Woodbury Transformations for Deep Generative Flows

You Lu [1] **Bert Huang** [1]

Abstract

Normalizing flows are deep generative models that allow efficient likelihood calculation and sampling. The core requirement for this advantage is that they are constructed using functions that can be efficiently inverted and for which the determinant of the function's Jacobian can be efficiently computed. Researchers have introduced various such flow operations, but few of these allow rich interactions among variables without incurring significant computational costs. In this paper, we introduce *Woodbury transformations*, which achieve efficient invertibility via the Woodbury matrix identity and efficient determinant calculation via Sylvester's determinant identity. In contrast with other operations used in state-of-the-art normalizing flows, Woodbury transformations enable (1) high-dimensional interactions, (2) efficient sampling, and (3) efficient likelihood evaluation. Other similar operations, such as 1x1 convolutions, emerging convolutions, or periodic convolutions allow at most two of these three advantages. In our experiments on multiple image datasets, we find that Woodbury transformations allow learning of higher-likelihood models than other flow architectures while still enjoying their efficiency advantages.

1. Introduction

Deep generative models are powerful tools for modeling complex distributions and have been applied to many tasks such as synthetic data generation (Oord et al., 2016a; Yu et al., 2017), domain adaption (Zhu et al., 2017), and structured prediction (Sohn et al., 2015). Examples of these models include autoregressive models (Graves, 2013; Oord et al., 2016b), variational autoencoders (Kingma & Welling, 2013; Rezende & Mohamed, 2015), generative adversarial networks (Goodfellow et al., 2014), and normalizing flows (Dinh et al., 2014; 2016; Kingma & Dhariwal, 2018). Normalizing flows are special because of two advantages: They allow efficient and exact computation of log-likelihood and sampling.

Flow-based models are composed of a series of invertible functions, which are specifically designed so that their inverse and determinant of the Jacobian are easy to compute. However, to preserve this computational efficiency, these functions usually cannot sufficiently encode dependencies among dimensions of a variable. For example, affine coupling layers (Dinh et al., 2014) split a variable to two parts and require the second part to only depend on the first. But they ignore the dependencies among dimensions in the second part.

To address this problem, Dinh et al. (2014; 2016) introduced a fixed permutation operation that reverses the ordering of the channels of pixel variables. Kingma & Dhariwal (2018) introduced a 1×1 convolution, which are a generalized permutation layer, that uses a weight matrix to model the interactions among dimensions along the channel axis. Their experiments demonstrate the importance of capturing dependencies among dimensions. Relatedly, Hoogeboom et al. (2019a) proposed emerging convolution operations, and Hoogeboom et al. (2019a) and Finz et al. (2019) proposed periodic convolution. These two convolution layers have $d \times d$ kernels that can model dependencies along the spatial axes in addition to the channel axis. However, the increase in representational power comes at a cost: These convolution operations do not scale well to high-dimensional variables. The emerging convolution is a combination of two autoregressive convolutions (Germain et al., 2015; Kingma et al., 2016), whose inverse is not parallelizable. To compute the inverse or determinant of the Jacobian, the periodic convolution requires transforming the input and the convolution kernel to Fourier space. This transformation is computationally costly.

In this paper, we develop *Woodbury transformations* for generative flows. Our method is also a generalized permutation layer and uses spatial and channel transformations to model dependencies among dimensions along spatial and channel axes. We use the Woodbury matrix identity (Woodbury, 1950) and Sylvester's determinant identity (Sylvester, 1851)

[1]Department of Computer Science, Virginia Tech, Blacksburg, VA. Correspondence to: You Lu <you.lu@vt.edu>.

Second workshop on *Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models* (ICML 2020), Virtual Conference

to compute the inverse and Jacobian determinant, respectively, so that both the training and sampling time complexities are linear to the input variable's size. We also develop a memory-efficient variant of the Woodbury transformation, which has the same advantage as the full transformation but uses significantly reduced memory when the variable is high-dimensional. In our experiments, we found that Woodbury transformations enable model quality comparable to many state-of-the-art flow architectures while maintaining significant efficiency advantages.

2. Deep Generative Flows

In this section, we briefly introduce the deep generative flows. More background knowledge can be found in the appendix.

A normalizing flow (Rezende & Mohamed, 2015) is composed of a series of invertible functions $\mathbf{f} = \mathbf{f}_1 \circ \mathbf{f}_2 \circ ... \circ \mathbf{f}_K$, which transform $\mathbf{x}$ to a latent code $\mathbf{z}$ drawn from a simple distribution. Therefore, with the *change of variables* formula, we can rewrite the log-likelihood $\log p_\theta(\mathbf{x})$ to be

$$\log p_\theta(\mathbf{x}) = \log p_Z(\mathbf{z}) + \sum_{i=1}^{K} \log \left| \det \left(\frac{\partial \mathbf{f}_i}{\partial \mathbf{r}_{i-1}} \right) \right|, \quad (1)$$

where $\mathbf{r}_i = \mathbf{f}_i(\mathbf{r}_{i-1})$, $\mathbf{r}_0 = \mathbf{x}$, and $\mathbf{r}_K = \mathbf{z}$.

Flow-based generative models (Dinh et al., 2014; 2016; Kingma & Dhariwal, 2018) are developed on the theory of normalizing flows. Each transformation function used in the models is a specifically designed neural network that has a tractable Jacobian determinant and inverse. We can sample from a trained flow $\mathbf{f}$ by computing $\mathbf{z} \sim p_Z(\mathbf{z})$, $\mathbf{x} = \mathbf{f}^{-1}(\mathbf{z})$.

There have been many operations, i.e., layers, proposed in recent years for generative flows. In our work, we will use the framework of Glow (Kingma & Dhariwal, 2018) to construct generative flows. Each Glow step is composed of a actnorm layer (Kingma & Dhariwal, 2018), an invertible convolution layer (Kingma & Dhariwal, 2018; Hoogeboom et al., 2019a; Finz et al., 2019), and an affine coupling layer (Dinh et al., 2014; 2016). The flow layers are connected together using the multi-scale architectures (Dinh et al., 2016), with *split layers* to factor out variables and *squeeze layers* to shuffle dimensions, resulting in an architecture with K flow steps and L levels. Due to the space limit, more background knowledge and other related work will be discussed in the appendix.

3. Woodbury Transformations

In this section, we introduce Woodbury transformations as an efficient means to model high-dimensional correlations.

3.1. Channel and Spatial Transformations

Suppose we reshape the input $\mathbf{x}$ to be a $c \times n$ matrix, where $n = hw$. Then the 1×1 convolution can be reinterpreted as a matrix transformation

$$\mathbf{y} = \mathbf{W}^{(c)}\mathbf{x}, \quad (2)$$

where $\mathbf{y}$ is also a $c \times n$ matrix, and $\mathbf{W}^{(c)}$ is a $c \times c$ matrix. For consistency, we will call this a channel transformation. For each column $\mathbf{x}_{:,i}$, the correlations among channels are modeled by $\mathbf{W}^{(c)}$. However, the correlation between any two rows $\mathbf{x}_{:,i}$ and $\mathbf{x}_{:,j}$ is not captured. Inspired by Eq. 2, we use a spatial transformation to model interactions among dimensions along the spatial axis

$$\mathbf{y} = \mathbf{x}\mathbf{W}^{(s)}, \quad (3)$$

where $\mathbf{W}^{(s)}$ is an $n \times n$ matrix that models the correlations of each row $\mathbf{x}_{i,:}$. Combining Equation 2 and Equation 3, we have

$$\begin{aligned} \mathbf{x}_c &= \mathbf{W}^{(c)}\mathbf{x}, \\ \mathbf{y} &= \mathbf{x}_c\mathbf{W}^{(s)}. \end{aligned} \quad (4)$$

For each dimension of output $\mathbf{y}_{i,j}$, we have

$$\mathbf{y}_{i,j} = \sum_{v=1}^{c} \left(\sum_{u=1}^{n} \mathbf{W}^{(c)}_{i,u} \cdot \mathbf{x}_{u,v} \right) \cdot \mathbf{W}^{(s)}_{v,j}.$$

Therefore, the spatial and channel transformations together can model the correlation between any pair of dimensions. However, in this preliminary form, directly using Eq. 4 is inefficient for large c or n. First, we would have to store two large matrices $\mathbf{W}^c$ and $\mathbf{W}^s$, so the space cost is $\mathcal{O}(c^2 + n^2)$. Second, the computational cost of Eq. 4 is $\mathcal{O}(c^2 n + n^2 c)$—quadratic in the input size. Third, the computational cost of the Jacobian determinant is $\mathcal{O}(c^3 + n^3)$, which is far too expensive in practice.

3.2. Woodbury Transformations

We solve the three scalability problems by using a low-rank factorization. Specifically, we define

$$\begin{aligned} \mathbf{W}^{(c)} &= \mathbf{I}^{(c)} + \mathbf{U}^{(c)}\mathbf{V}^{(c)}, \\ \mathbf{W}^{(s)} &= \mathbf{I}^{(s)} + \mathbf{U}^{(s)}\mathbf{V}^{(s)}, \end{aligned}$$

where $\mathbf{I}^{(c)}$ and $\mathbf{I}^{(s)}$ are c- and n-dimensional identity matrices, respectively. The matrices $\mathbf{U}^c$, $\mathbf{V}^c$, $\mathbf{U}^s$, and $\mathbf{V}^s$ are of size $c \times d_c$, $d_c \times c$, $n \times d_s$, and $d_c \times n$, respectively, where d_c and d_s are constant latent dimensions of these four matrices. Therefore, we can rewrite Equation 4 as

$$\begin{aligned} \mathbf{x}_c &= (\mathbf{I}^{(c)} + \mathbf{U}^{(c)}\mathbf{V}^{(c)})\mathbf{x}, \\ \mathbf{y} &= \mathbf{x}_c(\mathbf{I}^{(s)} + \mathbf{U}^{(s)}\mathbf{V}^{(s)}). \end{aligned} \quad (5)$$

We call Eq. 5 the Woodbury transformation because the Woodbury matrix identity (Woodbury, 1950) and Sylvester's determinant identity (Sylvester, 1851) allow efficient computation of its inverse and Jacobian determinant.

Woodbury matrix identity.[1] Let $\mathbf{I}^{(n)}$ and $\mathbf{I}^{(k)}$ be n- and k-dimensional identity matrices, respectively. Let $\mathbf{U}$ and $\mathbf{V}$ be $n \times k$ and $k \times n$ matrices, respectively. If $\mathbf{I}^{(k)} + \mathbf{VU}$ is invertible, then $(\mathbf{I}^{(n)} + \mathbf{UV})^{-1} = \mathbf{I}^{(n)} - \mathbf{U}(\mathbf{I}^{k} + \mathbf{VU})^{-1}\mathbf{V}$.

Sylvester's determinant identity. Let $\mathbf{I}^{(n)}$ and $\mathbf{I}^{(k)}$ be n- and k-dimensional identity matrices, respectively. Let $\mathbf{U}$ and $\mathbf{V}$ be $n \times k$ and $k \times n$ matrices, respectively. Then, $\det(\mathbf{I}^{(n)} + \mathbf{UV}) = \det(\mathbf{I}^{(k)} + \mathbf{VU})$.

Based on these two identities, we can efficiently compute the inverse and Jacobian determinant

$$\begin{aligned} \mathbf{x}_c &= \mathbf{y}(\mathbf{I}^{(s)} - \mathbf{U}^{(s)}(\mathbf{I}^{(d_s)} + \mathbf{V}^{(s)}\mathbf{U}^{(s)})^{-1}\mathbf{V}^{(s)}), \\ \mathbf{x} &= (\mathbf{I}^{(c)} - \mathbf{U}^{(c)}(\mathbf{I}^{(d_c)} + \mathbf{V}^{(c)}\mathbf{U}^{(c)})^{-1}\mathbf{V}^{(c)})\mathbf{x}_c, \end{aligned} \tag{6}$$

and

$$\begin{aligned} \log\left|\det\left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)\right| &= n\log\left|\det(\mathbf{I}^{(d_c)} + \mathbf{V}^{(c)}\mathbf{U}^{(c)})\right| \\ &+ c\log\left|\det(\mathbf{I}^{(d_s)} + \mathbf{V}^{(s)}\mathbf{U}^{(s)})\right| \end{aligned} \tag{7}$$

where $\mathbf{I}^{(d_c)}$ and $\mathbf{I}^{(d_s)}$ are d_c- and d_s-dimensional identity matrices, respectively.

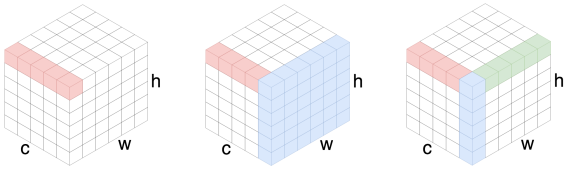
A Woodbury transformation is also a generalized permutation layer. We can directly replace an invertible convolution with a Woodbury transformation. In contrast with 1×1 convolutions, Woodbury transformations are able to model correlations along both channel and spatial axes. We illustrate this in Figure 1. The space complexity of Woodbury transformations is $\mathcal{O}(d(c+n))$, where d is the size of latent dimension. The computational complexity of training is $\mathcal{O}(d^2(c+n) + d^3)$, and the complexity of sampling is $\mathcal{O}(dcn + d^2(n+c) + d^3)$. Detailed analysis is in the appendix.

We do not restrict $\mathbf{U}$ and $\mathbf{V}$ to force $\mathbf{W}$ to be invertible. Based on analysis by Hoogeboom et al. (2019a), the training maximizes the log-likelihood, which implicitly pushes $\det(\mathbf{I} + \mathbf{VU})$ away from 0. Therefore, it is not necessary to explicitly force invertibility. In our experiments, the Woodbury transformations are as robust as other invertible convolution layers.

3.3. Memory-Efficient Variant

In Eq. 5, one potential challenge arises from the sizes of $\mathbf{U}^{(s)}$ and $\mathbf{V}^{((s))}$, which are linear in n. The challenge is that n may be large in some practical problems, e.g., high-resolution images. We develop a memory-efficient variant of Woodbury transformations, i.e., ME-Woodbury, to solve this problem. The ME version can effectively reduce space complexity from $\mathcal{O}(d(c+hw))$ to $\mathcal{O}(d(c+h+w))$.

[1] A more general version replaces $\mathbf{I}^{(n)}$ and $\mathbf{I}^{(k)}$ with arbitrary invertible $n \times n$ and $k \times k$ matrices. But this simplified version is sufficient for our tasks.



(a) 1×1 convolution (b) Woodbury (c) ME-Woodbury

Figure 1. Comparison of three transformations. The 1×1 convolution only operates along the channel axis. The Woodbury transformation operates along both the channel and spatial axes, modeling the dependencies of one channel directly via one transformation. The ME-Woodbury transformation operates along three axes. It uses two transformations to model spatial dependencies.

The difference between ME-Woodbury transformations and Woodbury transformations is that the ME form cannot directly model spatial correlations. As shown in Figure 1c, it uses two transformations, for height and width, together to model the spatial correlations. Therefore, for a specific channel k, when two dimensions $\mathbf{x}_{k,i,j}$ and $\mathbf{x}_{k,u,v}$ are in two different heights, and widths, their interaction will be modeled indirectly. In our experiments, we found that this limitation only slightly impacts ME-Woodbury's performance. More details on ME-Woodbury transformations are in the appendix.

4. Experiments

In this section, we compare the performance of Woodbury transformations against other modern flow architectures, measuring bit per-dimension ($\log_2$-likelihood). More experiments are in the appendix.

We train with the CIFAR-10 (Krizhevsky et al., 2009) and ImageNet (Russakovsky et al., 2015) datasets. We compare with three generalized permutation methods—1×1 convolution, emerging convolution, and periodic convolution—and two coupling layers—neural spline coupling (Durkan et al., 2019) and MaCow (Ma et al., 2019). We use Glow (Kingma & Dhariwal, 2018) as the basic flow architecture. For each method, we replace the corresponding layer. For example, to construct a flow with Woodbury transformations, we replace the 1×1 convolution with a Woodbury transformation, i.e., Eq. 5. For all generalized permutation methods, we use affine coupling. For each of the coupling layer baselines, we substitute it for the affine coupling. Ma et al. (2019) used steps containing a MaCow unit, i.e., 4 autoregressive convolution coupling layers, and a full Glow step. For fair comparison, we directly use the MaCow unit to replace the affine coupling. We tune the parameters of neural spline

coupling and MaCow so that their sizes are close to affine coupling. More details are in the appendix.

As discussed by Hoogeboom et al. (2019a), the models used in (Kingma & Dhariwal, 2018) are over-parameterized, and replacing the 1×1 convolution will not improve model performance. Moreover, training these models requires a large amount of computing resources. We follow Hoogeboom et al. (2019a) and test the performance of small models. For 32×32 images, we set the number of levels to $L = 3$ and the number of steps per-level to $K = 8$. For 64×64 images, we use $L = 4$ and $K = 16$.

Table 1. Quantitative measure of model fit (bits per-dimension).

	CIFAR-10 32x32	ImageNet 32x32	ImageNet 64x64
1×1 convolution	3.51	4.32	3.94
Emerging	3.48	4.26	3.91
Periodic	3.49	4.28	3.92
Neural spline	3.50	4.24	3.95
MaCow	3.48	4.34	4.15
ME-Woodbury	3.48	4.22	3.91
Woodbury	**3.47**	**4.20**	**3.87**

Table 2. Model sizes (number of parameters).

	32x32 images	64x64 images
1×1 convolution	11.02M	37.04M
Emerging	11.43M	40.37M
Periodic	11.21M	38.61M
Neural spline	**10.91M**	38.31M
MaCow	11.43M	37.83M
ME-Woodbury	11.02M	**36.98M**
Woodbury	11.10M	37.60M

The test-set likelihoods are listed in Table 1. Our scores are worse than those reported by Kingma & Dhariwal (2018); Hoogeboom et al. (2019a) because we use smaller models and we use a different training method. Kingma & Dhariwal (2018) trained their models with very large mini-batches, requiring parallelizing the training on multiple GPUs. In our experiments, we train each model on a single GPU, so we use small mini-batches. Based on the scores, 1×1 convolutions perform the worst. Emerging convolutions and periodic convolutions score better than the 1×1 convolutions, since they are more flexible and can model the dependencies along the spatial axes. Neural spline coupling works well on 32×32 images, but do slightly worse than 1×1 convolution on 64×64 images. We believe that, for larger images, since the variable dependencies become more complicated, we need more bins to draw better splines. MaCow does not work well on ImageNet, possibly because it needs to be combined with affine coupling layers, as specially designed in (Ma et al., 2019). This trend demonstrates the importance of permutation layers. They can model the interactions among dimensions and shuffle them, which coupling layers cannot do. Without a good permutation layer, a better coupling layer still cannot always improve the performance. The Woodbury transformation models perform the best, likely because they can model the interactions between the target dimension and all other dimensions, while the invertible convolutions only model the interactions between target dimension its neighbors. ME-Woodbury performs only slightly worse than the full version, showing that its restrictions provide a useful tradeoff between model quality and efficiency.

We list model sizes in Table 2. Despite modeling rich interactions, Woodbury transformations are not the largest. With 32×32 images, ME-Woodbury and 1×1 convolution are the same size. When the image size is 64×64, ME-Woodbury is the smallest. This is because we use the multi-scale architecture, to combine layers. The squeeze layer doubles the input variable's channels at each level, so larger L suggests larger c. The space complexities of invertible convolutions are $\mathcal{O}(c^2)$, while the space complexity of ME-Woodbury is linear to c. When c is large, the weight matrices of invertible convolutions are larger than the weight matrices of ME-Woodbury.

5. Conclusion

In this paper, we develop Woodbury transformations, which use the Woodbury matrix identity to compute the inverse transformations and Sylvester's determinant identity to compute Jacobian determinants. Our method has the same advantages as invertible $d \times d$ convolutions that can capture correlations among all dimensions. In contrast to the invertible $d \times d$ convolutions, our method is parallelizable and the computational complexity of our methods are linear to the input size, so that it is still efficient in computation when the input is high-dimensional. We test our models on multiple image datasets and they outperform state-of-the-art methods.

Acknowledgments

We thank NVIDIA's GPU Grant Program and Amazon's AWS Cloud Credits for Research program for their support.

References

Behrmann, J., Grathwohl, W., Chen, R. T., Duvenaud, D., and Jacobsen, J.-H. Invertible residual networks. *arXiv preprint arXiv:1811.00995*, 2018.

Berg, R. v. d., Hasenclever, L., Tomczak, J. M., and Welling, M. Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649*, 2018.

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pp. 6571–6583, 2018.

Chen, T. Q., Behrmann, J., Duvenaud, D. K., and Jacobsen, J.-H. Residual flows for invertible generative modeling. In *Advances in Neural Information Processing Systems*, pp. 9913–9923, 2019.

Dinh, L., Krueger, D., and Bengio, Y. NICE: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real NVP. *arXiv preprint arXiv:1605.08803*, 2016.

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. Neural spline flows. *arXiv preprint arXiv:1906.04032*, 2019.

Finz, M., Izmailov, P., Maddox, W., Kirichenko, P., and Wilson, A. G. Invertible convolutional networks. In *ICML Workshop on Invertible Neural Networks and Normalizing Flows*, 2019.

Germain, M., Gregor, K., Murray, I., and Larochelle, H. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pp. 881–889, 2015.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.

Grathwohl, W., Chen, R. T., Betterncourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Graves, A. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. Flow++: Improving flow-based generative models with variational dequantization and architecture design. *arXiv preprint arXiv:1902.00275*, 2019.

Hoogeboom, E., Berg, R. v. d., and Welling, M. Emerging convolutions for generative normalizing flows. *arXiv preprint arXiv:1901.11137*, 2019a.

Hoogeboom, E., Peters, J. W., Berg, R. v. d., and Welling, M. Integer discrete flows and lossless compression. *arXiv preprint arXiv:1905.07376*, 2019b.

Huang, C.-W., Krueger, D., Lacoste, A., and Courville, A. Neural autoregressive flows. *arXiv preprint arXiv:1804.00779*, 2018.

Karami, M., Schuurmans, D., Sohl-Dickstein, J., Dinh, L., and Duckworth, D. Invertible convolutional flow. In *Advances in Neural Information Processing Systems*, pp. 5636–5646, 2019.

Karras, T., Aila, T., Laine, S., and Lehtinen, J. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Kingma, D. P. and Welling, M. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, pp. 4743–4751, 2016.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Lu, Y. and Huang, B. Woodbury transformations for deep generative flows. *arXiv preprint arXiv:2002.12229*, 2020.

Ma, X., Kong, X., Zhang, S., and Hovy, E. Macow: Masked convolutional generative flow. In *Advances in Neural Information Processing Systems*, pp. 5891–5900. 2019.

Müller, T., McWilliams, B., Rousselle, F., Gross, M., and Novák, J. Neural importance sampling. *ACM Transactions on Graphics (TOG)*, 38(5):1–19, 2019.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016a.

Oord, A. v. d., Kalchbrenner, N., and Kavukcuoglu, K. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016b.

Papamakarios, G., Pavlakou, T., and Murray, I. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pp. 2338–2347, 2017.

Rezende, D. J. and Mohamed, S. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3): 211–252, 2015.

Sohn, K., Lee, H., and Yan, X. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, pp. 3483–3491, 2015.

Song, Y., Meng, C., and Ermon, S. Mintnet: Building invertible neural networks with masked convolutions. In *Advances in Neural Information Processing Systems*, pp. 11002–11012, 2019.

Sylvester, J. J. On the relation between the minor determinants of linearly equivalent quadratic functions. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 1(4):295–305, 1851.

Tran, D., Vafa, K., Agrawal, K. K., Dinh, L., and Poole, B. Discrete flows: Invertible generative models of discrete data. *arXiv preprint arXiv:1905.10347*, 2019.

Woodbury, M. A. Inverting modified matrices. 1950.

Yu, F., Zhang, Y., Song, S., Seff, A., and Xiao, J. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

Yu, L., Zhang, W., Wang, J., and Yu, Y. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of International Conference on Computer Vision*, pp. 2223–2232, 2017.

Ziegler, Z. M. and Rush, A. M. Latent normalizing flows for discrete sequences. *arXiv preprint arXiv:1901.10548*, 2019.

A. More Background

In this section, we introduce more detailed background knowledge.

A.1. Normalizing Flows

Let $\mathbf{x}$ be a high-dimensional continuous variable. We suppose that $\mathbf{x}$ is drawn from $p^*(\mathbf{x})$, which is the true data distribution. Given a collected dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_D\}$, we are interested in approximating $p^*(\mathbf{x})$ with a model $p_\theta(\mathbf{x})$. We optimize θ by minimizing the negative log-likelihood

$$\mathcal{L}(\mathcal{D}) = \sum_{i=1}^{D} -\log p_\theta(\mathbf{x}_i). \tag{8}$$

For some settings, variable $\tilde{\mathbf{x}}$ is discrete, e.g., image pixel values are often integers. In these cases, we dequantize $\tilde{\mathbf{x}}$ by adding continuous noise $\boldsymbol{\mu}$ to it, resulting in a continuous variable $\mathbf{x} = \tilde{\mathbf{x}} + \boldsymbol{\mu}$. As shown by Ho et al. (2019), the log-likelihood of $\tilde{\mathbf{x}}$ is lower-bounded by the log-likelihood of $\mathbf{x}$.

Normalizing flows enable computation of $p_\theta(\mathbf{x})$, even though it is usually intractable for many other model families. A normalizing flow (Rezende & Mohamed, 2015) is composed of a series of invertible functions $\mathbf{f} = \mathbf{f}_1 \circ \mathbf{f}_2 \circ ... \circ \mathbf{f}_K$, which transform $\mathbf{x}$ to a latent code $\mathbf{z}$ drawn from a simple distribution. Therefore, with the *change of variables* formula, we can rewrite $\log p_\theta(\mathbf{x})$ to be

$$\log p_\theta(\mathbf{x}) = \log p_Z(\mathbf{z}) + \sum_{i=1}^{K} \log \left| \det \left(\frac{\partial \mathbf{f}_i}{\partial \mathbf{r}_{i-1}} \right) \right|, \tag{9}$$

where $\mathbf{r}_i = \mathbf{f}_i(\mathbf{r}_{i-1})$, $\mathbf{r}_0 = \mathbf{x}$, and $\mathbf{r}_K = \mathbf{z}$.

A.2. Deep Generative Flows

Deep generative flows (Dinh et al., 2014; 2016; Kingma & Dhariwal, 2018), i.e., flow-based generative models, are developed on the theory of normalizing flows. Each transformation function used in the models is a specifically designed neural network layer that has a tractable Jacobian determinant and inverse. Given a trained flow f, we can easily sample from it

$$\mathbf{z} \sim p_Z(\mathbf{z}), \quad \mathbf{x} = f^{-1}(\mathbf{z}). \tag{10}$$

There have been many operations, i.e., layers, proposed in recent years for generative flows. In this section, we discuss some of the commonly used ones, and more related works will be discussed in Appendix B.

Actnorm layers (Kingma & Dhariwal, 2018) perform per-channel affine transformations of the activations using scale

and bias parameters to improve training stability and performance. The actnorm is formally expressed as

$$\mathbf{y}_{:,i,j} = \mathbf{s} \odot \mathbf{x}_{:,i,j} + \mathbf{b},$$

where both the input $\mathbf{x}$ and the output $\mathbf{y}$ are $c \times h \times w$ tensors, c is the channel dimension, and $h \times w$ are spatial dimensions. The parameters $\mathbf{s}$ and $\mathbf{b}$ are $c \times 1$ vectors.

Affine coupling layers (Dinh et al., 2014; 2016) split $\mathbf{x}$ into two parts, $\mathbf{x}_a, \mathbf{x}_b$. And then fix $\mathbf{x}_a$ and force $\mathbf{x}_b$ to only relate to $\mathbf{x}_a$, so that the Jacobian is a triangular matrix. Formally, affine coupling is computed as

$$\begin{aligned} \mathbf{x}_a, \mathbf{x}_b &= \text{split}(\mathbf{x}), \\ \mathbf{y}_a &= \mathbf{x}_a, \\ \mathbf{y}_b &= \mathbf{s}(\mathbf{x}_a) \odot \mathbf{x}_b + \mathbf{b}(\mathbf{x}_a), \\ \mathbf{y} &= \text{concat}(\mathbf{y}_a, \mathbf{y}_b), \end{aligned}$$

where $\mathbf{s}$ and $\mathbf{b}$ are two neural networks with $\mathbf{x}_a$ as input. The split and the concat split and concatenate the variables along the channel axis. Usually, s is restricted to be positive. An additive coupling layer is a special case when $\mathbf{s} = \mathbf{1}$.

Note that actnorm layers only rescale every dimension of $\mathbf{x}$, and the affine coupling layers restrict $\mathbf{x}_b$ to only relate to $\mathbf{x}_a$ but omit dependencies among different dimensions of $\mathbf{x}_b$. Therefore, we need additional layers to capture the local dependencies among dimensions.

Invertible convolutional layers (Kingma & Dhariwal, 2018; Hoogeboom et al., 2019a; Finz et al., 2019) are generalized permutation layers that can capture correlations among dimensions. The 1×1 convolution (Kingma & Dhariwal, 2018) is

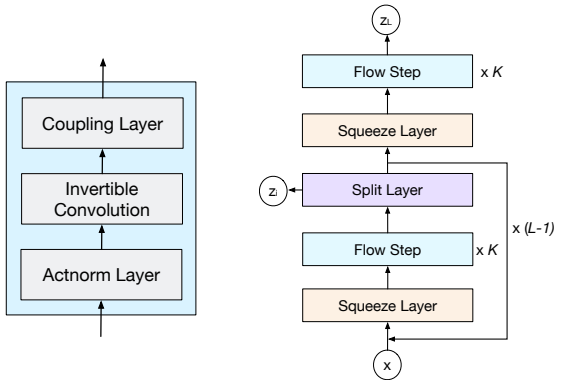
$$\mathbf{y}_{:,i,j} = \mathbf{M}\mathbf{x}_{:,i,j},$$

where $\mathbf{M}$ is a $c \times c$ matrix. The Jacobian of a 1×1 convolution is a block diagonal matrix, so that its log-determinant is $hw \log |\det(\mathbf{M})|$. Note that the 1×1 convolution only operates along the channel axis and ignores the dependencies along the spatial axes.

Emerging convolutions (Hoogeboom et al., 2019a) combine two autoregressive convolutions (Germain et al., 2015; Kingma et al., 2016). Each autoregressive convolution masks out some weights to force an autoregressive structure, so that the Jacobian is a triangular matrix and computing its determinant is efficient. Formally, an emerging convolution is computed as

$$\begin{aligned} \mathbf{M}'_1 &= \mathbf{M}_1 \odot \mathbf{A}_1, \\ \mathbf{M}'_2 &= \mathbf{M}_2 \odot \mathbf{A}_2, \\ \mathbf{y} &= \mathbf{M}'_2 \star (\mathbf{M}'_1 \star \mathbf{x}), \end{aligned}$$

where $\mathbf{M}_1, \mathbf{M}_2$ are convolutional kernels whose size is $c \times c \times d \times d$, and $\mathbf{A}_1, \mathbf{A}_2$ are binary masks. The symbol $\star$ represents the convolution operator.[2] An emerging convolutional layer has the same receptive fields as standard convolutional layers, which can capture correlations between a target pixel and its neighbor pixels. However, like other autoregressive convolutions, computing the inverse of an emerging convolution requires sequentially traversing each dimension of input, so its computation is not parallelizable and is a computational bottleneck when the input is high-dimensional.



(a) Flow step (b) Multi-scale architecture

Figure 2. Overview of architecture of generative flows. We can design the flow step by selecting a suitable convolutional layer and a coupling layer based on the task. Glow (Kingma & Dhariwal, 2018) uses 1×1 convolutions and affine coupling.

Periodic convolutions (Hoogeboom et al., 2019a; Finz et al., 2019) transform both the input and the kernel to the Fourier domain using discrete Fourier transformations, so that the convolution function becomes an element-wise matrix product whose Jacobian is a block diagonal matrix. A period convolution is computed as

$$\mathbf{y}_{u,:,:} = \sum_v \mathcal{F}^{-1}(\mathcal{F}(\mathbf{M}^{(p)}_{u,v,:,:}) \odot \mathcal{F}(\mathbf{x}_{v,:,:})),$$

where $\mathcal{F}$ is a discrete Fourier transformation, and $\mathbf{M}^{(p)}$ is the convolution kernel whose size is $c \times c \times d \times d$. The computational complexity of periodic convolutions is $\mathcal{O}(c^2 hw \log(hw) + c^3 hw)$. Thus, when the input is high-dimensional, both training and sampling become expensive.

Multi-scale architectures (Dinh et al., 2016) have been used to compose flow layers and generate rich models. This idea uses *split layers* to factor out variables and *squeeze layers* to shuffle dimensions, resulting in an architecture with number of flow steps K and number of levels L. We illustrate this architecture in Fig. 2.

[2] In practice, a convolutional layer is usually implemented as an aggregation of cross-correlations. We follow Hoogeboom et al. (2019a) and omit this detail.

B. Related Work

Rezende & Mohamed (2015) developed planar flows for variational inference $\mathbf{z}_{t+1} = \mathbf{z}_t + \mathbf{u}\delta(\mathbf{w}^T\mathbf{z}_t + b)$, where $\mathbf{z}$, $\mathbf{w}$, and $\mathbf{u}$ are d-dimensional vectors, $\delta()$ is an activation function, and b is a scalar.

Berg et al. (2018) generalized these to Sylvester flows $\mathbf{z}_{t+1} = \mathbf{Q}\mathbf{R}\delta(\tilde{\mathbf{R}}\mathbf{Q}^T\mathbf{z}_t + \mathbf{r})$, where $\mathbf{R}$ and $\tilde{\mathbf{R}}$ are upper triangular matrices, $\mathbf{Q}$ is composed of a set of orthonormal vectors, and $\mathbf{r}$ is a d-dimensional vector. The resulting Jacobian determinant can be efficiently computed via Sylvester's identity, just as our methods do. However, Woodbury transformations have key differences from Sylvester flows. First, the inputs to our layers are matrices rather than vectors, so our method operates on high-dimensional input, e.g., images. Second, though Sylvester flows are inverse functions, their inverse is intractable, so they cannot generate samples. Our layers can be inverted efficiently with the Woodbury identity. Third, our layers do not restrict the transformation matrices to be triangular or orthogonal. In fact, Woodbury transformations can be seen as another generalized variant of planar flows with $\delta(\mathbf{x}) = \mathbf{x}$, which can work on high-dimensional tensors, and whose inverse is tractable.

Normalizing flows have also been used for variational inference, density estimation, and generative modeling. Autoregressive flows (Kingma et al., 2016; Papamakarios et al., 2017; Huang et al., 2018; Ma et al., 2019) restrict each variable to depend on those that precede it in a sequence, forcing a triangular Jacobian. Non-linear coupling layers replace the affine transformation function. Specifically, spline flows (Müller et al., 2019; Durkan et al., 2019) use spline interpolation, and Flow++ (Ho et al., 2019) uses a mixture cumulative distribution function to define these functions. Flow++ also uses variational dequantization to prevent model collapse. Many works (Kingma & Dhariwal, 2018; Hoogeboom et al., 2019a; Finz et al., 2019; Karami et al., 2019) develop invertible convolutional flows to model interactions among dimensions. MintNet (Song et al., 2019) is a flexible architecture composed of multiple masked invertible layers. I-ResNet (Behrmann et al., 2018; Chen et al., 2019) uses discriminative deep network architecture as the flow. These two models require iterative methods to compute the inverse. Discrete flows (Tran et al., 2019; Hoogeboom et al., 2019b) and latent flows (Ziegler & Rush, 2019) can be applied to discrete data such as text. Continuous-time flows (Chen et al., 2018; Grathwohl et al., 2018) have been developed based on the theory of ordinary differential equations.

C. Woodbury Transformations

In this section, we introduce more details of our proposed methods.

C.1. Woodbury Transformations

Woodbury Transformations is composed of a channel transformation and a spatial transformation. To implement Woodbury transformations, we need to store four weight matrices, i.e., $\mathbf{U}^{(c)}, \mathbf{U}^{(s)}, \mathbf{V}^{(c)}$, and $\mathbf{V}^{(s)}$. To simplify our analysis, let $d_c \leq d$ and $d_s \leq d$, where d is a constant. This setting is also consistent with our experiments. The size of $\mathbf{U}^{(c)}$ and $\mathbf{V}^{(c)}$ is $\mathcal{O}(dc)$, and the size of $\mathbf{U}^{(c)}$ and $\mathbf{V}^{(c)}$ is $\mathcal{O}(dn)$. The space complexity is $\mathcal{O}(d(c+n))$.

For training and likelihood computation, the main computational bottleneck is computing $\mathbf{y}$ and the Jacobian determinant. To compute $\mathbf{y}$, we need to first compute the channel transformation and then compute the spatial transformation. The computational complexity is $\mathcal{O}(dcn)$. To compute the determinant, we need to first compute the matrix product of $\mathbf{V}$ and $\mathbf{U}$, and then compute the determinant. The computational complexity is $\mathcal{O}(d^2(c+n) + d^3)$.

For sampling, we need to compute the inverse transformations. With the Woodbury identity, we actually only need to compute the inverses of $\mathbf{I}^{(d_s)} + \mathbf{V}^{(s)}\mathbf{U}^{(s)}$ and $\mathbf{I}^{(d_c)} + \mathbf{V}^{(c)}\mathbf{U}^{(c)}$, which are computed with time complexity $\mathcal{O}(d^3)$. To implement the inverse transformations, we can compute the matrix chain multiplication, so we can avoid computing the product of two large matrices twice, yielding cost $\mathcal{O}(c^2 + n^2)$. For example, for the inverse spatial transformation, we can compute it as $\mathbf{x}_c = \mathbf{y} - ((\mathbf{y}\mathbf{U}^{(s)})(\mathbf{I}^{(d_s)} + \mathbf{V}^{(s)}\mathbf{U}^{(s)})^{-1})\mathbf{V}^{(s)}$, so that its complexity is $\mathcal{O}(d^3 + cd^2 + cnd)$. The total computational complexity is $\mathcal{O}(dcn + d^2(n+c) + d^3)$.

In practice, we found that for a high-dimensional input, a relatively small d is enough to obtain good performance, e.g., the input is $256 \times 256 \times 3$ images, and $d = 16$. In this situation, $nc \geq d^3$. Therefore, we can omit d and approximately see the spatial complexity as $\mathcal{O}(c+n)$, and the forward or inverse transformation as $\mathcal{O}(nc)$. They are all linear to the input size.

C.2. Memory-Efficient Woodbury transformations

Memory-Efficient Woodbury transformations can effectively reduce the space complexity. The main idea is to perform spatial transformations along the height and width axes separately, i.e., a height transformation and a width transformation. The transformations are:

$$\begin{aligned}
\mathbf{x}_c &= (\mathbf{I}^{(c)} + \mathbf{U}^{(c)}\mathbf{V}^{(c)})\mathbf{x}, \\
\mathbf{x}_w &= \text{reshape}(\mathbf{x}_c, (ch, w)), \\
\mathbf{x}_w &= \mathbf{x}_c(\mathbf{I}^{(w)} + \mathbf{U}^{(w)}\mathbf{V}^{(w)}), \\
\mathbf{x}_h &= \text{reshape}(\mathbf{x}_w, (cw, h)), \\
\mathbf{y} &= \mathbf{x}_h(\mathbf{I}^{(h)} + \mathbf{U}^{(h)}\mathbf{V}^{(h)}), \\
\mathbf{y} &= \text{reshape}(\mathbf{y}, (c, hw)),
\end{aligned} \tag{11}$$

where reshape$(\mathbf{x}, (n, m))$ reshapes $\mathbf{x}$ to be an $n \times m$ matrix. Matrices $\mathbf{I}^{(w)}$ and $\mathbf{I}^{(h)}$ are w- and h-dimensional identity matrices, respectively. Matrices $\mathbf{U}^{(w)}, \mathbf{V}^{(w)}, \mathbf{U}^{(h)}$, and $\mathbf{V}^{(h)}$ are $w \times d_w$, $d_w \times w$, $w \times d_w$, and $d_w \times w$ matrices, respectively, where d_w and d_h are constant latent dimensions.

Using the Woodbury matrix identity and the Sylvester's determinant identity, we can compute the inverse and Jacobian determinant:

$$\begin{aligned}
\mathbf{y} &= \text{reshape}(\mathbf{y}, (cw, h)), \\
\mathbf{x}_h &= \mathbf{y}(\mathbf{I}^{(h)} - \mathbf{U}^{(h)}(\mathbf{I}^{(d_h)} + \mathbf{V}^{(h)}\mathbf{U}^{(h)})^{-1}\mathbf{V}^{(h)}), \\
\mathbf{x}_w &= \text{reshape}(\mathbf{x}_h, (ch, w)), \\
\mathbf{x}_w &= \mathbf{x}_w(\mathbf{I}^{(w)} - \mathbf{U}^{(w)}(\mathbf{I}^{(d_w)} + \mathbf{V}^{(w)}\mathbf{U}^{(w)})^{-1}\mathbf{V}^{(w)}), \\
\mathbf{x}_c &= \text{reshape}(\mathbf{x}_w, (c, hw)), \\
\mathbf{x} &= (\mathbf{I}^{(c)} - \mathbf{U}^{(c)}(\mathbf{I}^{(d_c)} + \mathbf{V}^{(c)}\mathbf{U}^{(c)})^{-1}\mathbf{V}^{(c)})\mathbf{x}_c, \quad (12)
\end{aligned}$$

$$\begin{aligned}
\log\left|\det(\frac{\partial \mathbf{y}}{\partial \mathbf{x}})\right| &= hw \log\left|\det(\mathbf{I}^{(d_c)} + \mathbf{V}^{(c)}\mathbf{U}^{(c)})\right| \\
&+ ch \log\left|\det(\mathbf{I}^{(d_w)} + \mathbf{V}^{(w)}\mathbf{U}^{(w)})\right| \\
&+ cw \log\left|\det\left(\mathbf{I}^{(d_h)} + \mathbf{V}^{(h)}\mathbf{U}^{(h)}\right)\right| \quad (13)
\end{aligned}$$

where $\mathbf{I}^{(d_w)}$ and $\mathbf{I}^{(d_h)}$ are d_w- and d_h-dimensional identity matrices, respectively. The Jacobian of the reshape() is an identity matrix, so its log-determinant is 0.

We call Equation 11 the memory-efficient Woodbury transformation because it reduces space complexity from $\mathcal{O}(c + hw)$ to $\mathcal{O}(c+h+w)$. This method is effective when h and w are large. To analyze its complexity, we let all latent dimensions be less than d as before. The complexity of forward transformation is $\mathcal{O}(dchw)$; the complexity of computing the determinant is $\mathcal{O}(d(c+h+w)+d^3)$; and the complexity of computing the inverse is $\mathcal{O}(dchw+d^2(c+ch+cw)+d^3)$. The same as Woodbury transformations, when the input is high dimensional, we can omit d. Therefore, the computational complexities of the memory-efficient Woodbury transformation are also linear with the input size.

D. Experiments

In this section, we present more experiments and additional details to aid reproducibility.

D.1. Running Time

We follow Finz et al. (2019) and compare the per-sample running time of Woodbury transformations to other generalized permutations: 1×1 (Kingma & Dhariwal, 2018), emerging (Hoogeboom et al., 2019a), and periodic convolutions (Hoogeboom et al., 2019a; Finz et al., 2019). We test the training time and sampling time. In training, we compute (1) forward propagation, i.e., $\mathbf{y} = \mathbf{f}(\mathbf{x})$, of a given function $\mathbf{f}()$, (2) the Jacobian determinant, i.e., $\det\left(\left|\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right|\right)$, and (3) the gradient of parameters. For sampling, we compute the inverse of transformation $\mathbf{x} = \mathbf{f}^{-1}(\mathbf{y})$. For emerging and periodic convolutions, we use 3×3 kernels. For Woodbury transformations, we fix the latent dimension $d = 16$. For fair comparison, we implement all methods in Pytorch and run them on an Nvidia Titan V GPU. We follow Hoogeboom et al. (2019a) and implement the emerging convolution inverse in Cython, and we compute it on a 4 Ghz CPU (the GPU version is slower than the Cython version). We first fix the spatial size to be 64×64 and vary the channel number. We then fix the channel number to be 96 and vary the spatial size.

The results are shown in Figure 3. For training, the emerging convolution is the fastest. This is because its Jacobian is a triangular matrix, and computing its determinant is much more efficient than other methods, which require computing the determinants of weight matrices. The Woodbury transformation is slightly slower than the 1x1 convolution, since it contains two transformations. ME-Woodbury is slower than the normal variant, because it has three transformations. Emerging convolutions, Woodbury transformations, and 1x1 convolutions only slightly increase with input size, rather than increasing with $\mathcal{O}(c^3)$. This invariance to input size is likely because of how the GPU parallelizes computation. The periodic convolution is efficient only when the input size is small. When the size is large, it becomes slow, e.g., when the input size is $96 \times 64 \times 64$, it is around 30 times slower than Woodbury transformations. In our experiments, we found that the Fourier transformation requires a large amount of memory. According to Finz et al. (2019), the Fourier step may be the bottleneck that impacts periodic convolution's scalability.

For sampling, both 1×1 convolutions and Woodbury transformations are efficient. The 1×1 convolution is the fastest, and the Woodbury transformations are only slightly slower, due to the fact that they are richer transformations. Neither is sensitive to the change of input size. Emerging convolutions and periodic convolutions are much slower than Woodbury transformations, and their running time increases with the input size. When the input size is $96 \times 128 \times 128$, they are around 100 to 200 times slower than Woodbury transformations. This difference is because emerging convolutions must sequentially compute each dimension of the output and cannot make use of parallelization, and periodic transformations require conversion to Fourier form. Based on these results, we can conclude that both emerging convolution and periodic convolution do not scale well to high-dimensional inputs. In contrast, Woodbury transformations are efficient in both training and sampling.

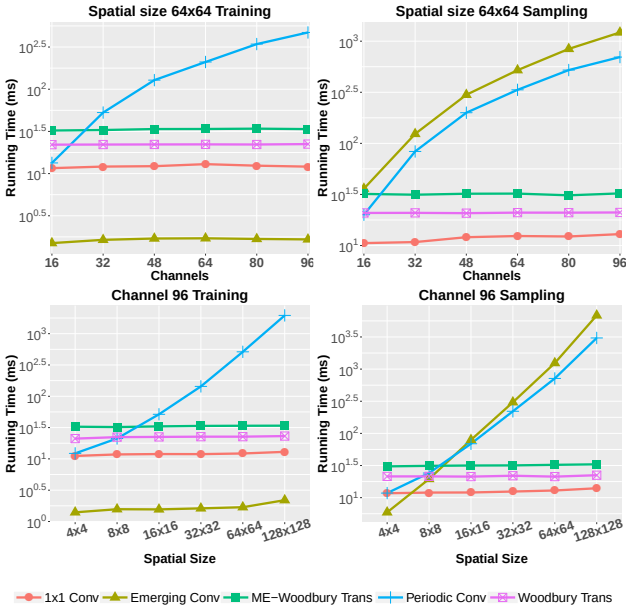


Figure 3. Running time comparisons. Emerging convolutions are inefficient in sampling, since their inverses are not parallelizable. Periodic convolutions are efficient only when the input size is small. Both 1×1 convolutions and Woodbury transformations are efficient in training and sampling.

D.2. Experiments of Quantitative Evaluation

In the experiments of qualitative evalution, we compare Woodbury transformations with 3 permutation layer baselines, i.e., 1x1 convolution, emerging convolution, and periodic coupling, and 2 coupling layer baselines, i.e., neural spline coupling, and MaCow. For all generalized permutation methods, we use affine coupling, which is composed of 3 convolutional layers, and the 2 latent layers have 512 channels. For the neural spline coupling, we set the number of spline bins to 4. The spline parameters are generated by a neural network, which is also composed of convolutional layers. For 32×32 images, we set the number of channels to 256, and for 64×64 images, we set it to 224. For MaCow, we directly substitue affine coupling for MaCow unit. For 32×32 images, we set the convolution channel to 384, and for 64×64 images, we set it to 296.

D.3. Hyper-parameter settings

We use Adam (Kingma & Ba, 2014) to tune the learning rates, with $\alpha = 0.001$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. We use uniform dequantization. The sizes of models we use, and mini-batch sizes for training in our experiments are listed in Table 4.

D.4. Latent Dimension Evaluation

We test the impact of latent dimension d on the performance of Woodbury-Glow. We train our models on CIFAR-10, and use bpd as metric. We vary d within $\{2, 4, 8, 16, 32\}$. The results are in Table 3. When $d < 8$, the model performance will be impacted. When $d > 16$, increasing d will not improve the bpd. This is probably because when d is too small, the latent features cannot represent the input variables well, and when d is too big, the models become hard to train. When $8 \leq d \leq 16$, the Woodbury transformations are powerful enough to model the interactions among dimensions. We also test two values of d, i.e., $16, 32$, of Woodbury-Glow on ImageNet 64×64. The bpds of both d are 3.87, which are consistent with our conclusion. Based on the results, when $8 \leq d \leq 16$, the Woodbury transformations are powerful enough to model the interactions among dimensions.

Table 3. Evaluation of different d (bits per-dimension).

	Woodbury	ME-Woodbury
$d = 2$	3.54	3.53
$d = 4$	3.51	3.51
$d = 8$	3.48	3.48
$d = 16$	3.47	3.48
$d = 32$	3.47	3.48

In all our experiments, we set the latent dimensions of Woodbury transformations, and ME-Woodbury transformations as in Table 5.

Table 4. Model sizes and mini-batch sizes.

Dataset	Mini-batch size	Levels(L)	Steps(K)	Coupling channels
CIFAR-10 32x32	64	3	8	512
ImageNet 32x32	64	3	8	512
ImageNet 64x64	32	4	16	512
LSUN Church 96x96	16	5	16	256
CelebA-HQ 64x64	8	4	8	512
CelebA-HQ 128x128	4	5	24	256
CelebA-HQ 256x256	4	6	16	256

Table 5. Latent dimensions of Woodbury transformations and ME-Woodbury transformations. The numbers in the brackets represent the latent dimension used in that level. For example, the $d_c : \{8, 8, 16\}$, represents that the settings of d_c at the three levels are 8, 8, and 16.

Dataset	Woodbury	ME-Woodbury
CIFAR-10 32x32	$d_c : \{8, 8, 16\}$ $d_s : \{16, 16, 8\}$	$d_c : \{8, 8, 16\}$ $d_h : \{16, 16, 8\}$ $d_w : \{16, 16, 8\}$
ImageNet 32x32	$d_c : \{8, 8, 16\}$ $d_s : \{16, 16, 8\}$	$d_c : \{8, 8, 16\}$ $d_h : \{16, 16, 8\}$ $d_w : \{16, 16, 8\}$
ImageNet 64x64	$d_c : \{8, 8, 16, 16\}$ $d_s : \{16, 16, 8, 8\}$	$d_c : \{8, 8, 16, 16\}$ $d_h : \{16, 16, 8, 8\}$ $d_w : \{16, 16, 8, 8\}$
LSUN Church 96x96	$d_c : \{8, 8, 16, 16, 16\}$ $d_s : \{16, 16, 16, 8, 8\}$	—
CelebA-HQ 64x64	$d_c : \{8, 8, 16, 16\}$ $d_s : \{16, 16, 8, 8\}$	—
CelebA-HQ 128x128	$d_c : \{8, 8, 16, 16, 16\}$ $d_s : \{16, 16, 16, 8, 8\}$	—
CelebA-HQ 256x256	$d_c : \{8, 8, 16, 16, 16, 16\}$ $d_s : \{16, 16, 16, 16, 8, 8\}$	—

E. Sample Quality Comparisons

We compare the samples generated by Woodbury-Glow and Glow models trained on the CelebA-HQ dataset. We follow Kingma & Dhariwal (2018) and randomly hold out 3,000 images as a test set. We use 5-bits images. We use 64×64, 128×128, 256×256 images. Due to the our limited computing resources, we use relatively small models. The model sizes and other settings are listed in Table 4 and Table 5. We generate samples from the models during different phases of training and display them in Figure 4, Figure 5, and Figure 6. For the 64×64 images, the samples show a clear trend where Woodbury-Glow more quickly learns to generate reasonable face shapes. After 100,000 iterations, it can already generate reasonable samples, while Glow's samples are heavily distorted. Woodbury-Glow samples are consistently smoother and more realistic than samples from Glow in all phases of training. The samples demonstrate Woodbury transformations' advantages. For the 128×128 images, both Glow and Woodbury-Glow generate distorted images at iteration 100,000, but Woodbury-Glow seems to improve in later stages, stabilizing the shapes of faces and structure of facial features. Glow, continues generating faces with distorted overall shapes as training continues. For the 256×256 images, neither model ever trains sufficiently to generate highly realistic faces, but Woodbury-Glow makes significantly more progress in these 300,000 iterations than Glow. Glow's samples at 300,000 are still mostly random swirls with an occasional recognizable face, while almost all of Woodbury-Glow's samples look like faces, though distorted. Due to limits on our computational resources, we stopped the higher resolution experiments at 300,000 iterations (rather than running to 600,000 iterations as we did for the 64×64 experiments in the main paper). With a larger model and longer training time, it seems Woodbury-Glow would reach higher sample quality much faster than Glow.

Table 6. Bit per-dimension results on CelebA-HQ

Size of images	Glow	Woodbury-Glow
64×64	1.27	**1.23**
128×128	1.09	**1.04**
256×256	**0.93**	**0.93**

The likelihoods of test set under the trained model are listed in Table 3. For the 64×64 and 128×128 images, Woodbury-Glow scores higher likelihood than Glow. For the 256×256 images, their likelihoods are almost identical, and are better than the score reported in (Kingma & Dhariwal, 2018). This may be due to three possible reasons: (1) We use affine coupling rather than additive coupling, which is a non-volume preserving layer and may improve the likelihoods; (2) Since the test set is randomly collected, it is different from the one used in (Kingma & Dhariwal, 2018); And (3) The model used in (Kingma & Dhariwal, 2018) is very large, so it may be somewhat over-fitting. Surprisingly, the clear difference in sample quality is not reflected by the likelihoods. This discrepancy may be because we use 5-bit images, and the images are all faces, so the dataset is less complicated than other datasets such as ImageNet. Moreover, even though Glow cannot generate reasonable 256×256 samples, the colors of these samples already match the colors of real images well, so these strange samples may non-intuitively be equivalently likely as the face-like samples from Woodbury-Glow.

F. Additional Samples

In this section, we include additional samples from Woodbury-Glow models trained on our various datasets. These samples complement our quantitative analysis. We train our models on CIFAR-10 (Krizhevsky et al., 2009), ImageNet (Russakovsky et al., 2015), the LSUN church dataset (Yu et al., 2015), and the CelebA-HQ dataset (Karras et al., 2017). Specifically, for ImageNet, we use 32×32 and 64×64 images. For the LSUN dataset, we use the same approach as Kingma & Dhariwal (2018) to resize the images to be 96×96. For the CelebA-HQ dataset, we use 64×64, 128×128, and 256×256 images. For LSUN and CelebA-HQ datasets, we use 5-bit images. The parameter settings of our models are in Table 4 and Table 5. Due to the file size limit, more figures can be found in (Lu & Huang, 2020)

Figure 4. Random samples of 64×64 images drawn with temperature 0.7 from a model trained on CelebA data.

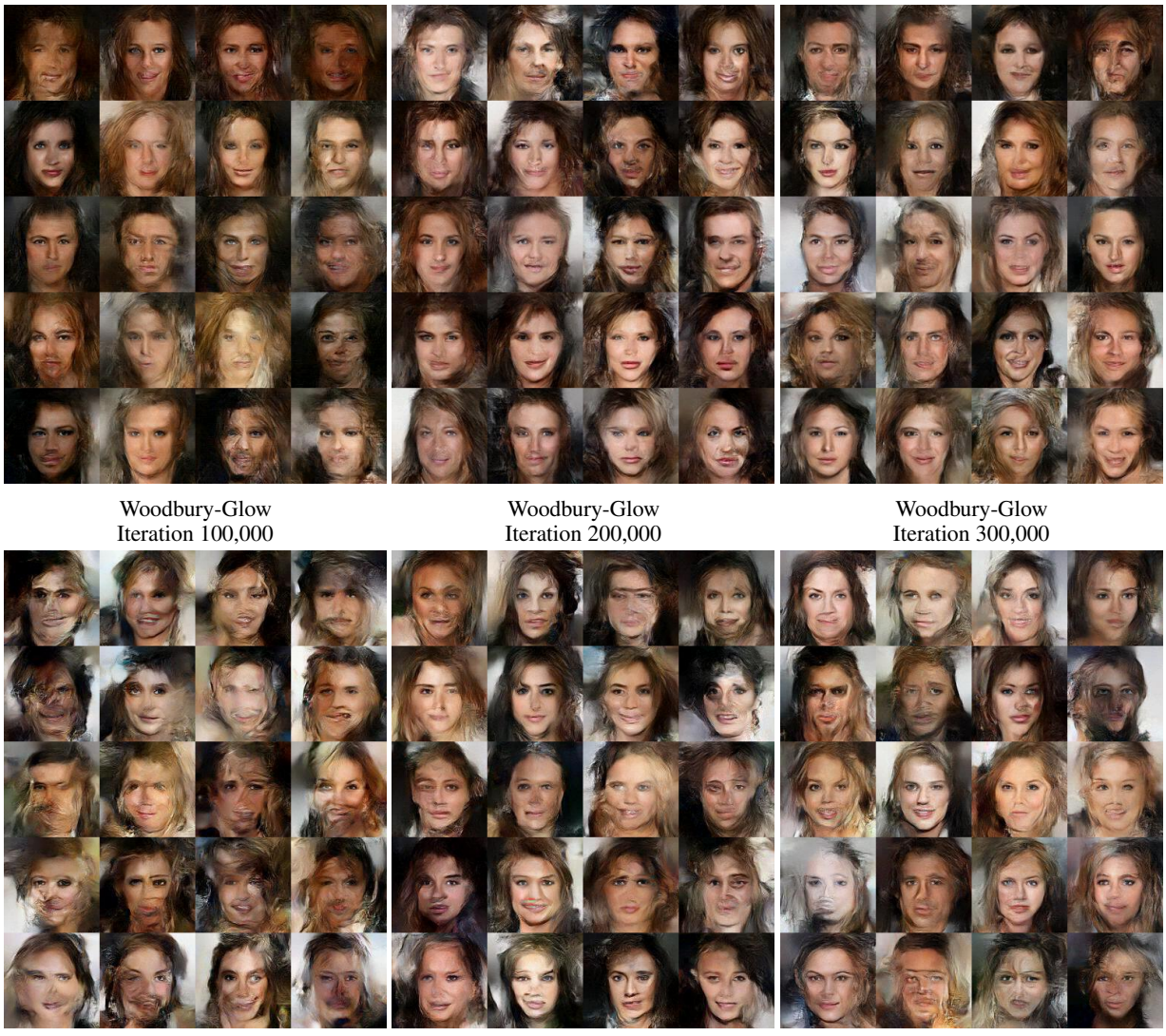


Figure 5. Random samples of 128×128 images drawn with temperature 0.7 from a model trained on CelebA data.

Figure 6. Random samples of 256×256 images drawn with temperature 0.7 from a model trained on CelebA data.