Residual Flows: Unbiased Generative Modeling with Norm-Learned i-ResNets

Ricky T. Q. Chen [*1] Jens Behrmann [*2] Jörn-Henrik Jacobsen [1]

Abstract

Flow-based generative models parameterize probability distributions through an invertible transformation and can be trained by maximum likelihood. Invertible residual networks provide a flexible family of transformations where only Lipschitz conditions rather than strict architectural constraints are needed for enforcing invertibility. However, prior work trained invertible residual networks for density estimation by relying on biased log-density estimates whose bias increased with the network's expressiveness. We give a tractable unbiased estima1te of the log density, and reduce the memory required during training by a factor of ten. Furthermore, we improve invertible residual blocks by proposing the use of activation functions that avoid gradient saturation and generalizing the Lipschitz condition to induced mixed norms. The resulting approach, called Residual Flows, achieves state-of-the-art performance on density estimation amongst flow-based models.

1. Introduction

Flow-based generative models are a class of models that make use of an invertible transformation f and the change of variables formula

$$\log p(x) = \log p(f(x)) + \log \left| \det \frac{df(x)}{dx} \right|, \tag{1}$$

where a simple base distribution is used for $\log p(f(x))$. This class of model can be trained with maximum likelihood, unlike variational autoencoders (Kingma and Welling, 2014) or generative adversarial nets (Goodfellow et al., 2014).

Recently Behrmann et al. (2019) showed that residual networks (He et al., 2016) composed of simple transformations

$$f(x) = x + g(x) \tag{2}$$

[*]Equal contribution [1]Vector Institute and University of Toronto [2]University of Bremen, Center for Industrial Mathematics.

First workshop on *Invertible Neural Networks and Normalizing Flows* (ICML 2019), Long Beach, CA, USA

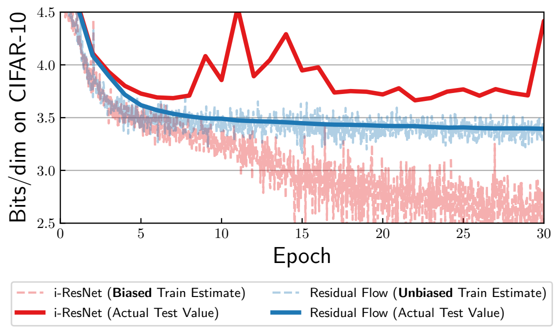

Figure 1: Biased estimator fails to optimize objective when using larger Lipschitz constants than the original i-ResNet.

are invertible if g is contractive, i.e. with Lipschitz constant strictly less than one. This can be easily satisfied by using spectral normalization (Miyato et al., 2018; Gouk et al., 2018), in contrast to the majority of approaches where structural dependencies are enforced on $f(x)$ to satisfy invertibility (Dinh et al., 2014; Rezende and Mohamed, 2015).

Applying (2) to (1), the following identity was shown

$$\log p(x) = \log p(f(x)) + \operatorname{tr}\left(\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} [J_g(x)]^k \right) \tag{3}$$

where $J_g(x) = \frac{dg(x)}{dx}$. Behrmann et al. (2019) used a finite truncation to approximate (3). However, this finite truncation can introduce significant bias into the optimization objective (see Figure 1), especially when the Lipschitz constant of g is closer to one, thus requiring a careful balance between bias and expressiveness.

In this work, we first introduce an unbiased estimator for (3), resulting in a "residual flow" model which allows principled training using maximum likelihood. We also show that a memory-efficient formulation can be constructed to directly estimate the gradients of (3). Furthermore, we describe a generalization of spectral norm to mixed norms for enforcing Lipschitz constraints where the order of the norms can be trained simultaneously with the main objective.

2. Residual Flows

2.1. Maximum Likelihood with Unbiased Log-density

To perform maximum likelihood with stochastic gradient descent, it is sufficient to have an unbiased estimator for

$$\max_\theta \mathbb{E}_{x \sim p_{\text{data}}(x)} \left[\log p_\theta(x)\right], \tag{4}$$

where p_{data} is the unknown data distribution and p_θ is the model distribution. Due to linearity of expectation, we focus on the infinite series in (3), written as

$$\sum_{k=1}^{\infty} \Delta_k, \quad \Delta_k = \frac{(-1)^{k+1}}{k} \operatorname{tr}([J_g(x)]^k). \tag{5}$$

McLeish (2011) and Rhee and Glynn (2012) showed that if $\sum_{k=1}^{\infty} |\Delta_k| < \infty$, then an unbiased estimator of this infinite series can be constructed via

$$\sum_{k=1}^{\infty} \Delta_k = \mathbb{E}_{n \sim p(N)} \left[\sum_{k=1}^{n} \frac{\Delta_k}{\mathbb{P}(N \geq k)} \right], \tag{6}$$

where N is a random variable with support over the positive integers.

We combine this estimator with the Skilling-Hutchinson trace estimator (Skilling, 1989; Hutchinson, 1990). Applying Fubini's theorem to swap the order of integration, we obtain an unbiased estimator for (3) as

$$\begin{aligned} \log p(x) &= \log p(f(x)) \\ &+ \mathbb{E}_{v,n} \left[\sum_{k=1}^{n} \frac{(-1)^{k+1}}{k} \frac{v^T [J_g(x)^k] v}{\mathbb{P}(N \geq k)} \right], \end{aligned} \tag{7}$$

where $n \sim p(N)$ and $v \sim \mathcal{N}(0, I)$. With this estimator, we can perform maximum likelihood training by backpropagating through (7) to obtain unbiased gradients.

2.2. Constant-Memory Backpropagation through Log-det Power Series

By differentiating directly through the power series (3), the gradient wrt. any parameter θ_i can be expressed as

$$\begin{aligned} &\frac{\partial}{\partial \theta} \log\det\left(I + J_g(x, \theta)\right) \\ &= \frac{\partial}{\partial \theta} \left(\sum_{k=1}^{\infty} (-1)^{k+1} \frac{\operatorname{tr}\left(J_g(x,\theta)^k\right)}{k} \right) \\ &= \operatorname{tr}\left(\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} \frac{\partial (J_g(x,\theta)^k)}{\partial \theta} \right). \end{aligned} \tag{8}$$

Unfortunately, this estimator requires each term to be stored in memory so the total memory cost is $\mathcal{O}(n \cdot m)$ where n is the number of computed terms and m is the number of residual blocks in the entire network. This is extremely memory-hungry during training, and a large sample of n can occasionally result in out of memory.

Instead, we can re-derive a power series specifically for estimating the gradients (see Appendix A) and get

$$\begin{aligned} &\frac{\partial}{\partial \theta} \log\det\left(I + J_g(x, \theta)\right) \\ &= \operatorname{tr}\left(\left[\sum_{k=0}^{\infty} (-1)^k J(x,\theta)^k \right] \frac{\partial (J_g(x,\theta))}{\partial \theta} \right). \end{aligned} \tag{9}$$

Written in this form, this power series does not need to be stored in memory since the gradient depends only on a single backward pass of g. Combining the unbiased estimator with this gradient power series reduces the memory required to $\mathcal{O}(m)$.

We can further reduce memory by partially performing the backpropagation during the forward pass. By taking advantage of $\log\det(I + J_g(x,\theta))$ being a scalar quantity, the gradient from the objective $\mathcal{L}$ is

$$\frac{\partial \mathcal{L}}{\partial \theta} = \underbrace{\frac{\partial \mathcal{L}}{\partial \log\det(I + J_g(x,\theta))}}_{\text{scalar}} \underbrace{\frac{\partial \log\det(I + J_g(x,\theta))}{\partial \theta}}_{\text{vector}}. \tag{10}$$

We compute $\frac{\partial \log\det(I + J_g(x,\theta))}{\partial \theta}$ along with the forward pass, release the memory used for computing this gradient, then simply multiply by $\frac{\partial \mathcal{L}}{\partial \log\det(I + J_g(x,\theta))}$ later after the main backprop. This is done per residual block, and reduces memory by another factor of m to $\mathcal{O}(1)$ with negligible overhead. Note that while these two tricks completely remove the memory cost from backpropagating through the $\log\det$ terms, computing the gradient from $\log p(f(x))$ still requires the same amount of memory as a single evaluation of the residual network.

2.3. Generalized Spectral Normalization

Spectral normalization (Miyato et al., 2018) is a popular algorithm that can approximately bound the Lipschitz constant of a neural network g. It effectively bounds the induced 2-norm of the Jacobian by bounding the norm of each weight matrix. However, the choice of norm is arbitrary, and we describe how to generalize to mixed norms. This allows the normalized weights to contain larger values, providing more expressive transformations while maintaining a less than unity Lipschitz constant and the invertibility of residual blocks (2).

If $g(x)$ is a neural network with pre-activation defined recursively using $z_l = W_l h_{l-1} + b_l$ and $h_l = \phi(z_l)$, with

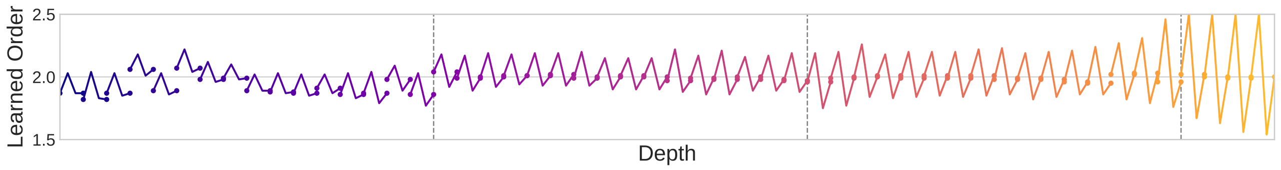

Figure 2: **Learned norm orders for each layer.** Each residual block is visualized as a single line. The input and two hidden states for each block use different normed spaces. We observe multiple trends: (i) the norms for the first hidden states are consistently higher than the input, and lower for the second. (ii) The orders for the hidden states drift farther away from 2 as depth increases. (iii) The ending order of one block and the starting order of the next are generally consistent and close to 2.

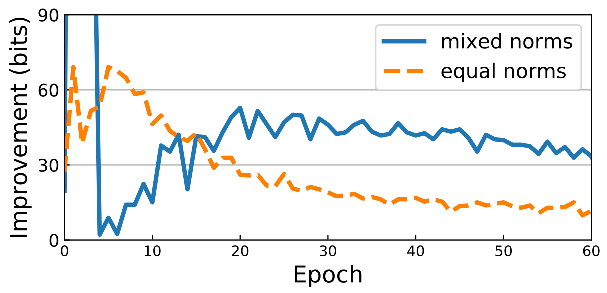

Figure 3: Improvement gained from using generalized spectral normalization. Learning with mixed norms is more effective than learning with equal norms, but both offer a small gain over spectral normalization with fixed 2-norm. Learned orders for mixed norms are visualized in Figure 2.

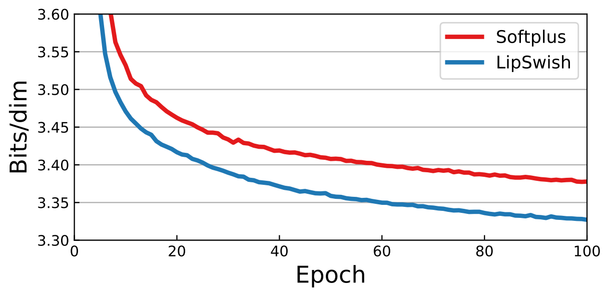

Figure 4: LipSwish converges much faster than activations with vanishing second derivatives such as softplus.

$x = z_0, g(x) = z_L$, then

$$\begin{aligned} ||J_g|| &= ||W_L \cdots W_2 \phi'(z_1) W_1 \phi'(z_0)|| \\ &\leq ||W_L|| \ \cdots ||W_2|| \ ||\phi'(z_1)|| \ ||W_1|| \ ||\phi'(z_0)||, \end{aligned} \tag{11}$$

where J_g is the Jacobian matrix of g, and $\phi'(z)$ are diagonal matrices containing the first derivatives of the activation functions. Note that the sub-multiplicative property in (11) holds for any induced matrix norm (Horn and Johnson, 2012). The inequality from sub-multiplicativity may even be tightened by using different normed vector spaces.

We generalize spectral normalization in two manners:

1. Using arbitrary p-norms to induce different Lipschitz constraints $||J_g||_p$.
2. Using different norms on the hidden states via mixed norms $||W||_{p\to q}$.

We use a a more general form of power iteration (Johnston, 2016), which becomes the power iteration for spectral normalization when $p = q = 2$. Assuming we choose activation functions where $\phi'(z) \leq 1$, then

$$||J_g||_{p_0} \leq ||W_L||_{p_{L-1}\to p_0} \cdots ||W_2||_{p_1\to p_2} ||W_1||_{p_0\to p_1}, \tag{12}$$

which implies that we can use any $p_0, p_1, \ldots, p_{L-1} \geq 1$ and g would still be a contractive mapping. As long as g maps back to the original normed vector space, Banach fixed point theorem holds and the residual transformation $f(x) = x + g(x)$ is invertible.

The Lipschitz constraint is enforced by scaling each weight matrix by the norm, $W' = \frac{W}{||W||}$. As such, we would like to induce a mixed norm such that $||W||_{p\to q}$ is as small as possible. Therefore during training, we optimize the p's and q's to minimize the norms $||W||_{p\to q}$.

2.4. On the Choice of Activation Functions

From (11), we see that the $\log\det$ terms contain the first-order derivatives of the activation functions $\phi'(z)$, while the gradients wrt. parameters depend multiplicatively on $\phi''(z)$. We thus require two properties from our activation functions:

1. The first derivatives must be bounded $|\phi'(z)| \leq 1$.
2. Since the Lipschitz constant is high (ie. more expressive) when $|\phi'(z)|$ is close to one, the second derivatives should not vanish when $|\phi'(z)|$ is close to one.

While many activation functions satisfy condition 1, most do not satisfy condition 2. We argue that the ELU activation used by Behrmann et al. (2019) is suboptimal, because if $\text{ELU}'(z) = 1$ then $\text{ELU}''(z) = 0$, so the gradients received by the parameters from the $\log\det$ terms are sparse. A similar case can be made for softplus since $|\phi'(z)| \to 1 \implies \phi''(z) \to 0$ while squashing functions such as tanh or sigmoid will encourage activations to be zero, requiring large weight matrices to be useful.

We find that good activation functions satisfying condition 2 are *smooth and non-monotonic* functions, such as Swish (Ramachandran et al., 2017). Swish by default does not satisfy condition 1, but it is easy to see that if we define

$$\text{LipSwish}(x) = \text{Swish}(x)/1.1 = x \cdot \sigma(\beta x)/1.1, \quad (13)$$

then $\frac{\partial}{\partial x}\text{LipSwish}(x) \leq 1$ for all values of β. LipSwish is a simple modification to Swish that exhibits a less than unity Lipschitz property. In our experiments, we parameterize β to be strictly positive by passing it through softplus.

3. Related Work

Power Series Estimation. The unbiased estimator we use was formulated by McLeish (2011) and Rhee and Glynn (2012), where the estimator was used in the context of solving stochastic differential equations.

Some recent works use Chebyshev polynomials to estimate the spectral functions of symmetric matrices Han et al. (2018); Adams et al. (2018); Ramesh and LeCun (2018); Boutsidis et al. (2008). These works estimate quantities that are similar to those presented in this work, but a key difference is that the Jacobian in our power series is in general not symmetric. (Han et al., 2018) proposed an unbiased estimator with a different assumption, and seems to not have been aware of the works by McLeish (2011) and Rhee and Glynn (2012).

Memory-efficient Backpropagation. The issue of computing gradients in a memory-efficient manner was explored by Gomez et al. (2017) and Chang et al. (2018) for residual networks with an architecture devised by (Dinh et al., 2014), and explored by Chen et al. (2018) for a continuous analogue of residual networks. We note that these are orthogonal to our contributions, as these works focus on the path-wise gradients from the output of the network, whereas we focus on the gradients from the $\log\det$ terms in the change of variables equation.

Lipschitz Activations Functions. Anil et al. (2018) proposed a sorting-based activation function that has an exactly unit Lipschitz property. Unfortunately, we cannot use it in our work as it is both non-smooth and has zero second-order derivatives.

Model	MNIST ↓	CIFAR10 ↓
Real NVP (Dinh et al., 2017)	1.06	3.49
Glow (Kingma and Dhariwal, 2018)	1.05	3.35
FFJORD (Grathwohl et al., 2019)	0.99	3.40
i-ResNet (Behrmann et al., 2019)	1.05	3.45
Flow++[1] (Ho et al., 2019)	–	**3.29**
Residual Flow (Ours)	**0.97**	3.30

Table 1: Results [bits/dim] on standard benchmark datasets.

4. Experiments

4.1. MNIST & CIFAR10

We use a similar architecture to Behrmann et al. (2019), except without the immediate downsampling at the image pixel-level. Removing this increases the amount of memory required as there are more spatial dimensions at every layer, but increases the overall performance. We are able to remove this first downsampling layer due to the memory-efficient backpropagation and unbiased estimator. Unlike prior works that use multiple GPUs, large batch sizes, and a few hundred epochs, Residual Flow models can be trained with the standard batch size of 64 and converges in 100-200 epochs for MNIST and CIFAR10. In contrast, Glow was trained for 1800 epochs (Kingma and Dhariwal, 2018) and Flow++ reported to not have fully converged after 400 epochs (Ho et al., 2019).

Table 1 reports the bits per dimension on standard benchmark datasets MNIST and CIFAR10. We achieve competitive performance to state-of-the-art flow-based models on both datasets. For evaluation, we compute 20 terms of the power series (3) and use the unbiased estimator (7) to estimate the remaining terms. This reduces the standard deviation of the unbiased estimator to a negligible level.

Norm-Learning. Table 1 shows preliminary results where the orders for generalized spectral normalization are parameterized to be between (1.5, 2.5). We find that some orders have saturated near the endpoints (shown in Figure 2), suggesting that a larger range can be used. Besides being a tool to improve performance, generalized spectral normalization may also be a useful tool for analyzing the underlying assumptions about the model, e.g. which layers are learning to be more expressive than others, indicated by norms where $p < q$.

[1]While the best bits/dim reported by Ho et al. (2019) is 3.09, we compare against their best *uniform dequantization* model.

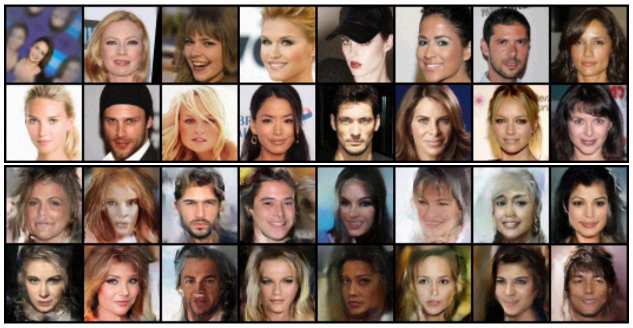

Figure 5: **Qualitative samples.** Real (top) and random samples (bottom) from a model trained on 5bit 64×64 CelebA. Most visually appealing samples were picked out of 5 random batches.

4.2. Ablation Experiments

We report some ablation experiments on the same architecture used by Behrmann et al. (2019) as our preliminary experiments were performed in that setting. Table 2 shows the improvements resulting from using an unbiased estimator and changing the activation function from ELU to LipSwish. The biased estimator uses 5 terms for CIFAR10 and 10 terms for MNIST. Even in this setting where the Lipschitz constant and bias are low, we find that we can gain non-trivial improvement from using an unbiased estimator, and using Swish results in some further gains.

Training Setting	MNIST $\downarrow$	CIFAR10 $\downarrow$
Biased Estimator + ELU	1.05	3.45
Unbiased Estimator + ELU	1.00	3.40
Unbiased Estimator + LipSwish	0.97	3.39

Table 2: Ablation results from preliminary experiments.

5. Conclusion

We have shown that invertible residual networks can be turned into powerful generative models. The resulting unbiased flow-based model, termed Residual Flow, achieves competitive performance to other recent works. In future work, we hope to better understand the effects of different norms for generalized spectral normalization, and apply Residual Flows to more difficult tasks.

References

Ryan P Adams, Jeffrey Pennington, Matthew J Johnson, Jamie Smith, Yaniv Ovadia, Brian Patton, and James Saunderson. Estimating the spectral density of large implicit matrices. *arXiv preprint arXiv:1802.03451*, 2018.

Cem Anil, James Lucas, and Roger Grosse. Sorting out lipschitz function approximation. *arXiv preprint arXiv:1811.05381*, 2018.

Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. *arXiv preprint arXiv:1811.00995*, 2019.

Christos Boutsidis, Michael W Mahoney, and Petros Drineas. Unsupervised feature selection for principal components analysis. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 61–69. ACM, 2008.

Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *AAAI Conference on Artificial Intelligence*, 2018.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018.

Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *International Conference on Learning Representations*, 2017.

Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. In *Advances in neural information processing systems*, pages 2214–2224, 2017.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael Cree. Regularisation of neural networks by enforcing lipschitz continuity. *arXiv preprint arXiv:1804.04368*, 2018.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *International Conference on Learning Representations*, 2019.

Insu Han, Haim Avron, and Jinwoo Shin. Stochastic chebyshev gradient descent for spectral optimization. In *Advances in Neural Information Processing Systems 31*, pages 7386–7396. 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. *arXiv preprint arXiv:1902.00275*, 2019.

Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge University Press, 2012.

Michael F Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 19(2):433–450, 1990.

Nathaniel Johnston. QETLAB: A MATLAB toolbox for quantum entanglement, version 0.9. http://qetlab.com, January 2016.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *International Conference on Learning Representations*, 2014.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224, 2018.

Don McLeish. A general method for debiasing a monte carlo estimator. *Monte Carlo Methods and Applications*, 17(4):301–315, 2011.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Aditya Ramesh and Yann LeCun. Backpropagation for implicit spectral densities. *arXiv preprint arXiv:1806.00499*, abs/1806.00499, 2018.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1530–1538, 2015.

Chang-han Rhee and Peter W Glynn. A new approach to unbiased estimation for sde's. In *Proceedings of the Winter Simulation Conference*, page 17. Winter Simulation Conference, 2012.

John Skilling. The eigenvalues of mega-dimensional matrices. In *Maximum Entropy and Bayesian Methods*, pages 455–466. Springer, 1989.

A. Memory-Efficient Gradient Estimation of Log-Determinant

Derivation of gradient estimator via differentiating power series:

$$\begin{aligned}\frac{\partial}{\partial \theta_i} \log \det \big(I + J_g(x,\theta)\big) &= \frac{\partial}{\partial \theta_i}\left(\sum_{k=1}^{\infty}(-1)^{k+1}\frac{\operatorname{tr}(J_g(x,\theta)^k)}{k}\right)\\ &= \operatorname{tr}\left(\sum_{k=1}^{\infty}\frac{(-1)^{k+1}}{k}\frac{\partial (J_g(x,\theta)^k)}{\partial \theta_i}\right)\end{aligned}$$

Derivation of memory-efficient gradient estimator:

$$\begin{aligned}&\frac{\partial}{\partial \theta_i} \log \det \big(I + J_g(x,\theta)\big)\\ &= \frac{1}{\det(I+J_g(x,\theta))}\left[\frac{\partial}{\partial \theta_i}\det\big(I+J_g(x,\theta)\big)\right] && (14)\\ &= \frac{1}{\det(I+J_g(x,\theta))}\left[\det(I+J_g(x,\theta))\operatorname{tr}\left((I+J(x,\theta))^{-1}\frac{\partial(I+J_g(x,\theta))}{\partial\theta_i}\right)\right] && (15)\\ &= \operatorname{tr}\left((I+J(x,\theta))^{-1}\frac{\partial(I+J_g(x,\theta))}{\partial\theta_i}\right)\\ &= \operatorname{tr}\left((I+J(x,\theta))^{-1}\frac{\partial(J_g(x,\theta))}{\partial\theta_i}\right)\\ &= \operatorname{tr}\left(\left[\sum_{k=0}^{\infty}(-1)^k J(x,\theta)^k\right]\frac{\partial(J_g(x,\theta))}{\partial\theta_i}\right) && (16)\end{aligned}$$

Note, that (14) follows from the chain rule of differentiation, for the derivative of the determinant in (15) see Matrix cookbook (eq 46) and (16) follows from the properties of a Neumann-Series which converges since $\|J\| < 1$.

Hence, if we are able to compute the trace exactly, both approaches will return the same values for a given truncation n. However, when estimating the trace via the Hutchinson trace estimator the estimation is not equal in general:

$$v^T\left(\sum_{k=1}^{\infty}\frac{(-1)^{k+1}}{k}\frac{\partial(J_g(x,\theta)^k)}{\partial\theta_i}\right)v \neq v^T\left(\left[\sum_{k=0}^{\infty}(-1)^k J_g^k(x,\theta)\right]\frac{\partial(J_g(x,\theta))}{\partial\theta_i}\right)v.$$

Another difference between both approaches is their memory consumption of the corresponding computational graph. The summation $\sum_{k=0}^{\infty}(-1)^k J_g^k(x,\theta)$ is not being tracked for the gradient, which allows to compute the gradient with constant memory (constant with respect to the truncation n).