Boring but important disclaimers:

- If you are not getting this from the GitHub repository or the associated Canvas page (e.g. CourseHero, Chegg etc.), you are probably getting the substandard version of these slides Don't pay money for those, because you can get the most updated version for free at

  `https://github.com/julianmak/OCES5303_ML_ocean`

  The repository principally contains the compiled products rather than the source for size reasons.
- Associated Python code (as Jupyter notebooks mostly) will be held on the same repository. The source data however might be big, so I am going to be naughty and possibly just refer you to where you might get the data if that is the case (e.g. JRA-55 data). I know I should make properly reproducible binders etc., but I didn't...
- I do not claim the compiled products and/or code are completely mistake free (e.g. I know I don't write Pythonic code). Use the material however you like, but use it at your own risk.
- As said on the repository, I have tried to honestly use content that is self made, open source or explicitly open for fair use, and citations should be there. If however you are the copyright holder and you want the material taken down, please flag up the issue accordingly and I will happily try and swap out the relevant material.

OCES 5303 :
ML methods in Ocean Sciences

Session 9: PINNs

Outline

- Physics Informed Neural Networks (PINNs)

  → "Physics" is neither here nor there, "principles" might be better...

  → enforcing principles through a **penalisation** in the loss function

  → extra PyTorch syntax to do taping needed

  → some explicit PyTorch syntax for using GPUs also

Figure: Example of unphysical behaviour.

Oceanic application

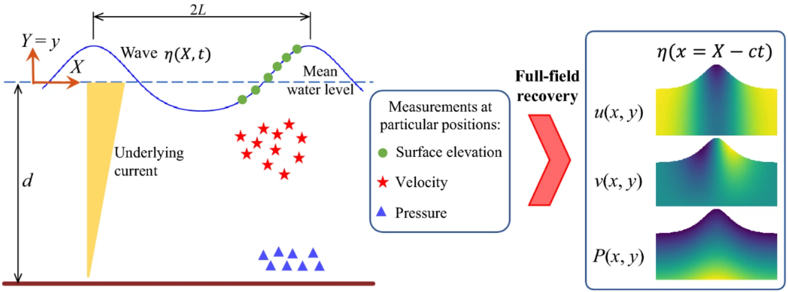

Figure: Problem statement for WaveNets with a PINNs architecture. From Fig. 1 of Chen *et al.* (2024).

- predicts sea surface η, then get the velocities u, v and pressure p according to some principles
  $\rightarrow$ can work with **sparse** training data

Oceanic application

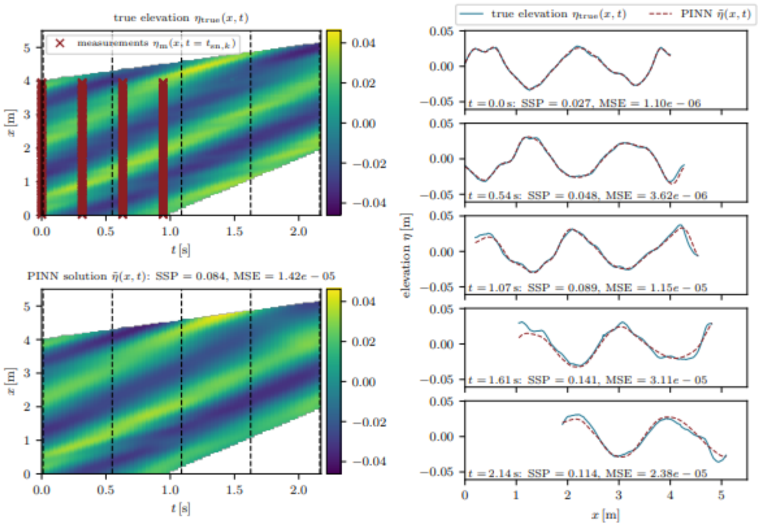

Figure: Nonlinear wave field reconstruction from observations. From Fig. 11 of Ehlers *et al.* (2025).

PINNs: motivating example

- suppose my data $f(t)$ satifies some principle given by ODE

$$\frac{\mathrm{d}f}{\mathrm{d}t} = rt(1-t)$$

$\rightarrow$ if $r = 1$ and $f(0) = 1$ then

$$f(t) = -\frac{1}{3}t^3 + \frac{1}{2}t^2 + 1$$

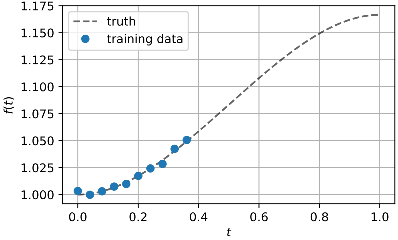

Figure: Example case from an ODE: can we train an MLP to predict what is given?

PINNs: motivating example

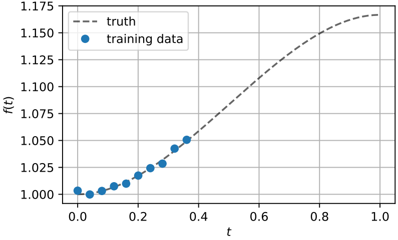

Figure: Example case from an ODE: can we train an MLP to predict what is given?

- goal: take a t and spit out a $f(t)$ (with a MLP say)

  $\rightarrow$ train on blue points, with some noise added to it

  $\rightarrow$ want to get predictions to lie on the black-dashed curve, should be simple right?

PINNs: motivating example

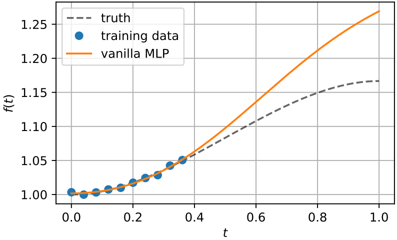

Figure: MLP output trained on the blue points, compared to true data.

- seven hidden layers, ten nodes each, 20k epochs, L^2 loss
- doesn't really work, not entirely surprising given we are extrapolating

  $\rightarrow$ we know data satisfies principles but the machine doesn't know that...

PINNs

Q. how to force the machine to recognise that the data comes from some principle?

$\rightarrow$ adding prior information to the model

$\rightarrow$ cf. us adding 'labels' (e.g. cGANs)

PINNs

Q. how to force the machine to recognise that the data comes from some principle?

$\rightarrow$ adding prior information to the model

$\rightarrow$ cf. us adding 'labels' (e.g. cGANs)

- idea: penalise outputs that do not satisfy the prescribed equations by modifying the loss function as

$$J = J_{\mathrm{data}} + J_{\mathrm{PINNs}}$$

$\rightarrow$ J_{data} the usual mismatch in training and predicted data

$\rightarrow$ J_{PINNs} is large if predicted values do not satisfy equations, e.g.,

$$J_{\mathrm{PINNs}} = \|F(\hat{f}; t)\|_{L^2}^2,$$

where $F = \mathrm{d}f/\mathrm{d}t - rt(1-t) = 0$ would be the constraining equation

PINNs

- adding a soft constraint, can proceed as usual?

PINNs

- adding a soft constraint, can proceed as usual?

!!! want $\mathrm{d}J/\mathrm{d}\theta$ for optimisation where θ is NN parameters, but note that

$$J_{\mathrm{PINNs}}(\theta) = \|F(\hat{f}(\theta); t)\|_{L^2}^2$$

so need to differentiate F...

Q. how do we go about doing that then?

PINNs

- adding a soft constraint, can proceed as usual?

!!! want $\mathrm{d}J/\mathrm{d}\theta$ for optimisation where θ is NN parameters, but note that

$$J_{\mathrm{PINNs}}(\theta) = \|F(\hat{f}(\theta); t)\|^2_{L^2}$$

so need to differentiate F...

Q. how do we go about doing that then?

- PyTorch does this I think (!?) through automatic differentation with `torch.autograd.grad`

  $\rightarrow$ write the ODE in a form PyTorch understands (see notebook)

  $\rightarrow$ ODE in terms of some elementary functions, work out the analytical derivative

  $\rightarrow$ evaluates those user-specified collocation points

PINNs

```python
def ode_logistic(t: torch.Tensor) -> torch.Tensor:
    R: float = 1.0
    return R * t * (1.0 - t)
```

Figure: Above sample ODE in a form PyTorch understands and can tape.

- can then construct and evaluate loss function J
- tape the operations accordingly

  → use `require_grad=True` and `create_graph=True`

  → construct and stores the relevant computation graph

  (cf. 04 with intro to NNs)

  → can then do back-propagation, compute gradients, pass to optimiser and update model as usual

- PyTorch magic to the rescue here...!

PINNs

- modify training loop slightly (see notebook)
- exactly the same settings as before but modifying the loss function

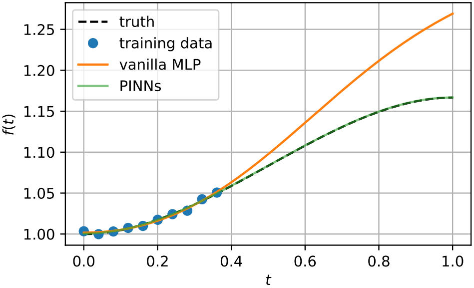

Figure: MLP PINNs output trained on the blue points, compared to true data.

- cf. curve fitting but the equation forces the shape of the curve

PINNs

- can do more complicated example:

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2},$$

$\rightarrow$ heat equation or diffusion equation, a PDE
$\rightarrow$ $u = u(x, t)$, and need some initial + boundary conditions

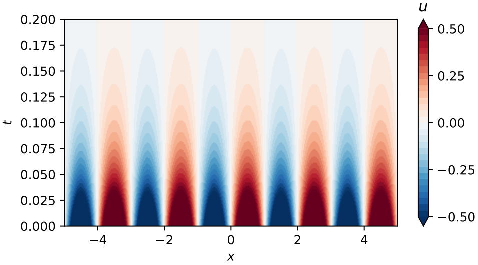

Figure: Diffusion equation as a Hovmöller plot.

- goal: take in (x, t) and spit out u

Digression: GPUs

- this one will be a bit slow, could speed it up probably with putting it on GPUs
  $\rightarrow$ on Colab you need to change the session to a GPU one
  $\rightarrow$ if you have a subscription you can get better GPUs

Digression: GPUs

- GPUs are fast for certain operations associated with neural networks

  → tends to be slow when exchanging data on/off the GPU

  → `model` and various piece of `data` needs to be **put on** the GPU

  → PyTorch does this with `.to(device)`

- once trained things needs to pull off from the GPU

  → `.detach()` and `.cpu()`, as well as `.numpy()` accordingly (see notebook)

- on Keras this is basically automatic

PINNs

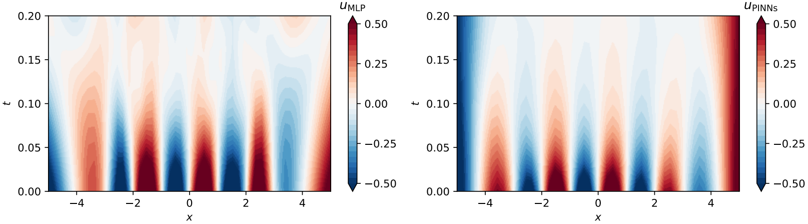

Figure: MLP output trained on some sample points with (left) no PINN and (right) with PINN.

- better in some sense but failing elsewhere
  $\rightarrow$ CNNs would be better probably

- did not do inference (e.g. **parameter estimation**) here, but could do that also

Demonstration

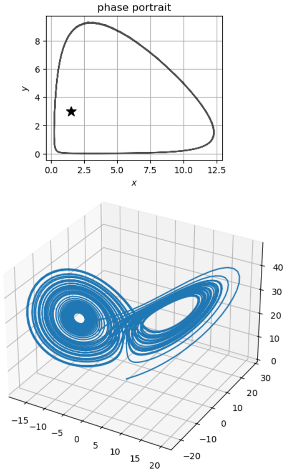

Figure: Phase diagrams of (top) Lotka-Volterra and (bot) Lorenz-63 model.

- idea behind PINNs

  → add “physics” penalisation to the loss function

  → try this for the Lotka-Volterra and KdV equation? (see 07)

  → try this with other equations?

  → could do **inference** calculations too

  → Keras implementation?

Moving to a Jupyter notebook →