Residual Flows for Invertible Generative Modeling

Ricky T. Q. Chen[1,3], Jens Behrmann[2], David Duvenaud[1,3], Jörn-Henrik Jacobsen[1,3]

University of Toronto[1], University of Bremen[2], Vector Institute[3]

Contributions

We build a powerful flow-based model based on the invertible residual network architecture (i-ResNet) (Behrmann *et al.*, 2019). We

1. Use a "Russian roulette" estimator to produce unbiased estimates of the log-density. This allows principled training as a flow-based model.
2. Formulate a gradient power series for computing the partial derivatives from the log determinant term in $\mathcal{O}(1)$ memory.
3. Motivate and investigate desirata for Lipschitz-constrained activation functions that avoid gradient saturation.
4. Generalize i-ResNets to induced mixed norms and learnable norm orders.

Background: Invertible (Flow-based) Generative Models

Maximum likelihood estimation. To perform maximum likelihood with stochastic gradient descent, we require

$$\nabla_\theta \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log p_\theta(x)] = \mathbb{E}_{x \sim p_{\text{data}}(x)} [\nabla_\theta \log p_\theta(x)] \tag{1}$$

Change of Variables. With an invertible transformation f, we can build a generative model

$$z \sim p(z), \; x = f^{-1}(z). \tag{2}$$

Then the log-density of x is given by

$$\log p(x) = \log p(f(x)) + \log \left| \det \frac{df(x)}{dx} \right|. \tag{3}$$

Flow-based generative models can be

1. sampled, if (2) can be computed or approximated up to some precision.
2. trained using maximum likelihood, if (3) can be unbiasedly estimated.

Background: Invertible Residual Networks (i-ResNets)

Residual networks are composed of simple transformations

$$y = f(x) = x + g(x) \tag{4}$$

Behrmann *et al.* (2019) proved that if g is a contractive mapping, ie. *Lipschitz strictly less than one*, then the residual block transformation (4) is invertible.

Sampling. The inverse f^{-1} can be efficiently computed by a fixed-point iteration

$$x^{(i+1)} = y - g(x^{(i)}) \tag{5}$$

which converges *superlinearly* by the Banach fixed-point theorem.

Log-density. The change of the variables can be applied to invertible residual networks

$$\log p(x) = \log p(f(x)) + \operatorname{tr}\left(\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} [J_g(x)]^k \right) \tag{6}$$

\+ Trace can be efficiently estimated using the *Skilling-Hutchinson* estimator.

– The infinite sum can be estimated by truncating to a fixed n. However, this introduces bias equal to the remaining terms.

This results in the biased estimator:

$$\log p(x) \approx \log p(f(x)) + \mathbb{E}_{v \sim \mathcal{N}(0,1)} \left(\sum_{k=1}^{n} \frac{(-1)^{k+1}}{k} v^T [J_g(x)]^k v \right) \tag{7}$$

where Behrmann *et al.* (2019) chose $n = 5, 10$.

Unbiased Log-density via "Russian Roulette" Estimator

Russian roulette estimator. To illustrate the idea, let Δ_k denote the k-th term of an infinite series, and suppose we always evaluate the first term then flip a coin $b \sim \text{Bernoulli}(q)$ to determine whether we stop or continue evaluating the remaining terms. By reweighting the remaining terms by $\frac{1}{1-q}$, we obtain

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

This unbiased estimator has probability q of being evaluated in finite time. We can obtain an estimator that is evaluated in finite time with probability one by applying this process infinitely many times to the remaining terms.

Residual Flows. Unbiased estimation of the log-density leads to our model.

$$\log p(x) = \log p(f(x)) + \mathbb{E}_{n,v} \left[\sum_{k=1}^{n} \frac{(-1)^{k+1}}{k} \frac{v^T [J_g(x)^k] v}{\mathbb{P}(N \geq k)} \right], \tag{8}$$

where $n \sim p(N)$ and $v \sim \mathcal{N}(0, I)$. We use a shifted geometric distribution for $p(N)$ with an expected compute of 4 terms.

Memory-efficient Gradient Estimation

Neumann gradient series. For estimating (1), we can either

(i) Estimate $\log p(x)$, then take gradient.

(ii) Take the gradient, then estimate $\nabla \log p(x)$.

The first option uses variable amount of memory depending on the sample of n:

$$\frac{\partial}{\partial \theta} \log \left| \det \frac{df(x)}{dx} \right| = \mathbb{E}_{n,v} \left[\sum_{k=1}^{n} \frac{(-1)^{k+1}}{k} \frac{\partial v^T (J_g(x,\theta)^k) v}{\partial \theta} \right]. \tag{9}$$

The second option, by using a Neumann series we obtain constant memory cost:

$$\frac{\partial}{\partial \theta} \log \left| \det \frac{df(x)}{dx} \right| = \mathbb{E}_{n,v} \left[\left(\sum_{k=0}^{n} \frac{(-1)^k}{\mathbb{P}(N \geq k)} v^T J(x,\theta)^k \right) \frac{\partial (J_g(x,\theta))}{\partial \theta} v \right] \tag{10}$$

Backward-in-forward. Since $\log \left| \det \frac{df(x)}{dx} \right|$ is a scalar quantity, we can compute its gradient early and free up memory. This reduces the amount of memory by a factor equal to the number of residual blocks with negligible cost.

	MNIST		CIFAR-10†		CIFAR-10		
	ELU	LipSwish	ELU	LipSwish	ELU	LipSwish	Relative
Naïve Backprop	92.0	192.1	33.3	66.4	120.2	263.5	100%
Neumann Gradient	13.4	31.2	5.5	11.3	17.6	40.8	15.7%
Backward-in-Forward	8.7	19.8	3.8	7.4	11.5	26.1	10.3%
Both Combined	4.9	13.6	3.0	5.9	6.6	18.0	7.1%

Table: Memory usage (GB) per minibatch of 64 samples when computing n=10 terms in the corresponding power series. †Uses immediate downsampling before any residual blocks.

LipSwish Activation for Enforcing Lipschitz Constraint

We motivate *smooth and non-monotonic* Lipschitz activation functions. This avoid "gradient saturation", which occurs if the 2nd derivative asymptotically approaches zero when the 1st derivative is close to one.

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

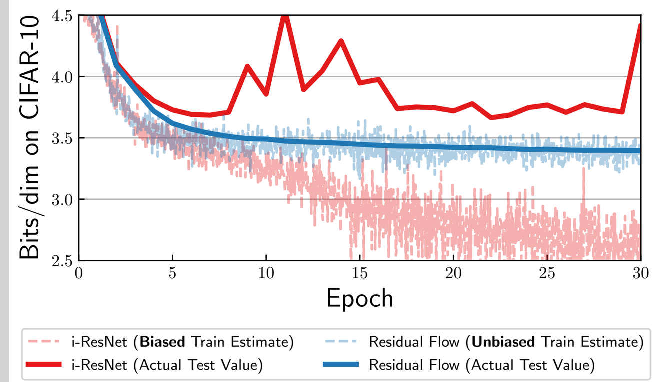
Density Modeling

We are competitive with existing state-of-the-art flow-based models on density estimation, when using uniform dequantization.

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 & 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**

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

Ablation Experiments



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: Ablation results. †Uses immediate downsampling before any residual blocks.

Qualitative Samples

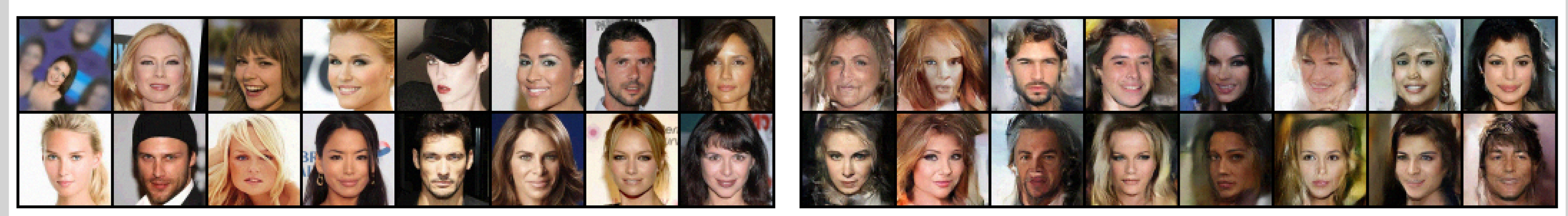


Figure: **Qualitative samples.** Real (left) and random samples (right) from a model trained on 5bit 64×64 CelebA. The most visually appealing samples were picked out of 5 random batches.

Hybrid Modeling

Residual blocks are better building blocks for hybrid models than coupling blocks. Trained using a weighted maximum likelihood objective similar to (Nalisnick *et al.*, 2019).

$$\mathbb{E}_{(x,y) \sim p_{\text{data}}} [\lambda \log p(x) + \log p(y|x)] \tag{11}$$

	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%**

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

References

- Behrmann *et al.*. "Invertible residual networks." (2019)
- Kahn. "Use of different monte carlo sampling techniques." (1955)
- Beatson & Adams. "Efficient Optimization of Loops and Limits with Randomized Telescoping Sums." (2019)
- Nalisnick *et al.*. "Hybrid Models with Deep and Invertible Features." (2019)