Self Normalizing Flows

T. Anderson Keller [1 2]   **Jorn W.T. Peters** [1 2]   **Priyank Jaini** [1 2]   **Emiel Hoogeboom** [1 2]   **Patrick Forré** [2]   **Max Welling** [2]

Abstract

Efficient gradient computation of the Jacobian determinant term is a core problem in many machine learning settings, and especially so in the normalizing flow framework. Most proposed flow models therefore either restrict to a function class with easy evaluation of the Jacobian determinant, or an efficient estimator thereof. However, these restrictions limit the performance of such density models, frequently requiring significant depth to reach desired performance levels. In this work, we propose *Self Normalizing Flows*, a flexible framework for training normalizing flows by replacing expensive terms in the gradient by learned approximate inverses at each layer. This reduces the computational complexity of each layer's exact update from $\mathcal{O}(D^3)$ to $\mathcal{O}(D^2)$, allowing for the training of flow architectures which were otherwise computationally infeasible, while also providing efficient sampling. We show experimentally that such models are remarkably stable and optimize to similar data likelihood values as their exact gradient counterparts, while training more quickly and surpassing the performance of functionally constrained counterparts.

1. Introduction

The framework of normalizing flows (Tabak & Turner, 2013) allows for powerful exact density estimation through the change of variables formula (Rudin, 1987). A significant challenge with this approach is the Jacobian determinant in the objective, which is generally expensive to compute. A large body of work has therefore focused on methods to evaluate the Jacobian determinant efficiently, usually by limiting the expressivity of the transformation. Two classes of functions have been proposed to achieve this: i) those with triangular Jacobians, such that the determinant only

depends on the diagonal (Bogachev et al., 2005; Marzouk et al., 2016; Jaini et al., 2019), and ii) those which are Lipschitz continuous such that Jacobian determinant can be approximated at each iteration through an infinite series (Behrmann et al., 2019; Grathwohl et al., 2019). The drawback of both of these approaches is that they rely on strong functional constraints.

Recently, a number of works have pursued an alternative approach to training unconstrained normalizing flows by avoiding the expensive computation of the Jacobian determinant altogether during training (Gresele et al., 2020; Krämer et al., 2020). However, these works restrict the parametrization of the transformation, or the parameter updates. As a result, it can be difficult to scale these to higher dimensional data such as images.

In this work we introduce a new framework to avoid the determinant computation during training, which we name the *Self Normalizing Framework*. Instead of computing the log Jacobian determinant, we approximate its gradient directly. This can be achieved through the insight that the derivative of the log Jacobian determinant is given by the inverse of the Jacobian itself. In the framework, flow components learn to approximate their own inverse through a self-supervised layer-wise reconstruction loss. Further, we then define the overall density model as a mixture of the probability induced by both the forward and inverse transformations and show how both transformations can be updated symmetrically using their respective learned inverses directly in the gradient. Ultimately, this avoids the $\mathcal{O}(D^3)$ complexity required for computing the determinant of each layer at each training iteration, instead substituting it with an additional backwards pass of order $\mathcal{O}(D^2)$ required to propagate the reconstruction error gradients.

2. Related Work

The field of normalizing flows can be broadly divided into linear and non-linear flows (Papamakarios et al., 2019). Non-linear flows are generally constructed either by constraining the Jacobian to be triangular (Kingma et al., 2016; Papamakarios et al., 2017; van den Berg et al., 2018; Huang et al., 2018; Jaini et al., 2019) or by constraining a residual function to be Lipschitz (Grathwohl et al., 2019; Behrmann et al., 2019; Chen et al., 2019; Perugachi-Diaz et al., 2020). Linear

---

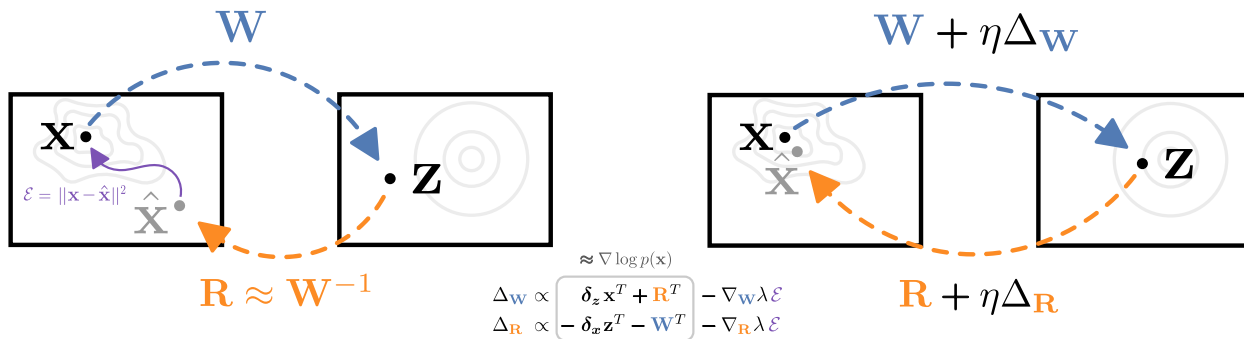[1]UvA-Bosch Delta Lab [2]University of Amsterdam, Netherlands. Correspondence to: T. Anderson Keller <t.anderson.keller@gmail.com>.

$$\mathbf{W} \qquad\qquad\qquad\qquad \mathbf{W} + \eta\Delta_{\mathbf{W}}$$

$$\mathcal{E} = \|\mathbf{x} - \hat{\mathbf{x}}\|^2$$

$$\mathbf{R} \approx \mathbf{W}^{-1} \qquad\qquad\qquad \mathbf{R} + \eta\Delta_{\mathbf{R}}$$

$$\approx \nabla \log p(\mathbf{x})$$

$$\Delta_{\mathbf{W}} \propto \boxed{\boldsymbol{\delta}_z \mathbf{x}^T + \mathbf{R}^T} - \nabla_{\mathbf{W}}\lambda\mathcal{E}$$
$$\Delta_{\mathbf{R}} \propto \boxed{-\boldsymbol{\delta}_x \mathbf{z}^T - \mathbf{W}^T} - \nabla_{\mathbf{R}}\lambda\mathcal{E}$$

Figure 1. Overview of self normalizing flows. A matrix $\boldsymbol{W}$ transforms $\boldsymbol{x}$ to $\boldsymbol{z}$. The matrix $\boldsymbol{R}$ is constrained to approximate the inverse of $\boldsymbol{W}$ with a reconstruction loss $\mathcal{E}$. The likelihood is efficiently optimized by approximating the gradient of the log Jacobian determinant with the learned inverse.

flows are constructed using a variety of methods. Examples of linear flows are 1×1 convolutions (Kingma & Dhariwal, 2018) which have block-diagonal structure, periodic convolutions (Karami et al., 2019; Hoogeboom et al., 2019) which leverage the frequency domain, Woodbury flows (Lu & Huang, 2020) that use low-rank transformations and relative gradient-based flows (Gresele et al., 2020) that re-frame optimization of fully connected linear flows. The disadvantage of these methods is that they are either constrained to a subset of transformations, or based on matrix decomposition structures that cannot straightforwardly be extended to convolutional weight sharing.

The work of Gresele et al. (2020) is most similar to ours in the goal of training unconstrained normalizing flows through the use of efficient gradient computations. In Gresele et al. (2020) this is achieved by applying a carefully constructed post-conditioner to the gradient of each layer, transforming it into the natural gradient (Cardoso & Laheld, 1996; Amari, 1998), thereby selectively canceling out the inverse normally required during training. However, since parameters are extensively shared for convolutions, natural gradient methods cannot be straightforwardly applied to convolutional layers. In contrast, the framework proposed in this paper makes use of the traditional gradient, allowing for more flexibility in parameterization, requiring only that an inverse function can be learned and maintained throughout training.

Related to our framework, the idea of using learned inverse functions in the setting of density estimation was proposed in early work on invertible neural networks (Rippel & Adams, 2013). In that work, similar to ours, both directions of the density model are parameterized and constrained to be approximate inverses through a reconstruction loss. The learned 'encoder' is then used to approximate the marginal distribution of the data in latent space by finding the best fit of a tractable parametric family (such as the Beta family), and the divergence of the approximate latent distribution from the target latent distribution is then min-

imized. Important differences with our work are that we compute the likelihood exactly through the change of variables formula, and use the Jacobian of our learned 'decoder' as an approximation to the gradient of the intractable Jacobian determinant term which arises. An advantage of this approach is that we no longer need to explicitly constrain our decoder to be well conditioned, or invertible, since this constraint is implicitly imposed through the maximization of the Jacobian determinant. Additionally, our framework novelly models the density of observations as a mixture of the density under the forward and inverse transformations, taking advantage of all learned parameters.

More broadly, the idea of greedy layer-wise learning has been explored in many forms for training neural networks (Bengio et al., 2006; Hinton et al., 2006; Löwe et al., 2019). One influential class of work uses stacked auto-encoders, or deep belief networks, for pre-training or representation learning (Bengio et al., 2006; Hinton et al., 2006; Vincent et al., 2008; Kosiorek et al., 2019). Our work leverages similar models and training paradigms, but introduces them into a modern flow framework, demonstrating additional potential uses for the learned feedback connections.

Another related class of work addresses the biological implausibility of backpropagation also through learned layer-wise autoencoders. Target propagation (Lecun, 1986; Bengio, 2014; Lee et al., 2015; Bengio, 2020; Meulemans et al., 2020) addresses the so-called weight transport problem by training auto-encoders at each layer of a network and using these learned feedback weights to propagate 'targets' to previous layers. Our method takes inspiration from this approach. Specifically, our method can be viewed as a hybrid of target propagation and backpropagation (Linnainmaa, 1976; Werbos, 1982; Rumelhart et al., 1986) particularly suited to unsupervised density estimation in the normalizing flow framework. The novelty of our approach in this regard lies in the use of the inverse weights directly in the update, rather than in the backward propagation of updates.

Figure 2. MNIST (left) and CIFAR-10 (right) samples generated from a self normalizing Glow model using the exact inverse (top) vs. the approximate learned inverse (bottom). From these samples we observe that the self normalizing flow models have learned to become good generative models of the data. Additionally, comparing the top and bottom rows, we see the inverse approximation is nearly exact.

3. A General Framework for Self Normalizing Flows

3.1. Preliminaries

Given an observation $\boldsymbol{x} \in \mathbb{R}^D$, it is assumed that $\boldsymbol{x}$ is generated from an underlying real vector $\boldsymbol{z} \in \mathbb{R}^D$ through an invertible and differentiable transformation g, and that $f = g^{-1}$ is also differentiable (i.e. g is a diffeomorphism). It is further assumed that $\boldsymbol{z}$ is a sample from a simple known underlying distribution $p_{\boldsymbol{Z}}$ such as a standard Gaussian. Then, the probability density $p_{\boldsymbol{X}}$ can be computed exactly using the change of variables formula:

$$p_{\boldsymbol{X}}(\boldsymbol{x}) = p_{\boldsymbol{Z}}(\boldsymbol{z}) \left| \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} \right| = p_{\boldsymbol{Z}}\left(g^{-1}(\boldsymbol{x})\right) \left| \boldsymbol{J}_{g^{-1}} \right|$$
$$= p_{\boldsymbol{Z}}\left(f(\boldsymbol{x})\right) \left| \boldsymbol{J}_f \right| \quad (1)$$

where the change of volume term $|\boldsymbol{J}_f| = \left| \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right|$ is the determinant of the Jacobian of the transformation between $\boldsymbol{z}$ and $\boldsymbol{x}$, evaluated at $\boldsymbol{x}$.

Typically, the functions f and g are defined as compositions of diffeomorphisms themselves, i.e. $g = g_0 \circ g_1 \circ \ldots g_K$ and $f = f_K \circ f_{K-1} \circ \ldots f_0$ where $g_k^{-1} = f_k$. This formulation takes advantage of the fact that a composition of diffeomorphic functions is also a diffeomorphism, meaning that if each f_k is invertible and differentiable, then so is the composition f and the change of variables formula in Equation 1 still holds.

Most approaches then propose defining and parameterizing only one direction of the flow, for example the 'forward' functions f_k, and compute the inverses g_k exactly when needed. The log-likelihood of the observations is then simultaneously maximized, with respect to a given f_k's vector of parameters $\boldsymbol{\theta}_k$, for all k, requiring the gradient. Using the identity $\frac{\partial}{\partial \boldsymbol{J}} \log |\boldsymbol{J}| = \boldsymbol{J}^{-T}$ we obtain the following gradient of the loss with respect to a given layer k's parameters:

$$\frac{\partial}{\partial \boldsymbol{\theta}_k} \log p_{\boldsymbol{X}}(\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{\theta}_k} \log p_{\boldsymbol{Z}}(f(\boldsymbol{x})) + \frac{\partial \left(\text{vec } \boldsymbol{J}_f\right)^T}{\partial \boldsymbol{\theta}_k} \left(\text{vec } \boldsymbol{J}_f^{-T}\right)$$
$$(2)$$

Following the conventions of Magnus (2010) for matrix derivatives, we make use of the vectorization operator vec which maps from $m \times n$ matricies to $mn \times 1$ column vectors

by stacking the columns of the matrix sequentially, and formulate the parameters $\boldsymbol{\theta}_k$ as column vectors.

3.2. Self Normalizing Framework

In order to avoid the inverse Jacobian in the gradient, we instead propose to define and parameterize *both* the forward and inverse functions f_k and g_k with parameters $\boldsymbol{\theta}_k$ and $\boldsymbol{\gamma}_k$ respectively. We then constrain the parameterized inverse g_k to be approximately equal to the true inverse f_k^{-1} through a layer-wise reconstruction loss. We can thus define our maximization objective as the mixture of the log-likelihoods induced by both models minus the reconstruction penalty:

$$\mathcal{L}(\boldsymbol{x}) = \frac{1}{2} \log p_{\boldsymbol{X}}^f(\boldsymbol{x}) + \frac{1}{2} \log p_{\boldsymbol{X}}^g(\boldsymbol{x}) - \lambda \sum_{k=0}^{K} ||g_k\left(f_k(\boldsymbol{h}_k)\right) - \boldsymbol{h}_k||_2^2$$
$$(3)$$

where $p_{\boldsymbol{X}}^f$ and $p_{\boldsymbol{X}}^g$ now denote the densities induced by both the forwards and inverse transformations separately, and $\boldsymbol{h}_k = \text{gradient\_stop}(f_{k-1} \circ \ldots f_0(\boldsymbol{x}))$ is the output of function f_{k-1} with the gradients blocked such that only g_k and f_k receive gradients from the reconstruction loss at layer k. We see that when $f = g^{-1}$ exactly, this is equivalent to the traditional normalizing flow framework.

By the inverse function theorem, we know that the inverse of the Jacobian of an invertible function is given by the Jacobian of the inverse function, i.e. $\boldsymbol{J}_f^{-1}(\boldsymbol{x}) = \boldsymbol{J}_{f^{-1}}(\boldsymbol{z})$. Therefore, we see that with the above parameterization and constraint, we can approximate both the change of variables formula, and the gradients for both functions, in terms of the Jacobians of the respective inverse functions. Explicitly:

$$\frac{\partial}{\partial \boldsymbol{\theta}_k} \log p_{\boldsymbol{X}}^f(\boldsymbol{x}) \approx \frac{\partial}{\partial \boldsymbol{\theta}_k} \log p_{\boldsymbol{Z}}(f(\boldsymbol{x})) + \frac{\partial \left(\text{vec } \boldsymbol{J}_f\right)^T}{\partial \boldsymbol{\theta}_k} \left(\text{vec } \boldsymbol{J}_g^T\right)$$
$$(4)$$
$$\frac{\partial}{\partial \boldsymbol{\gamma}_k} \log p_{\boldsymbol{X}}^g(\boldsymbol{x}) \approx \frac{\partial}{\partial \boldsymbol{\gamma}_k} \log p_{\boldsymbol{Z}}(g^{-1}(\boldsymbol{x})) - \frac{\partial \left(\text{vec } \boldsymbol{J}_g\right)^T}{\partial \boldsymbol{\gamma}_k} \left(\text{vec } \boldsymbol{J}_f^T\right)$$
$$(5)$$

where Equation 5 follows from the derivation of Equation 4 and the application of the derivative of the inverse. We note that the above approximation requires that the Jacobians of the functions are approximately inverses *in addition* to the functions themselves being approximate inverses. For the models presented in this work, this property is obtained for free since the Jacobian of a linear mapping is the matrix
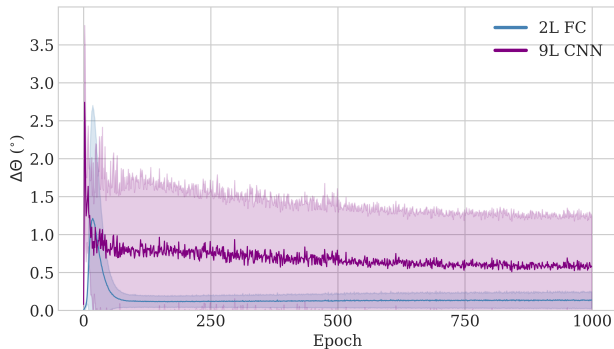
Figure 3. Angle (in degrees) between the exact gradient and the approximate gradient at each step of training. We see that the angle is close to 0.1 degrees for the fully connected network (FC), and less than 1 degree for the convolutional network (CNN), suggesting the approximation is close throughout training.

Figure 4. Time per batch vs. input dimensionality for a single fully connected layer, comparing the self normalizing and exact gradients. We see the self normalizing flow model scales much more favorably with input dimension.

representation of the map itself. However, for more complex mappings, this may not be exactly the case and should be constrained explicitly. We suggest this could be efficiently implemented with an additional loss analogous to a reconstruction loss but employing Jacobian vector products instead of matrix vector products (see Section A.4).

Although there are no known convergence guarantees for such a method, we observe in practice that, with sufficiently large values of λ, most models quickly converge to solutions which maintain the desired constraint. This phenomenon is demonstrated in Figure 3 where it is shown that the angle between the true gradient and the approximate gradient at each training iteration decreases to less than 1 degree over the course of training for both models tested. As a visual example of the quality of the inverse approximation, Figure 2 shows samples from the base distribution $p_{\boldsymbol{Z}}$ passed through both the true inverse f^{-1} (top) and the learned approximate inverse g (bottom) to generate samples from $p_{\boldsymbol{X}}$. As demonstrated by the nearly identical samples, the approximate inverse appears to be a very close match to the true inverse. The details of the models which generated these samples are in Section 5. Further mitigation strategies for potential optimization difficulties are discussed in Sections 6 and A.4. Finally, in Figure 4 we see the efficiency improvements of our method compared with the exact gradient. As expected, we see the self normalizing method scales much more favorably with input dimensionality than the exact gradient method.

3.3. Inference with Self Normalizing Flows

The above framework proposes approximate gradients which allow for the training of normalizing flow architectures without having to compute expensive determinants or matrix inverses. However, to compute the exact log-likelihood of an observation (i.e. perform inference), the exact log Jacobian determinant still has to be computed for
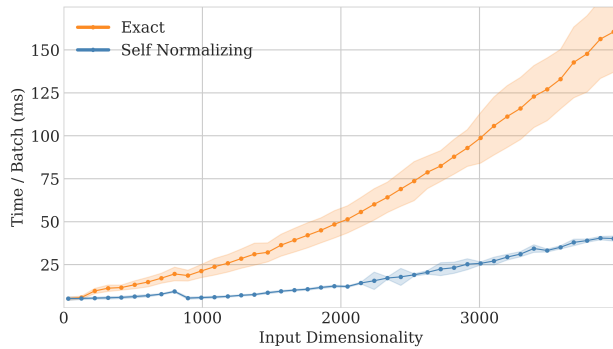
each input, which may be expensive.

To alleviate this limitation we take advantage of the compositional formulation of f, and the multiplicative property of the determinant. In detail, the determinant of the product of square matrices is equal to the product of their determinants. Therefore, assuming our flow is composed of square transformations f_k, the Jacobian is given by $\boldsymbol{J}_f = \prod_k \boldsymbol{J}_{f_k}$, and the log determinant of the Jacobian is given by:

$$\log |\boldsymbol{J}_f| = \sum_k^K \log |\boldsymbol{J}_{f_k}| \qquad (6)$$

For neural network architectures composed of sequential linear and nonlinear transformations, this allows us to separate the components of the Jacobian which are data-independent (e.g. those of linear layers) from those that are data-dependant (e.g. those of the activations). For example, given a network composed of L layers of the form $\boldsymbol{h}_l = \sigma_l(\boldsymbol{W}_l \boldsymbol{h}_{l-1})$ where σ_l is a point-wise non-linearity, combining Equations 1 and 6 yields:

$$\log p_{\boldsymbol{X}}(\boldsymbol{x}) = \log p_{\boldsymbol{Z}}(\boldsymbol{z}) + \sum_k^K \log |\boldsymbol{W}_k| + \sum_k^K \log |\boldsymbol{\Sigma}_k(\boldsymbol{x})| \qquad (7)$$

where $\boldsymbol{\Sigma}_k(\boldsymbol{x})$ is the Jacobian of the activation function at layer k, evaluated at the sample $\boldsymbol{x}$. Importantly, we observe that the data-independent terms, $\sum_k \log |\boldsymbol{W}_k|$, are the expensive part of inference, and the data-dependent terms, $\sum_k \log |\boldsymbol{\Sigma}_k(\boldsymbol{x})|$, are frequently cheap to compute analytically given the derivative of the activation. Therefore, as can be seen, once trained, the expensive data-independent terms must only be computed once and their values can then be reused for all future examples, effectively amortizing the cost of the determinant computation.
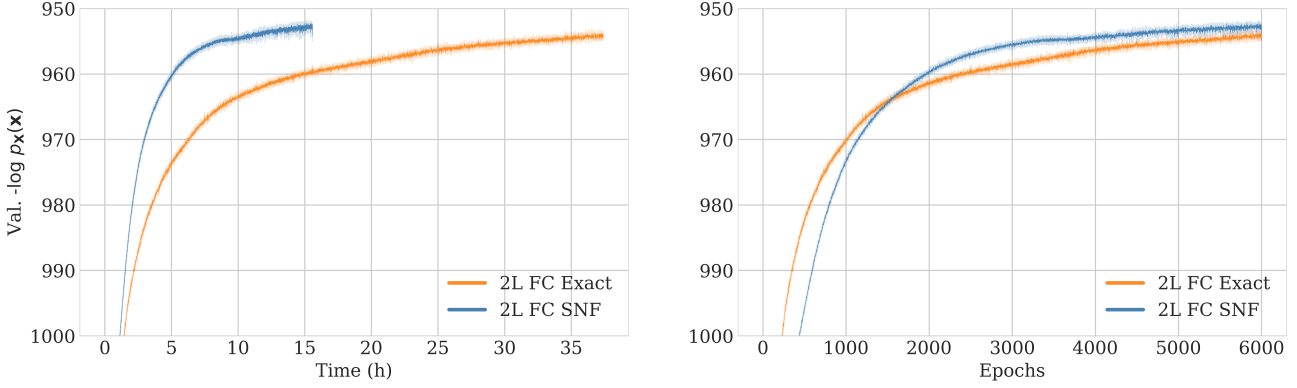
Figure 5. Negative log-likelihood on the MNIST validation set for a 2-layer fully connected flow trained with exact vs. self normalizing (SNF) gradients. Shown vs. training time (left) and vs. epochs (right). We see that both models converge to similar optima while the SNF model trains much more quickly.

4. Self Normalizing Flows

In this section we introduce two simple applications of the self normalizing framework: fully-connected and convolutional self normalizing flows.

4.1. Self Normalizing Fully Connected Layer

As a specific case of the self normalizing framework, we introduce a single fully connected self normalizing layer, as exemplified in Figure 1. Let $W, R \in \mathbb{R}^{D \times D}$, with $f(x) = Wx = z$, and $g(z) = Rz$, such that $W^{-1} \approx R$. We additionally denote the layer-wise reconstruction loss as $\mathcal{E}(x) = ||RWx - x||_2^2$, but leave the derivation of the gradients of this term for the appendix (see Section A.1) since these are efficiently computed by standard deep learning frameworks and require no approximations.

Taking the gradients of Equation 3 with respect to all parameters W and R, we get the following exact gradients:

$$\frac{\partial}{\partial W} \mathcal{L}(x) = \frac{1}{2} \frac{\partial}{\partial W} \log p_X^f(x) + \frac{1}{2} \frac{\partial}{\partial W} \log p_X^g(x) - \lambda \frac{\partial}{\partial W} \mathcal{E}$$
$$= \frac{1}{2} \left(\frac{\partial \log p_Z(Wx)}{\partial Wx} \frac{\partial Wx}{\partial W} + \frac{\partial \log |W|}{\partial W} \right) - \lambda \frac{\partial}{\partial W} \mathcal{E}$$
$$= \frac{1}{2} \left(\delta_z^f x^T + W^{-T} \right) - \lambda \frac{\partial}{\partial W} \mathcal{E} \qquad (8)$$

$$\frac{\partial}{\partial R} \mathcal{L}(x) = \frac{1}{2} \frac{\partial}{\partial R} \log p_X^f(x) + \frac{1}{2} \frac{\partial}{\partial R} \log p_X^g(x) - \frac{\partial}{\partial R} \lambda \mathcal{E}$$
$$= \frac{1}{2} \left(\frac{\partial \log p_Z(R^{-1}x)}{\partial R^{-1}x} \frac{\partial R^{-1}x}{\partial R} + \frac{\partial \log |R^{-1}|}{\partial R} \right) - \lambda \frac{\partial}{\partial R} \mathcal{E}$$
$$= \frac{1}{2} (-R^{-T} \delta_z^g x^T R^{-T} - R^{-T}) - \lambda \frac{\partial}{\partial R} \mathcal{E} \qquad (9)$$

where $\delta_z^f = \frac{\partial \log p_Z(Wx)}{\partial Wx}$ and $\delta_z^g = \frac{\partial \log p_Z(R^{-1}x)}{\partial R^{-1}x}$ can be computed by standard backpropagation. To avoid computing matrix inverses, using our framework we substitute W^{-1} with R in Equation 8, and symmetrically, all

instances of R^{-1} with W in Equation 9. Additionally, we approximate $\delta_z^f \approx \delta_z^g$ in Equation 9 such that we do not need to 'forward propagate' through the inverse model, and can re-use the backpropagated error from the forward model. This corresponds to assuming that $R^{-1}x \approx Wx$ for all x which follows directly from $R^{-1} \approx W$. Ultimately this results in the following approximations to gradient:

$$\frac{\partial}{\partial W} \mathcal{L}(x) \approx \frac{1}{2} \left(\delta_z^f x^T + R^T \right) - \lambda \frac{\partial}{\partial W} \mathcal{E} \qquad (10)$$

$$\frac{\partial}{\partial R} \mathcal{L}(x) \approx \frac{1}{2} \left(-\delta_x^f z^T - W^T \right) - \lambda \frac{\partial}{\partial R} \mathcal{E} \qquad (11)$$

where $\delta_x^f = \frac{\partial \log p_Z(Wx)}{\partial x} = W^T \delta_z^f$. We see that by using such a self normalizing layer, the gradient of the log-determinant of the Jacobian term, which originally required an expensive matrix inversion at each iteration, is approximately given by the weights of the inverse transformation – sidestepping computation of both the Jacobian and the inverse. Additionally, we see the δ_x^f term required for Equation 11 is already computed by traditional backpropagation, making the update efficient. Finally, we note the above gradients can trivially be extended to compositions of such layers, combined with non-linearities, by substituting the appropriate deltas for each layer, and the corresponding layer inputs and outputs for x and z respectively.

4.2. Self Normalizing Convolutional Layer

To construct a self normalizing convolutional layer, we consider the setting where both the forward transformation f, and the inverse transformation g are convolutional with the same kernel size. We note, importantly, that the inverse of a convolution operation is not necessarily another convolution. However, for sufficiently large λ, we observe that f is regularized such that it is restricted to the class of convolutions which is approximately invertible by a convolution.

We define the parameters of f to be the kernel $\boldsymbol{w}$ and similarly, the parameters of g to be the kernel $\boldsymbol{r}$. Then, letting $f(\boldsymbol{x}) = \boldsymbol{w} \star \boldsymbol{x} = \boldsymbol{z}$ and $g(\boldsymbol{z}) = \boldsymbol{r} \star \boldsymbol{z}$, such that $f^{-1} \approx g$ with $\boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^D$, we proceed to derive the exact gradients of the log-likelihood. Again, we ignore the reconstruction term for simplicity as it requires no approximations.

To make the derivation easier, we note that the convolution operation is a linear operation, and can therefore be represented in matrix form. We define a transformation $\boldsymbol{W} = \mathcal{T}(\boldsymbol{w})$, which maps between the convolutional kernel $\boldsymbol{w}$ and the corresponding matrix form of the convolution:

$$\boldsymbol{z} = \mathcal{T}(\boldsymbol{w})\boldsymbol{x} = \boldsymbol{w} \star \boldsymbol{x} \qquad (12)$$

Letting $\boldsymbol{w}$ be a column vector and again making use of the vectorization operator vec, we compute the exact gradient of the log-likelihood with respect to the kernel weights $\boldsymbol{w}$:

$$\frac{\partial}{\partial \boldsymbol{w}} \log p_{\boldsymbol{X}}^f(\boldsymbol{x})$$

$$= \frac{\partial (\operatorname{vec} \mathcal{T}(\boldsymbol{w}))^T}{\partial \boldsymbol{w}} \left(\frac{\partial \log p_{\boldsymbol{Z}}\left(\mathcal{T}(\boldsymbol{w})\boldsymbol{x}\right)}{\partial \operatorname{vec} \mathcal{T}(\boldsymbol{w})} + \frac{\partial \log |\mathcal{T}(\boldsymbol{w})|}{\partial \operatorname{vec} \mathcal{T}(\boldsymbol{w})} \right)$$

$$= \frac{\partial (\operatorname{vec} \mathcal{T}(\boldsymbol{w}))^T}{\partial \boldsymbol{w}} \left(\operatorname{vec}\left[\boldsymbol{\delta}_{\boldsymbol{z}}^f \boldsymbol{x}^T\right] + \operatorname{vec} \mathcal{T}(\boldsymbol{w})^{-T} \right)$$

$$= \boldsymbol{\delta}_{\boldsymbol{z}}^f \star \boldsymbol{x} + \frac{\partial (\operatorname{vec} \mathcal{T}(\boldsymbol{w}))^T}{\partial \boldsymbol{w}} \left(\operatorname{vec} \mathcal{T}(\boldsymbol{w})^{-T} \right) \qquad (13)$$

where we see that the first term $\frac{\partial (\operatorname{vec} \mathcal{T}(\boldsymbol{w}))^T}{\partial \boldsymbol{w}} \left(\operatorname{vec} \left[\boldsymbol{\delta}_{\boldsymbol{z}}^f \boldsymbol{x}^T\right]\right)$ is given by the convolution $\boldsymbol{\delta}_{\boldsymbol{z}}^f \star \boldsymbol{x}$, as is usually done with backpropagation in convolutional neural networks. Then, given our soft constraint $f^{-1} \approx g$, we can approximate $\mathcal{T}(\boldsymbol{w})^{-T}$ with $\mathcal{T}(\boldsymbol{r})^T$ giving us:

$$\frac{\partial}{\partial \boldsymbol{w}} \log p_{\boldsymbol{X}}^f(\boldsymbol{x}) \approx \boldsymbol{\delta}_{\boldsymbol{z}}^f \star \boldsymbol{x} + \frac{\partial (\operatorname{vec} \mathcal{T}(\boldsymbol{w}))^T}{\partial \boldsymbol{w}} \left(\operatorname{vec} \mathcal{T}(\boldsymbol{r})^T \right) \qquad (14)$$

To simplify the second term we note two points. First, the transpose of a convolution can similarly be achieved by standard convolution with a transformed kernel. We call this transformed kernel $\operatorname{flip}(\boldsymbol{r})$ such that $\mathcal{T}(\operatorname{flip}(\boldsymbol{r})) = \mathcal{T}(\boldsymbol{r})^T$. Succinctly, $\operatorname{flip}(\cdot)$ is implemented by swapping the input and output axes, and mirroring the spatial (height and width) dimensions of the kernel. The second point is that the partial derivative $\frac{\partial (\operatorname{vec} \mathcal{T}(\boldsymbol{w}))^T}{\partial \boldsymbol{w}}$ is given by a rectangular matrix, which, when multiplied by the vectorized form of the convolution matrix $\mathcal{T}(\boldsymbol{w})$ yields a constant multiple $\boldsymbol{m}$ element-wise multiplied with the kernel $\boldsymbol{w}$. Each multiple element m_i is given by the number of times the associated kernel element w_i is shared across the matrix $\mathcal{T}(\boldsymbol{w})$. We provide the derivation of this more thoroughly in the appendix, as well as an efficient method for calculating $\boldsymbol{m}$ for arbitrary convolutions (see Section A.2). In combination, we arrive at the approximate gradient of the likelihood with respect to the kernel $\boldsymbol{w}$:

$$\frac{\partial}{\partial \boldsymbol{w}} \log p_{\boldsymbol{X}}^f(\boldsymbol{x}) \approx \boldsymbol{\delta}_{\boldsymbol{z}}^f \star \boldsymbol{x} + \operatorname{flip}(\boldsymbol{r}) \odot \boldsymbol{m} \qquad (15)$$

We see that the symmetric derivation can be obtained for the gradient with respect to the kernel $\boldsymbol{r}$, as outlined below:

$$\frac{\partial}{\partial \boldsymbol{r}} \log p_{\boldsymbol{X}}^g(\boldsymbol{x})$$

$$= \frac{\partial (\operatorname{vec} \mathcal{T}(\boldsymbol{r}))^T}{\partial \boldsymbol{r}} \left(\frac{\partial \log p_{\boldsymbol{Z}}\left(\mathcal{T}(\boldsymbol{r})^{-1}\boldsymbol{x}\right)}{\partial \operatorname{vec} \mathcal{T}(\boldsymbol{r})} + \frac{\partial \log |\mathcal{T}(\boldsymbol{r})^{-1}|}{\partial \operatorname{vec} \mathcal{T}(\boldsymbol{r})} \right)$$

$$= \frac{\partial (\operatorname{vec} \mathcal{T}(\boldsymbol{r}))^T}{\partial \boldsymbol{r}} \left(\operatorname{vec}\left[-\mathcal{T}(\boldsymbol{r})^{-T}\boldsymbol{\delta}_{\boldsymbol{z}}^g \boldsymbol{x}^T \mathcal{T}(\boldsymbol{r})^{-T}\right] - \operatorname{vec} \mathcal{T}(\boldsymbol{r})^{-T} \right)$$

$$\approx \frac{\partial (\operatorname{vec} \mathcal{T}(\boldsymbol{r}))^T}{\partial \boldsymbol{r}} \left(\operatorname{vec}\left[-\mathcal{T}(\boldsymbol{w})^T\boldsymbol{\delta}_{\boldsymbol{z}}^f \boldsymbol{x}^T \mathcal{T}(\boldsymbol{w})^T\right] - \operatorname{vec} \mathcal{T}(\boldsymbol{w})^T \right)$$

$$= -\boldsymbol{\delta}_{\boldsymbol{x}}^f \star \boldsymbol{z} - \operatorname{flip}(\boldsymbol{w}) \odot \boldsymbol{m} \qquad (16)$$

5. Experiments

In our first set of experiments, we train simple flows composed of the above layers and invertible non-linearities on the MNIST dataset. To evaluate our proposed approximate gradients, we compare to baseline models of the same architectures trained with the exact gradient. These architectures are designed to be small, so that it is still possible to compute the exact gradients quickly. We additionally compare with similar recent approaches to training normalizing flows with linear and convolutional layers, namely the relative gradient method of Gresele et al. (2020) and the convolution parametrizations of Hoogeboom et al. (2019; 2020).

To evaluate the scalability of our method, we perform a second set of experiments where we integrate self normalizing flows into the Glow framework (Kingma & Dhariwal, 2018) as a replacement for the 1x1 convolutional mixing layers. In this framework we train models on MNIST, CIFAR-10, and the downsized Imagenet 32x32 dataset. All experimental results are from our re-implementations for consistency. In some cases, due to differing hyper-parameters or errors in prior work, this yielded slightly different results than those published. We provide extended explanations for these discrepancies, as well as a link to our code repository, in the appendix (See Section A.3).

5.1. MNIST

On the MNIST dataset we train three classes of models: 2-layer fully connected (FC) models with smooth-leaky-ReLU activations (Gresele et al., 2020), 9-layer convolutional (CNN) models with spline activations (Durkan et al., 2019), and 32-layer Glow models composed of affine coupling layers and 1x1 convolutional mixing layers (Kingma & Dhariwal, 2018). In all cases, we compare a self normalizing version with its exact gradient baseline.

As can be seen in Table 1 the models composed of self normalizing flow layers are nearly identical in performance to their exact gradient counterparts on the MNIST dataset. We see that the fully connected self normalizing model drastically outperforms the relative gradient method of Gresele

Table 1. Negative Log-likelihood in nats on the MNIST test set. Mean ± std. over 3 runs. Self normalizing flows (SNF) achieve comparable performance to their exact counterparts.

Model	$-\log p_{\boldsymbol{X}}(\boldsymbol{x})$
Relative Grad. FC 2-Layer	1096.5 ± 0.5
Exact Gradient FC 2-Layer	947.6 ± 0.2
SNF FC 2-Layer (ours)	947.1 ± 0.2
Emerging Conv. 9-Layer	645.7 ± 3.6
SNF Conv. 9-Layer (ours)	638.6 ± 0.9
Conv. Exponential 9-Layer	638.1 ± 1.0
Exact Gradient Conv. 9-Layer	637.4 ± 0.2
Glow 2L-16K	575.7 ± 0.8
SNF Glow 2L-16K (ours)	575.4 ± 1.4

et al. (2020), reaching the same performance as the exact gradient method. Additionally, we see that the self normalizing convolutional layer outperforms its constrained counterpart from Hoogeboom et al. (2019), likely due to the fact that the emerging convolution is unable to represent the 1x1 convolution explicitly. We hypothesize that the convolutional self normalizing flow model slightly under-performs the exact gradient method due to the convolutional-inverse constraint. We propose this constraint can be relaxed by using more complex inverse functions, potentially composed of multiple layers and non-linearities (see Section A.4), but leave this to future work. All training details can be found in the appendix (see Section A.3).

In Figure 5 the plot on the right shows that the qualitative convergence properties of the approximate gradient methods are very similar to those of the exact gradient, eventually converging to nearly the same validation likelihood. However, as can be seen in the plot on the left, due to the reduced computational complexity, the approximate gradient method trains in less than half the time, and even more quickly to approximate convergence. This timing comparison is demonstrated more exactly in Table 3 where the time per training batch and time per sample is computed for all models presented in this work. As can be seen, the self normalizing flow models are the fastest of all presented models, with the exception of the relative gradient method which appears to lag behind in likelihood performance. From this table we also see that the relative improvements to speed are directly related to the portions of the network which are replaced with self normalizing components. Since only the 1x1 convolution is replaced in the Glow framework, there is only a slight speed increase to be had.

Finally, we quantitatively measure the quality of the gradient approximation by measuring the alignment of directions of the approximate gradient and the exact gradient in Figure 3. Specifically, for models trained in the self normalizing

Table 2. Bits per dimension for large scale Glow models on two larger scale natural image datasets. Mean ± std. over 3 runs. Self normalizing flows (SNF) achieve comparable performance to their exact counterparts demonstrating that this method scales to large models (> 100 steps of flow). See Section A.3 for details.

Model	CIFAR-10	ImageNet32
Glow	3.36 ± 0.002	4.12 ± 0.002
SNF Glow	3.37 ± 0.004	4.14 ± 0.007

framework, we measure the 'angular error' between the approximate gradient and the true gradient at each training iteration, for each layer, and plot the average angular error over all layers for each training epoch. The shaded area denotes the standard deviation in angles. We make two observations from this figure. First, both the 2-layer FC model and the 9-Layer CNN appear to have an initial slight divergence from the true gradient, but then quickly align to less than 1-degree of error, suggesting they are close approximations to the true gradient direction. Second, we observe that the average angular error of the approximate gradient is significantly larger for the convolutional model than it is for the fully connected model. When comparing this with the results in Table 1 we see that this could be contributing to the slightly lower performance of the self normalizing model when compared with the exact gradient methods. We again hypothesize this could be due to convolutional inverse constraint, making the inverse approximation more challenging. We leave further exploration of this topic to future work but believe that the performance could be ameliorated with improved inverse approximations, potentially achieved through more complex constrained optimization techniques.

5.2. CIFAR-10 and ImageNet 32x32

For large scale experiments, we incorporate self normalizing flow layers into the Glow framework and train the models on CIFAR-10 and Imagenet 32x32. Specifically, we use the same model architectures as those proposed in Kingma & Dhariwal (2018), with some slight changes to the optimization parameters as detailed in Section A.3.

We observe that the self normalizing flow models are able to achieve competitive bits per dimension on both datasets (as seen in Table 2), while simultaneously training slightly faster than their exact gradient counterparts (as seen in Table 3). Importantly, we experimented with values of λ in the set $\{1, 10, 100, 1000\}$, and chose $\lambda = 1000$ for all CIFAR-10 and Imagenet models due to increased stability during training. We observed a slight reduction in final likelihood performance as a result of such a large reconstruction weight and believe this is a factor in the performance gap between the self normalizing and exact gradient models. We believe that with more tuning, or with a dynamically weighted con-

Table 3. Runtime comparison for the models presented in Tables 1 and 2. Hardware and implementation details are in Section A.3

Model	Dataset	Time / batch (ms)	Time / sample (ms)
Exact FC 2-Layer	MNIST	44.9 ± 4.4	61.5 ± 5.8
Relative Gradient FC 2-Layer	MNIST	7.0 ± 0.4	69.2 ± 5.6
Self Normalizing FC 2-Layer	MNIST	18.7 ± 0.8	38.6 ± 3.1
Exact Conv. 9-Layer	MNIST	372.2 ± 24.5	241.6 ± 12.9
Emerging Conv. 9-Layer	MNIST	305.0 ± 14.5	71.7 ± 8.8
Conv. Exponential 9-Layer	MNIST	304.4 ± 11.9	84.2 ± 9.2
Self Normalizing Conv. 9-Layer	MNIST	212.5 ± 37.3	29.9 ± 6.3
Glow 2L-16K	MNIST	583.4 ± 21.2	163.1 ± 21.8
Self Normalizing Glow 2L-16K	MNIST	476.4 ± 16.7	30.6 ± 2.2
Glow 3L-32K	CIFAR-10	1841.3 ± 85.4	126.3 ± 13.9
Self Normalizing Glow 3L-32K	CIFAR-10	1761.2 ± 104.5	97.8 ± 12.9
Glow 3L-48K	ImageNet 32x32	2397.2 ± 204.0	174.8 ± 16.7
Self Normalizing Glow 3L-48K	ImageNet 32x32	2047.9 ± 152.8	150.7 ± 20.8

strained optimization method such as that presented in Platt & Barr (1988), the self normalizing model is likely to match the exact gradient model even more closely. Preliminary experiments in this direction are shown in Section A.5.

Although it is clear that the glow framework is not the optimal setting for the application of self normalizing layers, given the determinant calculation of the 1x1 convolution is relatively quick to evaluate already, we present this work as a proof of scalability of our framework to models with greater than 100 steps of flow (as in the ImageNet 32x32 case), and to larger scale images.

6. Discussion

We see that the above framework yields an efficient update rule for flow-based models which appears to perform similarly to the exact gradient while taking significantly less time to train. However, in its current form, this approach is limited in a number of its own ways.

First, as described in Section 3.3, the evaluation of the exact log-likelihood is still expensive, requiring the computation of the exact log Jacobian determinant of a general function class. However, once a model is trained, the Jacobian determinants of the linear transformations only need to be computed once, and can then be re-used for all future likelihood evaluation, effectively amortizing the cost.

Second, there are no known optimization guarantees for our proposed model. Therefore, the model could converge to a sub-optimal trade-off between the negative log-likelihood and the reconstruction error, or even diverge if the inverse approximation is very poor. In practice, we observe that reconstruction error stays very low for most models when

initialized properly, and final likelihood values are only marginally impacted by the choice of λ. In future work, we intend to explore the possibility of augmented Lagrangian methods, the Modified Differential Method of Multipliers (Platt & Barr, 1988) and other constrained optimization techniques, which could provide better convergence guarantees.

We note one of the biggest constraints of the models presented here is that the inverse of the forward function may not always be given by a function of the same class, as for convolution. Although not evaluated in this work, we note that the general framework proposed here would allow for more complex asymmetric self normalizing flow components and we intend to evaluate these in future work.

Finally, as with all current exact likelihood flow methods, the dimensionality of the representation must stay consistent throughout the depth of the flow. Currently, this problem has been approached with flow components such as coupling layers and variational data augmentation, however these methods are either restrictive in their design or add significant noise. This is clearly one of the greatest architectural constraints for existing normalizing flows and remains so in this work.

7. Conclusion

In summary, we introduce Self Normalizing Flows, a new method to efficiently optimize normalizing flow layers. The method approximates the gradient of the log Jacobian determinant using learned inverses, allowing for the training of otherwise intractable normalizing flow architectures. We demonstrate that our method performs competitively with other models in the literature while simultaneously providing faster training and sampling.

References

Amari, S.-i. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

Behrmann, J., Grathwohl, W., Chen, R. T. Q., Duvenaud, D., and Jacobsen, J. Invertible residual networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 573–582. PMLR, 2019.

Bengio, Y. How auto-encoders could provide credit assignment in deep networks via target propagation. *CoRR*, abs/1407.7906, 2014.

Bengio, Y. Deriving differential target propagation from iterating approximate inverses, 2020.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19: Annual Conference on Neural Information Processing Systems 2006, NeurIPS 2006 Montréal, Canada*, pp. 153–160, 2006.

Biewald, L. Experiment tracking with weights and biases, 2020. URL https://www.wandb.com/. Software available from wandb.com.

Bogachev, V. I., Kolesnikov, A. V., and Medvedev, K. V. Triangular transformations of measures. *Sbornik: Mathematics*, 196(3):309, 2005.

Cardoso, J.-F. and Laheld, B. H. Equivariant adaptive source separation. *IEEE Transactions on signal processing*, 44 (21):3017–3030, 1996.

Chen, T. Q., Behrmann, J., Duvenaud, D., and Jacobsen, J. Residual flows for invertible generative modeling. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*, 2019.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real NVP. *CoRR*, abs/1605.08803, 2016.

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. Neural spline flows. In *Advances in Neural Information Processing Systems 32*, pp. 7511–7522. 2019.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. JMLR Workshop and Conference Proceedings.

Grathwohl, W., Chen, R. T. Q., Bettencourt, J., Sutskever, I., and Duvenaud, D. FFJORD: free-form continuous dynamics for scalable reversible generative models. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Gresele, L., Fissore, G., Javaloy, A., Schölkopf, B., and Hyvärinen, A. Relative gradient optimization of the jacobian term in unsupervised deep learning. *CoRR*, abs/2006.15090, 2020.

Gritsenko, A. A., Snoek, J., and Salimans, T. On the relationship between normalising flows and variational- and denoising autoencoders. In *Workshop on Deep Generative Models for Highly Structured Data, at 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Hinton, G. E., Osindero, S., and Teh, Y.-W. A fast learning algorithm for deep belief nets. *Neural Computation*, 18 (7):1527–1554, 2006.

Hoogeboom, E., van den Berg, R., and Welling, M. Emerging convolutions for generative normalizing flows. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2771–2780. PMLR, 2019.

Hoogeboom, E., Satorras, V. G., Tomczak, J. M., and Welling, M. The convolution exponential and generalized sylvester flows. *CoRR*, abs/2006.01910, 2020.

Huang, C., Krueger, D., Lacoste, A., and Courville, A. C. Neural autoregressive flows. In Dy, J. G. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML*, 2018.

Jaini, P., Selby, K. A., and Yu, Y. Sum-of-squares polynomial flow. *arXiv preprint arXiv:1905.02325*, 2019.

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.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

Kingma, D. P. and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pp. 10236–10245, 2018.

Kingma, D. P., Salimans, T., Józefowicz, R., Chen, X., Sutskever, I., and Welling, M. Improving variational autoencoders with inverse autoregressive flow. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016*, 2016.

Kosiorek, A., Sabour, S., Teh, Y. W., and Hinton, G. E. Stacked capsule autoencoders. In *Advances in Neural Information Processing Systems 32*, pp. 15512–15522. 2019.

Krämer, A., Köhler, J., and Noé, F. Training invertible linear layers through rank-one perturbations, 2020.

Lecun, Y. Learning processes in an asymmetric threshold network. In *Disordered systems and biological organization, Les Houches, France*, pp. 233–240. Springer-Verlag, 1986.

Lee, D.-H., Zhang, S., Fischer, A., and Bengio, Y. Difference target propagation. In *Proceedings of the 2015th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part I*, ECMLPKDD'15, pp. 498–515, 2015.

Linnainmaa, S. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2):146–160, 1976. ISSN 1572-9125. doi: 10.1007/BF01931367.

Löwe, S., O'Connor, P., and Veeling, B. Putting an end to end-to-end: Gradient-isolated learning of representations. In *Advances in Neural Information Processing Systems 32*, pp. 3039–3051. 2019.

Lu, Y. and Huang, B. Woodbury transformations for deep generative flows. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems*, 2020.

Magnus, J. R. On the concept of matrix derivative. *Journal of Multivariate Analysis*, 101(9):2200–2206, 2010.

Marzouk, Y., Moselhy, T., Parno, M., and Spantini, A. An introduction to sampling via measure transport. *arXiv preprint arXiv:1602.05023*, 2016.

Meulemans, A., Carzaniga, F. S., Suykens, J. A. K., Sacramento, J., and Grewe, B. F. A theoretical framework for target propagation, 2020.

Papamakarios, G., Pavlakou, T., and Murray, I. Masked autoregressive flow for density estimation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2017.

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. Normalizing flows for probabilistic modeling and inference, 2019.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. 2019.

Perugachi-Diaz, Y., Tomczak, J. M., and Bhulai, S. Invertible densenets. *CoRR*, abs/2010.02125, 2020.

Platt, J. and Barr, A. Constrained differential optimization. In *Neural Information Processing Systems*, pp. 612–621. American Institute of Physics, 1988.

Rezende, D. J. and Viola, F. Taming vaes, 2018.

Rippel, O. and Adams, R. P. High-dimensional probability estimation with deep density models, 2013.

Rudin, W. *Real and Complex Analysis. 1987*, volume 156. McGraw-Hill Book Company, 1987.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

Tabak, E. G. and Turner, C. V. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013.

van den Berg, R., Hasenclever, L., Tomczak, J. M., and Welling, M. Sylvester normalizing flows for variational inference. In Globerson, A. and Silva, R. (eds.), *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI*, pp. 393–402. AUAI Press, 2018.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pp. 1096–1103, New York, NY, USA, 2008. Association for Computing Machinery.

Werbos, P. J. Applications of advances in nonlinear sensitivity analysis. In Drenick, R. F. and Kozin, F. (eds.), *System Modeling and Optimization*, pp. 762–770, Berlin, Heidelberg, 1982. Springer Berlin Heidelberg. ISBN 978-3-540-39459-4.