NEURAL NETWORKS AS ORDINARY DIFFERENTIAL EQUATIONS

Presenter: Jodie Zhu (Dessa)

Facilitator: Helen Ngo & Lindsay Brin

The Paper

Neural Ordinary Differential Equations

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, David Duvenaud

10/22/2018 (v1: 6/19/2018) cs.LG | cs.AI | stat.ML

1806.07366v3 pdf

show similar | discuss

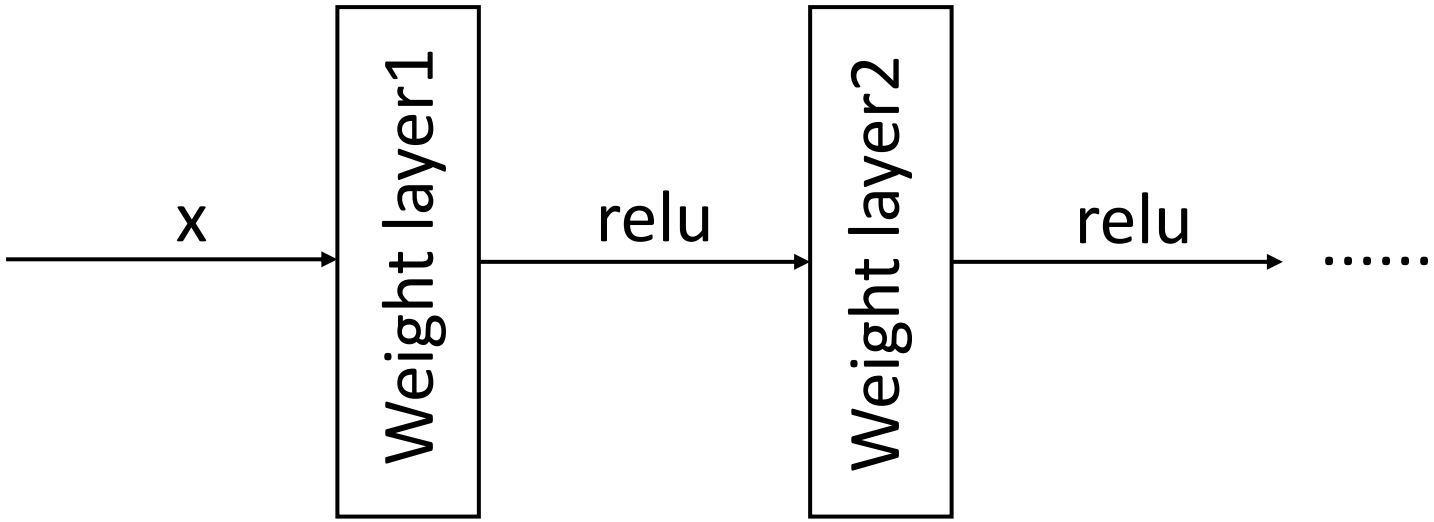
Source: http://www.arxiv-sanity.com/toptwtr?timefilter=month

Neural nets as discrete layers

New state Old state

$$h_{t+1} = f(h_t)$$

State: $h_1 = f(x)$ $h_2 = f(h_1)$ ……



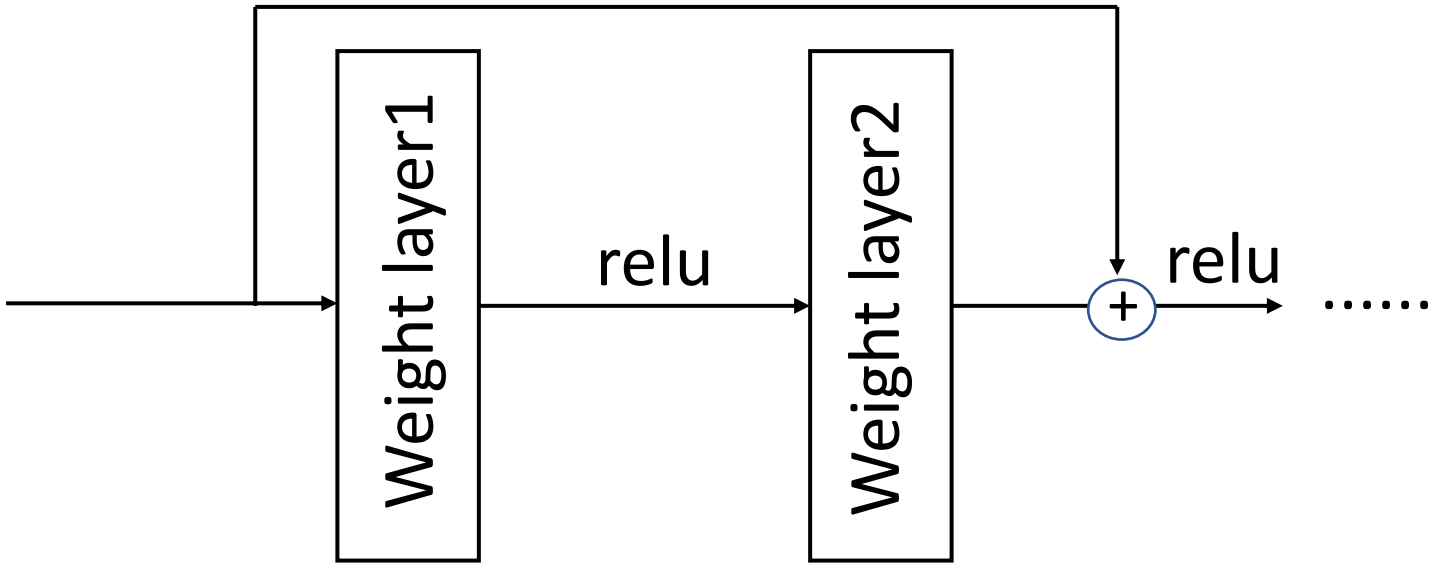
Residual neural networks

New state Old state

$$h_{t+1} = f(h_t)$$

Residuals

State: h_1 $h_2 = h_1 + f(h_1)$. ……



Euler's method and residual neural networks

$$h_{t+1} = h_t + f(h_t)$$

Euler's method and residual neural networks

$$h_{t+1} = h_t + f(h_t)$$
$$= h_t + \frac{\Delta t}{\Delta t} f(h_t)$$

Euler's method and residual neural networks

$$
\begin{aligned}
h_{t+1} &= h_t + f(h_t) \\
&= h_t + \frac{\Delta t}{\Delta t} f(h_t) \\
&= h_t + \Delta t \; G(h_t)
\end{aligned}
$$

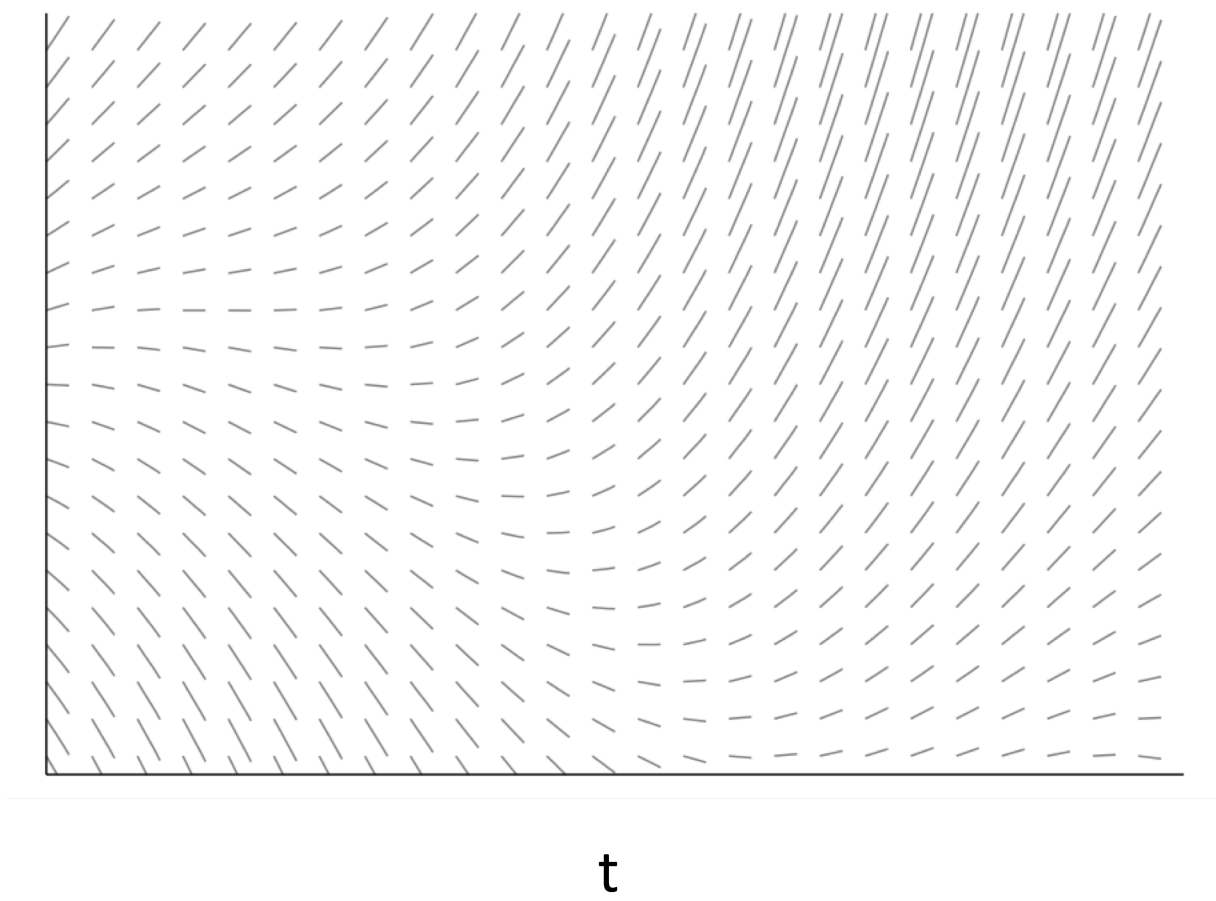
Euler's method and residual neural networks

$$h_{t+1} = h_t + f(h_t)$$
$$= h_t + \frac{\Delta t}{\Delta t} f(h_t)$$
$$= h_t + \Delta t\, G(h_t)$$

A single step of Euler's method for solving ODE

Background: ODE solvers

- State h changes with depth
- Depth-derivative: $\frac{dh}{dt} = G(h(t), t)$

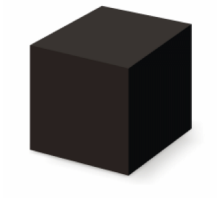


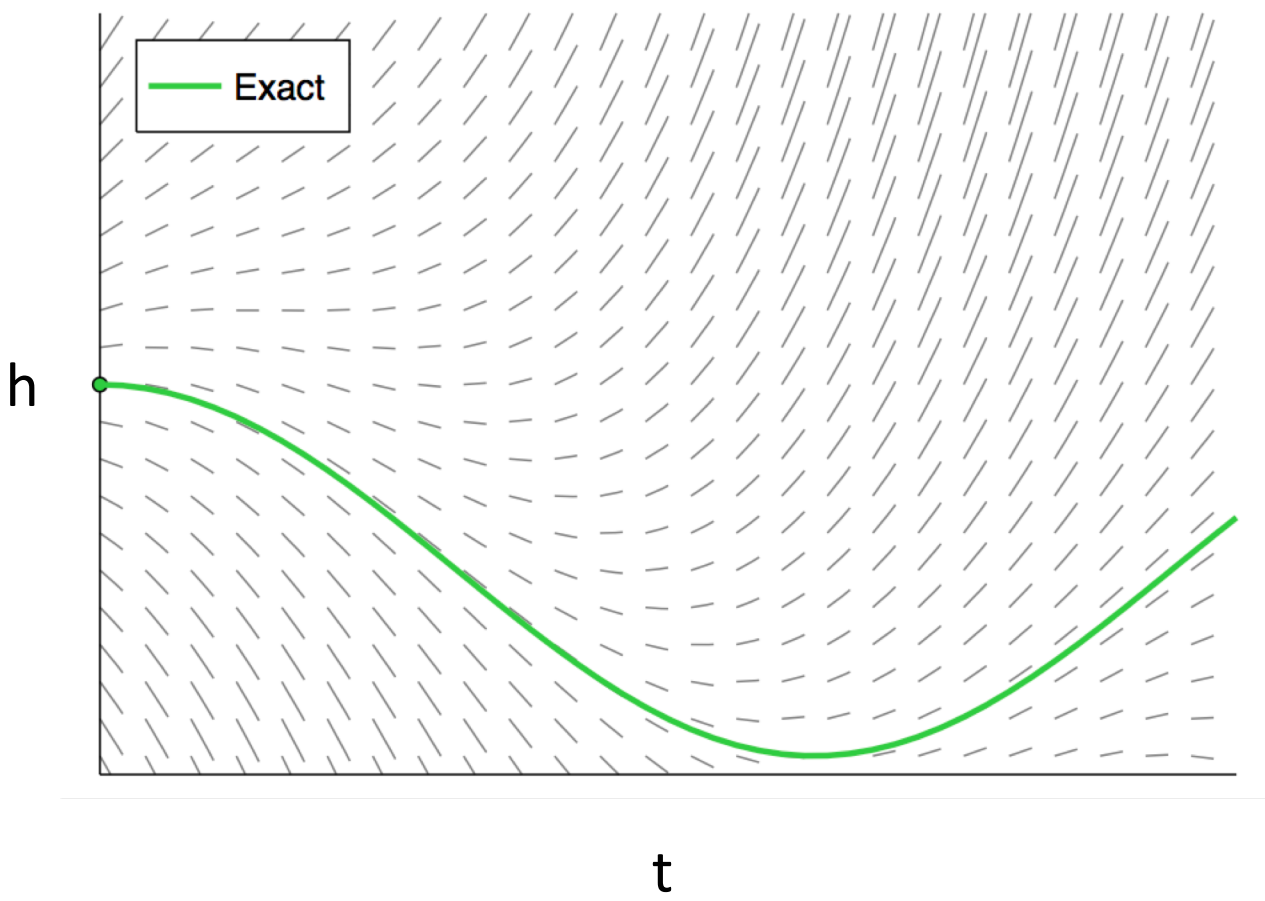
Source: https://en.wikipedia.org/wiki/Euler_method

Background: ODE solvers

- State h changes with depth
- Depth-derivative: $\frac{dh}{dt} = G(h(t), t)$
- Initial-value problem: given $\mathrm{h}(t_0)$, find

$$h(t_1) = h(t_0) + \int_{t_0}^{t_1} G(h(t), t, \theta)dt$$
$$= ODESolver(h(t_o), G, t0, t1, \theta)$$



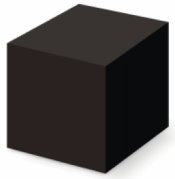


Source: https://en.wikipedia.org/wiki/Euler_method

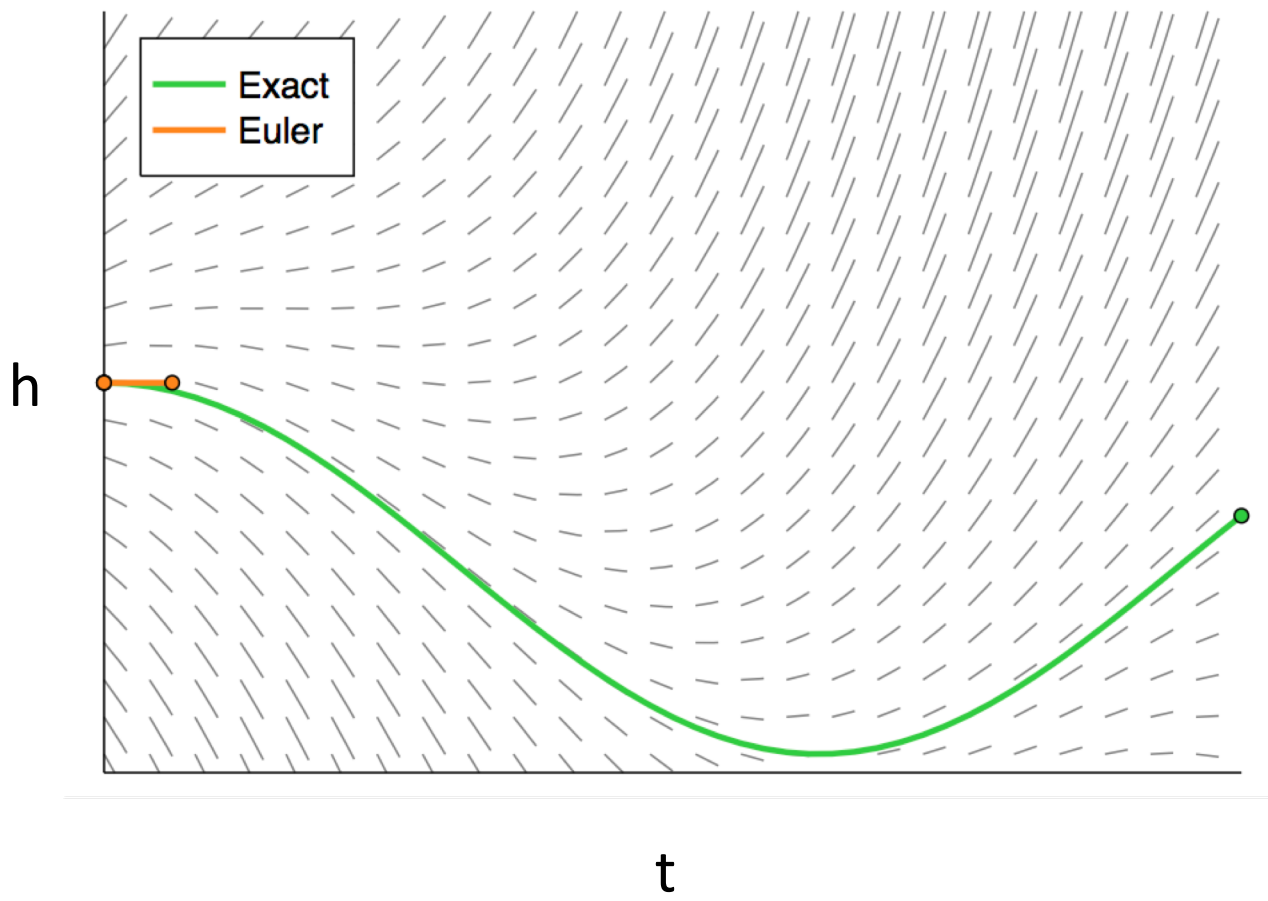
Background: ODE solvers

- State h changes with depth
- Depth-derivative: $\frac{dh}{dt} = G(h(t), t)$
- Initial-value problem: given $\mathrm{h}(t_0)$, find

$$h(t_1) = h(t_0) + \int_{t_0}^{t_1} G(h(t), t, \theta)dt$$
$$= ODESolver(h(t_o), G, t0, t1, \theta)$$



- Euler approximates with small steps:
  $h(t_1) = h(t_0) + (t_1 - t_0)G(h(t_0), t_0)$

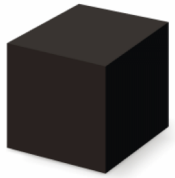


Source: https://en.wikipedia.org/wiki/Euler_method

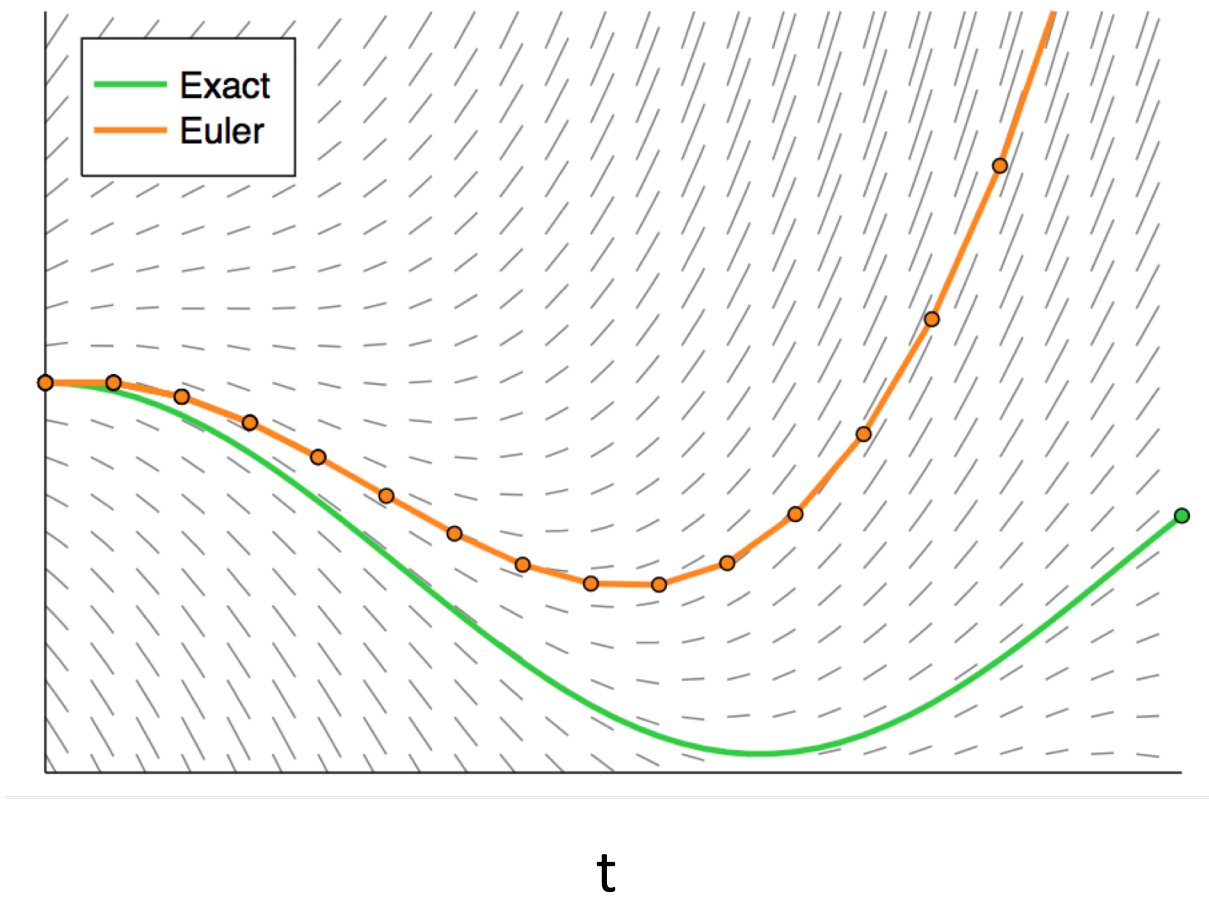
Background: ODE solvers

- Vector-valued h changes with depth
- Depth-derivative: $\frac{dh}{dt} = G(h(t), t)$
- Initial-value problem: given $\mathrm{h}(t_0)$, find

$$h(t_1) = h(t_0) + \int_{t_0}^{t_1} G(h(t), t, \theta)dt$$
$$= ODESolver(h(t_o), G, t0, t1, \theta)$$



- Euler approximates with small steps:
  $h(t_1) = h(t_0) + (t_1 - t_0)G(h(t_0), t_0)$

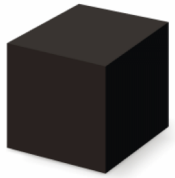


Source: https://en.wikipedia.org/wiki/Euler_method

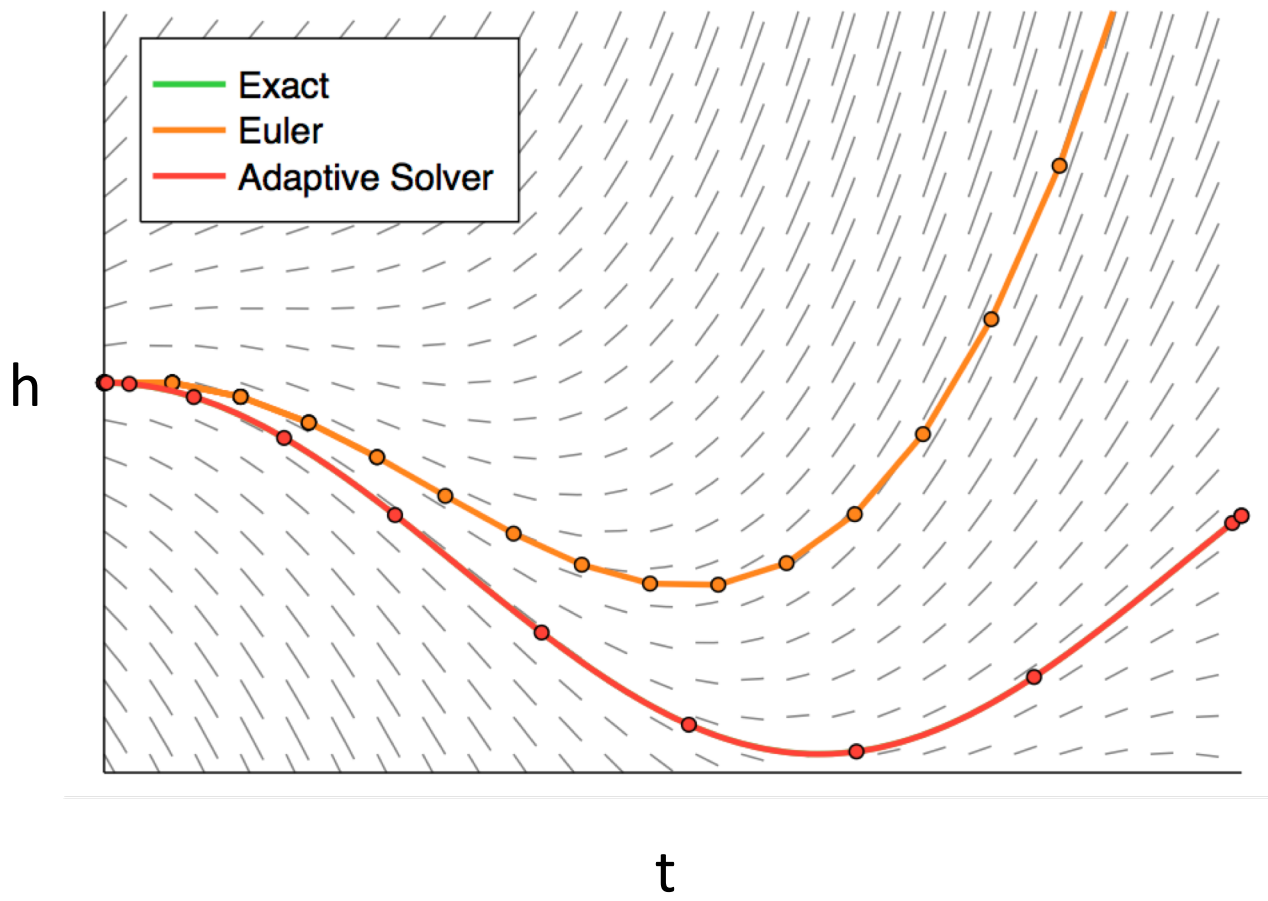
Background: ODE solvers

- Vector-valued h changes with depth
- Depth-derivative: $\frac{dh}{dt} = G(h(t), t)$
- Initial-value problem: given $\mathrm{h}(t_0)$, find

$$h(t_1) = h(t_0) + \int_{t_0}^{t_1} G(h(t), t, \theta)dt$$
$$= ODESolver(h(t_o), G, t0, t1, \theta)$$



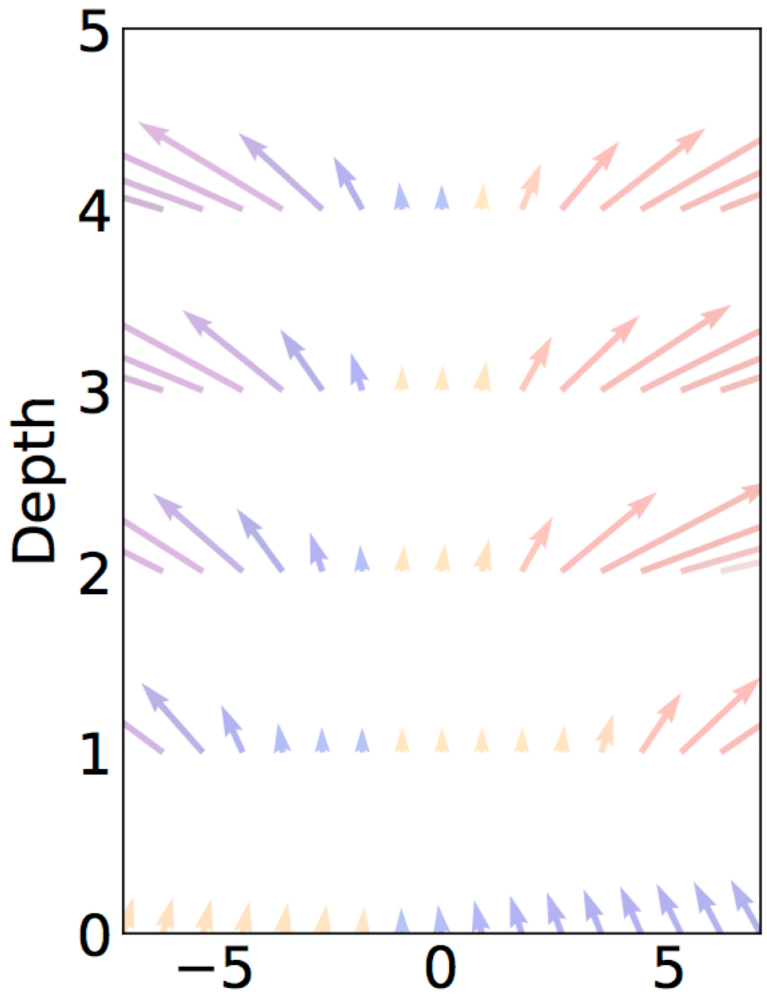
- Euler approximates with small steps:
  $h(t_1) = h(t_0) + (t_1 - t_0)G(h(t_0), t_0)$



Source: https://en.wikipedia.org/wiki/Euler_method

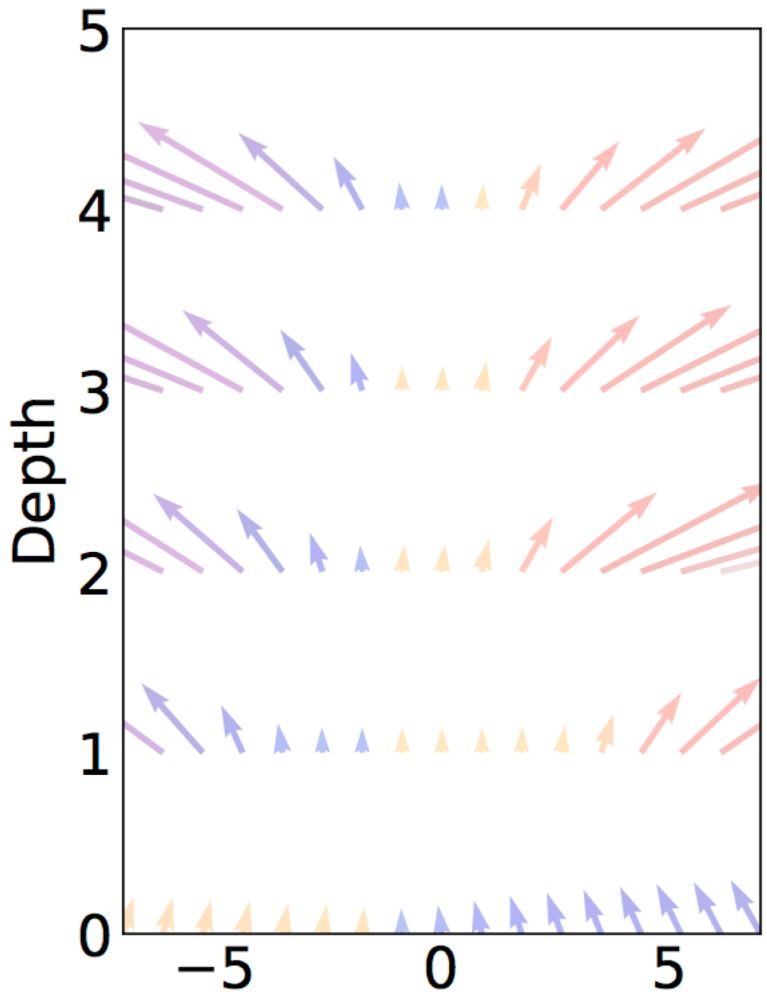
ResNet

```
def G(h, t, θ):
  return nnet(h, θ[t])
```



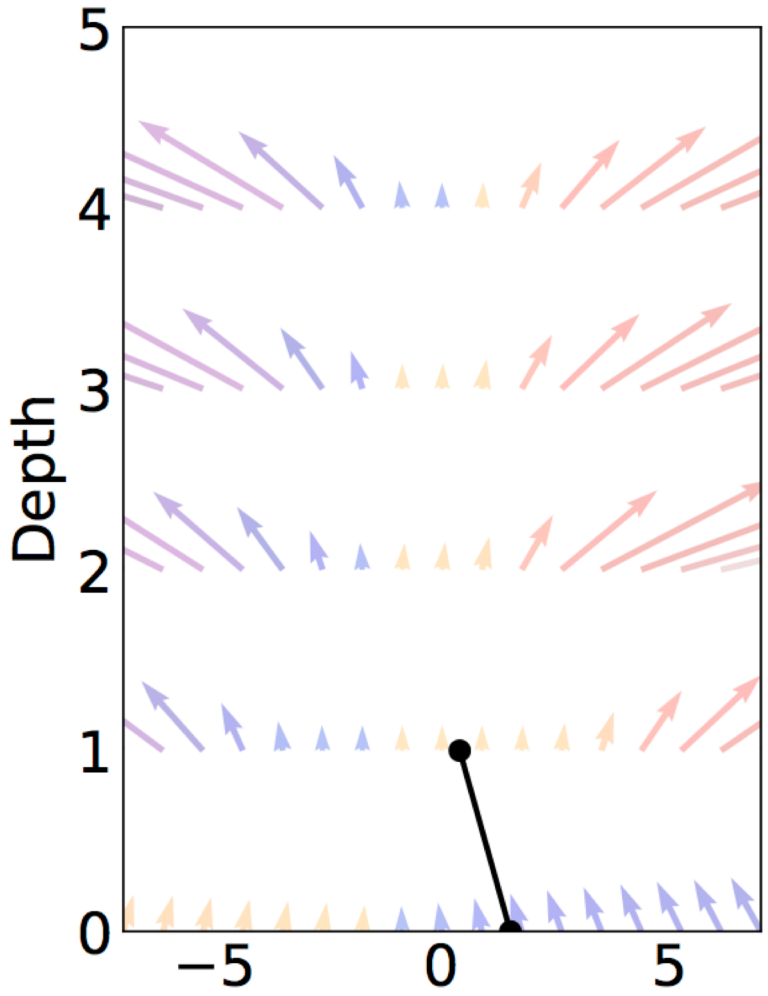
ResNet

```
def G(h, t, θ):
  return nnet(h, θ[t])

def resnet(h):
  for t in [1:T]:
    h = h + G(h, t, θ)
  return h
```



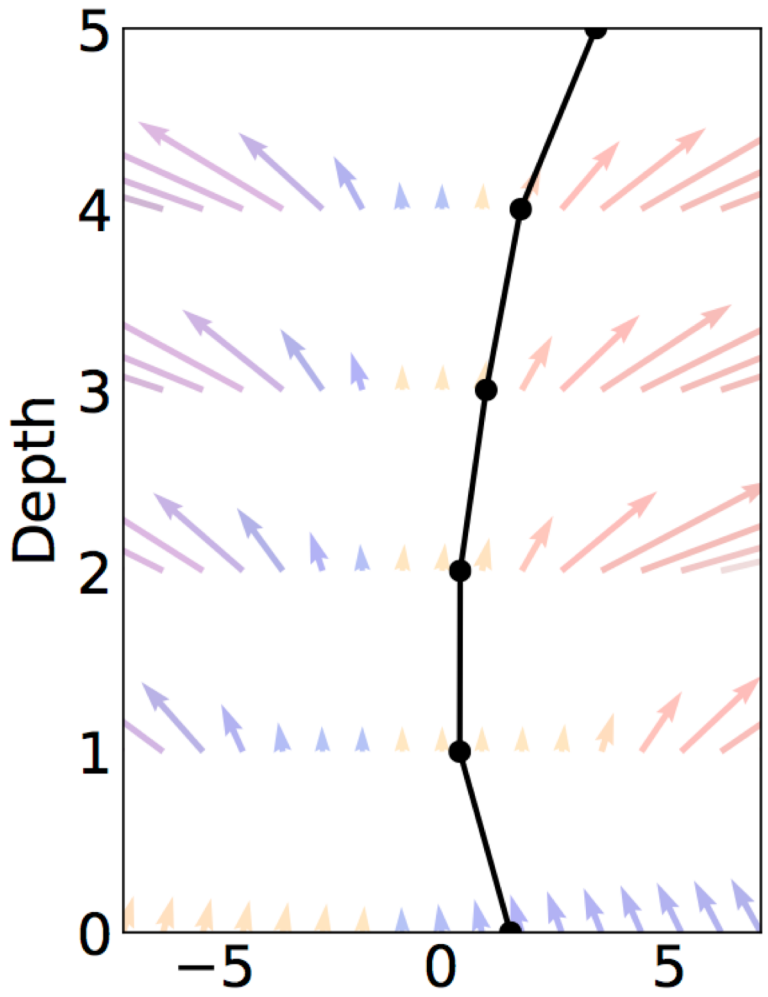
ResNet

```
def G(h, t, θ):
  return nnet(h, θ[t])

def resnet(h):
  for t in [1:T]:
    h = h + G(h, t, θ)
  return h
```



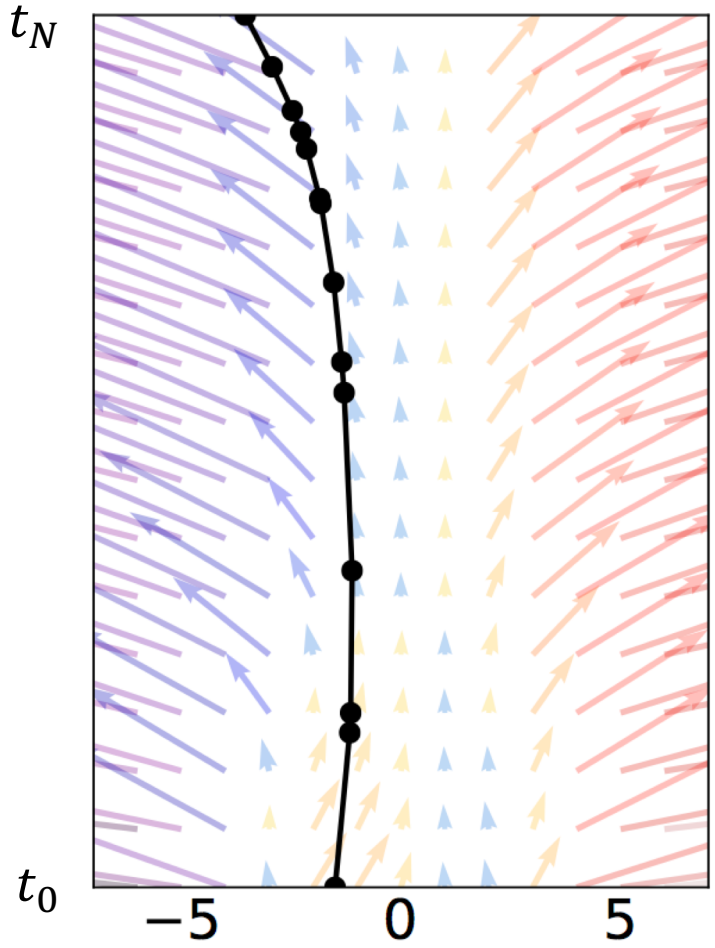
ResNet

```
def G(h, t, θ):
  return nnet(h, θ[t])

def resnet(h):
  for t in [1:T]:
    h = h + G(h, t, θ)
  return h
```

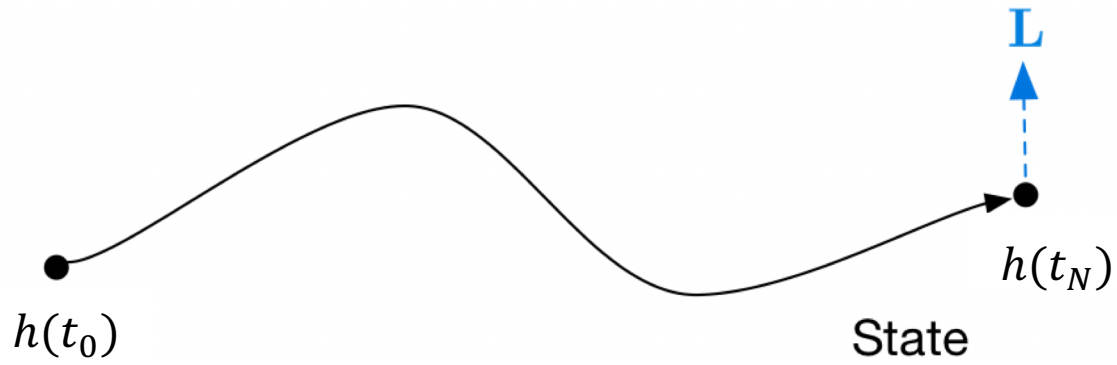


ODE-net

```
def G(h, t, θ):
  return nnet([h,t], θ)

def ODEnet(h, θ):
  return ODESolve(G, h, 0, 1, θ)
```



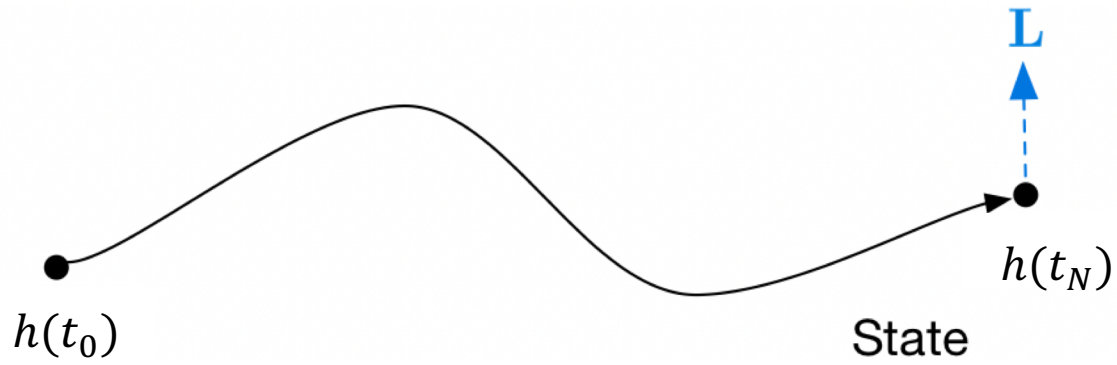
Training the beast



Loss function:

$$L(h(t_N)) = L(ODESolver(h(t_o), G, t_0, t_N, \theta))$$

Training the beast



Loss function:

$$L(h(t_N)) = L(ODESolver(h(t_o), G, t_0, t_N, \theta))$$

$$\frac{\partial L}{\partial h(t_0)}, \frac{\partial L}{\partial \theta}, \frac{\partial L}{\partial t_0}, \frac{\partial L}{\partial t_N} = ???$$

- Don't backprop through solver: High memory cost, extra numerical error
- Approximate the derivative, don't differentiate the approximation!

Backprop

Standard Backprop:

$$\frac{\partial \mathrm{h}_t}{\partial h_{t+1}} = \frac{\partial L}{\partial h_{t+1}} \frac{\partial G(h_t, \theta)}{\partial h_t}$$

$$\frac{\partial L}{\partial \theta_t} = \frac{\partial L}{\partial h_t} \frac{\partial G(h_t, \theta)}{\partial \theta_t}$$

Backprop

Standard Backprop:

$$\frac{\partial \mathrm{h}_t}{\partial h_{t+1}} = \frac{\partial L}{\partial h_{t+1}} \frac{\partial G(h_t, \theta)}{\partial h_t}$$

$$\frac{\partial L}{\partial \theta_t} = \frac{\partial L}{\partial h_t} \frac{\partial G(h_t, \theta)}{\partial \theta_t}$$

Adjoint sensitivity equations:

$$\mathrm{a(t)} = \frac{\partial L}{\partial h(t)}$$

Backprop

Standard Backprop:

$$\frac{\partial \mathrm{h}_t}{\partial h_{t+1}} = \frac{\partial L}{\partial h_{t+1}} \frac{\partial G(h_t, \theta)}{\partial h_t}$$

$$\frac{\partial L}{\partial \theta_t} = \frac{\partial L}{\partial h_t} \frac{\partial G(h_t, \theta)}{\partial \theta_t}$$

Adjoint sensitivity equations:

$$\mathrm{a(t)} = \frac{\partial L}{\partial h(t)}$$

Adjoint sensitivity:

$$\mathrm{a(t)} = \frac{\partial L}{\partial h(t)}$$

$$\frac{\partial a(t)}{\partial t} = a(t) \frac{\partial G(h(t), t, \theta)}{\partial h(t)}$$

$$\frac{\partial L}{\partial \theta_t} = \int_{t_N}^{t_0} a(t) \frac{\partial G(h(t), t, \theta)}{\partial \theta} dt$$

Backprop

Standard Backprop:

$$\frac{\partial \mathrm{h}_t}{\partial h_{t+1}} = \frac{\partial L}{\partial h_{t+1}} \frac{\partial G(h_t, \theta)}{\partial h_t}$$

$$\frac{\partial L}{\partial \theta_t} = \frac{\partial L}{\partial h_t} \frac{\partial G(h_t, \theta)}{\partial \theta_t}$$

Adjoint sensitivity equations:

$$\mathrm{a(t)} = \frac{\partial L}{\partial h(t)}$$

Adjoint sensitivity:

Use autodiff to define it!

$$\mathrm{a(t)} = \frac{\partial L}{\partial h(t)}$$

$$\frac{\partial a(t)}{\partial t} = a(t) \frac{\partial G(h(t), t, \theta)}{\partial h(t)}$$

$$\frac{\partial L}{\partial \theta_t} = \int_{t_N}^{t_0} a(t) \frac{\partial G(h(t), t, \theta)}{\partial \theta} dt$$

Training the beast

- Define this quantity as an adjoint state: $\mathrm{a(t)} = \frac{\partial L}{\partial h(t)}$
- Its dynamics are given by another ODE:

  Given $\mathrm{a}(t_N)$, find $a(t_0) = \int_{t_N}^{t_0} a(t) \frac{\partial G(h(t),t,\theta)}{\partial h(t)} dt$

$$\frac{\partial L}{\partial h(t_0)}$$

Training the beast

- Define this quantity as an adjoint state: $\mathrm{a}(\mathrm{t}) = \frac{\partial L}{\partial h(t)}$
- Its dynamics are given by another ODE:

  Given $\mathrm{a}(t_N)$, find $a(t_0) = \int_{t_N}^{t_0} a(t) \frac{\partial G(h(t),t,\theta)}{\partial h(t)} dt$

$$\frac{\partial L}{\partial h(t_0)}, \frac{\partial L}{\partial \theta}, \frac{\partial L}{\partial t_0}, \frac{\partial L}{\partial t_n} = ???$$

$$\frac{dL}{d\theta} = \int_{t_N}^{t_0} a(t) \frac{\partial G(h(t),t,\theta)}{\partial \theta} dt$$

$$\frac{dL}{dt_0} = \int_{t_N}^{t_0} a(t) \frac{\partial G(h(t),t,\theta)}{\partial t} dt$$

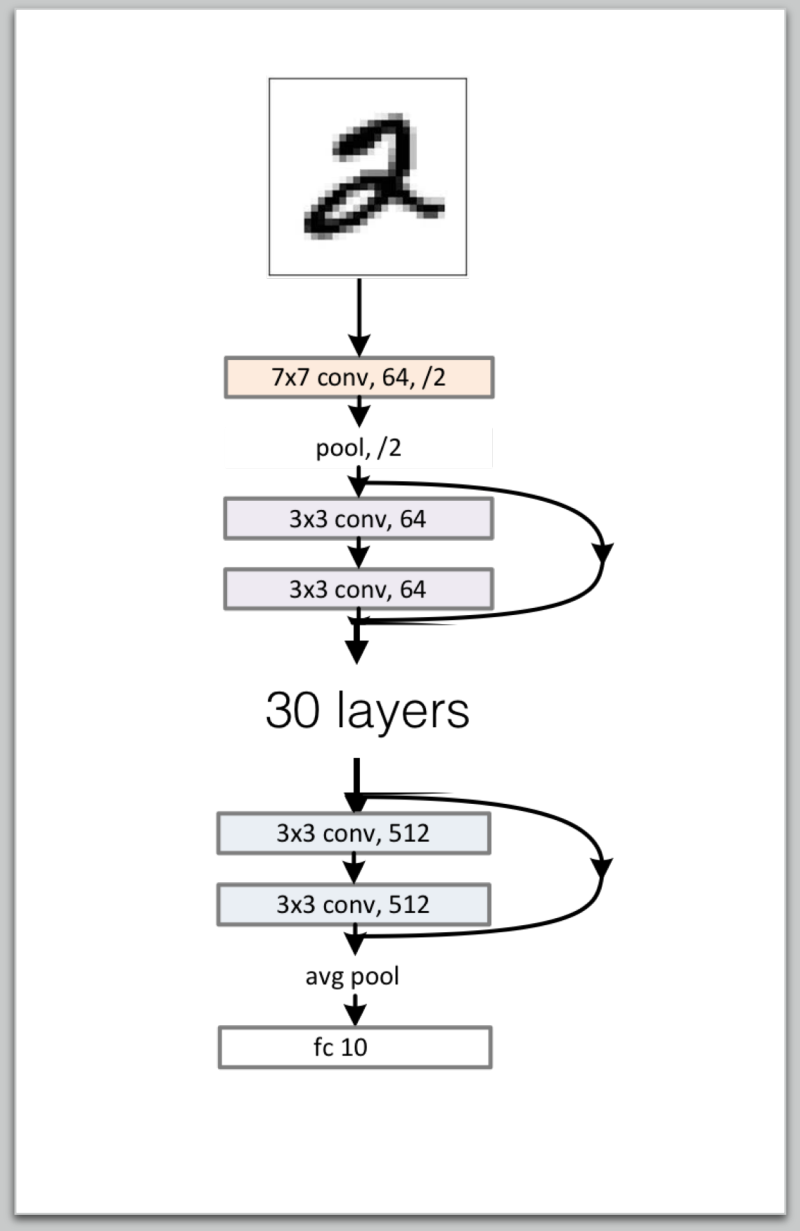
$$\frac{dL}{dt_N} = -a(t_N) \frac{\partial G(h(t),t,\theta)}{\partial t_N}$$

Break

Drop-in replacement for ResNet

	Test Error	# Params	Memory
1-Layer MLP	1.60%	0.24 M	-
ResNet	0.41%	0.60 M	O(L)

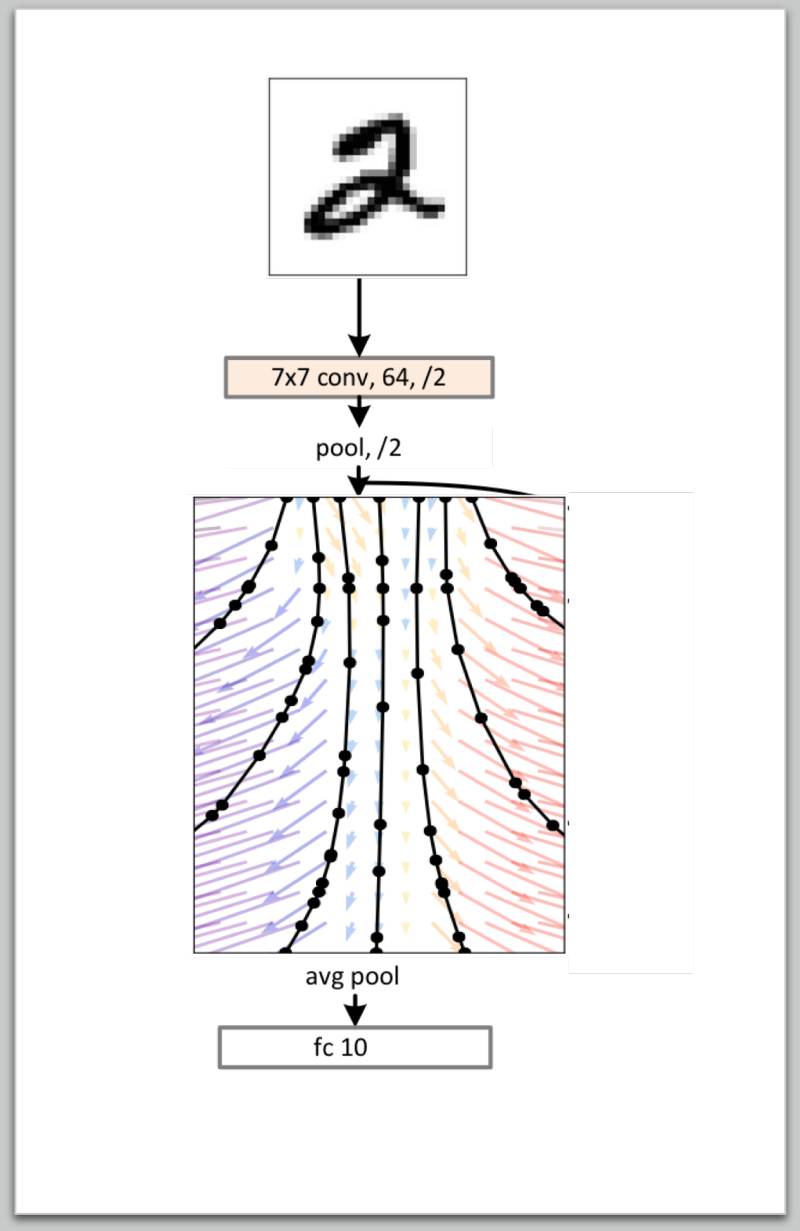
L: number of layers in ResNet



Drop-in replacement for ResNet

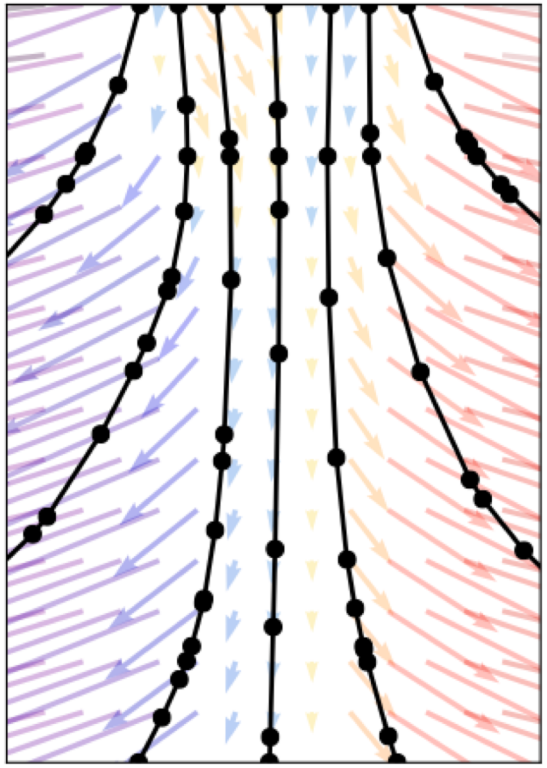
	Test Error	# Params	Memory
1-Layer MLP	1.60%	0.24 M	-
ResNet	0.41%	0.60 M	O(L)
ODE-Net	0.42%	0.22 M	O(1)

L: number of layers in ResNet



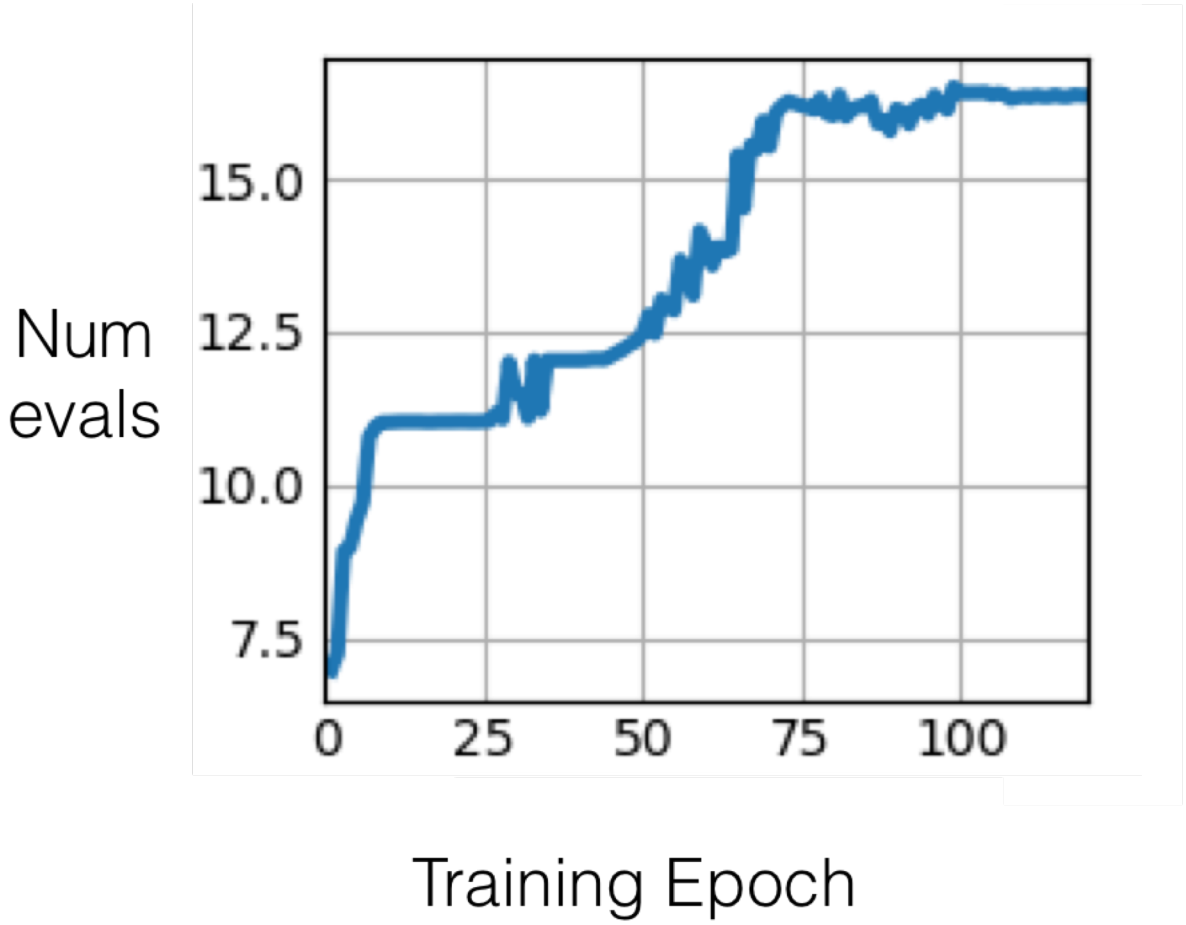
How deep are ODE-nets?

- No fixed number of layers
- Number of evaluations in ODE-nets ≈ Depth in ResNets



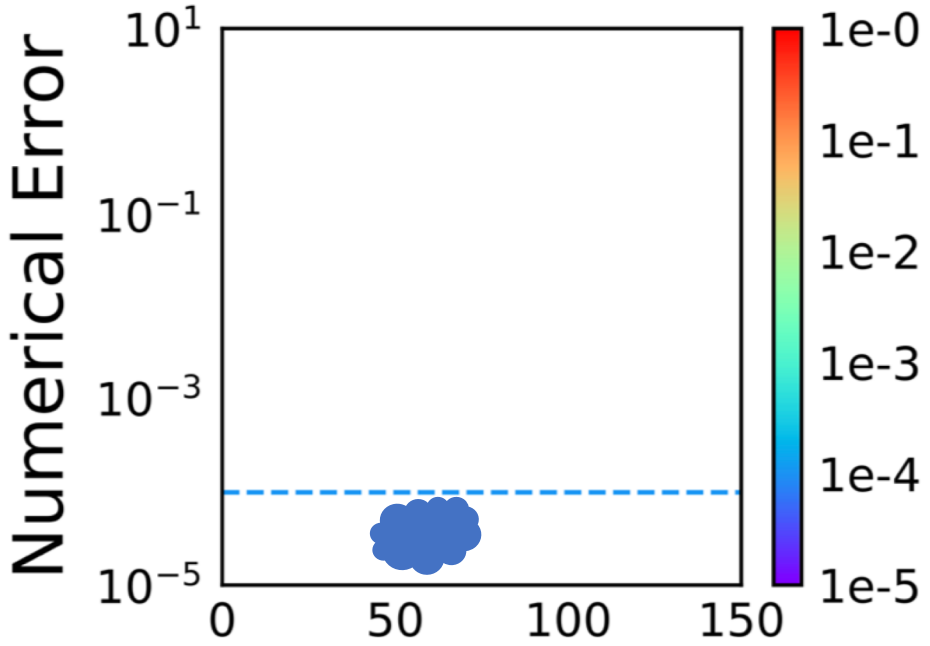
How deep are ODE-nets?

- No fixed number of layers
- Number of evaluations in ODE-nets ≈ Depth in ResNets
- Dynamics become more demanding to compute during training
- Downside: can't directly control the time-cost of the model during training



Explicit Error Control

ODESolver(h, x, t0, t1, θ, error)

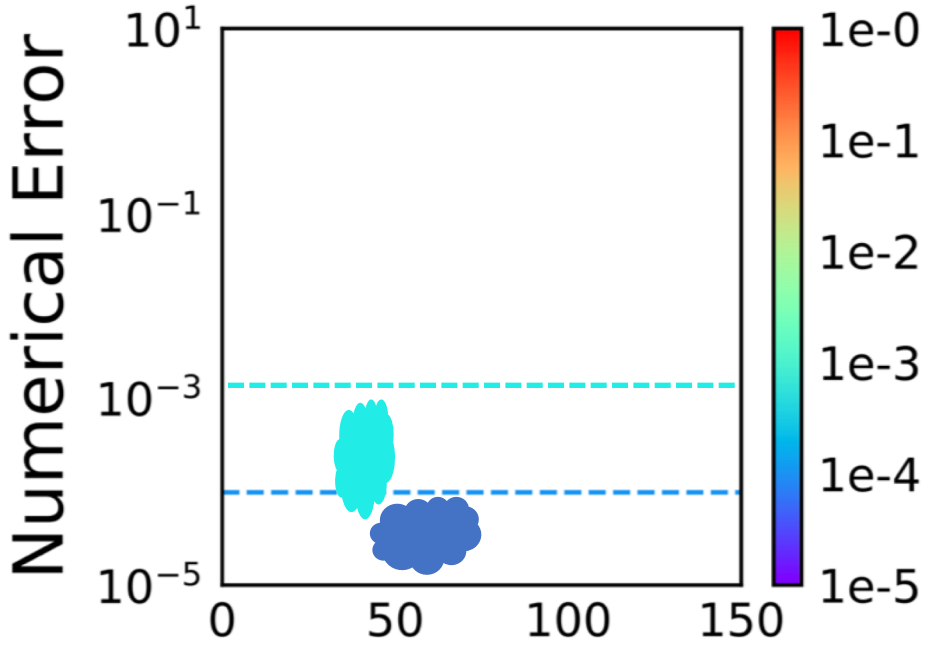


Number of dynamics evaluations

- Although we can't control the time cost of the model during training, we can control the time-cost at test time!!!!

Explicit Error Control

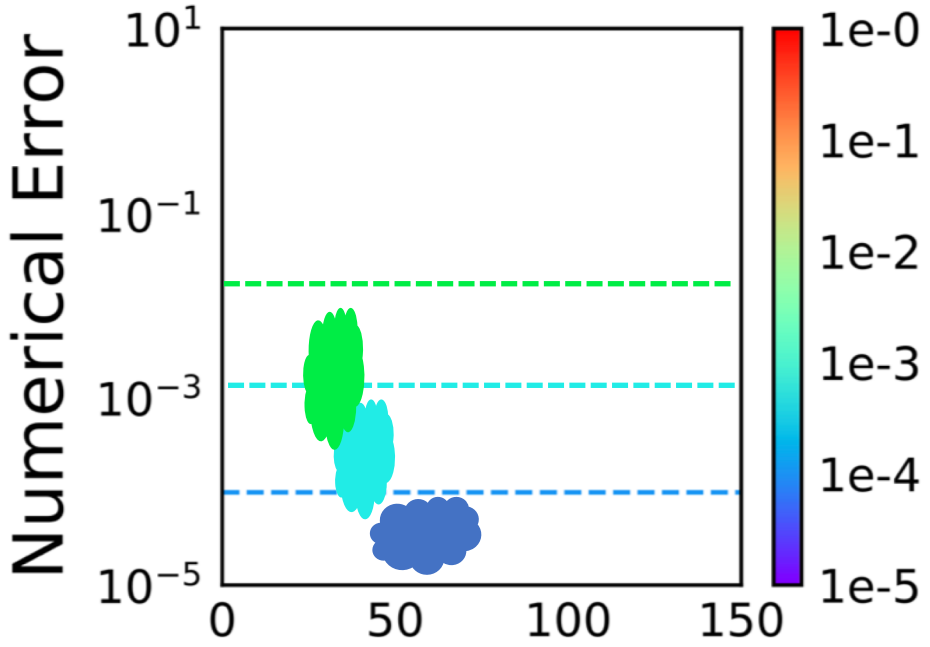
ODESolver(h, x, t0, t1, θ, error)



Number of dynamics evaluations

Explicit Error Control

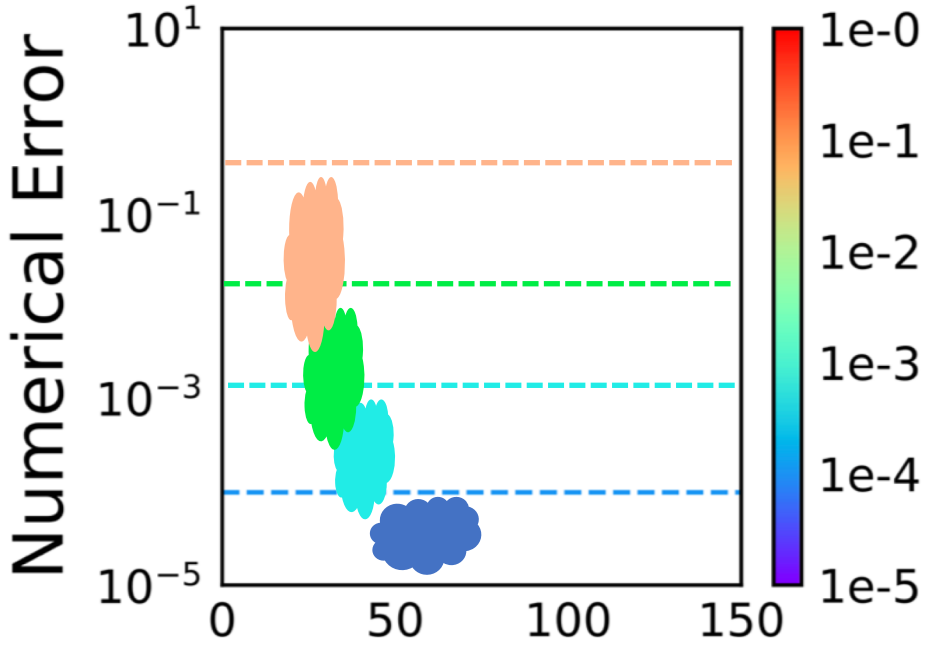
ODESolver(h, x, t0, t1, θ, error)



Number of dynamics evaluations

Explicit Error Control

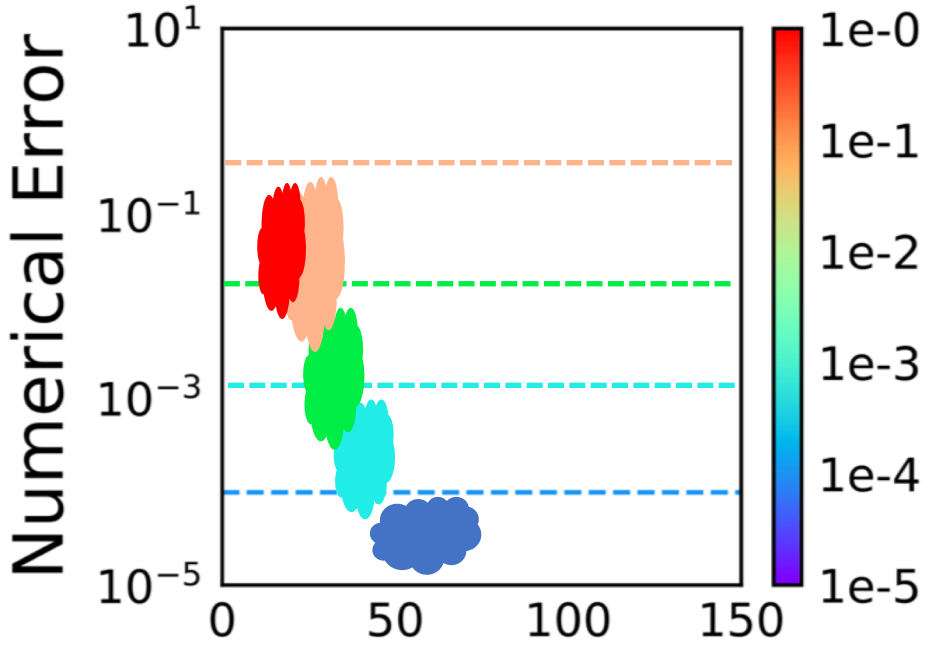
ODESolver(h, x, t0, t1, θ, error)



Number of dynamics evaluations

Explicit Error Control

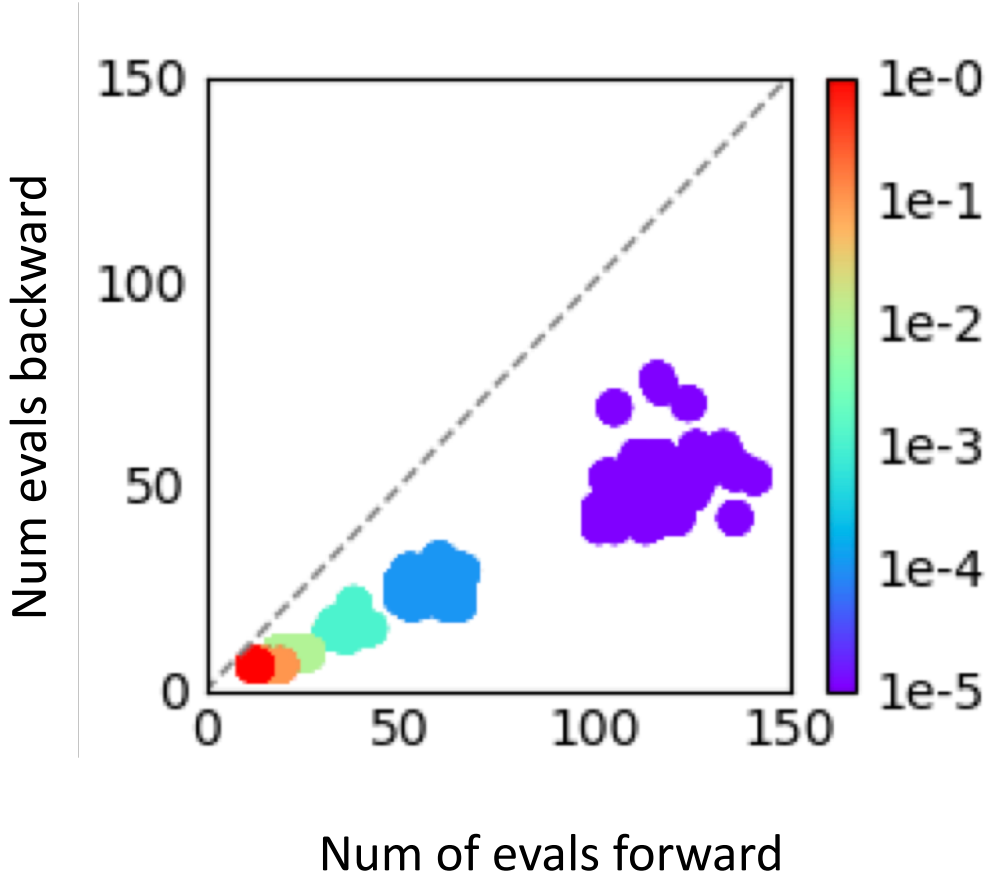
ODESolver(h, x, t0, t1, θ, error)



Number of dynamics evaluations

Reverse vs forward cost

Number of evaluations in the backward pass is roughly half of the forward (credit to adjoint sensitivity)



Major contributions

- New family of continuous-depth architectures (e.g. ResNets)
    1. No need to specific # of layers beforehand, just specific accuracy and it will train itself
    2. Adaptive computation
    3. Constant memory cost
- Other applications:

    Irregular time series data (medical history recorded at random time)

Discussions

- How to regularize the dynamics to need fewer evaluations?
- What basis for comparison would makes sense for the assessment of its utility generally?