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

Density estimation

Training data

$p(\text{dog}) = ?$

Density estimation is a fundamental problem in ML

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

Density estimation

Training data

$p(\text{dog}) = ?$

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_θ
- solve $\max_\theta \sum \log p_\theta(X_i)$

Density estimation


Training data

$p(\text{dog}) = ?$

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_θ
- solve $\max_\theta \sum \log p_\theta(X_i)$

Two issues arise

1. How to parameterize p_θ?
2. How to sample from the learned distribution p_θ? Ie, generate new data?

Density estimation *and generation* with normalizing flows

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

Density estimation

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

$$\log p_\theta(x) = \log p_{\mathcal{N}}(f_\theta(x)) + \log \det \left[\frac{df_\theta(x)}{dx} \right] \tag{1}$$

- here $\mathcal{N}$ is the standard normal distribution

Generation

- sample $z \sim \mathcal{N}$
- compute $x = f_\theta^{-1}(z)$

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.

That's nice, but. . .

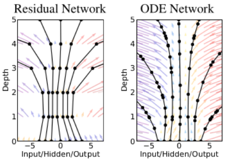
Unfortunately this approach has two difficult problems:

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



Source: [CRB+18], Figure 1

Neural ODEs [HR17, CRB+18] are generalizations of Residual Networks, where layer index t is a continuous variable called "time".

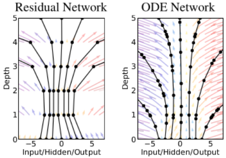
- Compare one layer of a Residual Network

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

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

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

Neural ODEs



Source: [CRB+18], Figure 1

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

$$\begin{cases} \dot{x} = v_\theta(x, t) \\ x(0) = x_0 \end{cases} \tag{2}$$

  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:

$$f_\theta(x_0) := x(T) = x_0 + \int_0^T v_\theta(x(s), s)\,\mathrm{d}s$$

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 θ?

- **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 μ, λ and x solve the following ODE (run backwards in time)

$$\begin{cases} \dot{\mu} = -\lambda^T \frac{d}{d\theta} v_\theta(x(t), t), & \mu(T) = 0 \\ \dot{\lambda} = -\lambda^T \nabla_x v_\theta(x(t), t), & \lambda(T) = \frac{dL(x(T))}{dx(T)} \\ \dot{x} = -v_\theta(x(t), t), & x(T) = x_T \end{cases}$$

Neural ODEs and Normalizing Flows

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

1. If v_θ is Lipschitz, then the solution map is invertible! (Picard-Lindelöf)

$$f_\theta^{-1}(x(T)) = x(T) + \int_T^0 v_\theta(x(s), s)\,\mathrm{d}s$$

- ie just need to solve the backwards dynamics

$$\begin{cases} \dot{x} = -v_\theta(x,t) \\ x(0) = x_T \end{cases} \tag{3}$$

Neural ODEs and Normalizing Flows

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]

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

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

- ie

$$\log \det \left[\frac{df_\theta(x)}{dx} \right] = \int_0^T \operatorname{div}(v)\,(x(s), s)\,\mathrm{d}s$$

FFJORD: density estimation

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_θ, solve the following log-likelihood optimization problem

$$\max_\theta \sum_i \log p_{\mathcal{N}}(z_i(T)) + \int_0^T \operatorname{div}(v_\theta)(z_i(s), s)\,\mathrm{d}s$$

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

$$\operatorname{div}(v_\theta)(x,s) = \sum_j \frac{\partial v_\theta^j(x,s)}{\partial x_j}$$

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

$$\begin{aligned} \operatorname{div}(v_\theta)(x, s) &= \mathrm{Tr}(\nabla_x v_\theta(x, t)) \\ &= \mathbb{E}_\eta\left[\eta^T \nabla_x v_\theta(x, t)\eta\right] \end{aligned}$$

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

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

Ie, $x = z + \int_T^0 v_\theta(z(s), s) \, \mathrm{d}s$

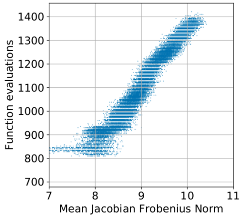
Need for regularity

A generic solution

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

Solver is slowed by number of function evaluations



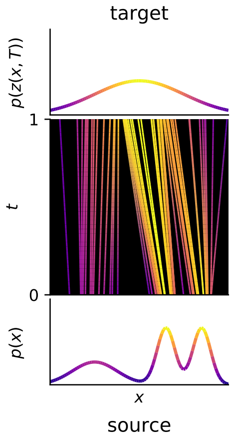
Function evaluations vs Jacobian norm

- 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 ∇v, we can reduce the number of function evaluations (time steps) taken by the ODE solver

Proposal: regularize objective to decrease # of FEs

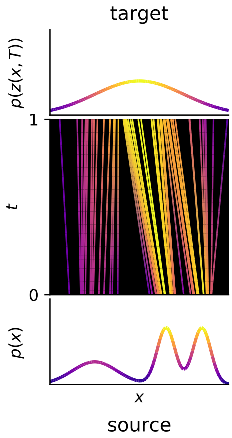


OT map

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

- $\int_0^T \|v_\theta(x(s),s)\|^2 \, \mathrm{d}s$, the kinetic energy. This is closely related to the Optimal Transport cost between distributions (Benamou-Brenier)
- $\int_0^T \|\nabla_x v_\theta(x(s),s)\|_F^2 \, \mathrm{d}s$, a Frobenius norm penalty on the Jacobian

Proposal: regularize objective to decrease # of FEs



OT map

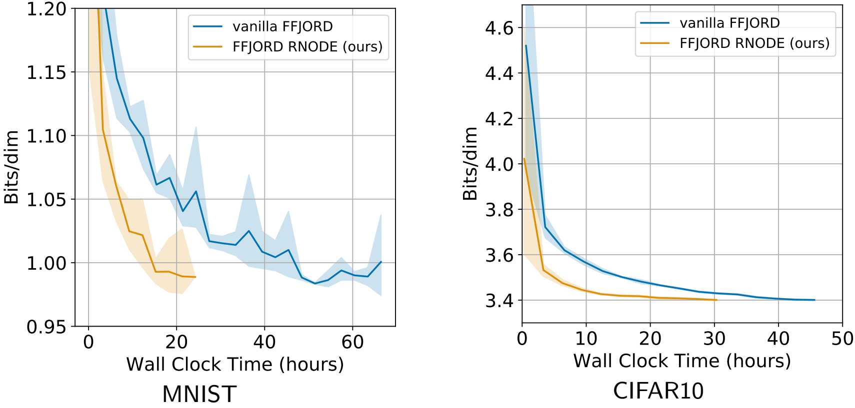
Add two terms to the objective to encourage regularity of trajectories:

- $\int_0^T \|v_\theta(x(s), s)\|^2 \,\mathrm{d}s$, the kinetic energy. This is closely related to the Optimal Transport cost between distributions (Benamou-Brenier)
- $\int_0^T \|\nabla_x v_\theta(x(s), s)\|_F^2 \,\mathrm{d}s$, a Frobenius norm penalty on the Jacobian
- Frobenius norms can be computed again with trace estimate:

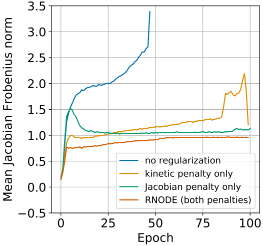
$$\|A\|_F^2 = \mathrm{Tr}(A^T A) = \mathbb{E}_\eta \left[\eta^T A^T A \eta\right] = \mathbb{E}\left[\|A\eta\|^2\right]$$

ie can recycle computations from divergence estimate

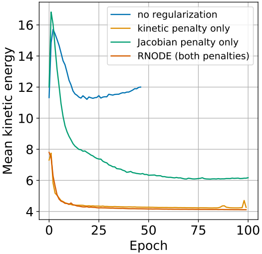
Test log-likelihood vs wall clock time



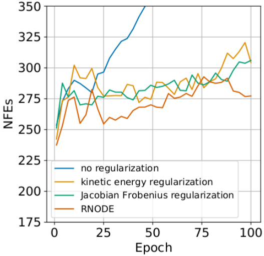
Ablation study: regularity measures on MNIST



Jacobian norm vs epoch



Kinetic energy vs epoch



Function evals vs epoch

Some numbers

	MNIST		CIFAR10		ImageNet64	
	bits/dim	time	bits/dim	time	bits/dim	time
FFJORD, original (Grathwohl et al., 2019)	0.99	-	3.40	$\geq$ 5 days	-	-
FFJORD, vanilla	0.974	68.47	3.363	91.33	-	-
FFJORD RNODE (ours)	0.973	24.37	3.398 (3.370)	31.84 (63.85)	3.723	64.07
RealNVP (Dinh et al., 2017)	1.06	-	3.49	-	3.98	-
i-ResNet (Behrmann et al., 2019)	1.05	-	3.45	-	-	-
Glow (Kingma & Dhariwal, 2018)	1.05	-	3.35	-	3.81	-
Flow++ (Ho et al., 2019)	-	-	3.28 / 3.09	-	- / 3.69	-
Residual Flow (Chen et al., 2019)	0.970	-	3.280	-	3.757	-

Some pictures: MNIST & CIFAR10

(a) real MNIST images

(b) real CIFAR10 images

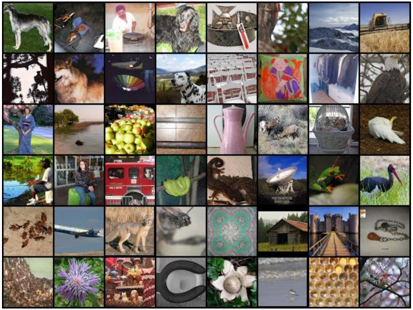
(c) vanilla FFJORD

(d) vanilla FFJORD

(e) FFJORD RNODE

(f) FFJORD RNODE

Some bigger pictures: ImageNet64



Real ImageNet64 images

Generated ImageNet64 images (FFJORD RNODE)

Comments

- 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

- TQ Chen, J Behrmann, D Duvenaud and JH Jacobsen, *Residual Flows for Invertible Generative Modeling*, NeurIPS 2019: 9913–9923.
- TQ Chen, Y Rubanova, J Bettencourt and D Duvenaud, *Neural Ordinary Differntial Equations*, NeurIPS 2018: 6572 – 6583.
- W Grathwohl, RT Chen, J Bettencourt, I Sutskever and D Duvenaud, *FFJORD: Free-Form Continuous Dynamics for Scalable Reversible Generative Models*, ICLR 2019.
- E Haber and L Ruthotto, *Stable architectures for deep neural networks*, Inverse Problems, **34** (2017), no. 1, 014004.
- L Pontryagin, EF Mischenko, VG Boltyanskii and RV Gamkrelidze, *The mathematical theory of optimal processes*, 1962.
- RJ Rezende and S Mohamed, *Variational Inference with Normalizing Flows*, ICML 2015: 1530–1538.

References II

EG Tabak and CV Turner, *A family of nonparametric density estimation algorithms*, Communication on Pure and Applied Mathematics, **66** (2013), no. 2, 145–164.

C Villani, *Topics in Optimal Transport*, Graduate studies in mathematics. AMS 2003, ISBN 9780821833124.