[illegible]

Thứ

Ngày . . No.

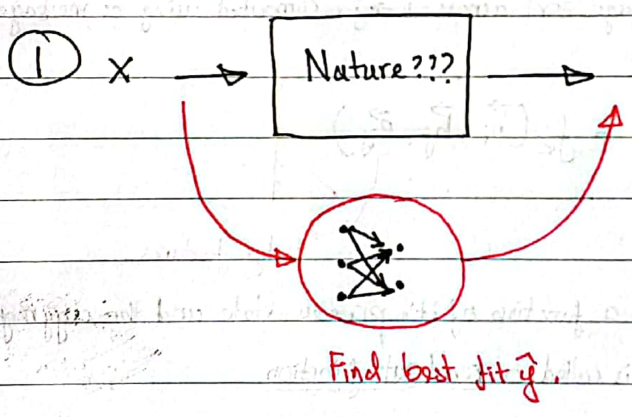
Personal Note #3: Graph Neural Diffusion.

① Prerequisites: ODE and Neural ODE.

ⓐ Understanding Ordinary Differential Equation.

→ Traditional data modelling method (regression): Given a bunch of paired input and output $\{(x_i, y_i)\}_{i=1}^{N}$, find a function $f: X \to Y$ to approx output directly.

→ The problem with this traditional paradigm is that the data is generated by nature factors (economic, social, ...). We have no reliable way to remodel the sophisticated natural generation process. So, we treats nature like a black box & bypass it.



→ $\hat{y} = ax + b$.

→ Params: a and b.

→ Goal: Find a^* and b^* that minimize MSE:

$$\mathcal{L}(a,b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

② Using ODE:

$$\begin{tikzcd} X \arrow[r] & \boxed{f} \arrow[r] & Y \end{tikzcd}$$

→ Goal: Find a robust mapping f from X to Y.

ⓑ Solve an ODE numerically: Euler's Method.

⊛ Modelling rate of change.

↳ So, instead of modelling the function itself directly, we model the rate of change.

↳ For every continuous differentiable function f, its rate of change is defined as the derivative:

$$f' = \frac{df}{dt}$$ (or dx, depending on input).

⊛ ODE basic form.

↳ In the basic form of an ODE, we allow the derivative to depend not only on x but also on y. This allows modelling flexibility.

↳ When integrating the derivative, the integral depends on at least 1 constant variable. To solve for this constant, we need an initial condition (x_0, y_0) (initial value problem).

⊛ $$\boxed{y'(x) = f(x,y) \text{ where } y(x_0) = y_0.}$$ → * Typical IVP.

⊛ Euler's Method.

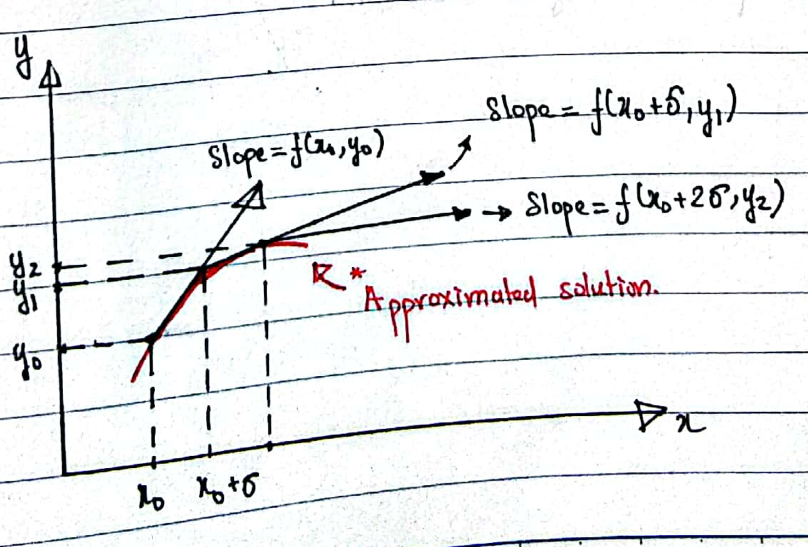
↳ Given a derivative of a function and an initial value :

$$\begin{cases} \dfrac{dy}{dx} = f(x,y) \\ y(x_0) = y_0. \end{cases}$$

↳ We know that $\frac{dy}{dx}(x_0, y_0)$ defines the slope of the line tangent to the curve at (x_0, y_0).

↳ Euler's method is derived from the basic form of tangential approximation:

⊛ $$\boxed{y(x+\delta) \approx y(x) + \delta \frac{dy}{dx}}$$ → * δ is a fixed step size. The smaller the step, the more accurate the approximation.



↳ In simple terms, Euler's method starts at an initial condition (x_0, y_0) then recursively increase x_0 by a fixed step size δ and evaluate the slope at each step to derive the approximated solution (curve).

(*) $$\boxed{\begin{cases} x_{n+1} = x_n + \delta \\ y_{n+1} = y_n + \delta f(x_n, y_n) \end{cases}}$$

(*) Optimising the ODE.

↳ Suppose we want to model a linear regression model $\hat{y}(x) = ax + b$.

→ $\frac{dy}{dx} = a$.

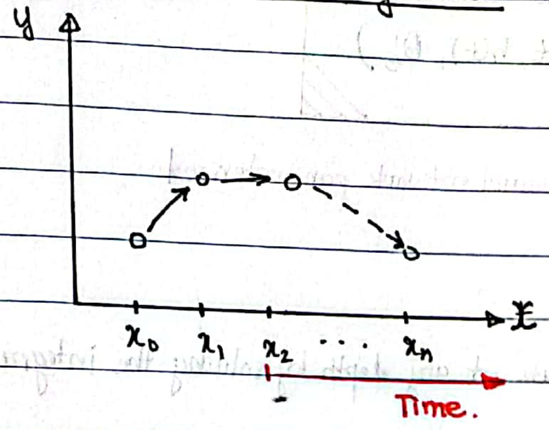
→ We have a set of input X and output Y with N data points.

↳ To model $\hat{y}$ using an ODE,

1. Choose an initial value (x_0, y_0) and a fixed stepsize δ.
2. Choose k data points to evaluate at $(k \leq N)$.
3. Add additional point from X so that Euler's method can be computed at a regular interval δ. This is my computation trajectory.
4. Evaluate the trajectory at the chosen k points.
5. Compute loss & gradient descent.

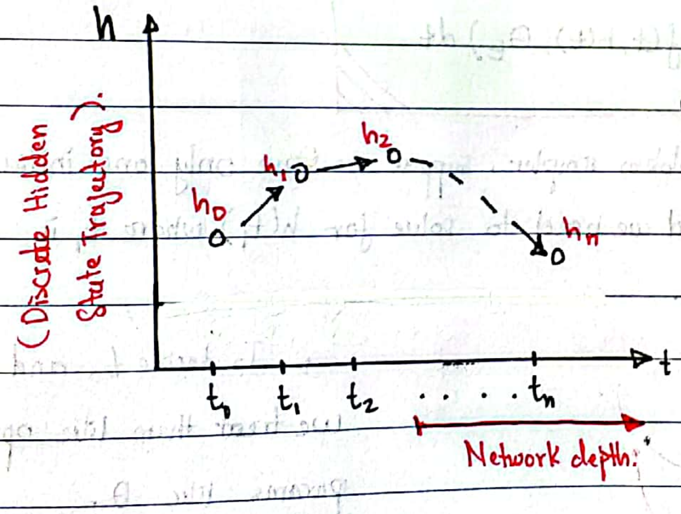
Ⓒ Neural ODE:

(*) Continuous Hidden State Dynamics.



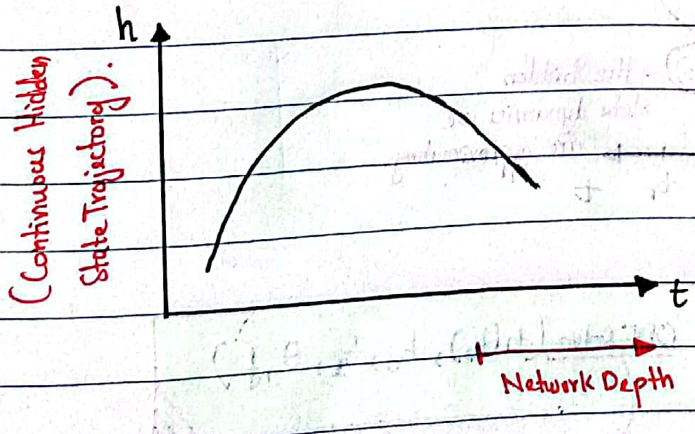
→ In Euler's method, we define the computation trajectory from some initial point (x_0, y_0) and recursively evaluating the ODE at fixed step size.

→ This defines a dynamic of y with respect to time.



→ The key idea of neural network is that as we add layers to our neural network, we add hidden states between input & output.

→ We can view the transformation made by each layer a dynamics of hidden state through network depth.



→ When we take the continuous limit of hidden state w.r.t network depth, we smooth out computation trajectory.

→ The hidden state can be evaluated at ANY depth. (Main idea of neural ODE).

↳ In Neural ODE, we parameterise this hidden state dynamics by :

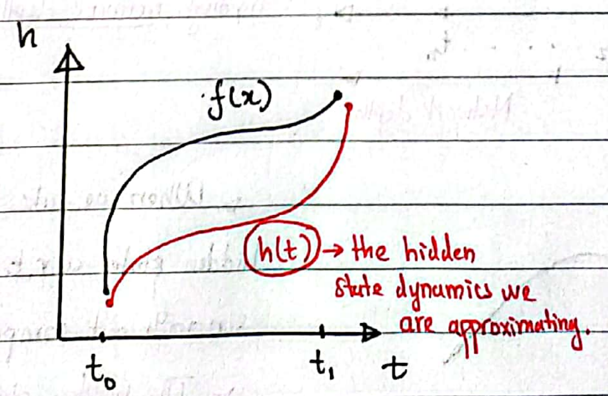
$$(*)\quad \boxed{\frac{dh(t)}{dt} = f(t, h(t), \theta_t)}$$

*↳ $f(t, h(t), \theta_t)$ is a neural network parameterized by θ_t at layer t.

↳ We can evaluate the hidden state at any depth by solving the integral

$$(*)\quad \boxed{h(t) = \int f(t, h(t), \theta_t)\, dt}$$

numerically. To make the problem simpler, suppose we have only one initial observation $(t_0, h(t_0))$ and we need to solve for $h(t_1)$ where t_1 is at the end of the trajectory.



→ To decide t_0 and t_1 we treat them like optimizable params like θ.

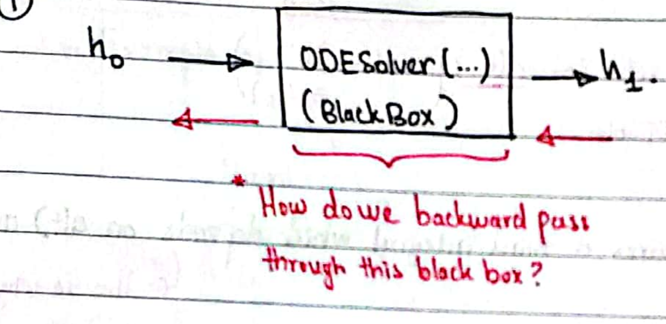
$$(*)\quad \boxed{\hat{y} = h(t_1) = \underline{\text{ODESolve}}(h(t_0), t_0, t_1, \theta, f)}$$

*A simple numerical ODE solver. (Euler, HyperEuler, ...).

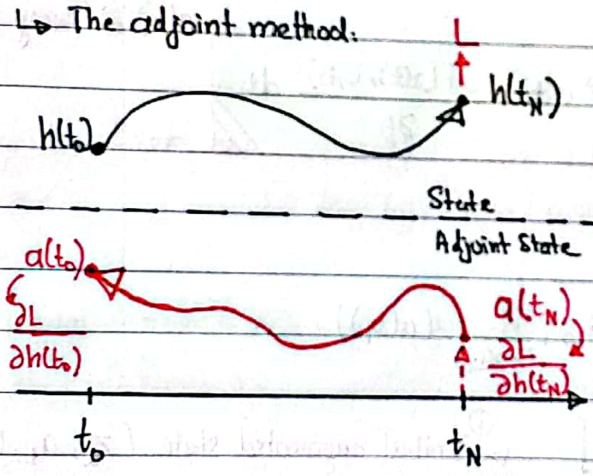
(*) Backpropagating through depth.

↳ The output of the ode net is computed by a "black box" ODE Solver to go from one state (h_0) to another (h_1). Then how do we backprop thru this "black box"?

(1)



↳ The adjoint method:


L
$h(t_N)$
$h(t_0)$
State
Adjoint State
$a(t_0)$
$\frac{\partial L}{\partial h(t_0)}$
$a(t_N)$
$\frac{\partial L}{\partial h(t_N)}$
t_0
t_N

→ The difficult thing about training a continuous depth network is back-propagating through the black-box ODE solver.

→ We do this using the adjoint sensitivity method.

↳ Consider optimizing a scalar-valued function $L(\cdot)$:

(*) $$L(z(t_1)) = L\left[\text{ODESolve}(z(t_0), f, t_0, t_1, \theta)\right]$$

* $z(t)$ = hidden state through time, we define it as $h(t)$ above, just a diff notation.

→ To optimize L, The first step is to determine $\frac{\partial L}{\partial z(t)}$. This is called the adjoint state:

(*) $$a(t) = \frac{\partial L}{\partial z(t)}.$$

Thứ

Ngày . . No.

↳ The dynamics of the adjoint state is defined by another ODE:

$$\boxed{\frac{da(t)}{dt} = -a(t)^T \frac{\partial f(z(t),t,\theta)}{\partial z}}$$

*We can solve for $\underline{a(t_0)}$ (which is $\frac{\partial L}{\partial z(t_0)}$) by calling ODESolve.

↳ Computing $\frac{\partial L}{\partial \theta}$ requires a third integral which depends on $a(t)$ and $z(t)$.

↳* This is why we need $a(t)$ at every backward step.

(*) $$\boxed{\frac{\partial L}{\partial \theta} = -\int_{t_1}^{t_0} a(t)^T \frac{\partial f(z(t),t,\theta)}{\partial \theta} dt}$$

↳ Backprop pseudocode:

```
Initialization: $\theta, t_0, t_1, z(t_1), \frac{\partial L}{\partial z(t_1)}$ $(a(t_1))$

$s_0 = [z(t_1), \frac{\partial L}{\partial z(t_1)}, 0_{|\theta|}]$ → ① Initial augmented state $(z_1, a_1, 0)^T$

def aug-dynamics$([z(t), a(t), \cdot], t, \theta)$: → ② Dynamics of aug state:
    return $[f(z(t), t, \theta), -a(t)^T \frac{\partial f}{\partial z}, -a(t)^T \frac{\partial f}{\partial \theta}]$   $(\frac{dz}{dt}, \frac{da}{dt}, \frac{dL}{d\theta dt})^T$

$[z(t_0), a(t_0), \frac{\partial L}{\partial \theta}]$ = ODESolve$(s_0$, aug-dynamics$, t_1, t_0, \theta)$

return $a(t_0), \frac{\partial L}{\partial \theta}$ #
```