1. Ordinary Differential Equations (ODEs)
   ○ Initial Value Problems
   ○ Numerical integration methods
   ○ Fundamental theorem of ODEs

2. Neural ODEs
   ○ Adjoint method
   ○ Applications

3. Recent research
1. Ordinary Differential Equations (ODEs)
   - Initial Value Problems
   - Numerical Integration methods
   - Fundamental theorem of ODEs

2. Neural ODEs

3. Recent research
Ordinary Differential Equations (ODEs)

1st order Ordinary Differential Equation:

\[
\frac{dx(t)}{dt} = f(x(t), t, \theta)
\]

\(x\) is a variable we are interested in,
\(t\) is (typically) time,
\(f\) is a function of \(x\) and \(t\), it is the differential,
\(\theta\) parameterizes \(f\) (optionally).
Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given;} \quad x(t_1) = ?$$

Many physical processes follow this template!
Ordinary Differential Equations (ODEs)

Initial value problem:

\[
\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ?
\]

Solution:

\[
x(t_1) = x(t_0) + \int_{t_0}^{t_1} f(x(t), t, \theta) \, dt
\]
Ordinary Differential Equations (ODEs)

Initial value problem:
\[ \frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ? \]

Solution:
\[ x(t_1) = x(t_0) + \int_{t_0}^{t_1} f(x(t), t, \theta) \, dt \]

Example:
\[ \frac{dx}{dt} = 2t; \quad x(0) = 2; \quad x(1) = ? \]
\[ \Rightarrow x(1) = x(0) + \int_0^1 2t \, dt \]
\[ = x(0) + (t^2)|_{t=1} - (t^2)|_{t=0} \]
\[ = 2 + 1^2 - 0^2 \]
\[ = 3 \]
Ordinary Differential Equations (ODEs)

Initial value problem:

\[
\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ?
\]

Solution:

\[
x(t_1) = x(t_0) + \int_{t_0}^{t_1} f(x(t), t, \theta) \, dt
\]

What if this cannot be analytically integrated?
Ordinary Differential Equations (ODEs)

Initial value problem:
\[
\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ?
\]

Solution:
\[
x(t_1) = x(t_0) + \int_{t_0}^{t_1} f(x(t), t, \theta) \, dt
\]

Example:
\[
\frac{dx}{dt} = 2xt; \quad x(0) = 3 \\
\Rightarrow \int \frac{1}{2x} \, dx = \int t \, dt \\
\Rightarrow \frac{1}{2} \log x = \frac{1}{2} t^2 + c_0 \\
\Rightarrow x(t) = ce^{t^2} \\
\Rightarrow x(0) = 3 \Rightarrow c = 2 \\
\therefore x(t) = 2e^{t^2} \\
\Rightarrow x(1) = 5.436
\]
Ordinary Differential Equations (ODEs)

Initial value problem:

\[
\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ?
\]

Solution:

\[
x(t_1) = x(t_0) + \int_{t_0}^{t_1} f(x(t), t, \theta) \, dt
\]

Approximations to \( \int_{t_0}^{t_1} f(x(t), t, \theta) \, dt \)

i.e. Numerical Integration:

- Euler method
- Runge-Kutta methods
- ...
Ordinary Differential Equations (ODEs)

Initial value problem:

\[
\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ?
\]

Solution:

\[
x(t_1) = x(t_0) + \int_{t_0}^{t_1} f(x(t), t, \theta) \, dt
\]

1st-order Runge-Kutta / Euler’s method:

\[
t_{n+1} = t_n + h \quad \rightarrow \quad \text{Step size } h
\]

\[
x(t_{n+1}) = x(t_n) + hf(x(t_n), t_n) \quad \rightarrow \quad \text{Update using derivative } f
\]

Initial value problem:

\[
\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ?
\]

Solution:

\[
x(t_1) = x(t_0) + \int_{t_0}^{t_1} f(x(t), t, \theta) \, dt
\]

1st-order Runge-Kutta / Euler’s method:

\[
t_{n+1} = t_n + h \\
x(t_{n+1}) = x(t_n) + hf(x(t_n), t_n)
\]

Example:

\[
\frac{dx}{dt} = f(x, t) = 2xt; \quad x(0) = 3; \quad x(1) = ?
\]

(Solution: \( x(t) = 2e^{t^2}; \quad x(1) = 5.436 \))

- \( h = 0.25 \)
- \( x(0.25) = x(0) + 0.25 \times f(x(0), 0) = 3 + 0.25 \times (2 \times 3 \times 0) = 3 \)
- \( x(0.5) = x(0.25) + 0.25 \times f(x(0.25), 0.25) = 3 + 0.25 \times (2 \times 3 \times 0.25) = 3.375 \)
- \( x(0.75) = x(0.5) + 0.25 \times f(x(0.5), 0.5) = 3.375 + 0.25 \times (2 \times 3.375 \times 0.5) = 4.21875 \)
- \( x(1) = x(0.75) + 0.25 \times f(x(0.75), 0.75) = 4.21875 + 0.25 \times (2 \times 4.21875 \times 0.75) = 5.8008 \)
Ordinary Differential Equations (ODEs)

Initial value problem:

\[
\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ?
\]

Solution:

\[
x(t_1) = x(t_0) + \int_{t_0}^{t_1} f(x(t), t, \theta) \, dt
\]

1st-order Runge-Kutta / Euler’s method:

\[
t_{n+1} = t_n + h
\]

\[
x(t_{n+1}) = x(t_n) + hf(x(t_n), t_n)
\]

Step size matters!

https://lpsa.swarthmore.edu/NumInt/NumIntFirst.html
Ordinary Differential Equations (ODEs)

Initial value problem:
\[ \frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given; } x(t_1) = ? \]

Solution:
\[ x(t_1) = x(t_0) + \int_{t_0}^{t_1} f(x(t), t, \theta) \, dt \]

1st-order Runge-Kutta / Euler’s method:
\[
\begin{align*}
  t_{n+1} &= t_n + h \\
  s_1 &= f(x(t_n), t_n) \\
  x(t_{n+1}) &= x(t_n) + h s_1
\end{align*}
\]
Ordinary Differential Equations (ODEs)

Initial value problem:

\[ \frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ? \]

Solution:

\[ x(t_1) = x(t_0) + \int_{t_0}^{t_1} f(x(t), t, \theta) \, dt \]

2nd-order Runge-Kutta method:

\[ t_{n+1} = t_n + h \]
\[ s_1 = f(x(t_n), t_n) \]
\[ s_2 = f(x(t_n + \frac{h}{2}), t_n + \frac{h}{2}) = f(x(t_n) + \frac{h}{2} s_1, t_n + \frac{h}{2}) \]
\[ x(t_{n+1}) = x(t_n) + hs_2 \]
Ordinary Differential Equations (ODEs)

Initial value problem:
\[
\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given; } x(t_1) = ?
\]

Solution:
\[
x(t_1) = x(t_0) + \int_{t_0}^{t_1} f(x(t), t, \theta) \, dt
\]

4th-order Runge-Kutta method:
\[
t_{n+1} = t_n + h
\]
\[
s_1 = f(x(t_n), t_n)
\]
\[
s_2 = f(x(t_n) + \frac{h}{2} s_1, t_n + \frac{h}{2})
\]
\[
s_3 = f(x(t_n) + \frac{h}{2} s_2, t_n + \frac{h}{2})
\]
\[
s_4 = f(x(t_n) + hs_3, t_n + h)
\]
\[
x(t_{n+1}) = x(t_n) + \frac{h}{6} (s_1 + 2s_2 + 2s_3 + s_4)
\]

Default ODE solver used in MATLAB:
Ordinary Differential Equations (ODEs)

Initial value problem:

\[ \frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given; } x(t_1) = ? \]

Solution:

\[ x(t_1) = x(t_0) + \int_{t_0}^{t_1} f(x(t), t, \theta) \, dt \]

Many other ODE solvers to choose from!
Ordinary Differential Equations (ODEs)

Initial value problem:
\[
\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given;} \quad x(t_1) = ?
\]

Solution:
\[
x(t_1) = x(t_0) + \int_{t_0}^{t_1} f(x(t), t, \theta) \, dt
\]

Considerations to choose an ODE solver:
- Stiff v/s Non-stiff ODE
- # of calculations per iteration
- Implicit v/s Explicit solver
- Single-step size v/s Multi-step size (adaptive)

Many other ODE solvers to choose from!

Ordinary Differential Equations (ODEs)

Initial value problem:

\[
\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given; } x(t_1) = ?
\]

Solution:

\[
x(t_1) = x(t_0) + \int_{t_0}^{t_1} f(x(t), t, \theta) \, dt
\]

\[
x(t_1) = \text{ODESolve}(f(x(t), t, \theta), x(t_0), t_0, t_1)
\]

Final time

Initial time

Initial value

Differential

Any ODE solver of our choice!
Ordinary Differential Equations (ODEs)

Initial value problem:

\[
\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given; } x(t_1) = ?
\]

Solution:

\[
x(t_1) = \text{ODESolve}( f(x(t), t, \theta), x(t_0), t_0, t_1 )
\]
Ordinary Differential Equations (ODEs)

Initial value problem:

\[ \frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given; } x(t_1) = ? \]

Solution:

\[ x(t_1) = \text{ODESolve}( f(x(t), t, \theta), x(t_0), t_0, t_1 ) \]

Fundamental Theorem of ODEs

Suppose \( f \) is continuously differentiable.

Then, the solution to the initial value problem is unique!

http://faculty.bard.edu/belk/math213/InitialValueProblems.pdf
Ordinary Differential Equations (ODEs)

Initial value problem:
\[
\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given; } x(t_1) = ?
\]

Solution:
\[
x(t_1) = \text{ODESolve}( f(x(t), t, \theta), x(t_0), t_0, t_1 )
\]

Fundamental Theorem of ODEs

Suppose \( f \) is continuously differentiable.

1. The solution curves for this differential equation completely fill the plane, and
2. Solution curves for different solutions do not intersect.

http://faculty.bard.edu/belk/math213/InitialValueProblems.pdf
Initial value problem:
\[
\frac{dx(t)}{dt} = f(x(t), t, \theta); \ x(t_0) \text{ is given; } x(t_1) = ?
\]

Solution:
\[
x(t_1) = \text{ODESolve}( f(x(t), t, \theta), x(t_0), t_0, t_1)
\]

Fundamental Theorem of ODEs

Suppose \( f \) is continuously differentiable.

1. The solution curves for this differential equation completely fill the plane.
2. Solution curves for different solutions do not intersect.

Geometrically, \( x(t) \) is a flow!

http:// faculty.bard.edu/belk/math213/InitialValueProblems.pdf

https://openreview.net/pdf?id=B1e9Y2NYvS
1. Ordinary Differential Equations (ODEs)
   - Initial Value Problems
   - Numerical Integration methods
   - Fundamental theorem of ODEs

2. Neural ODEs (Chen et al., 2018)
   - Adjoint method
   - Applications

3. Recent research
Neural ODEs (Chen et al., 2018)

Initial value problem:

\[ \frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ? \]

Solution:

\[ x(t_1) = \text{ODESolve}( f(x(t), t, \theta), x(t_0), t_0, t_1 ) \]

\[ f \text{ is a neural network!} \]

Paradigm shift: whereas earlier \( f \) was pre-defined/hand-designed according to the domain, here we would like to estimate an \( f \) that suits our objective.

Neural ODEs (Chen et al., 2018)

ODEs

$$\frac{dx(t)}{dt} = f(x(t), t, \theta)$$

$$x_{n+1} = x_n + h f(x_n, t_n, \theta)$$

Euler discretization

Vector notation

Residual networks

$$x_{l+1} = \text{ResBlock}(x_l, \theta)$$

$$x_{l+1} = x_l + g(x_1, \theta)$$

Skip connection


Neural ODEs (Chen et al., 2018)

\[
\frac{dx(t)}{dt} = f(x(t), t, \theta)
\]

**Euler discretization**

\[
x_{n+1} = x_n + h f(x_n, t_n, \theta)
\]

Forward propagation:

\[
x(t_1) = \text{ODESolve}(f(x(t), t, \theta), x(t_0), t_0, t_1)
\]

**Residual networks**

\[
x_{l+1} = \text{ResBlock}(x_l, \theta)
\]

\[
x_{l+1} = x_l + g(x_1, \theta)
\]

\[
y_{pred} = \text{ResNet}(x)
\]


Neural ODEs (Chen et al., 2018)

ODEs

\[
\frac{dx(t)}{dt} = f(x(t), t, \theta)
\]

Euler discretization

\[
x_{n+1} = x_n + h f(x_n, t_n, \theta)
\]

Forward propagation:
\[
x(t_1) = \text{ODESolve}(f(x(t), t, \theta), x(t_0), t_0, t_1)
\]

Residual networks

\[
x_{l+1} = \text{ResBlock}(x_l, \theta)
\]

\[
x_{l+1} = x_l + g(x_l, \theta)
\]

Skip connection

\[
y_{\text{pred}} = \text{ResNet}(x)
\]

Stacked ResBlocks

\[
L(y_{\text{pred}}) \rightarrow \frac{\partial L}{\partial \theta}
\]

Update \( \theta \) to reduce \( L \)

How to compute this?

Update \( \theta \) to reduce \( L \)


Neural ODEs (Chen et al., 2018)

ODEs

\[ \frac{dx(t)}{dt} = f(x(t), t, \theta) \]

\[ x_{n+1} = x_n + h f(x_n, t_n, \theta) \]

Euler discretization

Forward propagation:

\[ x(t_1) = \text{ODESolve}( f(x(t), t, \theta), x(t_0), t_0, t_1 ) \]

Back-propagate through the ODE Solver!

Update \( \theta \) to reduce \( L \)

Neural ODEs (Chen et al., 2018)

**ODEs**

\[
\frac{dx(t)}{dt} = f(x(t), t, \theta)
\]

Euler discretization

\[
x_{n+1} = x_n + hf(x_n, t_n, \theta)
\]

Forward propagation:

\[
x(t_1) = \text{ODESolve}(f(x(t), t, \theta), x(t_0), t_0, t_1)
\]

Back-propagate through the ODE Solver!

High memory cost - need to save all activations of all iterations of ODESolve.

**Can we do better?**

Yes.

Neural ODEs (Chen et al., 2018)

\[ L(\text{ODESolve}(f(x(t), t, \theta), x(t_0), t_0, t_1)) \rightarrow \frac{\partial L}{\partial \theta} \]

Adjoint method (Pontryagin et al., 1962)

adjoint \( a(t) = \frac{\partial L}{\partial x} ; \frac{da}{dt} = -a(t)^\top \frac{\partial f(x(t), t, \theta)}{\partial x} \)

\[
\frac{\partial L}{\partial \theta} = -\int_{t_1}^{t_0} a(t)^\top \frac{\partial f(x(t), t, \theta)}{\partial \theta} \, dt
\]

Neural ODEs (Chen et al., 2018)

\[ L(\text{ODESolve}( f(x(t), t, \theta), x(t_0), t_0, t_1 )) \rightarrow \frac{\partial L}{\partial \theta} \]

**Adjoint method** (Pontryagin et al., 1962)

adjoint \( a(t) = \frac{\partial L}{\partial x} \); \( \frac{da}{dt} = -a(t)^\top \frac{\partial f(x(t), t, \theta)}{\partial x} \)

\[ \frac{\partial L}{\partial \theta} = - \int_{t_0}^{t_1} a(t)^\top \frac{\partial f(x(t), t, \theta)}{\partial \theta} \, dt \]

*We need \( a(t) \)*

Neural ODEs (Chen et al., 2018)

\[ L(\text{ODESolve}( f(x(t), t, \theta), x(t_0), t_0, t_1 )) \rightarrow \frac{\partial L}{\partial \theta} \]

**Adjoint method** (Pontryagin et al., 1962)

adjoint \( a(t) = \frac{\partial L}{\partial x} ; \frac{da}{dt} = -a(t)^\top \frac{\partial f(x(t), t, \theta)}{\partial x} \)

Forward propagation: \( x(t_1) = \text{ODESolve}( f(x(t), t, \theta), x(t_0), t_0, t_1 ) \)

Neural ODEs (Chen et al., 2018)

\[ L(\text{ODESolve}( f(x(t), t, \theta), x(t_0), t_0, t_1 )) \rightarrow \frac{\partial L}{\partial \theta} \]

**Adjoint method** (Pontryagin et al., 1962)

Adjoint \( a(t) = \frac{\partial L}{\partial x} \); \( \frac{da}{dt} = -a(t)^\top \frac{\partial f(x(t), t, \theta)}{\partial x} \)

Forward propagation: \( x(t_1) = \text{ODESolve}( f(x(t), t, \theta), x(t_0), t_0, t_1 ) \Rightarrow a(t_1) = \frac{\partial L}{\partial x(t_1)} \)

*Can be computed using autodiff*

Adjoint method (Pontryagin et al., 1962)

adjoint $\mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{x}}$; $\frac{d\mathbf{a}}{dt} = -\mathbf{a}(t) \top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \mathbf{x}}$

Forward propagation: $\mathbf{x}(t_1) = \text{ODESolve} \left( f(\mathbf{x}(t), t, \theta), \mathbf{x}(t_0), t_0, t_1 \right) \Rightarrow \mathbf{a}(t_1) = \frac{\partial L}{\partial \mathbf{x}(t_1)}$

Can be computed using autodiff

We can use $\mathbf{a}(t_1)$ as initial value, and integrate backwards from $t_1$ to $t$ to get $\mathbf{a}(t)$.

We'll use $t_0$ as a proxy for $t$
Neural ODEs (Chen et al., 2018)

$$L(\text{ODESolve}( f(x(t), t, \theta), x(t_0), t_0, t_1 )) \rightarrow \frac{\partial L}{\partial \theta}$$

**Adjoint method** (Pontryagin et al., 1962)

adjoint $a(t) = \frac{\partial L}{\partial x}$; \[ \frac{da}{dt} = -a(t)^\top \frac{\partial f(x(t), t, \theta)}{\partial x} \]

Forward propagation: $x(t_1) = \text{ODESolve}( f(x(t), t, \theta), x(t_0), t_0, t_1 ) \Rightarrow a(t_1) = \frac{\partial L}{\partial x(t_1)}$

\[ \Rightarrow a(t_0) = \frac{\partial L}{\partial x(t_0)} = \text{ODESolve}( -a(t)^\top \frac{\partial f(x(t), t, \theta)}{\partial x}, \frac{\partial L}{\partial x(t_1)}, t_1, t_0 ) \]

Initial value

Backward integration from t1 to t0

Adjoint method (Pontryagin et al., 1962)

adjoint \( \mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{x}} \); \( \frac{d\mathbf{a}}{dt} = -\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \mathbf{x}} \)

Forward propagation: \( \mathbf{x}(t_1) = \text{ODESolve}( f(\mathbf{x}(t), t, \theta), \mathbf{x}(t_0), t_0, t_1 ) \) \( \Rightarrow \mathbf{a}(t_1) = \frac{\partial L}{\partial \mathbf{x}(t_1)} \)

\( \Rightarrow \mathbf{a}(t_0) = \frac{\partial L}{\partial \mathbf{x}(t_0)} = \text{ODESolve}( -\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \mathbf{x}}, \frac{\partial L}{\partial \mathbf{x}(t_1)}, t_1, t_0 ) \)

Vector-Jacobian Product
(can be efficiently evaluated by autodiff)

Neural ODEs (Chen et al., 2018)

\[ L(\text{ODESolve}(f(x(t), t, \theta), x(t_0), t_0, t_1)) \rightarrow \frac{\partial L}{\partial \theta} \]

**Adjoint method** (Pontryagin et al., 1962)

adjoint \( a(t) = \frac{\partial L}{\partial x} \); \( \frac{da}{dt} = -a(t)^\top \frac{\partial f(x(t), t, \theta)}{\partial x} \)

Forward propagation: \( x(t_1) = \text{ODESolve}(f(x(t), t, \theta), x(t_0), t_0, t_1) \Rightarrow a(t_1) = \frac{\partial L}{\partial x(t_1)} \)

\[ \Rightarrow a(t_0) = \frac{\partial L}{\partial x(t_0)} = \text{ODESolve}(-a(t)^\top \frac{\partial f(x(t), t, \theta)}{\partial x}, \frac{\partial L}{\partial x(t_1)}, t_1, t_0) \]

(And we don’t want to have saved \( x(t) \) in memory from forward-prop)

Adjoint method (Pontryagin et al., 1962)

\[
\begin{align*}
\text{adjoint } a(t) &= \frac{\partial L}{\partial x(t)} ; \quad \frac{da}{dt} = -a(t) \top \frac{\partial f(x(t), t, \theta)}{\partial x} \\
\text{Forward propagation: } x(t_1) &= \text{ODESolve}( f(x(t), t, \theta), x(t_0), t_0, t_1 ) \Rightarrow a(t_1) = \frac{\partial L}{\partial x(t_1)} \\
&x(t_0) = \text{ODESolve}( f(x(t), t, \theta), x(t_1), t_1, t_0 ) \\
\Rightarrow a(t_0) &= \frac{\partial L}{\partial x(t_0)} = \text{ODESolve}( -a(t) \top \frac{\partial f(x(t), t, \theta)}{\partial x}, \frac{\partial L}{\partial x(t_1)}, t_1, t_0 )
\end{align*}
\]

Neural ODEs (Chen et al., 2018)

\[
L(\text{ODESolve}(f(x(t), t, \theta), x(t_0), t_0, t_1)) \rightarrow \frac{\partial L}{\partial \theta}
\]

**Adjoint method** (Pontryagin et al., 1962)

Adjoint \( a(t) = \frac{\partial L}{\partial x} \); \( \frac{da}{dt} = -a(t)^T \frac{\partial f(x(t), t, \theta)}{\partial x} \)

Forward propagation: \( x(t_1) = \text{ODESolve}(f(x(t), t, \theta), x(t_0), t_0, t_1) \Rightarrow a(t_1) = \frac{\partial L}{\partial x(t_1)} \)

Back-propagation:

\[
x(t_0) = \text{ODESolve}(f(x(t), t, \theta), x(t_1), t_1, t_0)
\]

\[\Rightarrow a(t_0) = \frac{\partial L}{\partial x(t_0)} = \text{ODESolve}(-a(t)^T \frac{\partial f(x(t), t, \theta)}{\partial x}, \frac{\partial L}{\partial x(t_1)}, t_1, t_0)\]

\[
\therefore \frac{\partial L}{\partial \theta} = -\int_{t_1}^{t_0} a(t)^T \frac{\partial f(x(t), t, \theta)}{\partial \theta} dt
\]

Neural ODEs (Chen et al., 2018)

\[ L(\text{ODESolve}(f(x(t), t, \theta), x(t_0), t_0, t_1)) \rightarrow \frac{\partial L}{\partial \theta} \]

**Adjoint method** (Pontryagin et al., 1962)

adjoint \[ a(t) = \frac{\partial L}{\partial x} ; \frac{da}{dt} = -a(t) \top \frac{\partial f(x(t), t, \theta)}{\partial x} \]

Forward propagation: \[ x(t_1) = \text{ODESolve}(f(x(t), t, \theta), x(t_0), t_0, t_1) \Rightarrow a(t_1) = \frac{\partial L}{\partial x(t_1)} \]

Back-propagation:

\[ x(t_0) = \text{ODESolve}(f(x(t), t, \theta), x(t_1), t_1, t_0) \]

\[ \Rightarrow a(t_0) = \frac{\partial L}{\partial x(t_0)} = \text{ODESolve}(-a(t) \top \frac{\partial f(x(t), t, \theta)}{\partial x}, \frac{\partial L}{\partial x(t_1)}, t_1, t_0) \]

\[ \therefore \frac{\partial L}{\partial \theta} = - \int_{t_1}^{t_0} a(t) \top \frac{\partial f(x(t), t, \theta)}{\partial \theta} \, dt = \text{ODESolve}(-a(t) \top \frac{\partial f(x(t), t, \theta)}{\partial \theta}, 0_|\theta|, t_1, t_0) \]

Initial value is 0

Neural ODEs (Chen et al., 2018)

\[
L(\text{ODESolve}( f(\mathbf{x}(t), t, \theta), \mathbf{x}(t_0), t_0, t_1 )) \rightarrow \frac{\partial L}{\partial \theta}
\]

Adjoint method (Pontryagin et al., 1962)

adjoint \( \mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{x}} \); \( \frac{d\mathbf{a}}{dt} = -\mathbf{a}(t) \top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \mathbf{x}} \)

Forward propagation: \( \mathbf{x}(t_1) = \text{ODESolve}( f(\mathbf{x}(t), t, \theta), \mathbf{x}(t_0), t_0, t_1 ) \Rightarrow \mathbf{a}(t_1) = \frac{\partial L}{\partial \mathbf{x}(t_1)} \)

Back-propagation:

\[
\Rightarrow \mathbf{a}(t_0) = \frac{\partial L}{\partial \mathbf{x}(t_0)} = \text{ODESolve}( -\mathbf{a}(t) \top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \theta}, \mathbf{x}(t_1), t_1, t_0 )
\]

\[
\Rightarrow \frac{\partial L}{\partial \theta} = - \int_{t_1}^{t_0} \mathbf{a}(t) \top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \theta} \, dt = \text{ODESolve}( -\mathbf{a}(t) \top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \theta}, 0_{n \times n}, t_1, t_0 )
\]

**Neural ODEs (Chen et al., 2018)**

**Forward propagation:**
\[
x(t_1) = \text{ODESolve}( f(x(t), t, \theta), x(t_0), t_0, t_1 )
\]
Compute \( L(x(t_1)) \).
\[
a(t_1) = \frac{\partial L}{\partial x(t_1)}
\]

**Back-propagation:**
\[
\begin{bmatrix}
x(t_0) \\
\frac{\partial L}{\partial x(t_0)} \\
\frac{\partial L}{\partial \theta}
\end{bmatrix}
= \text{ODESolve}
\begin{bmatrix}
f(x(t), t, \theta) \\
-\mathbf{a}(t) \top \frac{\partial f(x(t), t, \theta)}{\partial x} \\
-\mathbf{a}(t) \top \frac{\partial f(x(t), t, \theta)}{\partial \theta}
\end{bmatrix}
\begin{bmatrix}
x(t_1) \\
\frac{\partial L}{\partial x(t_1)} \\
0_{|\theta|}
\end{bmatrix}
\]
Update \( \theta \) to reduce \( L \)

Neural ODEs (Chen et al., 2018)

Neural ODE \[ f(x(t), t, \theta) \]

\[ x(t_0) \rightarrow f(x(t), t, \theta) \rightarrow x(t_1) \]

Neural ODEs (Chen et al., 2018)

Neural ODEs describe a homeomorphism (flow).

- They preserve dimensionality.
- They form non-intersecting trajectories.


https://openreview.net/pdf?id=B1e9Y2NYvS
Neural ODEs (Chen et al., 2018)

Neural ODEs are reversible models!
Just integrate forward/backward in time.
Neural ODEs (Chen et al., 2018)

Applications

- Supervised Learning
- Continuous Normalizing Flows
- Generative Latent Models

Neural ODEs (Chen et al., 2018)

Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

ODE-Net:

\[ x(t_0) \xrightarrow{f(x(t), t, \theta)} x(t_1) \xrightarrow{\text{Linear}} y_{\text{pred}} \xrightarrow{L} y \]

~ Replacement for ResNets
Neural ODEs (Chen et al., 2018)

Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

Target distribution

(such as real image manifold)

Sample from target distribution (such as an image)

Noise distribution

Sample from noise distribution (such as Gaussian)

Neural ODE

\[ x(t_0) \xrightarrow{f(x(t), t, \theta)} x(t_1) \]
Neural ODEs (Chen et al., 2018)

Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

Likelihood estimation

using Change of Variables formula

\[ \mathbf{x}_1 = g(\mathbf{x}_0) \Rightarrow \log p(\mathbf{x}_1) = \log p(\mathbf{x}_0) - \log |\det \frac{\partial g}{\partial \mathbf{x}_0}| \]

Train \( f \) to maximize the likelihood of the samples from target distribution \( \log p(\mathbf{x}_1) \), by computing \( \mathbf{x}(t_0) \) using the Neural ODE with \( \mathbf{x}(t_1) \) as the initial value, and the Change of Variables formula.
Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

Neural ODEs (Chen et al., 2018)


Likelihood estimation using Change of Variables formula

Generate samples

Sample from the noise distribution, transform it into a sample from the target distribution using the trained Neural ODE.
Neural ODEs (Chen et al., 2018)

Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

Figure 6: Computation graph of the latent ODE model.
Neural ODEs (Chen et al., 2018)

Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

Encode the observed data

Figure 6: Computation graph of the latent ODE model.
Neural ODEs (Chen et al., 2018)

Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

Encode into a latent distribution (such as Gaussian)

Figure 6: Computation graph of the latent ODE model.

Encode into a latent distribution (such as Gaussian)
Neural ODEs (Chen et al., 2018)

Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

Sample from the latent distribution

Figure 6: Computation graph of the latent ODE model.
Neural ODEs (Chen et al., 2018)

Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

Use the sample as initial value to a Neural ODE

Figure 6: Computation graph of the latent ODE model.
Neural ODEs (Chen et al., 2018)

Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

Solve the Neural ODE for latent points in later time steps

Figure 6: Computation graph of the latent ODE model.
Neural ODEs (Chen et al., 2018)

Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

Figure 6: Computation graph of the latent ODE model.

Compute loss

Decode the latent points
Neural ODEs (Chen et al., 2018)

Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

Solve the Neural ODE for 
latent points in future time steps

ODE Solve \(z_{t_0}, f, \theta_f, t_0, \ldots, t_M\)

Decode the latent points

Future generation!

Figure 6: Computation graph of the latent ODE model.
1. Ordinary Differential Equations (ODEs)
   - Initial Value Problems
   - Numerical Integration methods
   - Fundamental theorem of ODEs

2. Neural ODEs (Chen et al., 2018)
   - Adjoint method
   - Applications

3. Recent research
FFJORD: Free-form Continuous Dynamics For Scalable Reversible Generative Models (Grathwohl et al., ICLR 2019)

- Essentially a better Continuous Normalizing Flow.
- Makes a better estimate for the log determinant term.
- “We demonstrate our approach on high-dimensional density estimation, image generation, and variational inference, achieving the state-of-the-art among exact likelihood methods with efficient sampling.”

Recent research

**Augmented Neural ODEs** (Dupont et al., NeurIPS 2019)

- Shows that Neural ODEs cannot model non-homeomorphisms (non-flows)
- **Augments** the state with additional dimensions to cover non-homeomorphisms
- Performs ablation study on toy examples and image classification

Recent research

ANODEV2: A Coupled Neural ODE Evolution Framework
(Zhang et al., NeurIPS 2019)

\[
\begin{align*}
    z(1) &= z(0) + \int_0^1 f(z(t), \theta(t)) dt \\
    \theta(t) &= \theta(0) + \int_0^t q(\theta(t), p) dt, \quad \theta(0) = \theta_0
\end{align*}
\]

“parent network”,
“weight network”.

- Network weights are also a function of time
- Separate “weight network” generates the weights of the function network at a given time

Recent research

Latent ODEs for Irregularly-Sampled Time Series
(Rubanova et al., NeurIPS 2019)

- Improves the generative latent variable framework for irregularly-sampled time series
- Essentially uses an ODE in the encoder where samples are missing
- Shows results on toy data, MuJoCo, PhysioNet

Recent research

Simple Video Generation using Neural ODEs
(David Kanaa*, Vikram Voleti*, Samira Kahou, Christopher Pal; NeurIPS 2019 Workshop)

- Video generation as a generative latent variable model using Neural ODEs

https://sites.google.com/view/neurips2019lire/accepted-papers?authuser=0
ODE2VAE: Deep generative second order ODEs with Bayesian neural networks (Yildiz et al., NeurIPS 2019)

- Uses 2nd-order Neural ODE
- Uses a Bayesian Neural Network
- Showed results modelling video generation as a generative latent variable model using (2nd-order Bayesian) Neural ODE

Figure 2: A schematic illustration of ODE2VAE model. Position encoder \((\mu_p, \sigma_p)\) maps the first item \(x_0\) of a high-dimensional data sequence into a distribution of the initial position \(s_0\) in a latent space. Velocity encoder \((\mu_v, \sigma_v)\) maps the first \(m\) high-dimensional data items \(x_{0:m}\) into a distribution of the initial velocity \(v_0\) in a latent space. Probabilistic latent dynamics are implemented by a second order ODE model \(f_W\) parameterised by a Bayesian deep neural network \((W)\). Data points in the original data domain are reconstructed by a decoder.

Recent research

On Robustness of Neural Ordinary Differential Equations
(Yan et al., ICLR 2020)

- Ablation study on adversarial attacks on ODE-Nets
- Introduces new regularization term to improve robustness

Figure 3: An illustration of the time-invariant property of ODEs. We can see that the curve $\tilde{z}_1(t)$ is exactly the horizontal translation of $z_1(t)$ on the interval $[T', \infty)$.

Recent research

How to Train Your Neural ODE (Finlay et al., 2020)

- Makes a link between the flow in Neural ODEs and optimal transport
- Introduces two new regularization terms to constrain flows to straight lines
- Speeds up training of Neural ODEs

Figure 1. Optimal transport map and a generic normalizing flow.

Additional References

- [http://faculty.bard.edu/belk/math213/InitialValueProblems.pdf](http://faculty.bard.edu/belk/math213/InitialValueProblems.pdf)
- [https://math.temple.edu/~queisser/assets/files/Talk3.pdf](https://math.temple.edu/~queisser/assets/files/Talk3.pdf)
- Textbook: [https://users.math.msu.edu/users/gnagy/teaching/ode.pdf](https://users.math.msu.edu/users/gnagy/teaching/ode.pdf)
- [https://lpsa.swarthmore.edu/NumInt/NumIntFirst.html](https://lpsa.swarthmore.edu/NumInt/NumIntFirst.html)
- [http://homepages.cae.wisc.edu/~blanchar/eps/ivp/ivp](http://homepages.cae.wisc.edu/~blanchar/eps/ivp/ivp)
Thank you!

voletiv.github.io