Physics-informed Neural Networks (PINNs)

- idea: to use physical prior knowledge explicitly
  (so far in SBI: knowledge only used to create training set via simulation)
  PINNs: incorporate knowledge into training
  equivariant networks: — " — into architecture ($\Rightarrow$ later)
- PINNs focus on surrogates for forward problem

SBI: $X = \phi(Y, \eta)$ $\quad Y \sim p(Y)$, $\eta \sim p(\eta)$

$\iff$ $p(X \mid Y)\, p(Y)$

surrogates learn likelihood

SBI usually learns posterior of inverse problem $p(Y \mid X)$

- typically, PINNs only give a single solution, not a distribution
  i.e. learn $\phi(Y, \eta)$ for Y and η fixed (known or learned)

- PINNs address the case where X is a function

surrogate $\hat{\phi}(t\,;\,\gamma,\eta)$ $\quad$ $\hat{\phi}(\vec{u}\,;\,\gamma,\eta)$

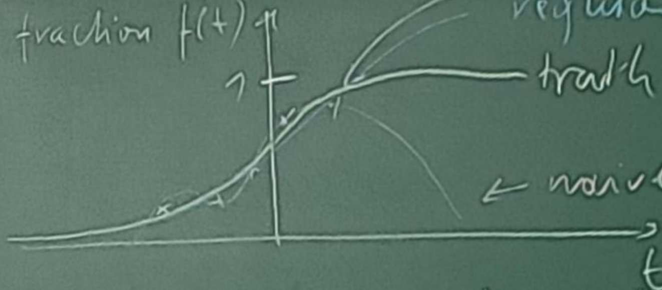
time $\quad$ space

$\hat{\phi}(\vec{u},t\,;\,\gamma,\eta)$

$\phi(t)$ is solution of a ordinary differential eq, $\phi(\vec{u}\,;\,\gamma,\eta)$ and $\phi(\vec{u},t\,;\,\gamma,\eta)$ solutions of partial differential eq

- physical prior knowledge: formula for the ODE or PDE
- why is this useful?
  - actual simulation may be too expensive
  - requires a lot less training data than traditional SBI
  - find a model that fits experimental data

example:



idea: use physical knowledge as a problem-specific regularizer

- here: growth follows "logistic rule": $\frac{d f(t)}{dt} = \lambda f(t) \left(1 - f(t)\right)$

  λ may or may not be known

- special case of SIR eq.

$$\frac{dS}{dt} = -\lambda \frac{S \cdot I}{N} \qquad \frac{dI}{dt} = \lambda \frac{S \cdot I}{N} - \mu I \qquad \frac{dR}{dt} = \mu I$$

set $\mu = 0$ and reformulate in terms of fractions; $[S] = \frac{S}{N}$ etc.

$$\frac{d[S]}{dt} = -\lambda [S] \cdot [I] \qquad \frac{d[I]}{dt} = \lambda [S] \cdot [I] \qquad \text{s.t. } [S] + [I] = 1$$

$$f(t) = [I](t) = \sigma(\lambda \cdot (t - t_0)) = \frac{1}{1 + \exp(-\lambda(t - t_0))}$$

- PINN approach to learning $\hat{f}(t; \lambda, t_0, \eta)$ ← observation noise
  - two kinds of points (later: 3)
    - data points $\hat{=}$ TS where $f(t)$ is (approximately) known (at least data for initial condition $t = 0$)
    - <u>collocation</u> points: points where we apply the regularizer

      $t \in CP$: check if ODE is fulfilled at t

  ⇒ regularization term

$$L_{ODE} = \frac{1}{M} \sum_{m=1}^{M} \Big(\underbrace{\frac{d\hat{f}(t_m)}{dt} - \lambda \hat{f}(t_m)(1 - \hat{f}(t_m))}_{\stackrel{!}{=} 0 \text{ if ODE is fulfilled}} \Big)^2$$

calculate by autodiff

(ideally $M \to \infty$ but finite M is sufficient in practice)

data term: $L_{data} = \frac{1}{N} \sum_{i=1}^{N} (f_i - \hat{f}(t_i))^2$

regression problem

$$\hat{f}, \hat{\lambda} = \underset{f, \lambda}{\arg\min} \; \gamma_{data} \mathcal{L}_{data} + \gamma_{ODE} \mathcal{L}_{ODE}$$

choice of hyperparameters γ_{data}, γ_{ODE} is important => later

result
- unknown ODE parameter $\hat{\lambda}$ is learned simultaneously
- $\hat{f}(t)$ fits the data and generalizes correctly to the domain of collocation points: $t_m \sim p(t_m)$ dom$(p(t_m))$

  (but not outside of this domain)

  [conjecture: average distance between nearest neighbors in $\{t_m\}_{m=1}^{M}$ should correspond to convergence radius of the Taylor series of "reliable" derivatives of $\hat{f}(t)$]
- we do not get a distribution over functions

  $p(f(t) \mid data)$

general case for ODEs: $-\ x(t)$, $\dot{x}(t) = \frac{dx(t)}{dt}$, $\ddot{x}(t) = \frac{d\dot{x}(t)}{dt} = \frac{d^2x(t)}{dt^2}$

$-$ ODE $\qquad ODE(x, \dot{x}, \ddot{x}; t, Y) = 0$

(0) define $TS = \{(t_i, x_i, \underbrace{\dot{x}_i, \ddot{x}_i}_{\text{optional}})\}_{i=1}^{N}$ and collocation set $CS = \{t_m \sim p(t_m)\}_{m=1}^{M}$

both real and $\underbrace{\text{simulated}}_{\text{if } Y \text{ is known}}$

[or resample CS for every batch]

(1) from $\tau = 1, \ldots, \tau_{max}$

(a) do a gradient step of $\nu_{data} L_{data} + \nu_{ODE} L_{ODE} + \nu_{bound} L_{bound}$
(usually ADAM)

optional include a loss for boundary conditions (if applicable)

ex logistic ODE $\quad 0 \leq f(t) \leq 1$

$$L_{bound} = \frac{1}{B} \sum_{b=1}^{B} \left(\mathrm{ReLU}(-f(t_b)) + \mathrm{ReLU}(f(t_b) - 1) \right)^2$$

general form of loss terms

$$L_{data} = \frac{1}{N}\sum_{n=1}^{N} \left(f_n - f(t_n)\right)^2 \qquad L_{ODE} = \frac{1}{M}\sum_{m=1}^{M} ODE(\ddot{f}, \dot{f}, f, t_m, Y)^2$$

$$L_{bound} = \frac{1}{B}\sum_{b=1}^{B} B(f, \dot{f}, \ddot{f}, t_b, Y)^2$$ boundary constraint functions

benefits of using neural networks

- universal approximators (if big enough) and good convergence in practice
  $\Rightarrow$ can in principle learn any $f^*(t, Y)$
- derivatives $\dot{f}(t)$, $\ddot{f}(t)$ easily computable by autodiff

disadvantage:

- for each set of data points (incl. initial conditions) and parameters Y
  learning must start from scratch $\Rightarrow$ no amortization / generalization
  problem is currently addressed $\Rightarrow$ later

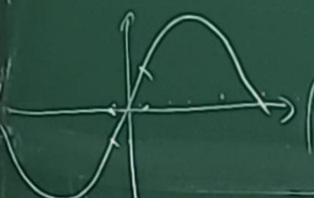
tricks to improve performance:

- use $\tanh(u)$ activation or recently $GELU(u) = \frac{u}{2}\cdot\overbrace{\left(1 + \mathrm{erf}\left(\frac{u}{\sqrt{2}}\right)\right)}^{\text{CDF of standard normal}}$

- standardize the problem (similar to scaling features to unit variance in standard regression)
  transform ODE into a "dimensionless" form
  - use fraction $[I] = \frac{I}{N}$ instead of counts I $\Rightarrow$ $0 \leq f(t) \leq 1$
  - choose units of free parameters γ "cleverly"
    [all γ_j should be $O(1)$?]

- network architecture
  - fully connected networks
  - random Fourier features
    - was shown [Naharam et al 2016 ?] that low frequency behavior of $f(t)$ converges much faster than high frequency behavior

    $\Rightarrow$ convert features (here: time coordinate) to the Fourier domain with random projections



$$\tilde{t} = \begin{bmatrix} \cos(\beta \cdot t) \\ \sin(\beta t) \end{bmatrix} \qquad \beta = [\beta_1, \dots, \beta_k] \qquad \beta_k \sim N(0, \sigma^2)$$

$$\sigma^2 \in [1, \dots, 100]$$

- random weight factorization [Wang et al. 2022]

normal linear layer $\tilde{z}^{(l)} = W^l \cdot z^{(l-1)} + b^l$

replace with $\tilde{z}^{(l)} = \operatorname{diag}(s^l)\, V^l \cdot z^{(l-1)} + b^l$

← explicit scaling of rows of V^l

s_i^l is an individual learning rate adjustment for every neuron

initialization: define $W_{ij}^l \sim N\left(0, \sigma^2 = \frac{2}{n_{in} + n_{out}}\right)$ standard Xavier initialization

$$s_i^l \sim N(\mu = 1, \sigma^2 = 0.01)$$

$$V^l = \operatorname{diag}(s^l)^{-1} \cdot W^l$$

[conjecture: similar to batchnorm, but less noisy because batch variance is not used]