Continuous Normalizing Flows

Vikram Voleti
PhD student, Mila, University of Montreal
Supervisor: Prof. Christopher Pal

voletiv.github.io

@ (virtual) Mila, Montreal, Canada
1. Ordinary Differential Equations

2. Neural ODEs

3. Continuous Normalizing Flows

4. Later research
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?

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
- ...

Vikram Voleti

Continuous Normalizing Flows
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:
\[ t_{n+1} = t_n + h \quad \text{Step size } h \]
\[ x(t_{n+1}) = x(t_n) + hf(x(t_n), t_n) \quad \text{Update using derivative } f \]

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:

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

Example:

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

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

\[ h = 0.25 \]
\[ x(0.25) = x(0) + 0.25 \cdot f(x(0), 0) = 3 + 0.25 \cdot (2 \cdot 3 \cdot 0) = 3 \]
\[ x(0.5) = x(0.25) + 0.25 \cdot f(x(0.25), 0.25) = 3 + 0.25 \cdot (2 \cdot 3 \cdot 0.25) = 3.375 \]
\[ x(0.75) = x(0.5) + 0.25 \cdot f(x(0.5), 0.5) = 3.375 + 0.25 \cdot (2 \cdot 3.375 \cdot 0.5) = 4.21875 \]
\[ x(1) = x(0.75) + 0.25 \cdot f(x(0.75), 0.75) = 4.21875 + 0.25 \cdot (2 \cdot 4.21875 \cdot 0.75) \approx 5.8008 \]
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:
\[
t_{n+1} = t_n + h \\
x(t_{n+1}) = x(t_n) + h f(x(t_n), t_n)
\]

Step size matters!

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

Initial value problem:

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

Solution:

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

★ 4th-order Runge-Kutta method:

\[ t_{n+1} = t_n + h \]
\[ s_1 = f(x(t_n), t_n) \]
\[ s_2 = f(x(t_n) + \frac{h}{2} s_1, t_n + \frac{h}{2}) \]
\[ s_3 = f(x(t_n) + \frac{h}{2} s_2, t_n + \frac{h}{2}) \]
\[ s_4 = f(x(t_n) + h s_3, t_n + h) \]
\[ x(t_{n+1}) = x(t_n) + \frac{h}{6} (s_1 + 2s_2 + 2s_3 + s_4) \]

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

Initial value problem:

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

Solution:

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

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

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:

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

Fundamental Theorem of ODEs

Suppose $f$ is continuously differentiable.

1. The solution curves for different initial conditions 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=B1e9Y2NYyS

Vikram Voleti

Continuous Normalizing Flows
1. Ordinary Differential Equations

2. Neural ODEs

3. Continuous Normalizing Flows
Neural ODEs (Chen et al., NeurIPS 2018)

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

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

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

**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**

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

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

_Euler discretization_

**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)

**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 \( L \)


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:

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

Back-propagate through the ODE Solver!

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

Can we do better?

Yes.

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

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

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

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:

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

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

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

Initial value is 0

Neural ODEs (Chen et al., 2018)

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:
\[
\begin{bmatrix}
  x(t_0) \\
  \frac{\partial L}{\partial x(t_0)} \\
  \frac{\partial L}{\partial \theta}
\end{bmatrix}
= \text{ODESolve}
\begin{bmatrix}
  f(x(t), t, \theta) \\
  -a(t)^\top \frac{\partial f(x(t), t, \theta)}{\partial x} \\
  -a(t)^\top \frac{\partial f(x(t), t, \theta)}{\partial \theta}
\end{bmatrix},
\begin{bmatrix}
  \frac{\partial L}{\partial x(t_1)} \\
  0_{|\theta|}
\end{bmatrix}, t_1, t_0
\]

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

Neural ODEs (Chen et al., 2018)

Neural ODEs describe a homeomorphism (flow).
- They preserve dimensionality.
- They form non-intersecting trajectories.
Neural ODEs (Chen et al., 2018)

Neural ODEs are **reversible** models!
Just integrate forward/backward in time.

1. Ordinary Differential Equations
2. Neural ODEs
3. Continuous Normalizing Flows
Continuous Normalizing Flows

Target distribution

\( \mathbf{x}_{im} \)

(such as real image manifold)

Sample from target distribution (such as an image)

Noise distribution

\( \mathbf{z} \)

Likelihood estimation using Change of Variables formula

\[ \mathbf{z} = g(\mathbf{x}) \Rightarrow \log p(\mathbf{x}) = \log p(\mathbf{z}) + \log | \det \frac{\partial \mathbf{z}}{\partial \mathbf{x}} | \]

Train \( f \) to maximize the likelihood of the samples from target distribution \( \log p(\mathbf{x}) \)

https://arxiv.org/abs/1810.01367
Continuous Normalizing Flows

“FFJORD”
(Free-Form Jacobian Of Reversible Dynamics)

Target distribution

\(X_{im}\)

\(f_\theta(x(t), t)\)

Noise distribution

\(Z\)

Neural ODE

Likelihood estimation using Change of Variables formula

Sample from the noise distribution, transform it into a sample from the target distribution using the trained Neural ODE.

Sample from target distribution (such as an image)

Sample from noise distribution (such as Gaussian)

(such as real image manifold)

Generate samples

https://arxiv.org/abs/1810.01367
**Continuous Normalizing Flows**

**FFJORD** (ICLR 2019)

Change of variables:

\[
\log p(x_{im}) - \log p(z) = \log \det \left| \frac{df_\theta}{dx(t)} \right|
\]

*Instantaneous* change of variables:

\[
\frac{\partial \log p(x(t))}{\partial t} = -\text{Tr} \left( \frac{\partial f_\theta}{\partial x(t)} \right)
\]

Initial value:

\[
\begin{bmatrix}
x_{im} \\
0
\end{bmatrix}
\]

\[
\begin{bmatrix}
z \\
\log p(x_{im}) - \log p(z)
\end{bmatrix} = \int_{t_0}^{t_1} \begin{bmatrix} f_\theta(x(t), t) \\ -\text{Tr} \left( \frac{\partial f_\theta}{\partial x(t)} \right) \end{bmatrix} dt
\]

Hutchinson’s trace estimator:

\[
\log p(z(t_1)) = \log p(z(t_0)) - \int_{t_0}^{t_1} \text{Tr} \left( \frac{\partial f}{\partial z(t)} \right) dt
\]

\[
= \log p(z(t_0)) - \int_{t_0}^{t_1} \mathbb{E}_{p(\epsilon)} \left[ \epsilon^T \frac{\partial f}{\partial z(t)} \epsilon \right] dt
\]

\[
= \log p(z(t_0)) - \mathbb{E}_{p(\epsilon)} \left[ \int_{t_0}^{t_1} \epsilon^T \frac{\partial f}{\partial z(t)} \epsilon dt \right]
\]

https://arxiv.org/abs/1810.01367
FFJORD (ICLR 2019)

Change of variables:
\[
\log p(x_{im}) - \log p(z) = \log \det | \frac{df_\theta}{dx(t)} |
\]

**Instantaneous** change of variables:
\[
\frac{\partial \log p(x(t))}{\partial t} = -\text{Tr} \left( \frac{\partial f_\theta}{\partial x(t)} \right)
\]

Initial value:
\[
\begin{bmatrix}
  x_{im} \\
  \log p(x_{im}) - \log p(z)
\end{bmatrix} = \int_{t_0}^{t_1} \begin{bmatrix}
  f_\theta(x(t), t) \\
  -\text{Tr} \left( \frac{\partial f_\theta}{\partial x(t)} \right)
\end{bmatrix} \, dt
\]

How to Train your Neural ODE (ICML 2020)

Introduces 2 regularization terms:
1) Kinetic energy of flow
2) Jacobian norm of flow

\[
K(\theta) = \int_{t_0}^{t_1} ||f(x(t), t, \theta)||_2^2 \, dt
\]
\[
B(\theta) = \int_{t_0}^{t_1} ||e^T \nabla_x f(x(t), t, \theta)||_2^2 \, dt
\]

STEER

Introduces temporal regularization:
\[
x(t_1) = x(t_0) + \int_{t_0}^{T} f_\theta(x(t), t) \, dt
\]
\[
= \text{ODESolve}(x(t_0), f_\theta, t_0, T)
\]
\[
T \sim \text{Uniform}(t_1 - b, t_1 + b)
\]
\[
b < t_1 - t_0
\]

<table>
<thead>
<tr>
<th>CIFAR10</th>
<th>ImageNet64</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPD</td>
<td>Time</td>
</tr>
<tr>
<td>3.40</td>
<td>≥5 days</td>
</tr>
</tbody>
</table>

https://arxiv.org/abs/1810.01367

<table>
<thead>
<tr>
<th>CIFAR10</th>
<th>ImageNet64</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPD</td>
<td>Time</td>
</tr>
<tr>
<td>3.38</td>
<td>31.84</td>
</tr>
<tr>
<td>3.83</td>
<td>64.1</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>CIFAR10</th>
<th>ImageNet64</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPD</td>
<td>Time</td>
</tr>
<tr>
<td>3.39</td>
<td>22.24</td>
</tr>
<tr>
<td>3.83</td>
<td>64.1</td>
</tr>
</tbody>
</table>

3. Image Generation

**FFJORD** (ICLR 2019)

<table>
<thead>
<tr>
<th>CIFAR10</th>
<th>ImageNet64</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPD</td>
<td>Time</td>
</tr>
<tr>
<td>3.40</td>
<td>≥5 days</td>
</tr>
</tbody>
</table>

**How to Train your Neural ODE** (ICML 2020)

<table>
<thead>
<tr>
<th>CIFAR10</th>
<th>ImageNet64</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPD</td>
<td>Time</td>
</tr>
<tr>
<td>3.38</td>
<td>31.84</td>
</tr>
<tr>
<td>3.83</td>
<td>64.1</td>
</tr>
</tbody>
</table>

**STEER**

<table>
<thead>
<tr>
<th>CIFAR10</th>
<th>ImageNet64</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPD</td>
<td>Time</td>
</tr>
<tr>
<td>3.39</td>
<td>22.24</td>
</tr>
</tbody>
</table>

https://arxiv.org/abs/1810.01367


Thank you!