The Convolution Exponential

Emiel Hoogeboom [1] **Victor Garcia Satorras** [1] **Jakub M. Tomczak** [2] **Max Welling** [1]

Abstract

This paper introduces a new method to build linear flows, by taking the exponential of a linear transformation. This linear transformation does not need to be invertible itself, and the exponential has the following desirable properties: it is guaranteed to be invertible, its inverse is straightforward to compute and the log Jacobian determinant is equal to the trace of the linear transformation. An important insight is that the exponential can be computed implicitly, which allows the use of convolutional layers. Using this insight, we develop new invertible transformations named *convolution exponentials* and *graph convolution exponentials*, which retain the equivariance of their underlying transformations. Empirically, we show that the convolution exponential outperforms other linear transformations in generative flows on CIFAR10 and the graph convolution exponential improves the performance of graph normalizing flows.

1. Introduction

Consider a variable $\mathbf{x} \in \mathbb{R}^d$ and an invertible function $f : \mathbb{R}^d \to \mathbb{R}^d$ that maps each $\mathbf{x}$ to a unique output $\mathbf{z} = f(\mathbf{x})$. In this case, the likelihood $p_X(\mathbf{x})$ can be expressed in terms of a base distribution p_Z and the Jacobian determinant of f:

$$p_X(\mathbf{x}) = p_Z(\mathbf{z}) \left| \frac{\mathrm{d}\mathbf{z}}{\mathrm{d}\mathbf{x}} \right|, \tag{1}$$

where p_Z is typically chosen to be a simple factorized distribution such as a Gaussian, and f is a function with learnable parameters that is referred to as a flow. Drawing a sample $\mathbf{x} \sim p_X$ is equivalent to drawing a sample $\mathbf{z} \sim p_Z$ and computing $\mathbf{x} = f^{-1}(\mathbf{z})$.

[1]UvA-Bosch Delta Lab, University of Amsterdam, the Netherlands [2]Vrije Universiteit Amsterdam, the Netherlands. Correspondence to: Emiel Hoogeboom <e.hoogeboom@uva.nl>.

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

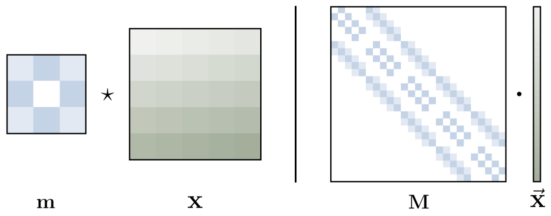


Figure 1. A convolution of a signal $\mathbf{x}$ with a kernel $\mathbf{m}$ (left) is equivalent to a matrix multiplication using a matrix $\mathbf{M}$ and a vectorized signal $\vec{\mathbf{x}}$ (right). In this example, $\mathbf{x}$ has a single channel with spatial dimensions 5×5. The convolution is zero-padded with one pixel on all sides. White square indicates zero values.

1.1. The Matrix Exponential

The matrix exponential gives a method to construct an invertible matrix from any dimensionality preserving linear transformation. For any square (possibly non-invertible) matrix $\mathbf{M}$, the matrix exponential is given by the power series:

$$\exp(\mathbf{M}) \equiv \mathbf{I} + \frac{\mathbf{M}}{1!} + \frac{\mathbf{M}^2}{2!} + \ldots = \sum_{i=0}^{\infty} \frac{\mathbf{M}^i}{i!}. \tag{2}$$

The matrix exponential is well-defined and the series always converges. Additionally, the matrix exponential has two very useful properties: *i)* computing the inverse of the matrix exponential has the same computational complexity as the exponential itself, and *ii)* the determinant of the matrix exponential can be computed easily using the trace:

$$\exp(\mathbf{M})^{-1} = \exp(-\mathbf{M}) \quad \text{and} \quad \log\det\left[\exp(\mathbf{M})\right] = \operatorname{Tr}\mathbf{M}.$$

The matrix exponenential has been largely used in the field of ODEs. Consider the linear ordinary differential equation $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{M}\mathbf{x}$. Given the initial condition $\mathbf{x}(t=0) = \mathbf{x}_0$, the solution for $\mathbf{x}(t)$ at time t can be written using the matrix exponential: $\mathbf{x}(t) = \exp(\mathbf{M} \cdot t) \cdot \mathbf{x}_0$.

1.2. Convolutions as Matrix Multiplications

Convolutional layers in deep learning can be expressed as matrix multiplications. Let $\mathbf{m} \star \mathbf{x}$ denote a convolution[1],

[1]In frameworks convolutions are typically implemented as *cross-correlations*. We follow literature convention and refer to

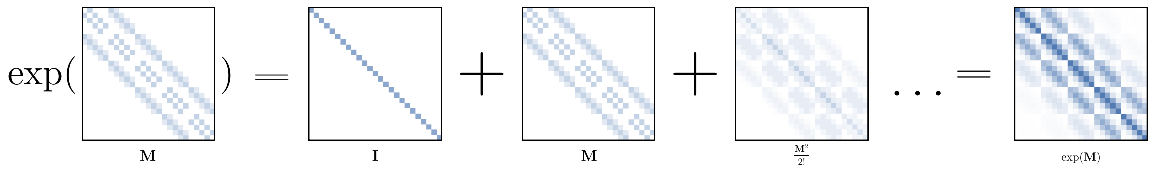


Figure 2. Visualization of the equivalent matrix exponential $\exp(\mathbf{M})$ where $\mathbf{M}$ represents a 2d convolution on a $1 \times 5 \times 5$ input (channel first). In this example the computation is explicit, however in practice the exponential is computed implicitly and the matrices $\mathbf{M}$ and $\exp(\mathbf{M})$ are never stored.

then there exists an equivalent matrix $\mathbf{M}$ such that the convolution is equivalent to the matrix multiplication $\mathbf{M}\vec{\mathbf{x}}$, where $\vec{\cdot}$ vectorizes $\mathbf{x}$. An example is provided in Figure 1. In these examples we use zero-padded convolutions, for periodic and reflective padded convolutions a slightly different equivalent matrix exists. An important detail to notice is that the equivalent matrix is typically unreasonably large to store in memory, its dimensions grow quadratically with the dimension of $\mathbf{x}$. For example, for 2d signals it has size $hwc \times hwc$, where h is height, w is width and c denotes number of channels. In practice the equivalent matrix is never stored but instead, it is a useful tool to utilize concepts from linear algebra.

2. The Convolution Exponential

We introduce a new method to build linear flows, by taking the exponential of a linear transformation. As the main example the exponential of a convolutional layer is taken, which we name the *convolution exponential*. Since a convolutional is linear, it can be expressed as a matrix multiplication (section 1.2). For a convolution with a kernel $\mathbf{m}$, there exists an associated equivalent matrix using the matrix $\mathbf{M}$ such that $\mathbf{m} \star \mathbf{x}$ and $\mathbf{M} \cdot \vec{\mathbf{x}}$ are equivalent. We define the convolution exponential:

$$\mathbf{z} = \mathbf{m} \star_e \mathbf{x}, \tag{3}$$

for a kernel $\mathbf{m}$ and signal $\mathbf{x}$ as the output of the matrix exponential of the equivalent matrix: $\vec{\mathbf{z}} = \exp(\mathbf{M}) \cdot \vec{\mathbf{x}}$, where the difference between $\mathbf{z}$ and $\vec{\mathbf{z}}$ is a vectorization or *reshape* operation that can be easily inverted. Notice that although $\star$ is a linear operation with respect to $\mathbf{m}$ and $\mathbf{x}$, the exponential operation $\star_e$ is only linear with respect to $\mathbf{x}$. Using the properties of the matrix exponential, the inverse is given by $(-\mathbf{m}) \star_e \mathbf{x}$, and the log Jacobian determinant is the trace of $\mathbf{M}$. For a 2d convolutional layer the trace is $hw \cdot \sum_c m_{c,c,m_y,m_x}$ given the 4d kernel tensor $\mathbf{m}$, where height is h, width is w, the spatial center of the kernel is given by m_y, m_x and c iterates over channels. As an example, consider the convolution in Figure 1. The exponential of its equivalent matrix is depicted in Figure 2. In contrast with a standard convolution, the convolution exponential guaranteed to be invertible, and computing the Jacobian determinant is computationally cheap.

Algorithm 1 Implicit matrix exponential

```
Inputs: $\mathbf{M}$, $\mathbf{x}$
Output: $\mathbf{z}$
let $\boldsymbol{\pi} \leftarrow \mathbf{x}$, $\mathbf{z} \leftarrow \mathbf{x}$
for $i = 1, \dots, T$ do
    $\boldsymbol{\pi} \leftarrow \mathbf{M} \cdot \boldsymbol{\pi} / i$
    $\mathbf{z} \leftarrow \mathbf{z} + \boldsymbol{\pi}$
end for
```

Algorithm 2 General linear exponential

```
Inputs: $\mathbf{x}$, linear function $L : \mathcal{X} \to \mathcal{X}$
Output: $\mathbf{z}$
let $\boldsymbol{\pi} \leftarrow \mathbf{x}$, $\mathbf{z} \leftarrow \mathbf{x}$
for $i = 1, \dots, T$ do
    $\boldsymbol{\pi} \leftarrow L(\boldsymbol{\pi}) / i$
    $\mathbf{z} \leftarrow \mathbf{z} + \boldsymbol{\pi}$
end for
```

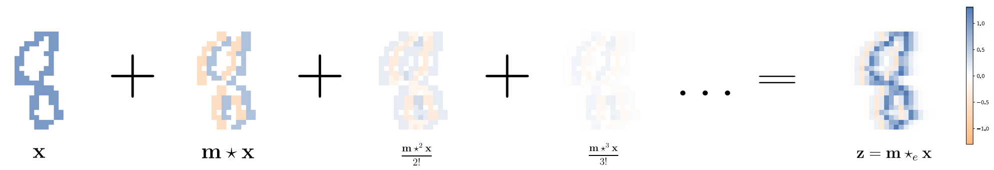
Implicit iterative computation
Due to the popularity of the matrix exponential as a solutions to ODEs, numerous methods to compute the matrix exponential with high numerical precision exist (Arioli et al., 1996; Moler & Van Loan, 2003) which where used by Goliński et al. (2019) to construct orthogonal matrices for flows. However, these methods typically rely on having the matrix $\mathbf{M}$ in memory, which is very expensive for transformations such as convolutional layers. Instead, we propose to solve the exponential using matrix vector products $\mathbf{M}\vec{\mathbf{x}}$. The *exponential* matrix vector product $\exp(\mathbf{M})\vec{\mathbf{x}}$ can be computed implicitly using the power series, multiplied by any vector $\vec{\mathbf{x}}$ using only matrix-vector multiplications:

$$\exp(\mathbf{M}) \cdot \vec{\mathbf{x}} = \vec{\mathbf{x}} + \frac{\mathbf{M} \cdot \vec{\mathbf{x}}}{1!} + \frac{\mathbf{M}^2 \cdot \vec{\mathbf{x}}}{2!} + \dots = \sum_{i=0}^{\infty} \frac{\mathbf{M}^i \cdot \vec{\mathbf{x}}}{i!}, \tag{4}$$

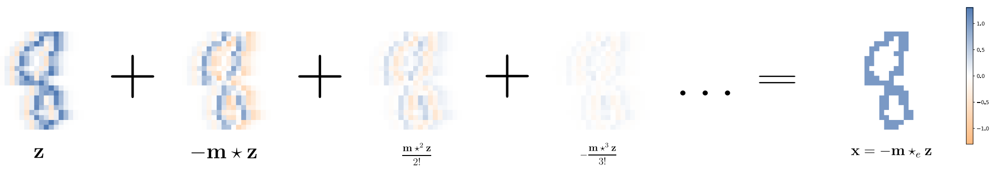
where the term $\mathbf{M}^2 \cdot \mathbf{x}$ can be expressed as two matrix vector multiplications $\mathbf{M}(\mathbf{M} \cdot \mathbf{x})$. Further, computation from previous terms can be efficienty re-used as described in Algorithm 1. Using this, the convolution exponential can be directly computed using the series:

$$\mathbf{m} \star_e \mathbf{x} = \mathbf{x} + \frac{\mathbf{m} \star \mathbf{x}}{1!} + \frac{\mathbf{m} \star (\mathbf{m} \star \mathbf{x})}{2!} + \dots, \tag{5}$$

them as *convolutions* in text. In equations $\star$ denotes a *cross-correlations* and $*$ is a convolution.


$\mathbf{x}$
$\mathbf{m} \star \mathbf{x}$
$\frac{\mathbf{m} \star^2 \mathbf{x}}{2!}$
$\frac{\mathbf{m} \star^3 \mathbf{x}}{3!}$
$\mathbf{z} = \mathbf{m} \star_e \mathbf{x}$
1.0
0.5
0.0
−0.5
−1.0

(a) Forward computation $\mathbf{z} = \mathbf{m} \star_e \mathbf{x}$.


$\mathbf{z}$
$-\mathbf{m} \star \mathbf{z}$
$\frac{\mathbf{m} \star^2 \mathbf{z}}{2!}$
$-\frac{\mathbf{m} \star^3 \mathbf{z}}{3!}$
$\mathbf{x} = -\mathbf{m} \star_e \mathbf{z}$
1.0
0.5
0.0
−0.5
−1.0

(b) Reverse computation $\mathbf{x} = -\mathbf{m} \star_e \mathbf{z}$.

Figure 3. Visualization of the feature maps in the convolution exponential with the edge filter $\mathbf{m} = [0.6, 0, -0.6]$. Note that the notation $\mathbf{w} \star^2 \mathbf{x}$ simply means $\mathbf{w} \star (\mathbf{w} \star \mathbf{x})$, *that is* two subsequent convolutions on $\mathbf{x}$. Similarly for any n the expression $\mathbf{w} \star^n \mathbf{x} = \mathbf{w} \star (\mathbf{w} \star^{n-1} \mathbf{x})$.

which can be done efficiently by simply setting $L(\mathbf{x}) = \mathbf{m} \star \mathbf{x}$ in Algorithm 2. A visual example of the implicit computation is presented in Figure 3.

Power series convergence

Even though the exponential can be solved implicitly, it is uncertain how many terms of the series will need to be expanded for accurate results. Moreover, it is also uncertain that the series can be computed with high numerical precision. To resolve both issues, we constrain the induced matrix norm of the linear transformation. Given the p-norm on the matrix $\mathbf{M}$, a theoretical upper bound for the size of the terms in the power series can be computed using the inequality: $||\mathbf{M}^i \mathbf{x}||_p \leq ||\mathbf{M}||_p^i ||\mathbf{x}||_p$. Hence, an upper bound for relative size of the norm of a term at iteration i, is given by $||\mathbf{M}||_p^i / i!$. Notice that the factorial term in the denominator causes the exponential series to converges very fast, which is depicted in Figure 4.

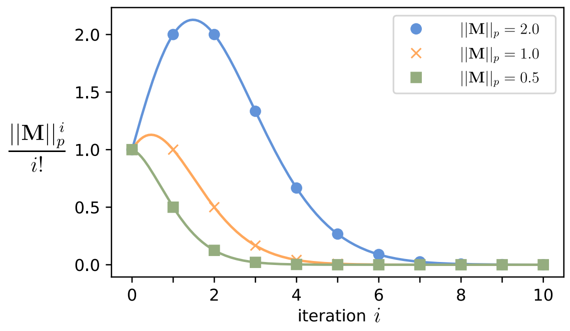


Figure 4. Upper bound of the norm of a term in the power series $||\mathbf{M}^i \mathbf{x}||_p / i!$ at iteration i, relative to the size of the input $||\mathbf{x}||_p$ given a matrix norm.

In our experiments we constrain $\mathbf{M}$ using spectral normalization (Miyato et al., 2018; Gouk et al., 2018), which constrains the ℓ_2 norm of the matrix ($p = 2$) and can be computed efficiently for convolutional layers and standard linear layers. Even though the algorithm produces a lower bound on the ℓ_2 norm, in practice the bound is sufficiently close to produce convergence behaviour as shown in Figure 4. Moreover, the figure depicts worst-case behaviour given the norm, and typically the series converges far more rapidly. In experiments we normalize the convolutional layer using a ℓ_2 coefficient of 0.9 and we find that expanding around 5 or 6 terms of the series is generally sufficient.

2.1. Graph Convolution Exponential

In this section we extend the Convolution Exponential to graph structured data. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes $v \in \mathcal{V}$ and edges $e \in \mathcal{E}$. We define a matrix of nodes $\times$ features $\mathbf{X} \in \mathbb{R}^{N \times nf}$, an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ and a degree matrix $D_{ii} = \sum_j A_{ij}$. Employing a similar notation as (Kipf & Welling, 2016), a linear graph convolutional layer $\text{GCL} : \mathbb{R}^{N \times nf} \rightarrow \mathbb{R}^{N \times nf}$ can be defined as:

$$\text{GCL}_\theta(\mathbf{X}) = \mathbf{I}\mathbf{X}\boldsymbol{\theta}_0 + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{X}\boldsymbol{\theta}_1, \qquad (6)$$

where $\boldsymbol{\theta}_0, \boldsymbol{\theta}_1 \in \mathbb{R}^{nf \times nf}$ are free parameters. Since the output in the graph convolution linearly depends on its inputs, it can also be expressed as a product of some equivalent matrix $\mathbf{M} \in \mathbb{R}^{N \cdot nf \times N \cdot nf}$ with a vectorized signal $\vec{\mathbf{X}} \in \mathbb{R}^{N \cdot nf}$. Note that the trace of this equivalent matrix $\text{Tr } \mathbf{M}$ is equal to the trace of $\boldsymbol{\theta}_0$, multiplied by the number of nodes, *i.e.* $\text{Tr } \mathbf{M} = N \text{ Tr } \boldsymbol{\theta}_0$. This is because the adjacency matrix $\mathbf{A}$ contains zeros on its diagonal and all self-connections are parametrized by $\boldsymbol{\theta}_0$. The proofs to obtain $\mathbf{M}$ from equation 6 and its trace $\text{Tr } \mathbf{M}$ are shown in the Appendix.

The graph convolution exponential can be easily computed by replacing L with the function GCL in Algorithm 2. Since the size and structure of graphs may vary, the norm of $||\mathbf{M}||$ changes depending on this structure even if the parameters $\boldsymbol{\theta}_0$ and $\boldsymbol{\theta}_1$ remain unchanged. As a rule of thumb we find that the graph convolution exponential converges quickly when the norm $||\boldsymbol{\theta}_0||_2$ is constrained to one divided by the maximum number of neighbours, and $||\boldsymbol{\theta}_1||_2$ to one (as it is already normalized via $\mathbf{D}$).

2.2. General Linear Exponentials and Equivariance

In the previous section we generalized the exponential to convolutions and graph convolutions. Graph convolutions can be viewed as permutation equivariant convolutions (Maron et al., 2019). In this section we ask the question whether exponentiation and equivariance commute, *i.e.* which equivariant convolutions will remain equivariant after exponentiation.

Consider a feature field $\boldsymbol{h}(\boldsymbol{x})$ with components $h_c(x_i)$ where c indexes the channel dimensions of the capsules and x_i is the $i'th$ pixel at position x_i. Note that feature fields can consist of multiple capsules but we will restrict ourselves without loss of generality to only one. We now construct a resized vector $\boldsymbol{h}$ which combines the indices into $a = (c, i)$. Next we define a (not necessarily linear) operator K which maps $K : \boldsymbol{h} \rightarrow \boldsymbol{h'}$. Equivariance under K is defined as $[K, M] = KM - MK = 0$ where M is a general kernel that maps one layer of the neural network to the next layer. It states that first performing the map M (usually a convolution) and then the symmetry transform in the activation layer is the same as first transforming the input layer and then convolving. Note that neither K nor M need to be invertible, but that we did require that the symmetry transformation in the input layer and the activation layer are the same (this is less general than the usual equivariance constraint which is of the form $K_1 M = M K_2$). Subject to that constraint, this definition is however very general and encompasses group convolutions (Cohen & Welling, 2016; Dieleman et al., 2016) and permutation equivariant graph convolutions (Maron et al., 2019).

Since $[K, M] = 0$ it follows that $[K^n, M^m] = 0$ for positive powers n, m. Moreover, any linear combination of any collection of powers commutes as well, which in turn implies the statement $[K, \exp M] = 0$ proving that the exponential of the map M is still equivariant. In particular it shows that both the exponential of group convolutions and permutation equivariant graph convolutions are still equivariant.

3. Related Work

Deep generative models can be broadly divided in likelihood based model such as autoregressive models (ARMs) (Germain et al., 2015), Variational AutoEncoders (VAEs) (Kingma & Welling, 2014), Normalizing flows (Rezende & Mohamed, 2015), and adversarial methods (Goodfellow et al., 2014). Normalizing flows are particularly attractive because they admit exact likelihood estimation and can be designed for fast sampling. Several works have studied equivariance in flow-based models (Köhler et al., 2019; Rezende et al., 2019).

Linear flows are generally used to mix information in-between triangular maps. Existing transformations in literature are permutations (Dinh et al., 2017), orthogonal transformations (Tomczak & Welling, 2016; Goliński et al., 2019), 1×1 convolutions (Kingma & Dhariwal, 2018), low-rank Woodbury transformations (Lu & Huang, 2020), emerging convolutions (Hoogeboom et al., 2019a), and periodic convolutions (Finzi et al., 2019; Karami et al., 2019; Hoogeboom et al., 2019a). From these transformations only periodic and emerging convolutions have a convolutional parametrization. However, periodicity is generally not a good inductive bias for images, and since emerging convolutions are autoregressive, their inverse requires the solution to an iterative problem. Notice that Goliński et al. (2019) utilize the matrix exponential to construct orthogonal transformations. However, their method cannot be utilized for convolutional transformations since they compute the exponential matrix explicitly. Our linear exponential can also be seen as a linear ODE (Chen et al., 2018), but the methods are used for different purposes and are computed differently.

4. Experiments

Because image data needs to be dequantized, we optimize the expected lowerbound (ELBO) of the log-likelihood. The performance is compared in terms of negative ELBO and negative log-likelihood (approximated with 1000 importance weighting samples) in bits per dimension on CIFAR10.

4.1. Mixing for generative flows

In this experiment the convolution exponential is utilized as a linear layer in-between affine coupling layers. For a fair comparison, all the methods are implemented in the same framework, and are optimized using the same procedure. For details regarding architecture and optimization see Appendix A. The convolution exponential is compared to other linear mixing layers from literature: 1×1 convolutions (Kingma & Dhariwal, 2018), emerging convolutions (Hoogeboom et al., 2019a), and Woodbury transformations (Lu & Huang, 2020). The number of intermediate channels in the coupling layers are adjusted slightly such that each method has an approximately equal parameter budget. The experiments show that our method outperforms all other methods measured in negative ELBO and log-likelihood (see Table 1). Interestingly, even though emerging convolutions also have a convolutional parametrization, their performance is worse than the convolution exponential. This indicates that the autoregressive factorization of emerging convolutions somewhat limits their flexibility, and the expo-

Table 1. Generative modelling performance with a generative flow. Results computed using $\log_2$ averaged over dimensions, i.e. bits per dimension. Results were obtained by re-implementing the relevant method in the same framework for a fair comparison. Models have an approximately equal parameter budget.

Mixing type	CIFAR10	
	$-$ELBO	$-\log P(x)$
1×1 (Kingma & Dhariwal, 2018)	3.285 ± 0.008	3.266 ± 0.007
Emerging (Hoogeboom et al., 2019a)	3.245 ± 0.002	3.226 ± 0.002
Woodbury (Lu & Huang, 2020)	3.247 ± 0.003	3.228 ± 0.003
Convolution Exponential	$\mathbf{3.237} \pm 0.002$	$\mathbf{3.218} \pm 0.003$

Table 2. Benchmark over different models for synthetic graph datasets. Per-node Negative Log Likelihood (NLL) is reported in nats.

Model	MoG-4	MoG-9	MoG-16	MoG-Ring
Dataset entropy	3.63 $\pm$0.000	4.26 $\pm$0.013	-	$\leq$4.05 $\pm$0.001
Baseline Coupling Flow	3.89 $\pm$0.012	6.14 $\pm$0.012	7.20 $\pm$0.021	4.35 $\pm$0.023
Graph Normalizing Flow	3.69 $\pm$0.016	4.60 $\pm$0.067	5.38 $\pm$0.048	4.22 $\pm$0.025
with Graph Convolution Exponential	$\mathbf{3.68}$ $\pm$0.017	$\mathbf{4.52}$ $\pm$0.047	$\mathbf{5.26}$ $\pm$0.047	$\mathbf{4.19}$ $\pm$0.036

nential parametrization works better.

4.2. Graph Normalizing Flows

In this section we compare our Graph Convolution Exponential with other methods from the literature. As a first baseline we use a baseline coupling flow (Dinh et al., 2017) that does not exploit the graph structure of the data. The second baseline is a Graph Normalizing Flows that uses graph coupling layers as described in (Liu et al., 2019). Since normalizing flows for edges of the graph is an open problem, following (Liu et al., 2019) we assume a fully connected adjacency matrix. Our method then adds a graph convolution exponential layer preceding every coupling layer. For further implementation details refer to Appendix A.1. Following (Liu et al., 2019) we test the methods on the graph datasets Mixture of Gaussian (MoG) and Mixture of Gaussians Ring (MoG-Ring), which are essentially mixtures of permutation of Gaussians. The original MoG dataset considers 4 Gaussians, which we extend to 9 and 16 points obtaining two new datasets MoG-9 and MoG-16 to study performance when the number of nodes increase. Results are presented in Table 2. Adding the graph convolution exponential improves the performance in all four datasets. The improvement becomes larger as the number of nodes increases (e.g. MoG-9 and MoG-16), which is coherent with the intuition that our Graph Convolution Exponential propagates information among nodes in the mixing layer.

5. Conclusion

In this paper we introduced a new simple method to construct invertible transformations, by taking the *exponential* of any linear transformation. Unlike prior work, we observe that the exponential can be computed *implicitly*. Using this we developed new invertible transformations named *convolution exponentials* and *graph convolution exponentials*, and showed that they retain their equivariance properties under exponentiation.

References

Arioli, M., Codenotti, B., and Fassino, C. The padé method for computing the matrix exponential. *Linear Algebra and its Applications*, 240:111 – 130, 1996. ISSN 0024-3795.

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pp. 6572–6583, 2018.

Cohen, T. and Welling, M. Group equivariant convolutional networks. In *Proceedings of the 33nd International Conference on Machine Learning, ICML*, volume 48, pp. 2990–2999. JMLR.org, 2016.

Dieleman, S., Fauw, J. D., and Kavukcuoglu, K. Exploiting cyclic symmetry in convolutional neural networks. In Balcan, M. and Weinberger, K. Q. (eds.), *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pp. 1889–1898, 2016.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using Real NVP. *5th International Conference on Learning Representations, ICLR*, 2017.

Finzi, M., Izmailov, P., Maddox, W., Kirichenko, P., and Wilson, A. G. Invertible convolutional networks. *Workshop on Invertible Neural Nets 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.

Goliński, A., Lezcano-Casado, M., and Rainforth, T. Improving normalizing flows via better orthogonal parameterizations. *Workshop on Invertible Neural Nets and Normalizing Flows*, 2019.

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.

Gouk, H., Frank, E., Pfahringer, B., and Cree, M. J. Regularisation of neural networks by enforcing lipschitz continuity. *CoRR*, abs/1804.04368, 2018.

Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. Flow++: Improving flow-based generative models with variational dequantization and architecture design. *36th International Conference on Machine Learning*, 2019.

Hoogeboom, E., Berg, R. v. d., and Welling, M. Emerging convolutions for generative normalizing flows. *Proceedings of the 36th International Conference on Machine Learning*, 2019a.

Hoogeboom, E., Peters, J. W. T., van den Berg, R., and Welling, M. Integer discrete flows and lossless compression. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*, 2019b.

Karami, M., Schuurmans, D., Sohl-Dickstein, J., Dinh, L., and Duckworth, D. Invertible convolutional flow. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 5636–5646, 2019.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR*, 2015.

Kingma, D. P. and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pp. 10236–10245, 2018.

Kingma, D. P. and Welling, M. Auto-Encoding Variational Bayes. In *Proceedings of the 2nd International Conference on Learning Representations*, 2014.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Köhler, J., Klein, L., and Noé, F. Equivariant flows: sampling configurations for multi-body systems with symmetric energies. *CoRR*, abs/1910.00753, 2019.

Liu, J., Kumar, A., Ba, J., Kiros, J., and Swersky, K. Graph normalizing flows. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 13556–13566, 2019.

Lu, Y. and Huang, B. Woodbury transformations for deep generative flows. *CoRR*, abs/2002.12229, 2020.

Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. Invariant and equivariant graph networks. In *7th International Conference on Learning Representations, ICLR*, 2019.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. *CoRR*, abs/1802.05957, 2018.

Moler, C. and Van Loan, C. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM review*, 45(1):3–49, 2003.

Rezende, D. and Mohamed, S. Variational Inference with Normalizing Flows. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1530–1538. PMLR, 2015.

Rezende, D. J., Racanière, S., Higgins, I., and Toth, P. Equivariant hamiltonian flows. *CoRR*, abs/1909.13739, 2019.

Tomczak, J. M. and Welling, M. Improving variational auto-encoders using householder flow. *arXiv preprint arXiv:1611.09630*, 2016.

A. Experimental Details

A.1. Mixing experiment

We train on the first 40000 images of CIFAR10, using the remaining 10000 for validation. The final performance is shown on the conventional 10000 test images. The flow architecture is multi-scale following (Kingma & Dhariwal, 2018): Each level starts with a squeeze operation, and then 10 subflows which each consist of a linear mixing layer and an affine coupling layer (Dinh et al., 2017). The coupling architecture utilizes densenets as described in (Hoogeboom et al., 2019b). Further, we use variational dequantization (Ho et al., 2019), using the same flow architecture as for the density estimation, but using less subflows. Following (Dinh et al., 2017; Kingma & Dhariwal, 2018) after each level (except the final level) half the variables are transformed by another coupling layer and then factored-out. The final base distribution p_Z is a diagonal Gaussian with mean and standard deviation. All methods are optimized using a batch size of 256 using the Adam optimizer (Kingma & Ba, 2015) with a learning rate of 0.001 with standard settings. More details are given in Table 3. Notice that convexp mixing utilizes a convolution exponential and a 1×1 convolutions, as it tends to map close to the identity by the construction of the power series. Results are obtained by running models three times after random weight initialization, and the mean of the values is reported. Runs require approximately four to five days to complete. Results are obtained by running on four NVIDIA GeForce GTX 1080Ti GPUs, CUDA Version: 10.1.

A.2. Graph Normalizing Flow experiment

The normalizing flows in the graph experiments all utilize three subflows, where a subflow consists of an actnorm layer (Kingma & Dhariwal, 2018), a 1×1 convolution and an affine coupling layer (Dinh et al., 2017). In the model that utilizes the graph convolution exponential, the convolution exponentional precedes each coupling layer. In the baseline coupling flow, the neural networks inside the coupling layers are 4-layer Multi Layer Perceptrons (MLPs) with Leaky Relu activations. In the graph normalizing flow, the neural networks inside the coupling layers are graph neural networks where node and edge operations are performed by a 2-layer and a 3-layer MLPs respectively with ReLU activations. All above mentioned neural networks utilize 64 hidden features.

All experiments are optimized for $35,000$ iterations, using a batch size of 256 and a starting learning rate of 2^{-4} with a learning rate decay factor of 0.1 every $15,000$ iterations. For testing we used $1,280,000$ samples, i.e. 5.000 iterations with a batch size of 256.

Table 3. Architecture settings and optimization settings for the mixing experiments.

Model	levels	subflows	epochs	lr decay	densenet depth	densenet growth	deq. levels	deq. subflows
1×1	2	10	1000	0.995	8	64	1	4
Emerging	2	10	1000	0.995	8	63	1	4
Woodbury	2	10	1000	0.995	8	63	1	4
ConvExp	2	10	1000	0.995	8	63	1	4