NF 3: Improved, Neural, Residual Flows

CS 598: Deep Generative and Dynamical Models

Instructor: Arindam Banerjee

October 7, 2021

De-quantization

- De-quantization: Images $x \in \{0, \ldots, 255\}^D$
  - Create noisy image: $y = x + u$, $u \in [0, 1[^D$, density $p(y)$ on $[0, 256]^D$
  - Fit models $p_{\text{model}}(y)$ based on y
- Can be interpreted as lower bound maximization on

$$P_{\text{model}}(x) = \int_{[0,1)^D} p_{\text{model}}(x + u) du$$

- With $x \sim P_{\text{data}}$ and $y \sim p_{\text{data}}$

$$\mathbb{E}_{y \sim p_{\text{data}}}[\log p_{\text{model}}(y)] \leq \mathbb{E}_{x \sim P_{\text{data}}}[\log P_{\text{model}}(x)]$$

Variational Dequantization

- Rather than using uniform, use $q(\mathrm{u}|\mathrm{x})$

$$\mathbb{E}_{\mathrm{x}\sim P_{\text{data}}}[\log P_{\text{model}}(\mathrm{x})] \geq \mathbb{E}_{\mathrm{x}\sim P_{\text{data}}}\mathbb{E}_{\mathrm{u}\sim q(\cdot|\mathrm{x})}\left[\log \frac{p_{\text{model}}(\mathrm{x}+\mathrm{u})}{q(\mathrm{u}|\mathrm{x})}\right]$$

- Use a conditional flow for u, i.e., $\mathrm{u} = q_{\mathrm{x}}(\epsilon), \epsilon \sim p(\epsilon) = \mathcal{N}(\epsilon; 0, \mathbb{I})$
- With a flow model for u, we have

$$\mathbb{E}_{\mathrm{x}\sim P_{\text{data}}}[\log P_{\text{model}}(\mathrm{x})] \geq \mathbb{E}_{\mathrm{x}\sim P_{\text{data}}}\mathbb{E}_{\epsilon\sim p}\left[\log \frac{p_{\text{model}}(\mathrm{x}+q_{\mathrm{x}}(\epsilon))}{p(\epsilon)|\partial q_{\mathrm{x}}/\partial\epsilon|^{-1}}\right]$$

- Flow model for p_{model}
  - Invertible f with $x + q_{\mathrm{x}}(\epsilon) = f^{-1}(\mathrm{z})$
  - Can use SGD, $f(\mathrm{x} + q_{\mathrm{x}}(\epsilon))$ is differentiable w.r.t. parameters in f, q
- Uniform q is a special case, inexpressive, poor lower bound
- VI idea for (flow based) dequantization for fitting a flow model

Improved Coupling Layers

- Paramtereized flow $y = f_\theta(x)$
- Affine coupling layers: splits $x = (x_1, x_2)$ to get

$$y_1 = x_1 \qquad y_2 = x_2 \cdot \exp(a_\theta(x_1)) + b_\theta(x_1)$$

  - $a_\theta(x_1), b_\theta(x_1)$ are neural networks
  - But still just an affine transform of x_2
- The inverse map is

$$x_1 = y_1$$
$$x_2 = (y_2 - b_\theta(y_1)) \cdot \exp(-a_\theta(y_1))$$
$$\log \left| \frac{\partial y}{\partial x} \right| = 1^T a_\theta(x)$$

- Can we do more general elementwise non-linear transformations?

Coupling with Mixture of CDFs

- Mixture of k sigmoids, viewed as CDF of logistics

$$\text{MixLogCDF}(x; \pi, \mu.\text{s}) = \sum_{i=1}^{K} \pi_i \sigma((x - \mu_i) \cdot \exp(-s_i))$$

  - π are the mixture probabilities, $\pi^T 1 = 1, \pi \geq 0$
  - μ, s determine the mean and scaling respectively
- Such mapping is followed by inverse sigmoid, and affine transform

$$\text{x} \mapsto \sigma^{-1}(\text{MixLogCDF}(\text{x}; \pi, \mu, \text{s})) \cdot \exp(a) + b)$$

- Non-linear coupling transformation

$$\text{y}_1 = \text{x}_1$$
$$\text{y}_2 = \sigma^{-1}(\text{MixLogCDF}(\text{x}_2; \pi_\theta(\text{x}_1), \mu_\theta(\text{x}_1), \text{s}_\theta(\text{x})) \cdot \exp(a_\theta(\text{x}_1) + b_\theta(\text{x}_1))$$

- Can be inverted by bisection, since CDF is monotonic increasing
- Jacobian is based on density function of logistic mixtures

Elementwise Models with Attention

- Elementwise transform parameters $\pi, \mu, \mathsf{s}, \mathsf{a}, b$
- Multi-head self-attention, similar to transformers
- Each block has two layers, connected as residuals

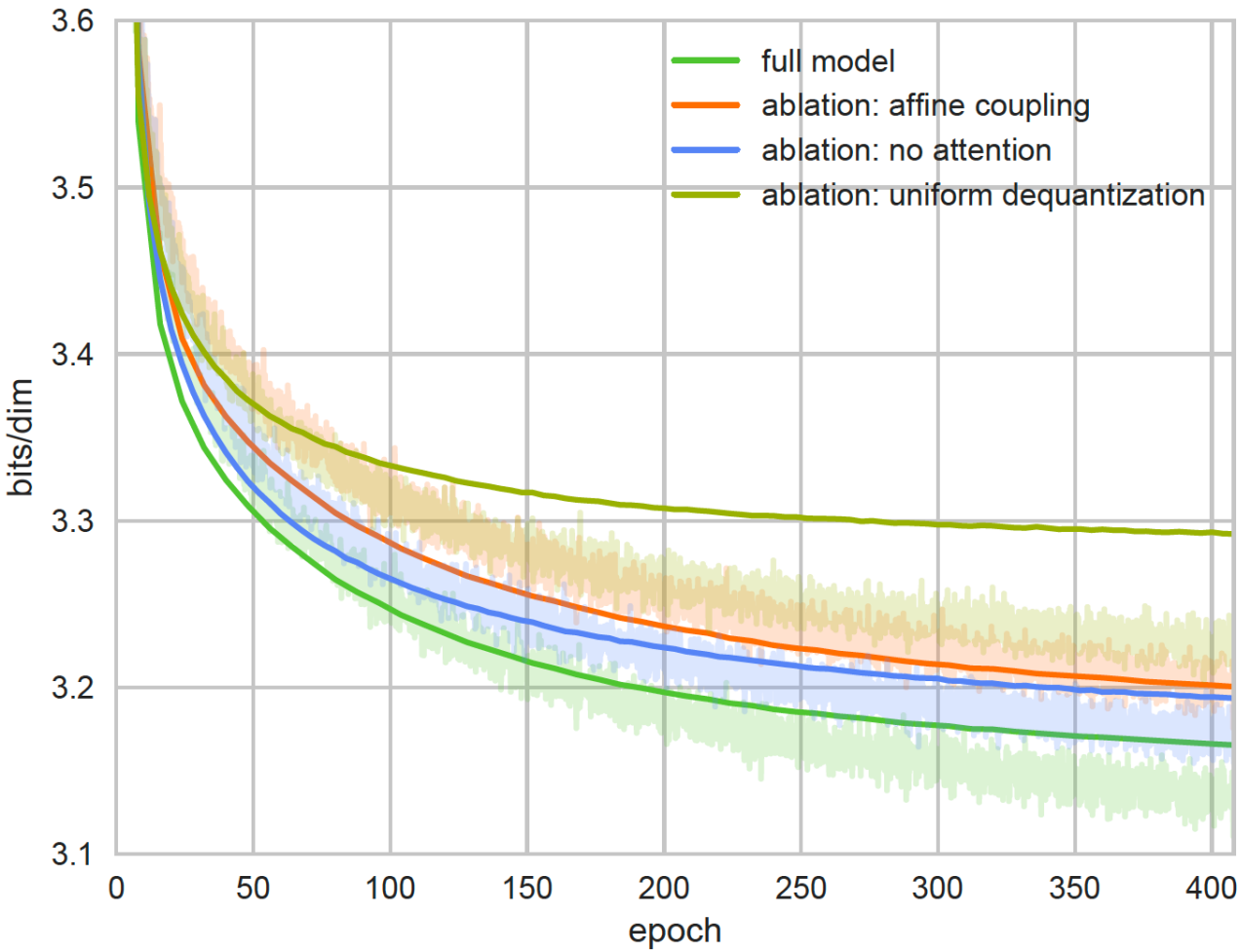
$$
\begin{aligned}
\text{Conv} &= \text{Input} \to \text{Nonlinearity} \\
&\to \text{Conv}_{3\times 3} \to \text{Nonlinearity} \to \text{Gate} \\
\text{Attn} &= \text{Input} \to \text{Conv}_{1\times 1} \\
&\to \text{MultiHeadSelfAttention} \to \text{Gate}
\end{aligned}
$$

Results: Bits/dim

Table 1. Unconditional image modeling results in bits/dim

Model family	Model	CIFAR10	ImageNet 32x32	ImageNet 64x64
Non-autoregressive	RealNVP (Dinh et al., 2016)	3.49	4.28	–
	Glow (Kingma & Dhariwal, 2018)	3.35	4.09	3.81
	IAF-VAE (Kingma et al., 2016)	3.11	–	–
	Flow++ (ours)	**3.08**	**3.86**	**3.69**
Autoregressive	Multiscale PixelCNN (Reed et al., 2017)	–	3.95	3.70
	PixelCNN (van den Oord et al., 2016b)	3.14	–	–
	PixelRNN (van den Oord et al., 2016b)	3.00	3.86	3.63
	Gated PixelCNN (van den Oord et al., 2016c)	3.03	3.83	3.57
	PixelCNN++ (Salimans et al., 2017)	2.92	–	–
	Image Transformer (Parmar et al., 2018)	2.90	3.77	–
	PixelSNAIL (Chen et al., 2017)	2.85	3.80	3.52

Results: Ablation

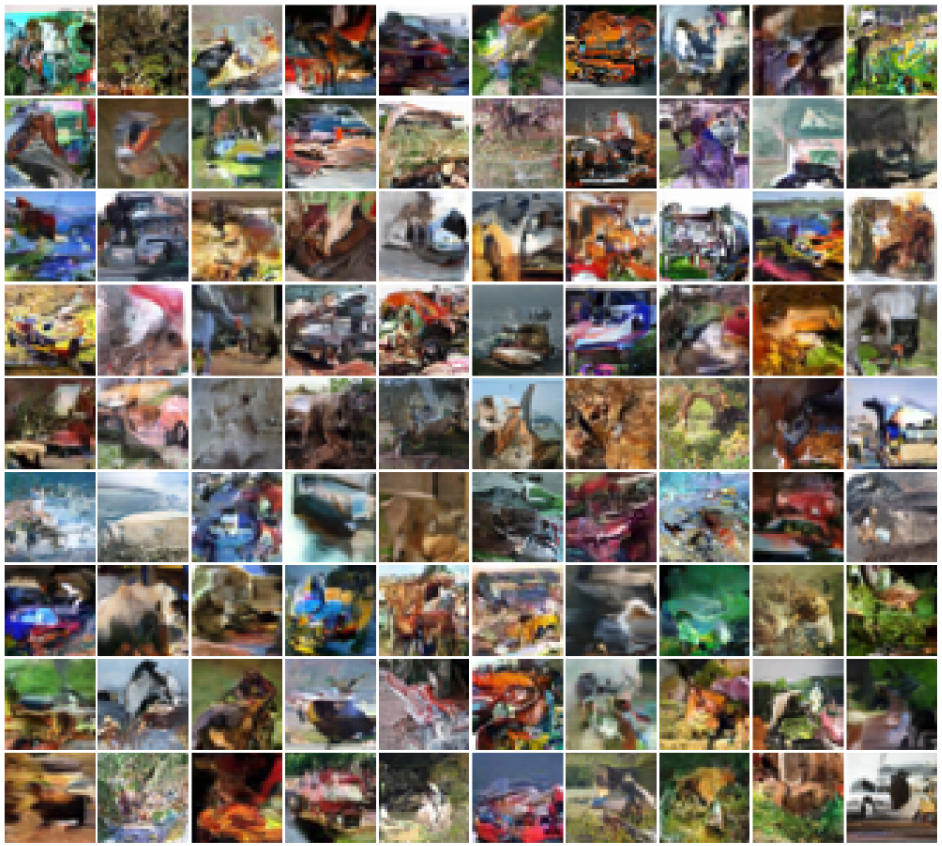


Results: Ablation

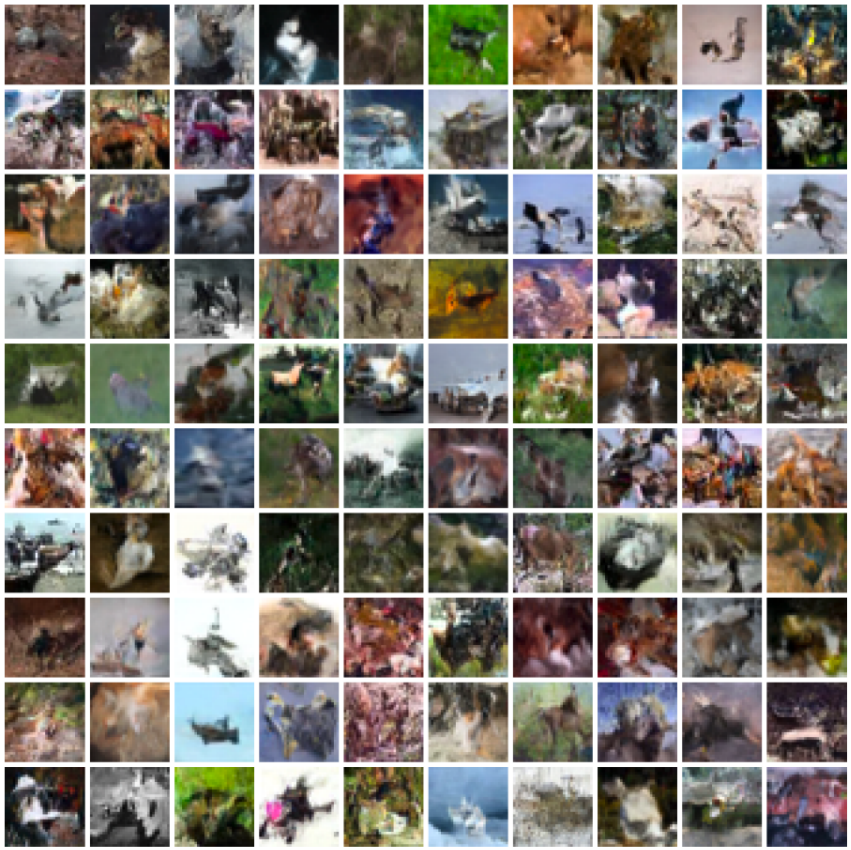
Table 2. CIFAR10 ablation results after 400 epochs of training. Models not converged for the purposes of ablation study.

Ablation	bits/dim	parameters
uniform dequantization	3.292	32.3M
affine coupling	3.200	32.0M
no self-attention	3.193	31.4M
Flow++ (not converged for ablation)	**3.165**	**31.4M**

Results: Samples, Cifar-10



(a) PixelCNN

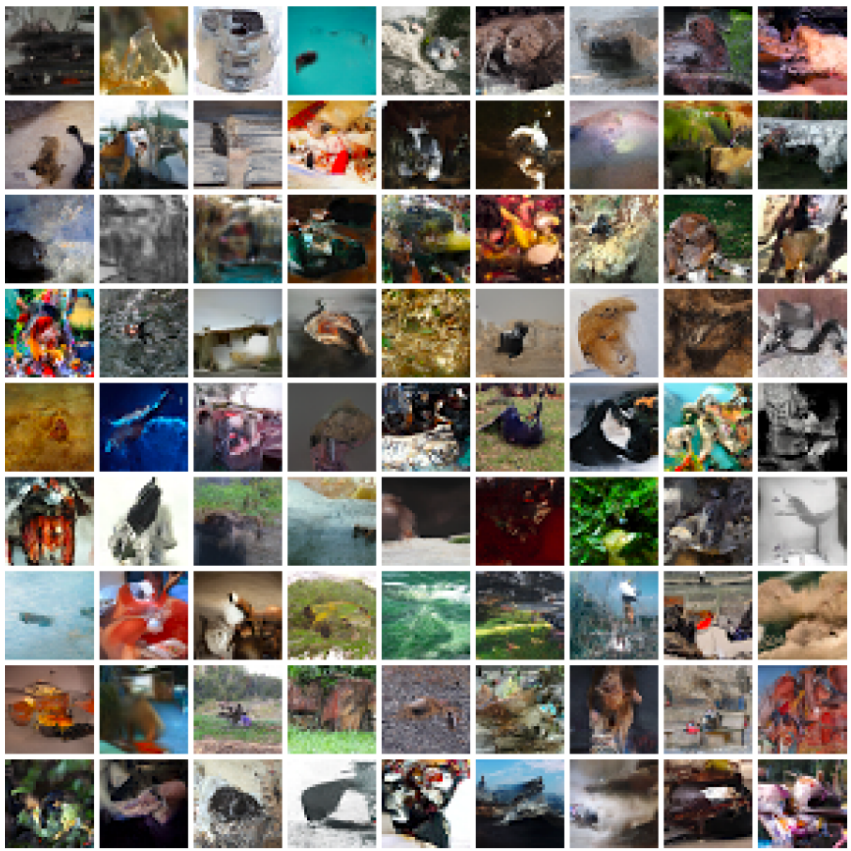


(b) Flow++

Figure 2. CIFAR 10 Samples. Left: samples from van den Oord et al. (2016b). Right: samples from Flow++, which captures local dependencies well and generates good samples at the quality level of PixelCNN, but with the advantage of efficient sampling.

Results: Samples, ImageNet

(a) PixelCNN



(b) Flow++

Figure 3. 32x32 ImageNet Samples. Left: samples from van den Oord et al. (2016b). Right: samples from Flow++. Note that diversity of samples from Flow++ matches the diversity of samples from an autoregressive model on this dataset, which is much larger than CIFAR10.

Results: Samples, CelebA

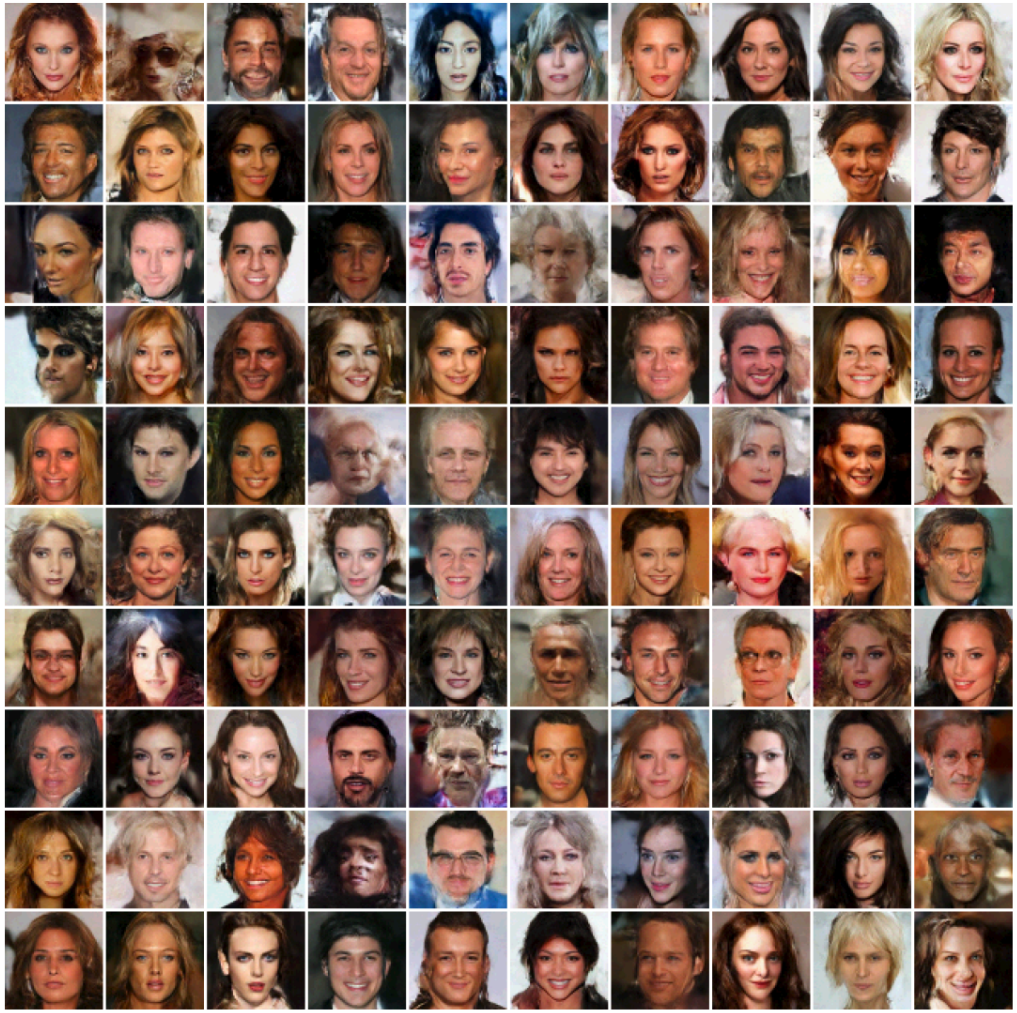


Figure 4. Samples from Flow++ trained on 5-bit 64x64 CelebA, without low-temperature sampling.

Flow models

- Invertible mapping $f_\theta : \mathcal{X} \mapsto \mathcal{Y}$, $\mathrm{y} = f_\theta(\mathrm{x})$

$$p_Y(\mathrm{y}) = \left| \frac{\partial f(\mathrm{x})}{\partial \mathrm{x}} \right|^{-1} p_X(\mathrm{x})$$

- Minimize $KL(p_Y(\mathrm{y}) \| p_{\text{target}}(\mathrm{y}))$
- Maximum-likelihood, density estimation
  - $p_X(\mathrm{x})$ is a complex data distribution
  - $p_{\text{target}}(\mathrm{y}))$ is a simple distribution, say isotropic Gaussian
  - f_θ maps x to y, so $p_X(\mathrm{x})$ can be computed
- Variational inference
  - $p_X(\mathrm{x})$ is a simple distribution, e.g., Gaussian from encoder network
  - $p_{\text{target}}(\mathrm{y}))$ is a complex distribution, approximating true posterior
  - f_θ maps simple distribution to more complex/flexible

Desiderata: Key Computational Steps

- Sampling $\mathrm{x} \sim p(\mathrm{x})$
- Computing $\mathrm{y} = f(\mathrm{x})$
- Computing gradient of log-likelihood of $y = f(\mathrm{x})$
- Computing the gradient of the log-det of the Jacobian of f

Affine Autoregressive Flows

- Order x, y
- Compute y_t as a function of $x_{1:t-1}$
- A conditioner c and transformation τ

$$y_t = \tau(c(x_{1:t-1}), x_t)$$

- Focus has been on affine transformations, for simplicity

$$\tau(\mu, \sigma, x_t) = \mu + \sigma x_t$$
$$\tau(\mu, \sigma, x_t) = \sigma x_t + (1 - \sigma)\mu$$

- Affine is not necessary, properties we need
  - τ must be an invertible function of x_t
  - $\frac{dy_t}{dx_t}$ must be tractable to compute

Neural Autoregressive Flows (NAFs)

- Use deep nets for complex monotonic transform

$$\tau(c(x_{1:t-1}), x_t) = \text{DNN}(x_t; \phi = c(x_{1:t-1}))$$

- NAF use monotonic transforms, view as CDF

> **Proposition 1.** *Using strictly positive weights and strictly monotonic activation functions for τ_c is sufficient for the entire network to be strictly monotonic.*

- Can model multi-modality by going from affine to monotonic

Example: Multi-modal Modeling

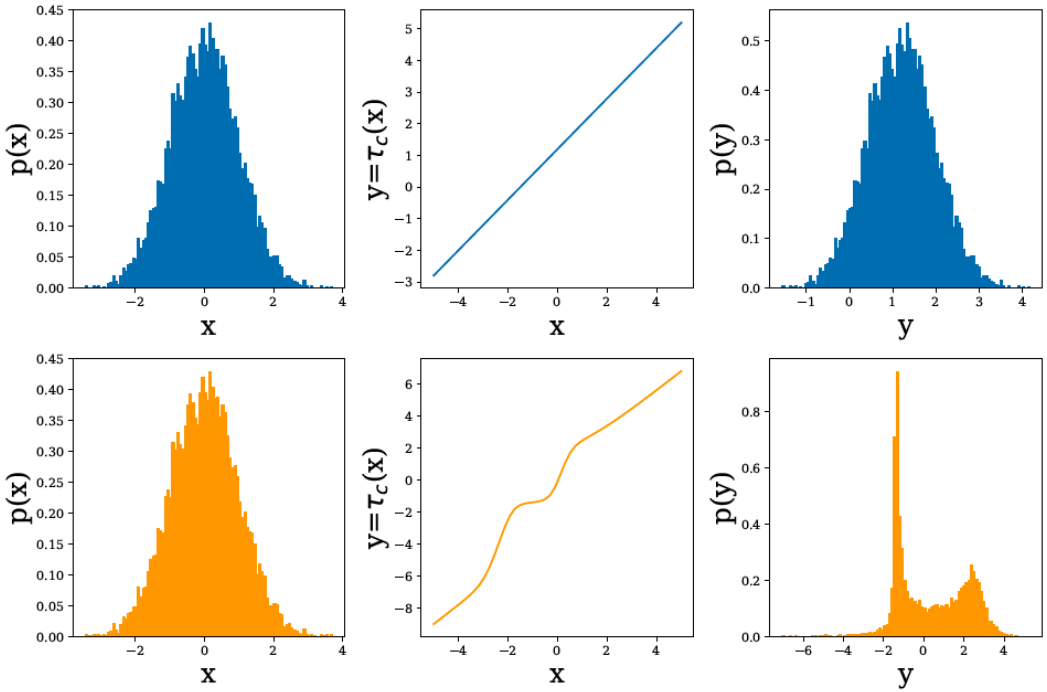


Figure 5. Illustration of the effects of traditional IAF (top), and our proposed NAF (bottom). Areas where the slope of the transformer τ_c is greater/less than 1, are compressed/expanded (respectively) in the output distribution. Inflection points in $\tau_c(x_t)$ (middle) can transform a unimodal $p(x_t)$ (left) into a multimodal $p(y_t)$ (right); NAF allows for such inflection points, whereas IAF does not.

Architectures

- Two types of architectures
- Deep Sigmoidal Flows (DSF)
  - Single hidden layer, with sigmoid and inverse sigmoid

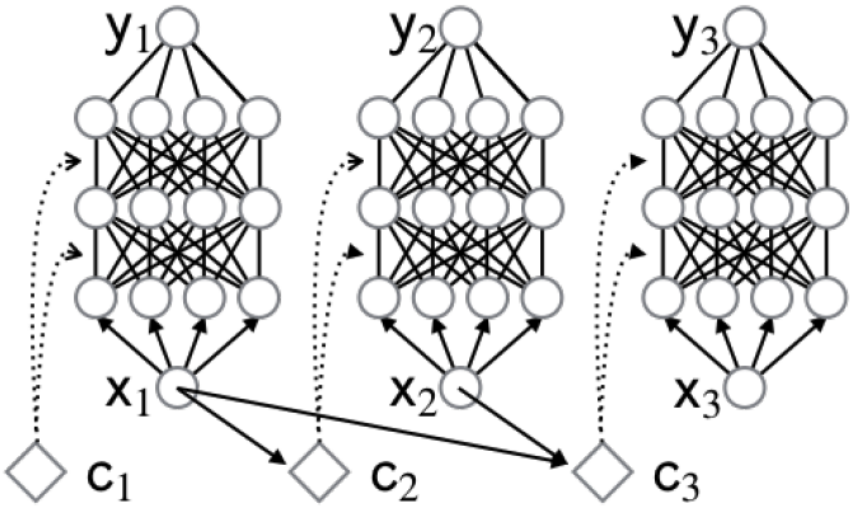
$$y_t = \sigma^{-1}(\mathrm{w} \cdot \sigma(a \cdot x_t + b)) \ , \quad \mathrm{w}^T 1 = 1, w > 0, a > 0$$

- Deep Dense Sigmoidal Flows (DDSF)
  - Stacking layers of DSF looks like a MLP with bottlenecks
  - Dense version avoids the bottleneck

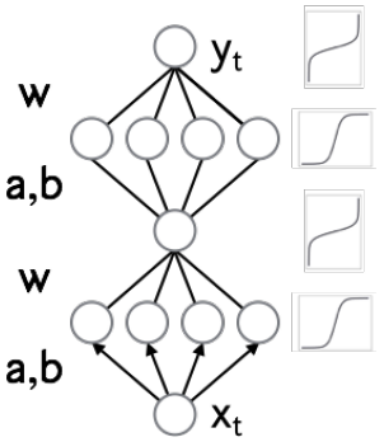
$$h^{(l+1)} = \sigma^{-1}(\mathrm{w} \cdot \sigma(a^{(l+1)} \odot u^{(l+1)} \cdot h^{(l)} + b^{(l+1)}))$$

    - $h_0 = x, h_L = y$; $d_0 = d_L = 1$
    - w continues to be a distribution, all parameters except b are positive
- Jacobian can be computed by chain rule

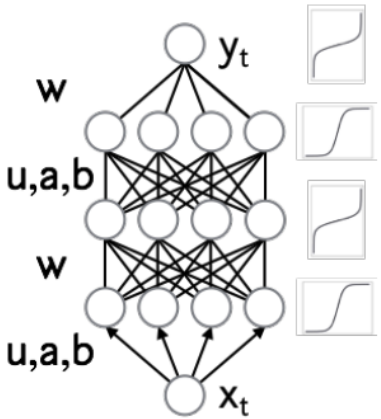
NAF Architectures: DSF, DDSF



(a) Neural autoregressive flows (NAF)



(b) DSF



(c) DDSF

Universal Density Approximators

Proposition 2. (DSF universally transforms uniform random variables into any desired random variables) *Let Y be a random vector in $\mathbb{R}^m$ and assume Y has a strictly positive and continuous probability density distribution. Let $X \sim \text{Unif}((0,1)^m)$. Then there exists a sequence of functions $(G_n)_{n\geq 1}$ parameterized by autoregressive neural networks in the following form*

$$G(\mathbf{x})_t = \sigma^{-1}\left(\mathcal{S}\left(x_t; \mathcal{C}_t(x_{1:t-1})\right)\right) \tag{12}$$

where $\mathcal{C}_t = (a_{tj}, b_{tj}, \tau_{tj})_{j=1}^n$ are functions of $x_{1:t-1}$, such that $Y_n \doteq G_n(X)$ converges in distribution to Y.

Universal Density Approximators

Proposition 3. (DSF universally transforms any random variables into uniformly distributed random variables) *Let X be a random vector in an open set $\mathcal{U} \subset \mathbb{R}^m$. Assume X has a positive and continuous probability density distribution. Let $Y \sim \text{Unif}((0,1)^m)$. Then there exists a sequence of functions $(H_n)_{n\geq 1}$ parameterized by autoregressive neural networks in the following form*

$$H(\mathbf{x})_t = \mathcal{S}\left(x_t; \mathcal{C}_t(x_{1:t-1})\right) \tag{13}$$

where $\mathcal{C}_t = (a_{tj}, b_{tj}, \tau_{tj})_{j=1}^n$ are functions of $x_{1:t-1}$, such that $Y_n \doteq H_n(X)$ converges in distribution to Y.

Universal Density Approximators

Theorem 1. (DSF universally transforms any random variables into any desired random variables) *Let X be a random vector in an open set $\mathcal{U} \subset \mathbb{R}^m$. Let Y be a random vector in $\mathbb{R}^m$. Assume both X and Y have a positive and continuous probability density distribution. Then there exists a sequence of functions $(K_n)_{n \geq 1}$ parameterized by autoregressive neural networks in the following form*

$$K(\mathbf{x})_t = \sigma^{-1}\left(\mathcal{S}\left(x_t; \mathcal{C}_t(x_{1:t-1})\right)\right) \tag{14}$$

where $\mathcal{C}_t = (a_{tj}, b_{tj}, \tau_{tj})_{j=1}^n$ are functions of $x_{1:t-1}$, such that $Y_n \doteq K_n(X)$ converges in distribution to Y.

Results: Log-likelihood

Table 2. Test log-likelihood and error bars of 2 standard deviations on the 5 datasets (5 trials of experiments). Neural autoregressive flows (NAFs) produce state-of-the-art density estimation results on all 5 datasets. The numbers (5 or 10) in parantheses indicate the number of transformations which were stacked; for TAN (Oliva et al., 2018), we include their best results, achieved using different architectures on different datasets. We also include validation results to give future researchers a fair way of comparing their methods with ours during development.

Model	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
MADE MoG	0.40 ± 0.01	8.47 ± 0.02	-15.15 ± 0.02	-12.27 ± 0.47	153.71 ± 0.28
MAF-affine (5)	0.14 ± 0.01	9.07 ± 0.02	-17.70 ± 0.02	-11.75 ± 0.44	155.69 ± 0.28
MAF-affine (10)	0.24 ± 0.01	10.08 ± 0.02	-17.73 ± 0.02	-12.24 ± 0.45	154.93 ± 0.28
MAF-affine MoG (5)	0.30 ± 0.01	9.59 ± 0.02	-17.39 ± 0.02	-11.68 ± 0.44	156.36 ± 0.28
TAN (various architectures)	0.48 ± 0.01	11.19 ± 0.02	-15.12 ± 0.02	-11.01 ± 0.48	157.03 ± 0.07
MAF-DDSF (5)	$\mathbf{0.62 \pm 0.01}$	11.91 ± 0.13	$\mathbf{-15.09 \pm 0.40}$	$\mathbf{-8.86 \pm 0.15}$	$\mathbf{157.73 \pm 0.04}$
MAF-DDSF (10)	0.60 ± 0.02	$\mathbf{11.96 \pm 0.33}$	-15.32 ± 0.23	-9.01 ± 0.01	157.43 ± 0.30
MAF-DDSF (5) valid	0.63 ± 0.01	11.91 ± 0.13	15.10 ± 0.42	-8.38 ± 0.13	172.89 ± 0.04
MAF-DDSF (10) valid	0.60 ± 0.02	11.95 ± 0.33	15.34 ± 0.24	-8.50 ± 0.03	172.58 ± 0.32

Results: Variational Inference

Table 1. Using DSF to improve variational inference. We report the number of affine IAF with our implementation. We note that the log likelihood reported by Kingma et al. (2016) is 78.88. The average and standard deviation are carried out with 5 trials of experiments with different random seeds.

Model	ELBO	$\log p(x)$
VAE	85.00 ± 0.03	81.66 ± 0.05
IAF-affine	82.25 ± 0.05	80.05 ± 0.04
IAF-DSF	81.92 ± 0.04	79.86 ± 0.01

Results: Grid of Gaussians

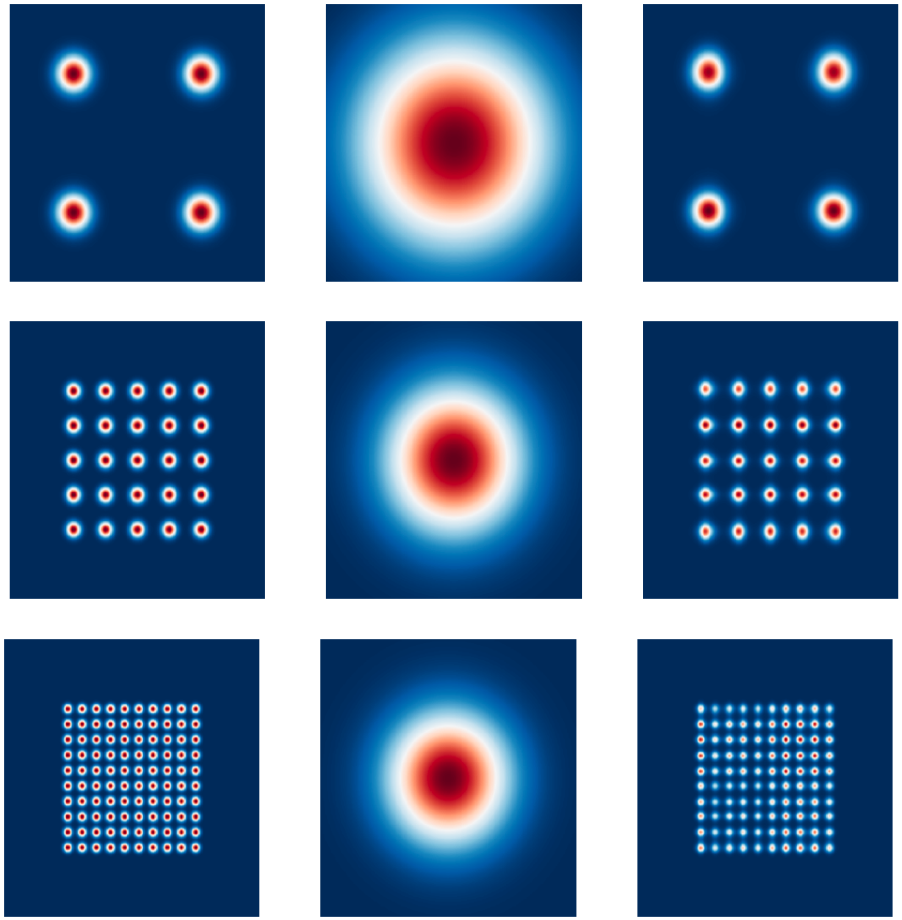


Figure 6. Fitting grid of Gaussian distributions using maximum likelihood. Left: true distribution. Center: affine autoregressive flow (AAF). Right: neural autoregressive flow (NAF)

Results: Learning Curves

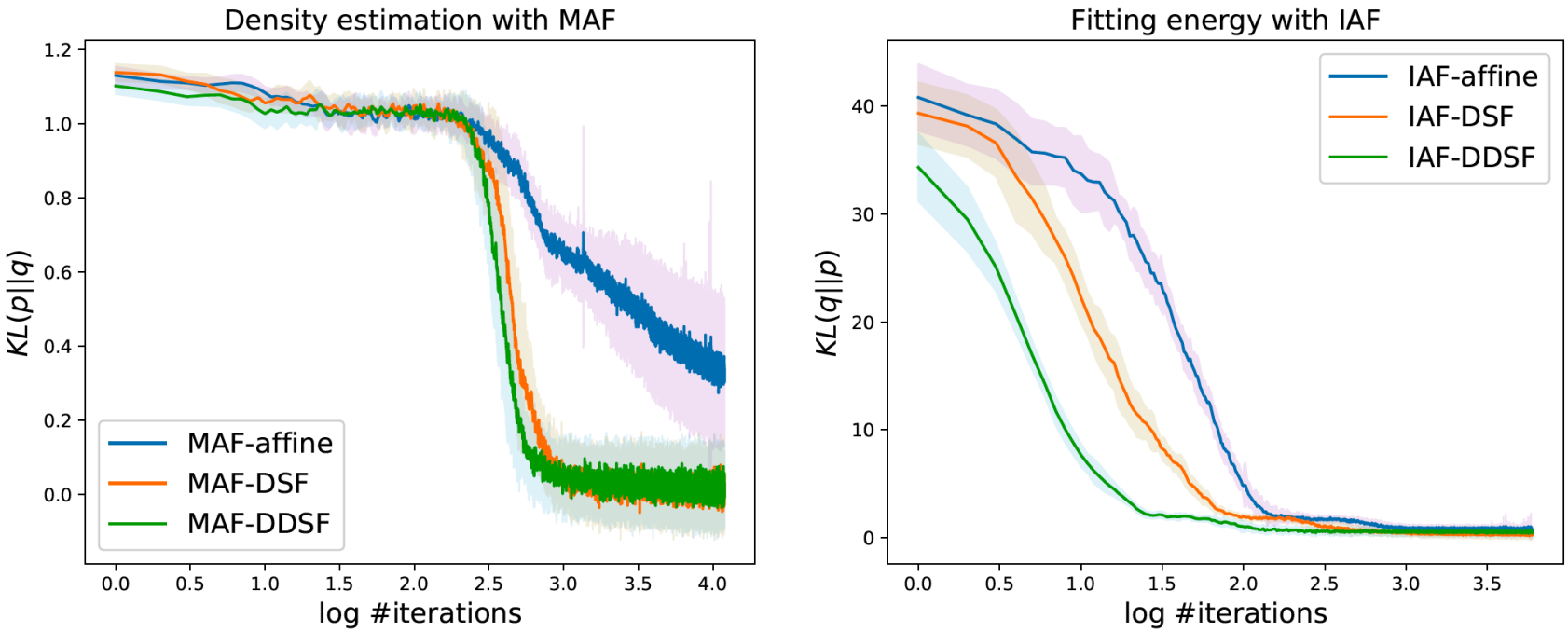
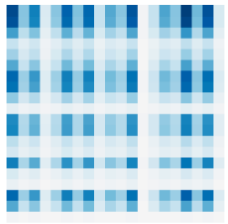
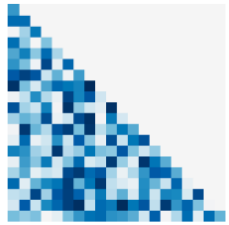


Figure 7. Learning curve of MAF-style and IAF-style training. q denotes our trained model, and p denotes the target.

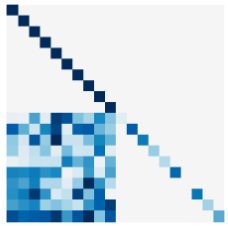
Jacobian Structure for Flows



(a) Det. Identities (Low Rank)



(b) Autoregressive (Lower Triangular)



(c) Coupling (Structured Sparsity)

(d) **Unbiased Est.** (Free-form)

Figure 1: **Pathways to designing scalable normalizing flows** and their enforced Jacobian structure. Residual Flows fall under unbiased estimation with free-form Jacobian.

Invertible Residual Networks (i-ResNets)

- Recall change of variables $y = f(x)$

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

- Residual networks: $y = x + g(x)$
  - Transformation is invertible if g is a contraction
  - With Jacobian $J_g = \frac{dg(x)}{dx}$, we have

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

- In practice, truncate the infinite series
  - Biased estimate of the stochastic gradient
  - Bias is worse with increase in dimension and Lipschitz constant of g

Unbiased Estimation of Infinite Series

- Estimating an infinite series, k-th term Δ_k
  - Evaluate first term Δ_1
  - Draw $b \sim \text{Bern}(q)$ to decide whether to stop or continue

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

  - Estimator by drawing the number of terms evaluated

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

Theorem 1 (Unbiased log density estimator). *Let $f(x) = x + g(x)$ with $\text{Lip}(g) < 1$ and N be a random variable with support over the positive integers. Then*

$$\log p(x) = \log p(f(x)) + \mathbb{E}_{n,v}\left[\sum_{k=1}^{n}\frac{(-1)^{k+1}}{k}\frac{v^T[J_g(x)^k]v}{\mathbb{P}(N \geq k)}\right], \tag{6}$$

where $n \sim p(N)$ and $v \sim \mathcal{N}(0, I)$.

Biased vs. Unbiased Stochastic Gradients

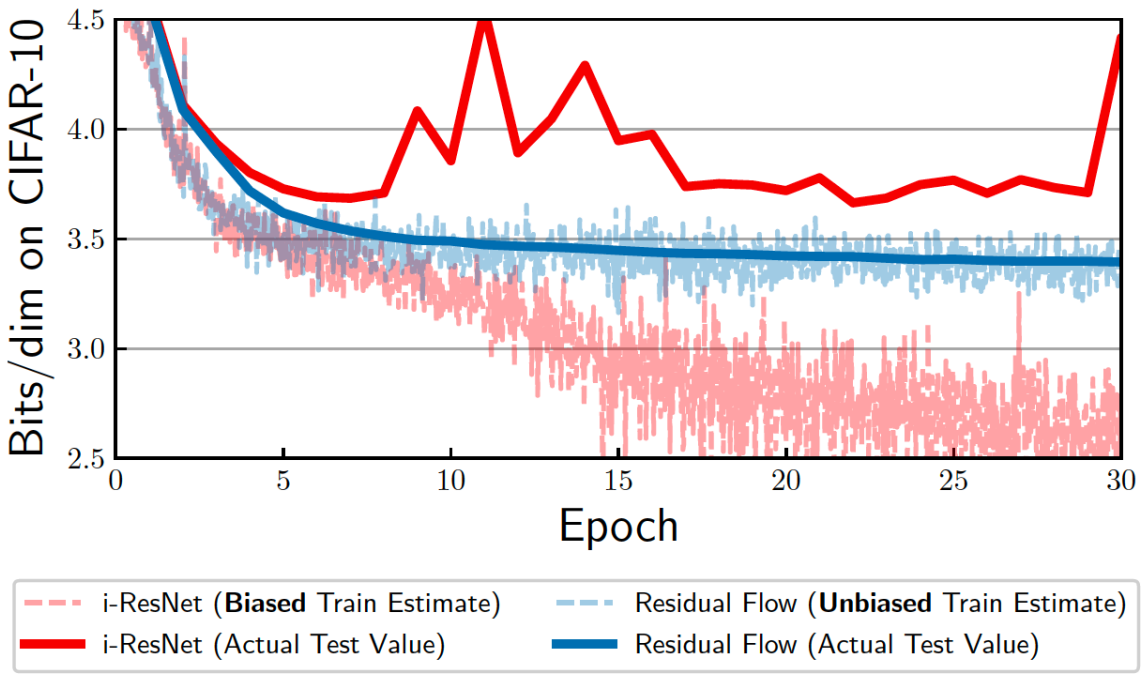


Figure 2: i-ResNets suffer from substantial bias when using expressive networks, whereas Residual Flows principledly perform maximum likelihood with unbiased stochastic gradients.

Memory Efficient Backpropagation

- Naive backprop is problematic

$$\frac{\partial}{\partial \theta} \log \det(I + J_g(\mathrm{x}, \theta)) = \mathbb{E}_{n,v} \left[\sum_{k=1}^{n} \frac{(-1)^{k+1}}{k} \frac{\partial v^T (J_g(\mathrm{x}, \theta))^k) v}{\partial \theta} \right]$$

Theorem 2 (Unbiased log-determinant gradient estimator). *Let* $\mathrm{Lip}(g) < 1$ *and* N *be a random variable with support over positive integers. Then*

$$\frac{\partial}{\partial \theta} \log \det \left(I + J_g(x, \theta)\right) = \mathbb{E}_{n,v} \left[\left(\sum_{k=0}^{n} \frac{(-1)^k}{\mathbb{P}(N \geq k)} v^T J(x, \theta)^k \right) \frac{\partial (J_g(x, \theta))}{\partial \theta} v \right], \qquad (8)$$

where $n \sim p(N)$ *and* $v \sim \mathcal{N}(0, I)$.

- Additional efficiencies by chain rule on log-det, a scaler

Memory Usage

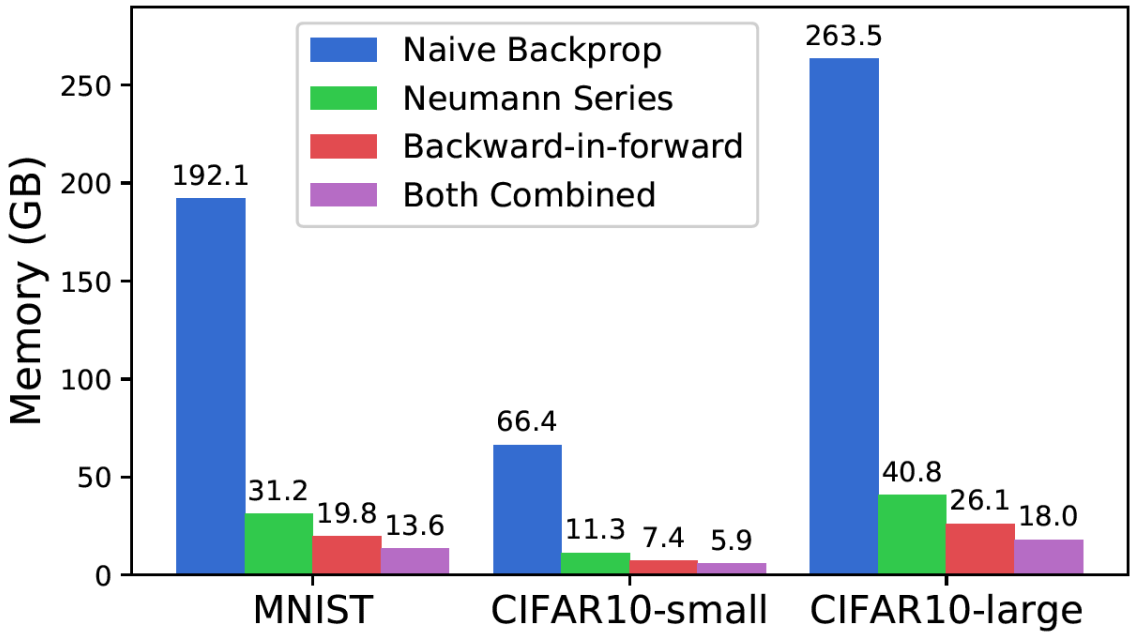


Figure 3: Memory usage (GB) per minibatch of 64 samples when computing n=10 terms in the corresponding power series. *CIFAR10-small* uses immediate downsampling before any residual blocks.

Avoiding Derivative Saturation

- Training depends on Hessian, i.e., derivative of the Jacobian
- Gradient saturation leads to slow training
- Desirable properties
  - Bounded derivatives $|\phi'(z)| \leq 1$
  - Second derivative should not be zero when $|\phi'(z)|$ close to 1
- Good choices: smooth and non-monotonic
- Scaled version of the Swish function

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

Activation Functions

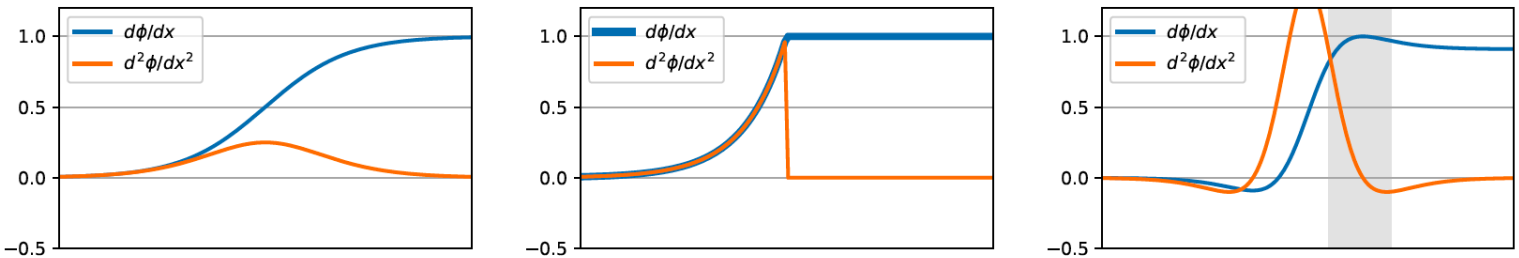


Figure 4: Common smooth Lipschitz activation functions ϕ usually have vanishing ϕ'' when ϕ' is maximal. LipSwish has a non-vanishing ϕ'' in the region where ϕ' is close to one.

Results: Bits/dim on Benchmarks

Table 1: 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	ImageNet 64	CelebA-HQ 256
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	1.03
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)	**0.970**	**3.280**	**4.010**	**3.757**	**0.992**

Results: Samples, CelebA

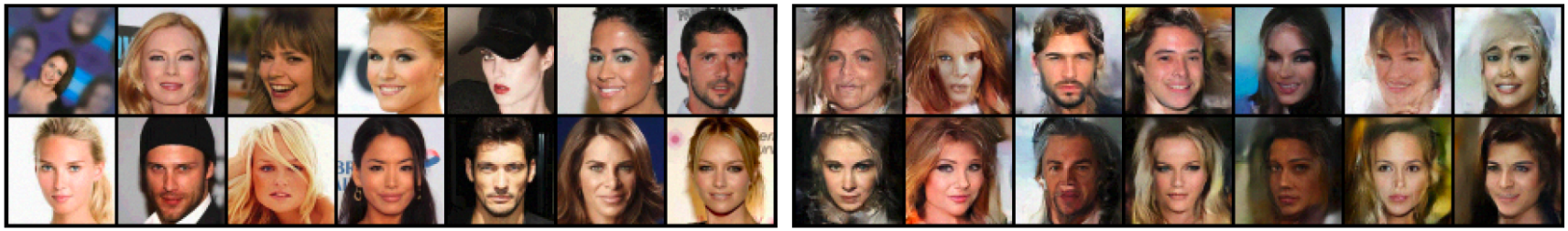
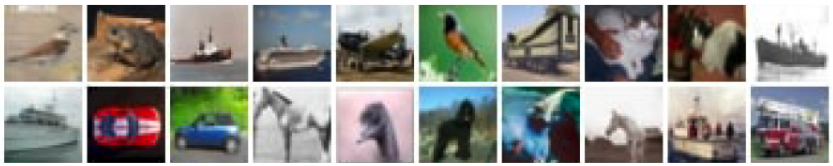


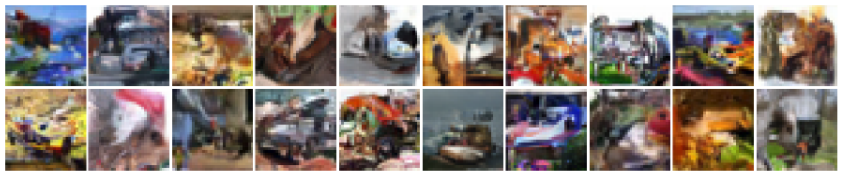
Figure 5: **Qualitative samples.** Real (left) and random samples (right) from a model trained on 5bit 64×64 CelebA. The most visually appealing samples were picked out of 5 random batches.

Results: Samples, CIFAR-10

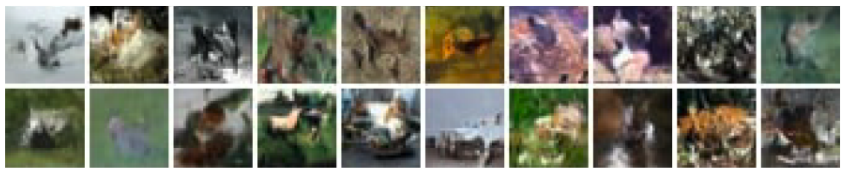


CIFAR-10 Real Data

Residual Flow (3.29 bits/dim)



PixelCNN (3.14 bits/dim)



Variational Dequantized Flow++ (3.08 bits/dim)

Figure 6: Random samples from Residual Flow are more globally coherent. PixelCNN (Oord et al., 2016) and Flow++ samples reprinted from Ho et al. (2019).

Results: FID, CIFAR-10

Table 2: Lower FID implies better sample quality. *Results taken from Ostrovski et al. (2018).

Model	CIFAR10 FID
PixelCNN*	65.93
PixelIQN*	49.46
i-ResNet	65.01
Glow	46.90
Residual Flow	**46.37**
DCGAN*	37.11
WGAN-GP*	36.40

Results: Reduced Entropy Sampling

Figure 7: Reduced entropy sampling does not equate with proper temperature annealing for general flow-based models. Naïvely reducing entropy results in samples that exhibit black hair and background, indicating that samples are not converging to the mode of the distribution.

Results: Effect of Activation Function

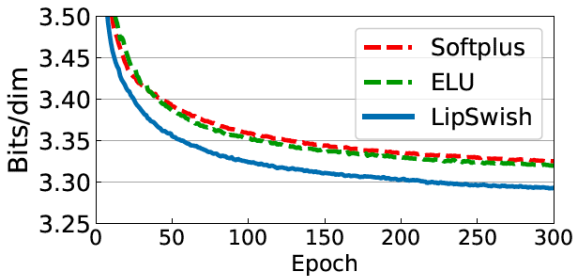


Figure 8: Effect of activation functions on CIFAR-10.

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**	**3.28**

Table 3: Ablation results. †Uses immediate downsampling before any residual blocks.

Results: Hybrid Modeling

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%**

References

- J. Ho, X. Chen, A. Srinivas, Y. Duan, P. Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design.. ICML, 2019.
- C.-W. Huang, D. Krueger, A. Lacoste, A. Courville. Neural Autoregressive Flows. ICML, 2018.
- R. Chen, J. Behrmann, D. Duvenaud, J. Jacobsen. Residual Flows for Invertible Generative Modeling. NeurIPS, 2019.