Residual Flows for Invertible Generative Modeling

Ricky T. Q. Chen, Jens Behrmann, David Duvenaud,

Jörn-Henrik Jacobsen

Motivation

- Flow based generative models aim to invert a transformation (*noising*) and trained with ML
- iResnet offers invertibility directly through flexible architecture
- Calculating exact log likelihood requires trace of Jacobian, proposed estimation results in biased estimator
- They proposed a memory and computation efficient unbiased estimator for Jacobian log-det
  - Do not require ODE solvers unlike FM
  - Can directly calculate the likelihood -> Stochastic estimate of Jacobian log-det
  - iResnets are discrete and inverse requires iteration (*simulation in CNF*)
  - Unlike CNF, calculation of network's Lipschitz constant

Recap

- iResNet
  - Forward pass $y = f(x) = x + g(x)$ remains bijective, x is recoverable from y
    - g is the NN part of the residual block
  - Requires Lipschitz continuity of <:1
    - There is a guaranteed unique inverse -> contractiveness ensures inversion leads to a fixed point
  - Process can be reverted through fixed point iteration (do the inversion *n* times)

$$x^{(i)} = y - g(x^{(i-1)})$$

  - Reverthing requires calculation the trace of Jacobian
    - Also, ensuring Lipschitz constraint on g requires calculation of it as well

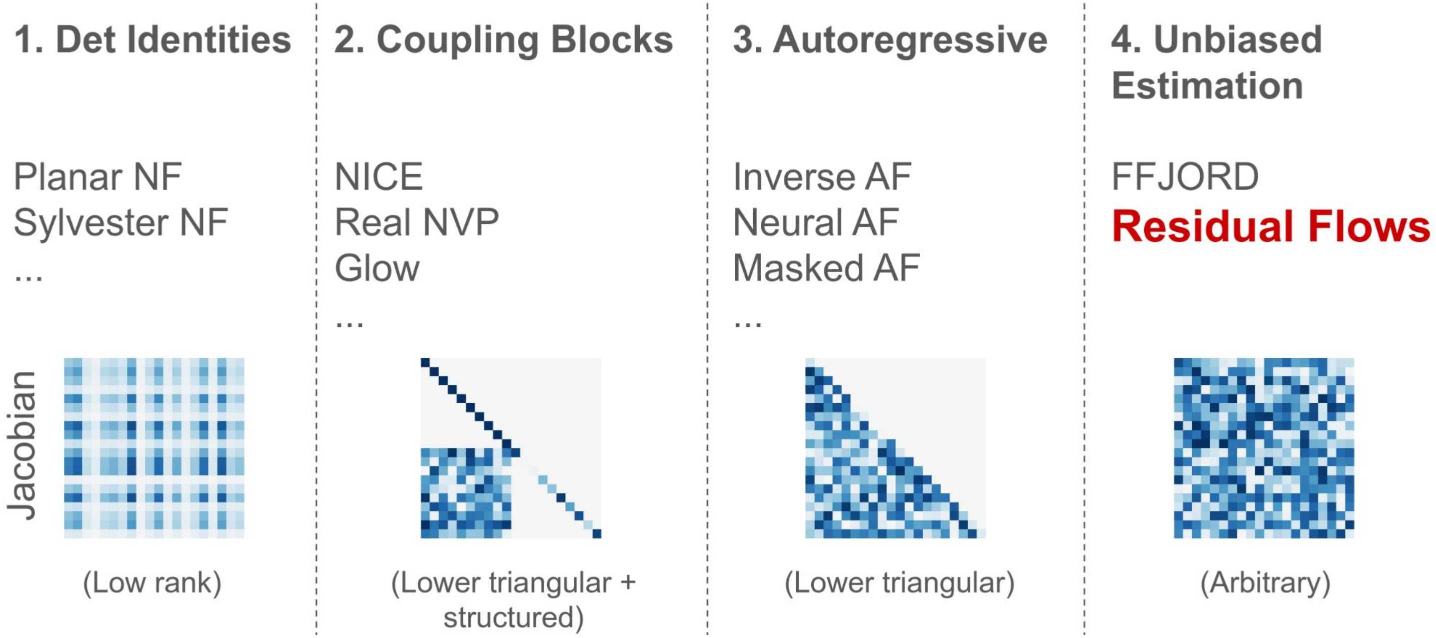
$$||J_g(x)||_2 = ||W_L \dots W_2 \phi'(z_1) W_1 \phi'(z_2)||_2$$

  - Can be used for estimating a tractable density

$$\mathcal{L}_{\text{NLL}} = -\frac{1}{N} \sum_{i=1}^{N} \left[\log p_Z(f(x_i)) + \log \left| \det \frac{\partial f}{\partial x_i} \right| \right]$$ where p_z is a gaussian

Problem of existing Flow Approaches

- They rely on specific architectures when predicting Jacobian during density estimation
- Results in **biased estimator**, **slow inference**



- Model can overfit on the bias without maxing ML
- Less flexibility in the architecture

Change of Variables for Estimating Density

- For an invertible transformation F, the density estimation:

$$\ln p_x(x) = \ln p_z(z) + \ln|\det J_F(x)|,$$

- Given that Residual Network f(x):

$$y = f(x) = x + g(x).$$

- Det of Jacobian can be written as a power series from a book I dont have access to

$$\log p(x) = \log p(f(x)) + \operatorname{tr}\left(\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} [J_g(x)]^k\right)$$

- The trace can be estimated by Skilling-Hutchinson estimator

$\operatorname{tr}(A) = \mathbb{E}_{p(v)}\left[v^T A v\right]$, where v is a random variable (generally normal dist)

$$\mathbb{E}_v\left[\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} v^T [J_g(x)]^k v\right]$$

  - The SH estimator is Lipschitz bounded as well (See App. proof. Theorem 1)

- Change of Variables Theorem
  - In Bishop:

$$p_y(y) = p_x(x)\left|\frac{\mathrm{d}x}{\mathrm{d}y}\right|$$

  - More Formally

$$\int_{\varphi(U)} f(\mathbf{v})\, d\mathbf{v} = \int_U f(\varphi(\mathbf{u}))\, |\det(D\varphi)(\mathbf{u})|\, d\mathbf{u}.$$

- Change of Variables in flows

$$\underbrace{\log p(x)}_{\text{<Model distribution>}} = \underbrace{\log p(f(x))}_{\text{<Base distribution>}} + \log\left|\det \frac{df(x)}{dx}\right|$$

How to avoid calculating infinite series

- Truncation:
  - Just calculate first *n* terms

$$\mathbb{E}_v\left[\sum_{k=1}^{\infty}\frac{(-1)^{k+1}}{k}v^T[J_g(x)]^k v\right]\approx \mathbb{E}_v\left[\sum_{k=1}^{n}\frac{(-1)^{k+1}}{k}v^T[J_g(x)]^k v\right]$$

for Each residual block **do**
  Lip constraint: $\hat{W}_j := \mathrm{SN}(W_j, x)$ for linear Layer W_j.
  Draw v from $\mathcal{N}(0, I)$
  $w^T := v^T$
  $\ln\det := 0$
  for $k = 1$ **to** n **do**
    $w^T := w^T J_g$ (vector-Jacobian product)
    $\ln\det := \ln\det + (-1)^{k+1} w^T v/k$
  end for

- Results in biased estimator which requires tradeoff
  - Reduce d to reduce the Lip const
  - Increase n to diminish the bias term
  - Also model can overfit on on bias

$$\underbrace{\mathbb{E}_v\left[\sum_{k=1}^{n}\frac{(-1)^{k+1}}{k}v^T[J_g(x)]^k v\right]}_{\text{biased estimator}} + \underbrace{\mathrm{tr}\left(\sum_{k=n+1}^{\infty}\frac{(-1)^{k+1}}{k}[J_g(x)]^k\right)}_{\text{bias}} \in \mathcal{O}\left(\frac{d}{1-\mathrm{Lip}(g)} \times \mathrm{Lip}(g)^n\right)$$

How to avoid calculating infinite series

- With an **unbiased estimator**:
  - Calculate the first term of the series Δ_1
  - Flip a coin for the next terms, and weight the the calculated terms:

$$\mathbb{E}\left[\Delta_1 + \left[\tfrac{1}{1-q}\textstyle\sum_{k=2}^{\infty}\Delta_k\right]\mathbb{1}_{b=0} + [0]\mathbb{1}_{b=1}\right]$$

    - Has a probability of running finite time with probability of q
    - It is an unbiased estimator because:

$$= \Delta_1 + \left[\tfrac{1}{1-q}\textstyle\sum_{k=2}^{\infty}\Delta_k\right](1-q)$$

$$= \textstyle\sum_{k=1}^{\infty}\Delta_k$$

How to avoid calculating infinite series

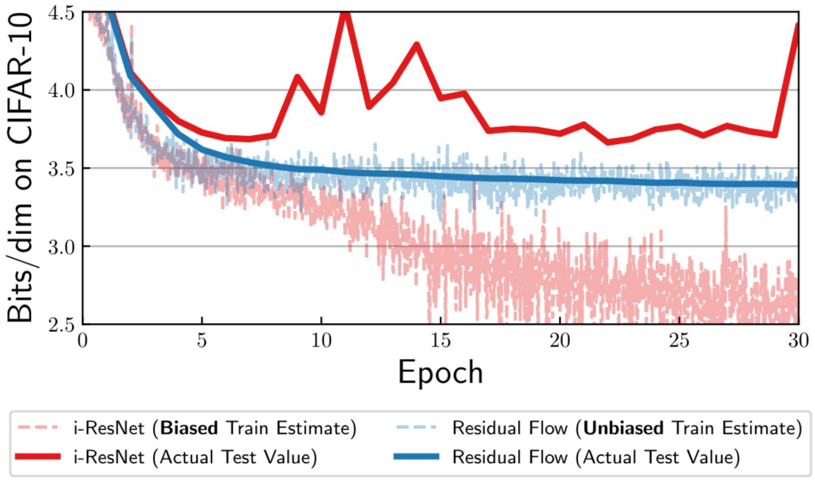
- With an **unbiased estimator**:
  - Modify it so that It can run in finite terms with q=1
    - Apply the same trick for each next term (*still an unbiased estimator*)

$$\sum_{k=1}^{\infty} \Delta_k = \mathbb{E}_{n\sim p(N)}\left[\sum_{k=1}^{n} \frac{\Delta_k}{\mathbb{P}(N\geq k)}\right]$$

kth term is weighted by prob of seeing >=k tosses

- Final LLH: $\log p(x) = \log p(f(x)) + \mathbb{E}_{n,v}\left[\sum_{k=1}^{n} \frac{(-1)^{k+1}}{k} \frac{v^T[J_g(x)]^k v}{\mathbb{P}(N\geq k)}\right]$
  - Two levels of stochasticity:
    - Coin toss
    - Skilling-Hutchinson estimator

Calculate first 2 terms, then sample from Geom(0.5).



Engineering Problems

- OOM error further into the training:
  - # calculated terms could get high during training
    - Terms need to backpropagated $\mathbb{E}_{n,v}\left[\sum_{k=1}^{n} \alpha_k \frac{\partial v^T [J_g(x)]^k v}{\partial \theta}\right]$
  - Use Neumann gradient series (?) to take the differentiation out of the power-series
    - Number of derivatives stored in memory becomes independent of n

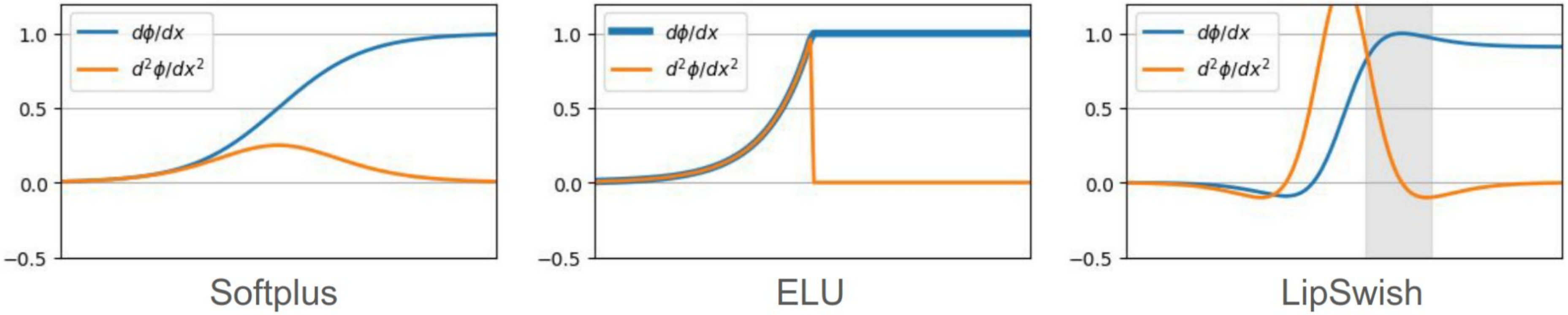
$$\mathbb{E}_{n,v}\left[\left(\sum_{k=1}^{n} \alpha_k v^T [J_g(x)]^k\right) \frac{\partial J_g(x) v}{\partial \theta}\right]$$

Only work if the series is finite

Engineering Problems

- Lipschitz constraint requires regularization, such as spectral norm
- Lipschitz constrained activation functions can have a saturation in their derivatives
  - Manifest as second derivatives vanish
  - Swish is a good candidate, but breaks the uniform Lipschitz constraint
    - Scale it!

$$\mathrm{LipSwish}(x) = \mathrm{Swish}(x)/1.1$$



Thank you for listening!