Training neural ODEs for density estimation

Chris Finlay

in collaboration with
Jörn-Henrik Jacobsen, Levon Nurbekyan and Adam Oberman
Paper: “How to train your Neural ODE”

IPAM HJB II
April 23, 2020
Table of Contents

Background
  Density estimation with Normalizing flows
  Neural ODEs

FFJORD: Neural ODEs + Normalizing flows

Regularized neural ODEs

Results
Density estimation is a fundamental problem in ML

- Given data \( \{X_1, \ldots, X_n\} \) drawn from an unknown distribution \( p(x) \), estimate \( p \)

\[ p(\text{dog}) = ? \]
Density estimation is a fundamental problem in ML

- Given data \( \{X_1, \ldots, X_n\} \) drawn from an *unknown* distribution \( p(x) \), estimate \( p \)

Typically done through maximum log-likelihood

- select a family of models \( p_\theta \)
- solve \( \max_\theta \sum \log p_\theta(X_i) \)

\( p(\text{dog}) = ? \)
Density estimation is a fundamental problem in ML

- Given data \( \{X_1, \ldots, X_n\} \) drawn from an unknown distribution \( p(x) \), estimate \( p \)

Typically done through maximum log-likelihood

- select a family of models \( p_\theta \)
- solve \( \max_\theta \sum \log p_\theta(X_i) \)

Two issues arise

1. How to parameterize \( p_\theta \)?
2. How to sample from the learned distribution \( p_\theta \)? Ie, generate new data?

\( p(\text{dog}) = ? \)
Density estimation \textit{and generation} with normalizing flows

Today, the gospel of Normalizing Flows [TT13, RM15]

\textbf{Density estimation}

- Suppose we have an invertible map $f_\theta : X \mapsto Z$
- Change of variables applied to log-densities:

\[
\log p_\theta(x) = \log p_N(f_\theta(x)) + \log \det \left[ \frac{df_\theta(x)}{dx} \right]
\]  

- Here $N$ is the standard normal distribution

\textbf{Generation}

- Sample $z \sim N$
- Compute $x = f_\theta^{-1}(z)$
Unfortunately this approach has two difficult problems:

1. Constructing invertible deep neural networks is hard. (but see eg [CBD+19])
2. Need to compute Jacobian determinants. Also hard.
That’s nice, but... 

Unfortunately this approach has two difficult problems:

1. Constructing invertible deep neural networks is hard. (but see eg [CBD+]19)
2. Need to compute Jacobian determinants. Also hard.

Both of these tasks can be made easier by putting structural constraints on the model. However these constraints tend to degrade model performance.

Is there an easier way?
Neural ODEs [HR17, CRB⁺18] are generalizations of Residual Networks, where layer index $t$ is a continuous variable called “time”.

- Compare one layer of a Residual Network

$$x^{t+1} = x^t + v_\theta(x^t, t)$$

- with an Euler step discretization of the ODE $\dot{x} = v_\theta(x, t)$, with step size $\tau$

$$x^{t+1} = x^t + \tau v_\theta(x^t, t)$$
Neural ODEs

- Rather than fixing the number of layers (time steps) in the ResNet beforehand
- Solve the ODE

\[
\begin{aligned}
\dot{x} &= v_\theta(x, t) \\
x(0) &= x_0
\end{aligned}
\]

(2)

with an adaptive ODE solver, up to some end time $T$. Here $x_0$ is the input to the “network”

- The function so defined is the solution map:

\[
f_\theta(x_0) := x(T) = x_0 + \int_0^T v_\theta(x(s), s) \, ds
\]

**Benefits:** memory savings; tradeoff between accuracy and speed
Aside: Differentiating losses of neural ODEs

Suppose we have a loss function $L(x(T))$ which depends only on the final state $x(T)$. How do we differentiate wrt $\theta$?

▶ **Naive approach:** backpropagate through the computation graph of the ODE solver

▶ **Optimal control:** use sensitivity analysis [PMB$^+$62]. The gradient $\frac{dl}{d\theta}$ is given by

\[
\frac{dL(x(T))}{d\theta} = \mu(0)
\]

where $\mu$, $\lambda$ and $x$ solve the following ODE (run backwards in time)

\[
\begin{align*}
\dot{\mu} &= -\lambda^T \frac{d}{d\theta} v_\theta(x(t), t), \\
\dot{\lambda} &= -\lambda^T \nabla_x v_\theta(x(t), t), \\
\dot{x} &= -v_\theta(x(t), t),
\end{align*}
\]

\[
\begin{align*}
\mu(T) &= 0 \\
\lambda(T) &= \frac{dL(x(T))}{dx(T)} \\
x(T) &= x_T
\end{align*}
\]
Neural ODEs and Normalizing Flows

We can overcome the two difficulties of normalizing flows by exploiting properties of dynamical systems

1. If $v_{\theta}$ is Lipschitz, then the solution map is invertible! (Picard-Lindelöf)

$$f_{\theta}^{-1}(x(T)) = x(T) + \int_{0}^{T} v_{\theta}(x(s), s) \, ds$$

- ie just need to solve the backwards dynamics

$$\begin{cases} 
\dot{x} = -v_{\theta}(x, t) \\
x(0) = x_T 
\end{cases}$$

(3)
We can overcome the two difficulties of normalizing flows by exploiting properties of dynamical systems.

2. Log-determinants (of particles solving the ODE) have a beautiful time derivative [Vil03, p. 114]

\[
\frac{d}{ds} \log \det \left[ \frac{dx(s)}{dx_0} \right] = \text{div} \left( v \right) \left( x(s), s \right)
\]

where \( \text{div} \left( \cdot \right) \) is the divergence operator, \( \text{div} \left( f \right) = \sum_i \frac{\partial f_i}{\partial x_i} \)

\[\text{ie}\]
\[
\log \det \left[ \frac{df_\theta(x)}{dx} \right] = \int_0^T \text{div} \left( v \right) \left( x(s), s \right) ds
\]
Putting these two observations together, we arrive at the FFJORD algorithm (Free-form Jacobian Of Reversible Dynamics) [GCB⁺19]

**Density estimation**

To learn the distribution $p_\theta$, solve the following log-likelihood optimization problem

$$
\max_\theta \sum_i \log p_N(z_i(T)) + \int_0^T \text{div} \ (v_\theta) (z_i(s), s) \, ds
$$

where $z_i(s)$ satisfies the ODE

$$
\begin{cases}
\dot{z}_i(s) = v_\theta(z_i(s), s) \\
z_i(0) = x_i
\end{cases}
$$
That’s nice, but... Isn’t it really hard to compute the divergence term

$$\text{div}(v_\theta)(x, s) = \sum_j \frac{\partial v^j_\theta(x, s)}{\partial x_j}$$

in high dimensions?
Trace estimates

- Divergence is the trace of the Jacobian $\nabla_x v_\theta(x, t)$

- So we can use trace estimates to approximate

$$\text{div} (v_\theta)(x, s) = \text{Tr}(\nabla_x v_\theta(x, t)) = \mathbb{E}_\eta \left[ \eta^T \nabla_x v_\theta(x, t) \eta \right]$$

where $\eta \sim \mathcal{N}(0, 1)$

- this can be computed quickly with reverse mode automatic differentiation
FFJORD: generation

FFJORD \[GCB^+19\] generation (density sampling) is simple

**Generation**

Sample \( z \sim \mathcal{N} \), and solve

\[
\begin{aligned}
\dot{x}(s) &= -v_\theta(x(s), s) \\
x(0) &= z
\end{aligned}
\]

\( \text{le, } x = z + \int_0^T v_\theta(z(s), s) \, ds \)
3. Regularized neural ODEs

Need for regularity

FFJORD is promising, but there are no constraints placed on the paths the particles take

- As long as source (data) distribution is mapped to target (normal) distribution, the log-likelihood is maximized
- ie solutions are not unique
- If the particle paths are “wobbly” the adaptive ODE solver has to take many tiny steps, with many function evaluations. This is time consuming
the number of function evaluations (time steps) taken by the solver is related to the total derivative

\[
\frac{dv(x(t), t)}{dt} = [\nabla v(x(t), t)] v(x(t), t) + \frac{\partial}{\partial t} v(x(t), t)
\]

In other words, if we can control the size of \( v \) and \( \nabla v \), we can reduce the number of function evaluations (time steps) taken by the ODE solver.
Proposal: regularize objective to decrease \# of FEs

Add two terms to the objective to encourage regularity of trajectories:

1. \( \int_0^T \| \nu_\theta(x(s), s) \|^2 ds \), the kinetic energy. This is closely related to the Optimal Transport cost between distributions (Benamou-Brenier)

2. \( \int_0^T \| \nabla_x \nu_\theta(x(s), s) \|_F^2 ds \), a Frobenius norm penalty on the Jacobian
Proposal: regularize objective to decrease $\#$ of FEs

Add two terms to the objective to encourage regularity of trajectories:

$\int_0^T \|v_\theta(x(s), s)\|^2 \, ds$, the kinetic energy. This is closely related to the Optimal Transport cost between distributions (Benamou-Brenier)

$\int_0^T \|\nabla_x v_\theta(x(s), s)\|^2_F \, ds$, a Frobenius norm penalty on the Jacobian

Frobenius norms can be computed again with trace estimate:

$$\|A\|^2_F = \text{Tr}(A^T A) = \mathbb{E}_\eta \left[ \eta^T A^T A \eta \right]$$

$$= \mathbb{E} \left[ \|A\eta\|^2 \right]$$

ie can recycle computations from divergence estimate
4. Results

Test log-likelihood vs wall clock time

**MNIST**

**CIFAR10**
4. Results

Ablation study: regularity measures on MNIST

**Jacobian norm vs epoch**

**Kinetic energy vs epoch**

**Function evals vs epoch**
### 4. Results

Some numbers

<table>
<thead>
<tr>
<th></th>
<th>MNIST</th>
<th>CIFAR10</th>
<th>IMAGENET64</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BITS/DIM</td>
<td>TIME</td>
<td>BITS/DIM</td>
</tr>
<tr>
<td><strong>FFJORD, ORIGINAL</strong></td>
<td>0.99</td>
<td>-</td>
<td>3.40</td>
</tr>
<tr>
<td><strong>FFJORD, VANILLA</strong></td>
<td>0.974</td>
<td>68.47</td>
<td>3.363</td>
</tr>
<tr>
<td><strong>FFJORD RNODE (OURS)</strong></td>
<td>0.973</td>
<td>24.37</td>
<td>3.398 (3.370)</td>
</tr>
<tr>
<td><strong>REALNVP</strong></td>
<td>1.06</td>
<td>-</td>
<td>3.49</td>
</tr>
<tr>
<td><strong>1-RESNet</strong></td>
<td>1.05</td>
<td>-</td>
<td>3.45</td>
</tr>
<tr>
<td><strong>GLOW</strong></td>
<td>1.05</td>
<td>-</td>
<td>3.35</td>
</tr>
<tr>
<td><strong>FLOW++</strong></td>
<td>-</td>
<td>-</td>
<td>3.28 / 3.09</td>
</tr>
<tr>
<td><strong>Residual Flow</strong></td>
<td>0.970</td>
<td>-</td>
<td>3.280</td>
</tr>
</tbody>
</table>
Some pictures: MNIST & CIFAR10

(a) real MNIST images
(b) real CIFAR10 images
(c) vanilla FFJORD
(d) vanilla FFJORD
(e) FFJORD RNODE
(f) FFJORD RNODE
4. Results

Some bigger pictures: ImageNet64

Real ImageNet64 images

Generated ImageNet64 images (FFJORD RNODE)
The Jacobian norm penalty can be viewed as a continuous time analogue of layer-wise Lipschitz regularization in ResNets. This helps ensure particle paths do not cross, and helps keep networks numerically invertible.

Without regularization, it is very difficult to train neural ODEs with fixed step-size ODE solvers. Need adaptive ODE solvers.

However with regularization, neural ODEs can be trained with fixed step-size ODE solvers (eg a four stage Runge-Kutta scheme).

On large images (> 64 pixel width) vanilla FFJORD will not train: adaptive solver’s time step underflows.
References I


