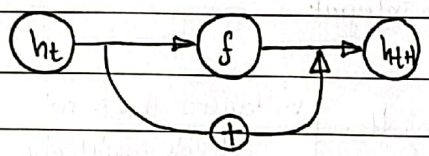
Notes on Neural ODE

Date: No.

① Background.

↳ In ResNet, the author proposes that given a set of (x, y) pairs. Instead of learning the direct relationship $f: X \to y$, we aim to learn the rate of change using residual connection in the network architecture.



(*) $h(t+1) = f(h(t)) + h(t)$. → * This is a discrete process.

↳ The idea of Neural ODE is to model this rate of change in a continuous manner. So, the hidden state dynamics is defined as an ODE:

(1) $\frac{dh(t)}{dt} = f(h(t), t, \theta)$ → * f is a neural network parameterized by θ.

② Related Concepts:

(*) Basic form of ODE.

↳ In the basic form of ODE, the derivative will depend on both x and y.

↳ When integrating the derivative, the solution is often dependent on 1 or many constants. So, we need an initial known point (x_0, y_0) of the solution to solve the ODE.

→ Basic ODE form: (2) $y'(x) = f(x, y)$ given $y(x_0) = y_0$.

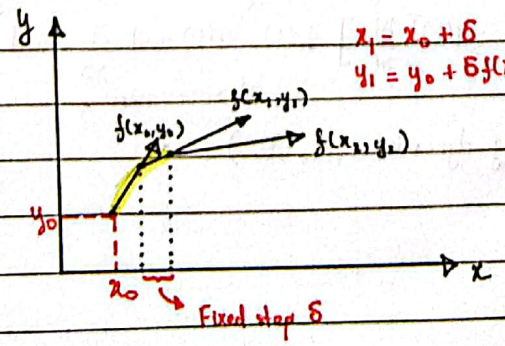
(*) Solving ODE numerically.

↳ One of the numerical method for solving an ODE is the Euler's method.

↳ The Euler method relies on the tangent approximation to "guess" the solution's trajectory starting from an initial point (x_0, y_0).

→ Tangent approximation formula:

$$y(x+\delta) \approx y(x) + \delta f(x, y)$$



→ The euler's method use this to recursively update the computation trajectory.

(3) $x_{n+1} = x_n + \delta$

$y_{n+1} = y_n + \delta f(x_n, y_n)$

Date: No.

③ Optimising a Neural ODE.

↳ Now in Neural ODE, the main idea is to model the derivative of the hidden state. This way we can calculate the hidden state at any depth.

(*) Forward pass.

↳ To calculate the hidden state, we compute the integral:

④ $$h(t_1) = \int_{t_0}^{t_1} f(t, h(t), \theta)\, dt$$

→ * However, this is not solvable analytically.

↳ Since we already know Euler's method, we will solve ④ numerically instead:

⑤ $$h(t_1) = \text{ODESolve}(h(t_0), t_0, t_1, f, \theta)$$

(*) Backward pass:

↳ Since the ODESolver is a blackbox, we cannot perform backprop using regular chain rule. Instead, we use a method called adjoint state $a(t) = \frac{\partial L}{\partial h(t)}$ and we treat it as a dynamic system:

⑥ $$\frac{da(t)}{dt} = -a(t)^T \frac{\partial f(t, h(t), \theta)}{\partial h}$$

→ * This is also solvable using the ODESolver. (from $t_1 \to t_0$)

↳ Our final goal is to compute $\frac{\partial L}{\partial \theta}$, which depends on both $a(t)$ and $h(t)$:

⑦ $$\frac{\partial L}{\partial \theta} = -\int_{t_1}^{t_0} a(t)^T \frac{\partial f(t, h(t), \theta)}{\partial \theta}\, dt$$

→ * Also solvable using ODESolver (from $t_1 \to t_0$).

(*) Back propagation pseudocode.

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

$s_0 = [h(t_1), a(t_1), 0_{|\theta|}]$

def aug_dynamics($[h(t), a(t), \cdot], t, \theta$): → * Augmented dynamics: Concat the dynamics of $h(t)$, $a(t)$ and $\frac{\partial L}{\partial \theta}$ and solve all at once here

return $[f(h(t), t, \theta), -a(t)^T \frac{\partial f}{\partial h}, -a(t)^T \frac{\partial f}{\partial \theta}]$

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

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