Residual Flows

for Invertible Generative Modeling

Ricky T. Q. Chen, Jens Behrmann,
David Duvenaud, Jörn-Henrik Jacobsen

Pathways to Designing a Normalizing Flow

1. Require an invertible architecture.

- Coupling layers, autoregressive, etc.

2. Require efficient computation of a change of variables equation.

$$\log p(x) = \log p(f(x)) + \log \left| \det \frac{df(x)}{dx} \right|$$

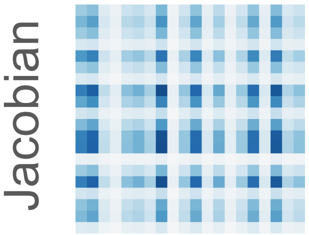
<Model distribution> <Base distribution>

(or a continuous version) $\log p(x(t_N)) = \log p(x(t_0)) + \int_{t_0}^{t_N} \operatorname{tr}\left(\frac{\partial f(x(t), t)}{\partial x(t)}\right) dt$

Pathways to Designing a *Scalable* Normalizing Flow

1. Det Identities

Planar NF
Sylvester NF
...

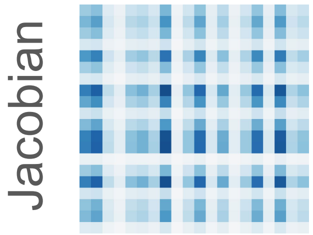


(Low rank)

Pathways to Designing a *Scalable* Normalizing Flow

1. Det Identities

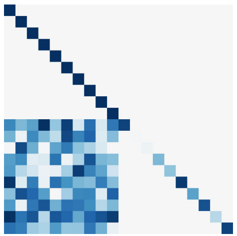
Planar NF
Sylvester NF
...



(Low rank)

2. Coupling Blocks

NICE
Real NVP
Glow
...



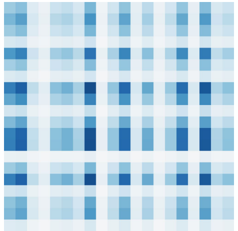
(Lower triangular + structured)

Pathways to Designing a *Scalable* Normalizing Flow

1. Det Identities

Planar NF
Sylvester NF
...

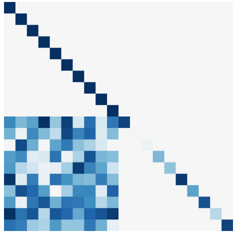
Jacobian



(Low rank)

2. Coupling Blocks

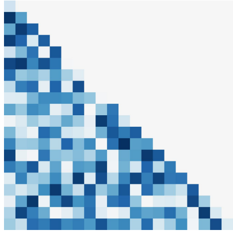
NICE
Real NVP
Glow
...



(Lower triangular + structured)

3. Autoregressive

Inverse AF
Neural AF
Masked AF
...



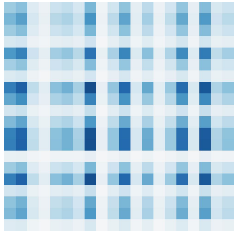
(Lower triangular)

Pathways to Designing a *Scalable* Normalizing Flow

1. Det Identities

Planar NF
Sylvester NF
...

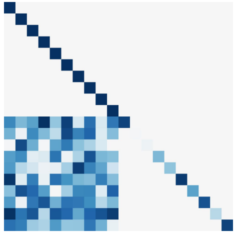
Jacobian



(Low rank)

2. Coupling Blocks

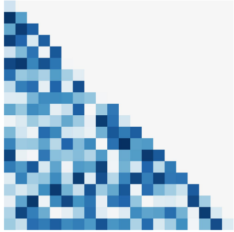
NICE
Real NVP
Glow
...



(Lower triangular + structured)

3. Autoregressive

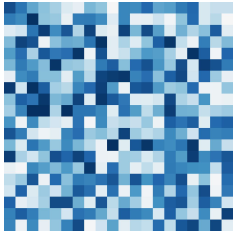
Inverse AF
Neural AF
Masked AF
...



(Lower triangular)

4. Unbiased Estimation

FFJORD
Residual Flows



(Arbitrary)

Unbiased Estimation → Flow-based Model

Benefits of Flow-based Generative Models:

- Trainable with either the reverse- or forward-KL (a.k.a. maximum likelihood).
- Generally possible to sample from the model.

Maximum Likelihood Training:

Stochastic Gradients
$$\nabla_\theta \mathbb{E}_{x \sim p_{\text{data}}(x)} \left[\log p_\theta(x)\right] = \mathbb{E}_{x \sim p_{\text{data}}(x)} \left[\nabla_\theta \log p_\theta(x)\right]$$

Log-Likelihood
$$\log p_\theta(x) = \log p(f(x)) + \log \left|\det \frac{df_\theta(x)}{dx}\right|$$

Invertible Residual Networks (i-ResNet)

It can be shown that residual blocks

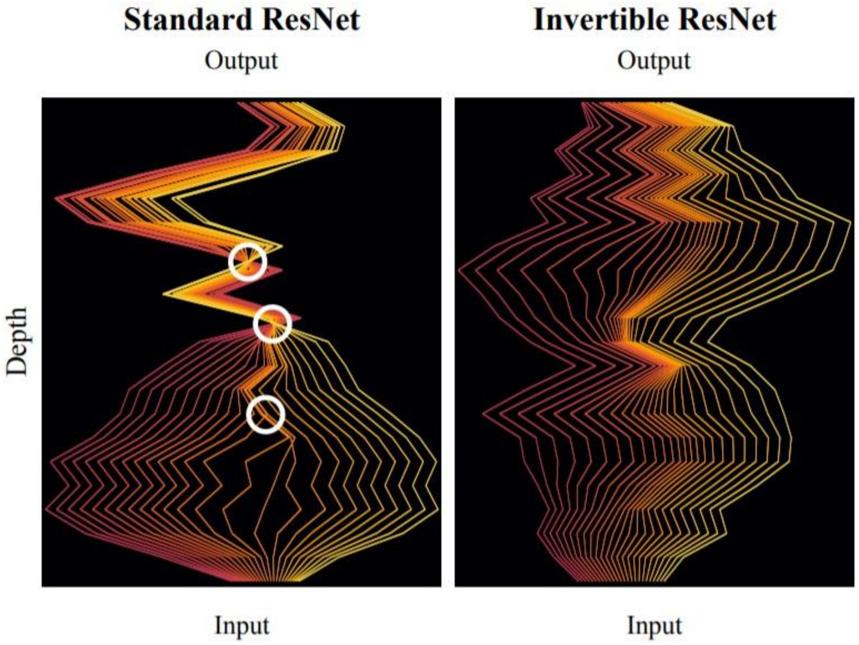
$$y = f(x) = x + g(x)$$

can be inverted by fixed-point iteration

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

and has a unique inverse (ie. invertible) if

$$\mathrm{Lip}(g) < 1$$



(Behrmann et al. 2019)

Satisfying Lipschitz Condition on g(x)

Parametering g(x) as a deep neural network with pre-activation:

$$z_l = W_l h_{l-1} + b_l \text{ and } h_l = \phi(z_l)$$

The Lipschitz constant of g(x) can be expressed as:

$$\begin{aligned} ||J_g(x)||_2 &= ||W_L \ldots W_2 \phi'(z_1) W_1 \phi'(z_2)||_2 \\ &\leq ||W_L||_2 \ldots ||W_2||_2 \; ||\phi'(z_1)||_2 \; ||W_1||_2 \; ||\phi'(z_2)||_2 \end{aligned}$$

1. Choose Lipschitz-constrained activation functions $\phi'(z) \leq 1$.
2. Bound the spectral norm of weight matrices.

(Behrmann et al. 2019)

Applying Change of Variables to i-ResNets

If

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

Then

$$\log p(x) = \log p(f(x)) + \log \left| \det \frac{df(x)}{dx} \right|$$

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

(Behrmann et al. 2019)

Applying Change of Variables to i-ResNets

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

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

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

Skilling-Hutchinson trace estimator:

$$\operatorname{tr}(M) = \mathbb{E}\left[v^T M v\right]$$

(Behrmann et al. 2019)

Decoupling Training Objective and Bias

The i-ResNet used a biased estimator of the log-likelihood.

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

Decoupling Training Objective and Bias

The i-ResNet used a biased estimator of the log-likelihood.

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

This bias is large when:

- Scaling to higher dimensional data.
- The Lipschitz constant of network is large.

Decoupling Training Objective and Bias

The i-ResNet used a biased estimator of the log-likelihood.

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

This bias is large when:

- Scaling to higher dimensional data.
- The Lipschitz constant of network is large.

Thus, requires **carefully trading off between bias and expressiveness.**

Decoupling Training Objective and Bias

Enter the “Russian roulette” estimator (Kahn, 1955). Suppose we want to estimate

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

(Require $\sum_{k=1}^{\infty} |\Delta_k| < \infty$)

Decoupling Training Objective and Bias

Enter the “Russian roulette” estimator (Kahn, 1955). Suppose we want to estimate

$$\sum_{k=1}^{\infty} \Delta_k \qquad \text{(Require } \sum_{k=1}^{\infty} |\Delta_k| < \infty\text{)}$$

Flip a coin b with probability q.

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

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

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

Decoupling Training Objective and Bias

Enter the “Russian roulette” estimator (Kahn, 1955). Suppose we want to estimate

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

(Require $\sum_{k=1}^{\infty} |\Delta_k| < \infty$)

Flip a coin b with probability q.

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

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

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

Has probability q of being evaluated in **finite** time.

Decoupling Training Objective and Bias

If we repeatedly apply the same procedure *infinitely many times*, we obtain an unbiased estimator of the infinite series.

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

Computed in **finite** time with **prob. 1**!!

Decoupling Training Objective and Bias

If we repeatedly apply the same procedure *infinitely many times*, we obtain an unbiased estimator of the infinite series.

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

Directly sample the first successful coin toss.

Computed in **finite** time with **prob. 1**!!

Decoupling Training Objective and Bias

If we repeatedly apply the same procedure *infinitely many times*, we obtain an unbiased estimator of the infinite series.

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

Computed in **finite** time with **prob. 1**!!

Directly sample the first successful coin toss.

k-th term is weighted by prob. of seeing >= k tosses.

Decoupling Training Objective and Bias

If we repeatedly apply the same procedure *infinitely many times*, we obtain an unbiased estimator of the infinite series.

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

Directly sample the first successful coin toss.

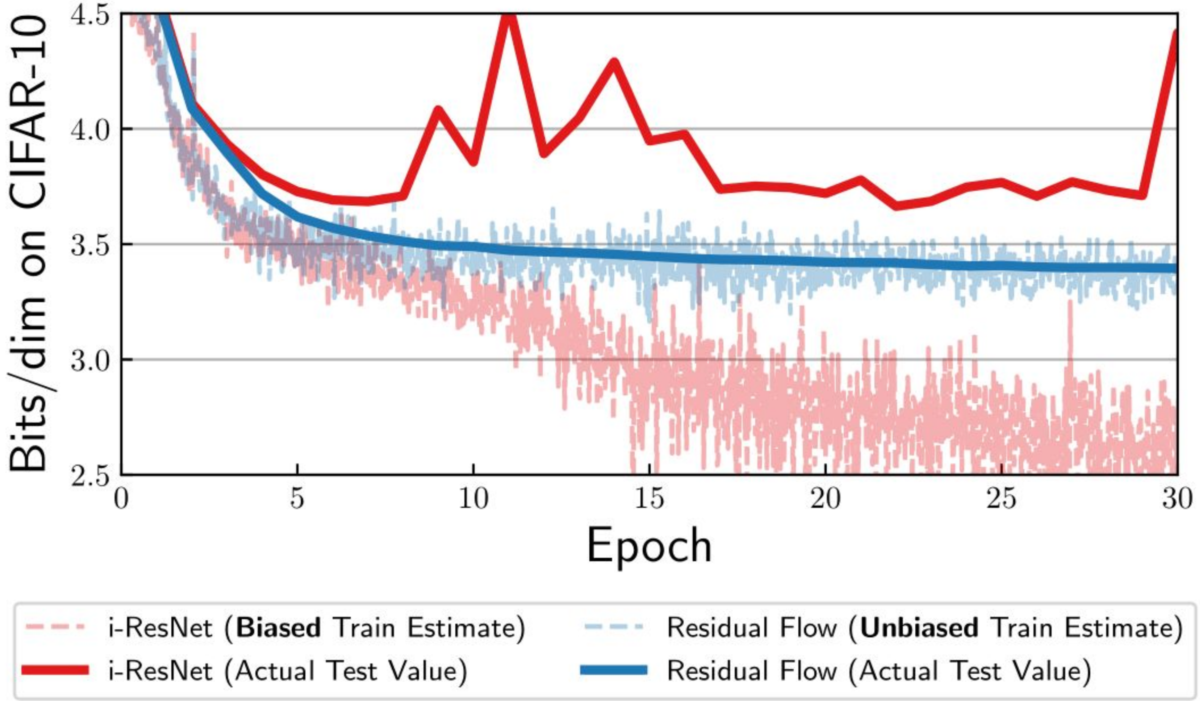
k-th term is weighted by prob. of seeing >= k tosses.

Computed in **finite** time with **prob. 1**!!

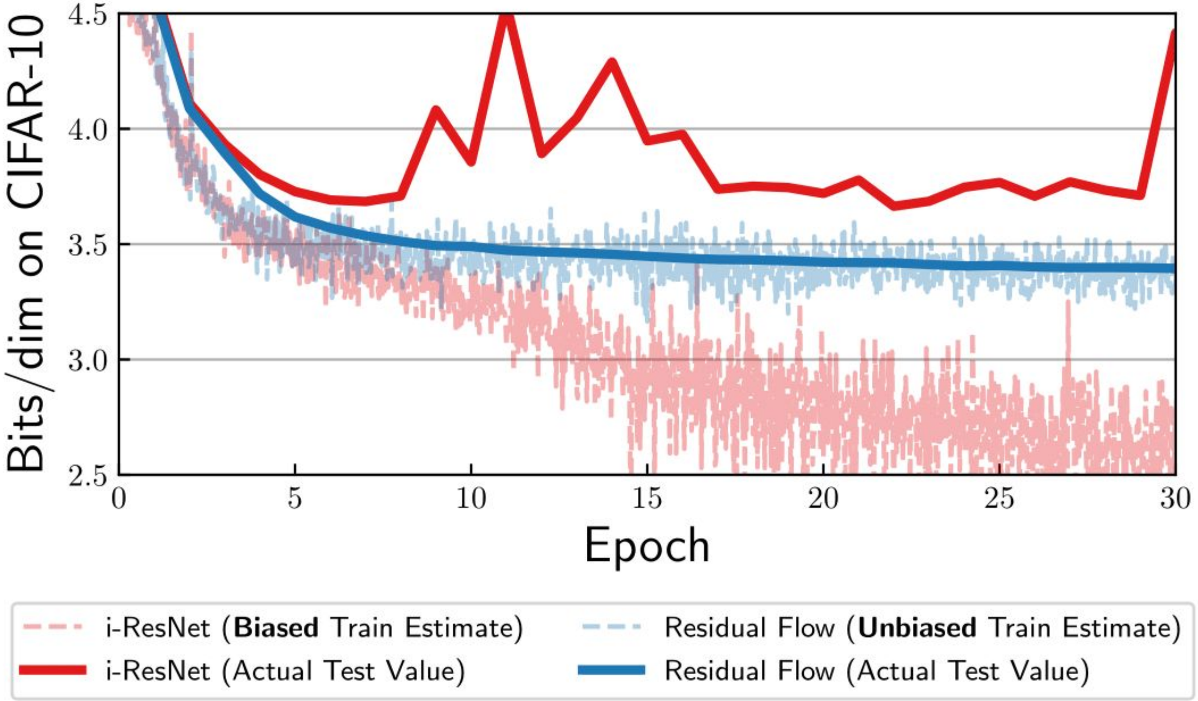
Residual Flow:

$$\log p(x) = \log p(f(x)) + \mathbb{E}_{n,v} \left[\sum_{k=1}^{n} \frac{(-1)^{k+1}}{k} \frac{v^T [J_g(x)]^k v}{\mathbb{P}(N \geq k)} \right]$$

Decoupling Training Objective and Bias



Decoupling Training Objective and Bias



Unbiased but...
variable compute
and memory!

Dealing with Variable Memory Usage

$$\mathbb{E}_{n,v}\left[\sum_{k=1}^{n} \alpha_k v^T [J_g(x)]^k v\right] \qquad \alpha_k = \frac{(-1)^{k+1}}{k} \frac{1}{\mathbb{P}(N \geq k)}$$

Dealing with Variable Memory Usage

$$\mathbb{E}_{n,v}\left[\sum_{k=1}^{n} \alpha_k v^T [J_g(x)]^k v\right] \qquad \alpha_k = \frac{(-1)^{k+1}}{k}\frac{1}{\mathbb{P}(N \geq k)}$$

Naive gradient computation:

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

1. Estimate
2. Differentiate

Dealing with Variable Memory Usage

$$\mathbb{E}_{n,v}\left[\sum_{k=1}^{n} \alpha_k v^T [J_g(x)]^k v\right] \qquad \alpha_k = \frac{(-1)^{k+1}}{k}\frac{1}{\mathbb{P}(N \geq k)}$$

Naive gradient computation:

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

1. Estimate
2. Differentiate

Alternative (Neumann series) gradient formulation:

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

1. Analytically Differentiate
2. Estimate

Dealing with Variable Memory Usage

$$\mathbb{E}_{n,v}\left[\sum_{k=1}^{n} \alpha_k v^T [J_g(x)]^k v\right] \qquad \alpha_k = \frac{(-1)^{k+1}}{k}\frac{1}{\mathbb{P}(N \geq k)}$$

Naive gradient computation:

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

Alternative (Neumann series) gradient formulation:

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

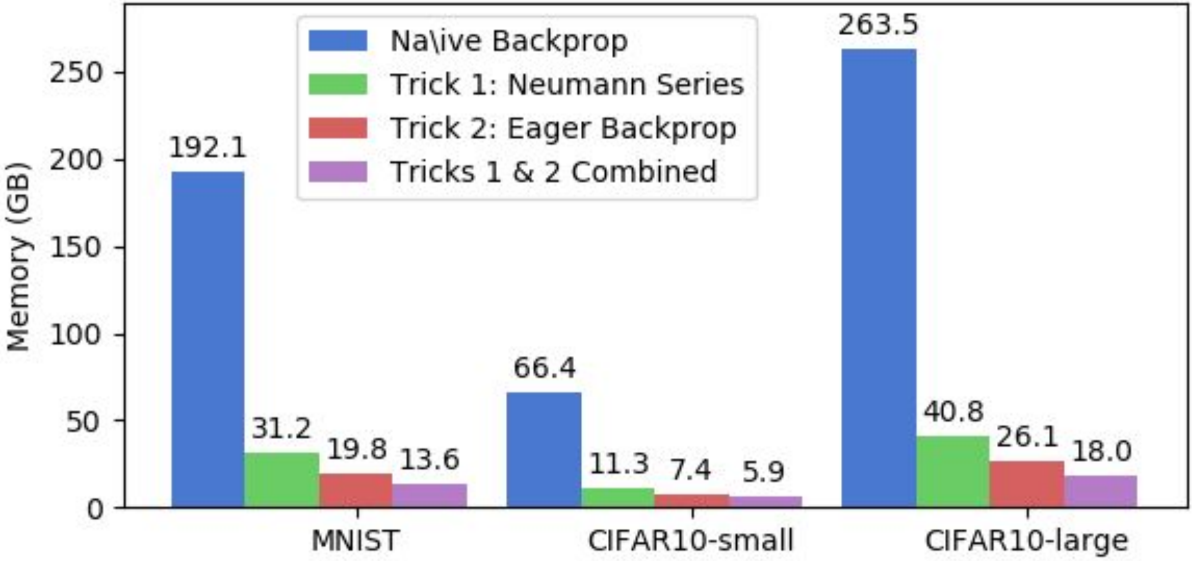
Don't need to store random number of terms in memory!!

1. Estimate
2. Differentiate

↓

1. Analytically Differentiate
2. Estimate

Dealing with Variable Memory Usage



Architecture used by i-ResNet

More expressive architectures

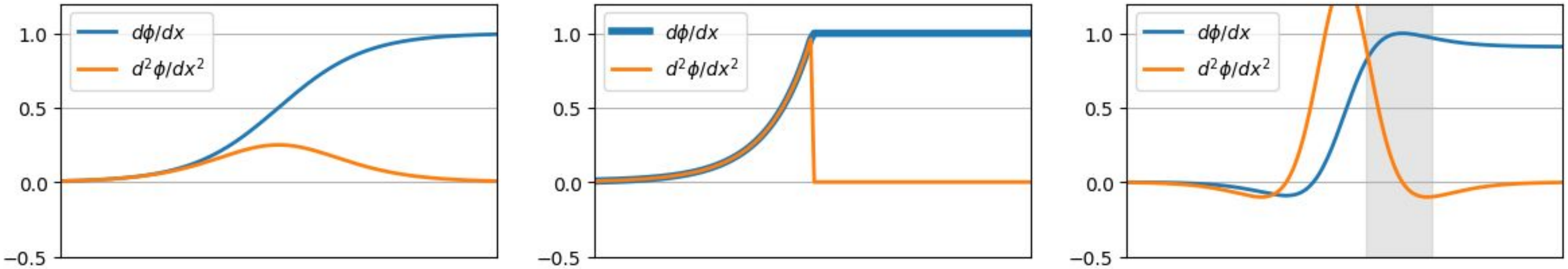
Gradient Saturation of Lipschitz Act Fns

Log-likelihood depends on first-order derivatives.

→ Lipschitz activation functions have bounded derivative.

Gradient depends on second-order derivatives.

→ Lipschitz activation fns can lead to “gradient saturation”.



Softplus ELU LipSwish

Gradient Saturation of Lipschitz Act Fns

Log-likelihood depends on first-order derivatives.

$\rightarrow$ Lipschitz activation functions have bounded derivative.

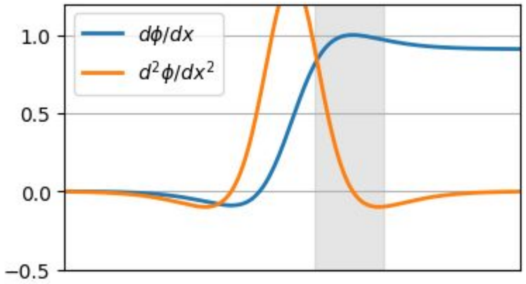
Gradient depends on second-order derivatives.

$\rightarrow$ Lipschitz activation fns can lead to “gradient saturation”.

$$\left(\text{because } \frac{d}{dx}\text{Swish}(x) \lesssim 1.1\right)$$

$$\text{LipSwish}(x) = \text{Swish}(x)/1.1 = \sigma(\beta x)x$$

(Ramachandran et al., 2017)



LipSwish

Generalizing i-ResNet and Spectral Normalization

Recall

$$\begin{aligned}||J_g(x)||_2 &= ||W_L \dots W_2 \phi'(z_1) W_1 \phi'(z_2)||_2 \\ &\leq ||W_L||_2 \dots ||W_2||_2 \; ||\phi'(z_1)||_2 \; ||W_1||_2 \; ||\phi'(z_2)||_2\end{aligned}$$

Generalizing i-ResNet and Spectral Normalization

Recall

$$\begin{aligned} ||J_g(x)||_p &= ||W_L \dots W_2 \phi'(z_1) W_1 \phi'(z_2)||_p \\ &\leq ||W_L||_p \dots ||W_2||_p \, ||\phi'(z_1)||_p \, ||W_1||_p \, ||\phi'(z_2)||_p \end{aligned}$$

$$||A||_p = \sup_{x \neq 0} \frac{||Ax||_p}{||x||_p}$$

Generalizing i-ResNet and Spectral Normalization

Recall

$$
\begin{aligned}
||J_g(x)||_p &= ||W_L \ldots W_2 \phi'(z_1) W_1 \phi'(z_2)||_p \\
&\leq ||W_L||_p \ldots ||W_2||_p \; ||\phi'(z_1)||_p \; ||W_1||_p \; ||\phi'(z_2)||_p
\end{aligned}
$$

$$
||A||_{p \to q} = \sup_{x \neq 0} \frac{||Ax||_q}{||x||_p}
$$

Generalizing i-ResNet and Spectral Normalization

Recall

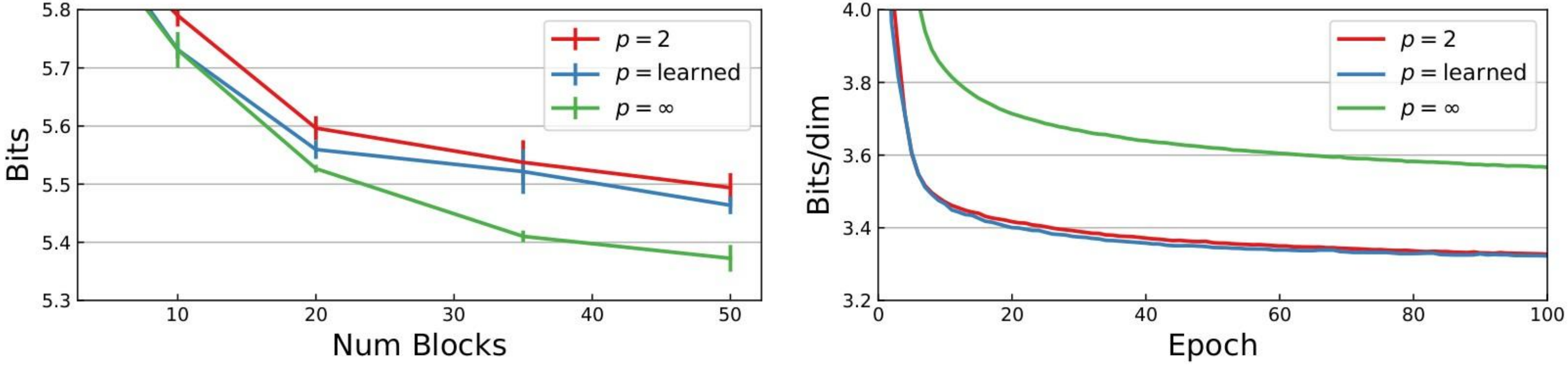
$$
\begin{aligned}
||J_g(x)||_p &= ||W_L \dots W_2 \phi'(z_1) W_1 \phi'(z_2)||_p \\
&\leq ||W_1||_{p \to p_1} \; ||W_2||_{p_1 \to p_2} \cdots ||W_L||_{p_{L-1} \to p}
\end{aligned}
$$

$$
||A||_{p \to q} = \sup_{x \neq 0} \frac{||Ax||_q}{||x||_p}
$$

Power iteration for mixed norms:
(Johnston, “QETLAB.” 2016)

Generalizing i-ResNet and Spectral Normalization

Learn the norm orders p's and q's!



(a) Checkerboard 2D

(b) CIFAR-10

Figure 2: **Lipschitz constraints with different induced matrix norms**.

Power iteration for mixed norms:
(Johnston, "QETLAB." 2016)

Density Estimation Experiments

Contribution Summary:

- **[Residual Flow]** Unbiased estimator of log-likelihood.
- Memory-efficient computation of log-likelihood.
- LipSwish activation function.

Table 2: Results [bits/dim] on standard benchmark datasets for density estimation. In brackets are models that used "variational dequantization" (Ho et al., 2019), which we don't compare against.

Model	MNIST	CIFAR-10	ImageNet 32×32	ImageNet 64×64
Real NVP (Dinh et al., 2017)	1.06	3.49	4.28	3.98
Glow (Kingma and Dhariwal, 2018)	1.05	3.35	4.09	3.81
FFJORD (Grathwohl et al., 2019)	0.99	3.40	—	—
Flow++ (Ho et al., 2019)	—	**3.29** (3.09)	— (3.86)	— (3.69)
i-ResNet (Behrmann et al., 2019)	1.05	3.45	—	—
Residual Flow (Ours)	**0.97**	**3.29**	**4.02**	**3.78**

Density Estimation Experiments

Contribution Summary:

- **[Residual Flow]** Unbiased estimator of log-likelihood.
- Memory-efficient computation of log-likelihood.
- LipSwish activation function.

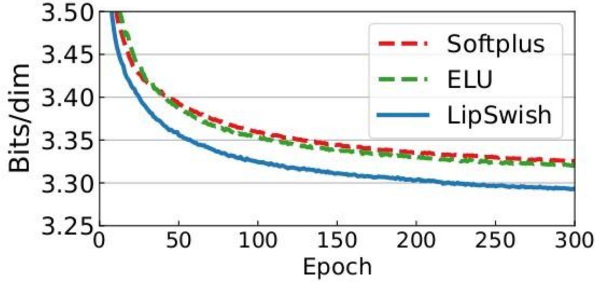


Figure 5: Effect of activation functions on CIFAR-10.

Training Setting	MNIST	CIFAR-10†	CIFAR-10
i-ResNet + ELU	1.05	3.45	3.66∼4.78
Residual Flow + ELU	1.00	3.40	3.32
Residual Flow + LipSwish	**0.97**	**3.39**	**3.29**

Table 3: Ablation results.

Qualitative Samples

CelebA:

Data

Residual Flow

CIFAR10:

Data

Residual Flow

PixelCNN

Flow++

Joint Generative and Discriminative Representations

Coupling blocks have difficulty learning both a generative model and a discriminative classifier.

Following *Nalisnick et al. (2019)*, we train using *weighted* maximum likelihood.

$$\mathbb{E}_{(x,y)\sim p_{\mathrm{data}}}\left[\lambda \log p_\theta(x) + \log p_\theta(y|x)\right]$$

Joint Generative and Discriminative Representations

Hybrid models using weighted maximum likelihood:

$$\mathbb{E}_{(x,y)\sim p_{\text{data}}}\left[\lambda \log p_\theta(x) + \log p_\theta(y|x)\right]$$

Table 4: Comparison of residual vs. coupling blocks for the hybrid modeling task.

	MNIST					SVHN				
	$\lambda = 0$	$\lambda = 1/D$		$\lambda = 1$		$\lambda = 0$	$\lambda = 1/D$		$\lambda = 1$	
Block Type	Acc↑	BPD↓	Acc↑	BPD↓	Acc↑	Acc↑	BPD↓	Acc↑	BPD↓	Acc↑
Nalisnick et al. (2019)	99.33%	1.26	97.78%	–	–	95.74%	2.40	94.77%	–	–
Coupling	99.50%	1.18	98.45%	1.04	95.42%	96.27%	2.73	95.15%	2.21	46.22%
+ 1×1 Conv	**99.56%**	1.15	98.93%	1.03	94.22%	**96.72%**	2.61	95.49%	2.17	46.58%
Residual	99.53%	**1.01**	**99.46%**	**0.99**	**98.69%**	**96.72%**	**2.29**	**95.79%**	**2.06**	**58.52%**

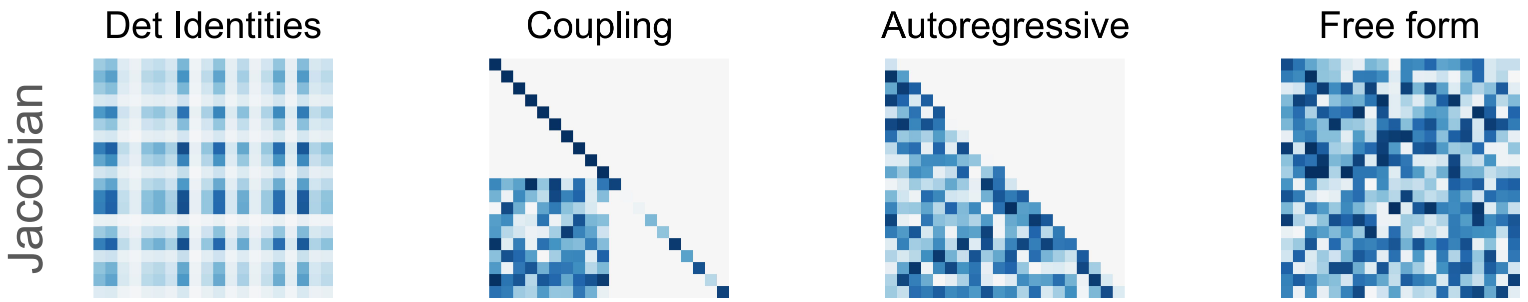
results on CIFAR-10.

$\lambda = 0$	$\lambda = 1/D$		$\lambda = 1$	
Acc↑	BPD↓	Acc↑	BPD↓	Acc↑
89.77%	4.30	87.58%	3.54	67.62%
90.82%	4.09	87.96%	3.47	67.38%
91.78%	**3.62**	**90.47%**	**3.39**	**70.32%**

Summary of Residual Flows

An approach to flow-based modeling requiring only Lipschitz constraints.

- Unbiased estimate of log-likelihood.
- Memory-efficient training.
- LipSwish for 1-Lipschitz activation function.
- Generalized spectral normalization.



Thanks for Listening!

Co-Authors:

Jens Behrmann

David Duvenaud

Jörn-Henrik Jacobsen