximately satisfied

- UDEs provide instead full guarantees
- ...But approximate satisfisfaction might be good if the DE is not fully reliable

PINNs do not rely on DE integration: they are an integration method

- This makes them faster than UDEs at inference and possibly training time
- ...But don't forget that changing the boundary conditions requires retraining!

Some Remarks

If you are looking for additional information

- There's a very well done PyTorch library for PINNs
- A well-known library is also available for JAX
- The PINN idea can be generalized, leading to Neural Operators
- ...Which map boundary conditions into integrated differential equations