4 - Normalising flows

Shan-Conrad Wolf

1 Normalising flows

A *normalising flow* consists of the transformation of one probability distribution into another probability distribution through the application of a series of invertible mappings. Possible applications of a normalising flow include generative models, flexible variational inference, and density estimation, with issues such as scalability of each of these applications depending on the specifics of the invertible mappings involved. These notes provide a short introduction to normalising flows by examining the motivation behind the non-parametric normalising flow proposed in [1] and discussing how deep learning can be applied to produce parametric normalising flows.

2 Non-parametric normalising flows

Consider a set of independent observations $\{x^j\}_{j=1}^m$ drawn from some underlying probability density $\rho(x)$, where $x \in \mathbb{R}^n$. If we can find an invertible transformation $y = y(x)$ such that y is distributed according to a known probability distribution $\mu(y)$, we can straightforwardly perform density estimation as follows:

$$\rho(x) = J_y(x)\mu(y(x)) \tag{1}$$

where $J_y(x) = |\det\left(\frac{\partial y}{\partial x}\right)|$ is the Jacobian of the transformation $y = y(x)$. If we can also sample from μ, then data generation via the inverse mapping is similarly straightforward:

$$\begin{aligned} Y &\sim \mu, \\ X &= x(Y). \end{aligned} \tag{2}$$

If μ and ρ are very similar, finding a suitable transformation $y = y(x)$ won't be too difficult. Conversely, if μ and ρ are very different, it may be very difficult to find a suitable transformation. It's therefore useful to think of our transformation as a flow whereby we slowly try to get from ρ to μ. Additionally, there may be multiple invertible transformations mapping ρ to μ, if there exists some symmetry g such that $\mu(g(y)) = \mu(y)$. Putting restrictions on the types of flows allowed may then allow us to restrict the function space of suitable invertible mappings $y = y(x)$. We therefore define the an invertible flow $z = \phi_t(x)$ such that

$$\begin{aligned} \phi_0(x) &= x, \\ \lim_{t\to\infty} \phi_t(x) &= y(x). \end{aligned} \tag{3}$$

Just like in equation 1, our estimate of $\rho(x)$ at time t is given by

$$\tilde{\rho}_t(x) = J_{\phi_t}(x)\mu(\phi_t(x)) \tag{4}$$

and has the properties that

$$\begin{aligned} \tilde{\rho}_0(x) &= \mu(x), \\ \lim_{t\to\infty} \tilde{\rho}_t(x) &= \rho(x). \end{aligned} \tag{5}$$

2.1 Log-likelihood ascent

So how can we construct a flow ϕ_t that has these properties? Recalling that the KL-divergence is a measure of the distance between two probability distributions, it might make sense to have a flow that moves the probability mass in a direction that decreases the KL-divergence as quickly as possible. Of course, we can always write $\mathrm{KL}(\rho(x) \mid\mid \tilde{\rho}_t(x)) = -L_\rho[\phi_t] - H(\rho)$, where $H(\rho)$ is the (constant) entropy of the probability distribution ρ and $L_\rho[\phi_t]$ is the log-likelihood of our current density estimate under the data distribution, and so descending the KL-divergence is the same as ascending the log-likelihood. At this point, it's worth saying that 'as quickly as possible' is a little misleading, since, by reparametrising the flow's time-variable, we can make $\tilde{\rho}_t$ approach ρ in as short a time as we like. What we really mean is that, for each (infinitesimal) time step, we want our flow to move each chunk of probability mass in a direction that leads to the greatest decrease possible in the KL-divergence. We'll do this now.

The log-likelihood $L_\rho[\phi]$ can be written as follows:

$$L_\rho[\phi_t] = \int \left(\log\left(J_{\phi_t}(x)\right) + \log\left(\mu\left(\phi_t(x)\right)\right)\right)\rho(x)dx. \tag{6}$$

Taking the functional derivative with respect to ϕ_t, [1] we have

$$\frac{\delta L_\rho}{\delta \phi_t} = J_{\phi_t}(x)\left(\frac{\nabla_z \mu(z)}{\mu(z)}\rho_t(z) - \nabla_z \rho_t(z)\right), \tag{7}$$

where z = $\phi_t(x)$, and

$$\rho_t(z) = \left.\frac{\rho(x)}{J_{\phi_t}(x)}\right|_{x=x(z)} \tag{8}$$

is the current distribution of the transported chunks of probability mass.

What we could do now is set $\dot{\phi}_t$ equal to $\delta L_\rho / \delta \phi_t$. However, what we'll actually do is drop the $J_{\phi_t}(x)$ term in equation 7 and set $\dot{\phi}_t$ equal to the resulting expression:

$$\dot{\phi}_t(x) = u_t(\phi_t(x)), \tag{9}$$

where u_t is the velocity field for the flow and has the form

$$u_t(z) = \frac{\nabla_z \mu(z)}{\mu(z)}\rho_t(z) - \nabla_z \rho_t(z). \tag{10}$$

But why did we do this? After all, unless $J_{\phi_t}(x)$ happens to be independent of x, $u_t(\phi_t(x))$ will not be proportional to $\delta L_\rho / \delta \phi_t$, and so with each time step we won't be moving the probability mass in the direction of steepest ascent of the log-likelihood. Yes, it's true that we'll be moving in a direction of ascent, since the Jacobian $J_{\phi_t}(x)$ is positive, but we still need a good reason to decide purposefully not to move in the direction of steepest ascent. The reason for our choice can be seen in the following equation:

$$\left.\frac{\delta L_\rho\left[\varphi \circ \phi_t\right]}{\delta \varphi}\right|_{\varphi = id} = \left(\frac{\nabla_z \mu(z)}{\mu(z)}\rho_t(z) - \nabla_z \rho_t(z)\right), \tag{11}$$

What this equation means is that we evolve the flow in the direction of steepest ascent on a modified log-likelihood function which uses the current sample z rather than the original x. That is, we have a genuinely local algorithm that can be thought of as a series of maps that at each time moves the current distribution ρ_t towards μ in the direction that gets the current distribution (and not the original distribution) closer to μ as quickly as possible. This provides the inspiration for the algorithm we'll see later, wherein we perform a series of simple invertible transformations that at each time moves the current distribution of our data towards the a distribution of our choosing.

2.2 Dual problem

The dual of equation 9 involves looking at the evolution of the probability distribution $\rho_t(z)$ itself rather than the evolution of the chunks of probability mass that make it up. Since ρ_t is a probability distribution, it satisfies conservation of probability mass, and thus satisfies the Liouville equation: [2]

[1] If you're not sure what this means, ignore the integral sign and pretend that ϕ_t is just a finite set of coordinate variables. Then differentiate as usual. Of course, ϕ_t is not a coordinate variable but rather a field, that is, an uncountably infinite set of coordinates.

[2] Conservation of probability mass means that the rate at which probability mass decreases in a fixed volume must equal the rate at which probability mass leaves that volume. That is, $\frac{\partial}{\partial t}\int_V \rho_t dV = -\int_{\partial V}(\rho_t u_t) \cdot dS$. Applying the divergence theorem and noting that this holds for all volumes V, we obtain the Liouville equation.

$$\frac{\partial \rho_t}{\partial t} + \nabla \cdot (\rho_t u_t) = 0 \tag{12}$$

Substituting in our expression for u_t from equation 10, we have:

$$\frac{\partial \rho_t}{\partial t} = \nabla \cdot \left(\mu^2 \nabla \left(\frac{1}{2} \left(\frac{\rho_t}{\mu} \right)^2 \right) \right) \tag{13}$$

The chunks of probability mass flow from their initial positions x to $y(x)$ via the flow $\phi_t(x)$, and the corresponding probability distribution evolves from the (unknown) initial ρ towards the target μ. Simultaneously, $\tilde{\rho}_t$, which is our current estimate (as calculated from our known target distribution μ) of the the unknown distribution ρ, evolves from μ towards ρ. This gives rise to the name of the paper, 'Dual ascent of the log-likelihood'.

If ρ is known, [3] we can put equation 13 in a PDE solver and precisely examine how the flow from the primal problem behaves.

2.3 Convergence in the continuous case

We want to prove that ρ_t converges to μ as t tends to infinity. Let's look at the time derivative of the KL-divergence between them. Noting that the entropy of μ is constant and using equation 13, we have

$$\frac{d}{dt} D_{\mathrm{KL}}(\mu, \rho_t) = -\frac{d}{dt} \int \mu \log(\rho_t) dz = -\int \frac{\mu}{\rho_t} \frac{\partial \rho_t}{\partial t} dz = -\int \frac{\mu^3}{\rho_t} \left| \nabla \left(\frac{\rho_t}{\mu} \right) \right|^2 dz \leq 0, \tag{14}$$

with equality if and only if $\rho_t = \mu$. Thus the Kullback-Liebler divergence of μ and ρ_t does not stop decreasing until $\rho_t(z)$ reaches its target μ. Actually proving convergence is more technical and we won't go into it, but the basic idea behind the proof is still similar to the one behind the lemma from real analysis that a decreasing sequence that is bounded below converges.

2.4 One-dimensional version of the flow

We'd like to be able to scale the finite-sample algorithm we see in the next section to high dimensions. A good starting point would therefore be to look for a coordinate ascent version of the continuous case algorithm just described.

Suppose the chunks of probability mass are only allowed to flow in some direction θ and with a speed that depends only on their coordinate in that direction. We then define a new orthogonal coordinate system with θ as one of the directions

$$x = \begin{pmatrix} x_\theta \\ x_\perp \end{pmatrix}, \qquad \phi_t = \begin{pmatrix} \phi_\theta \\ \phi_\perp \end{pmatrix}, \tag{15}$$

and restrict ϕ_t to take the form

$$\phi_t = \begin{pmatrix} \phi_\theta(x_\theta) \\ x_\perp \end{pmatrix}. \tag{16}$$

The log-likelihood from equation 6 becomes

$$L_\rho \left[\phi_t \right] = \int \left(\log \left(\frac{d\phi_\theta}{dx_\theta} \right) + \log \left(\mu \left(\phi_t(x) \right) \right) \right) \rho(x) dx. \tag{17}$$

If, in addition, we can factorise the target density μ as

$$\mu(x) = \mu_\theta(x_\theta) \mu_\perp(x_\perp), \tag{18}$$

then

$$L_\rho \left[\phi_t \right] = L_{\bar{\rho}} \left[\phi_\theta \right] + \tilde{L}, \tag{19}$$

where

$$\bar{\rho}(x_\theta) = \int \rho(x) dx_\perp \tag{20}$$

[3] which, of course, it isn't, else there'd be no problem to solve!

is the marginal density associated with direction θ,

$$L_{\bar{\rho}}\left[\phi_\theta\right] = \int \left(\log \left(\frac{d\phi_\theta}{dx_\theta} \right) + \log \left(\mu_\theta \left(\phi_\theta(x_\theta) \right) \right) \right) \bar{\rho}(x_\theta) dx_\theta \tag{21}$$

is a one-dimensional log-likelihood functional, and

$$\tilde{L} = \int \log \left(\mu_\perp(x_\perp) \right) \rho(x) dx \tag{22}$$

does not depend on the flow ϕ_t. Helpfully, equation 22 is just a one-dimensional version of equation 6, and so everything we saw in sections 2.1 to 2.3 hold for this flow too. In particular, the one-dimensional flow will only stop when $\bar{\rho} = \mu_\theta$.

This one-dimensional flow only correctly adjusts one marginal of ρ_t, namely, the one corresponding to the θ direction. We therefore still need to adjust the other marginals. Now, suppose that our target distribution μ can be factorised as in equation 18 for any direction θ, and that, at each time, we pick a random direction θ and follow the flow in that direction for one time step. As we make these time steps infinitesimal, all our expressions become linear, [4] and our overall flow becomes a superposition of infinitesimal one-dimensional flows. Because, for each direction θ, the flow only stops when the corresponding marginal $\bar{\rho}$ equals μ_θ, we deduce that the flow makes ρ_t converge to μ and $\tilde{\rho}_t$ converge to ρ.

It turns out that requiring μ to be factorisable as in equation 18 for every direction θ ends up choosing μ for us, since the only distribution satisfying this requirement is the isotropic Gaussian. Indeed, we might have chosen this as our target distribution anyway, in the hope of benefitting from the convergence robustness that the central limit theorem can provide against observational and numerical noise.

2.5 The finite-sample algorithm

Returning to the finite-sample algorithm, we seek a sequence of invertible mappings $(\phi_t)_{t\in\mathbb{Z}_{\geq 0}}$ that we can successively apply in order to map the data $\{x^j\}_{j=1}^m$ (which we refer to as Lagrangian markers) to points $\{y^j\}_{j=1}^m$ that are roughly distributed according to an isotropic normal distribution with mean 0 and precision λ of our choosing. We write $\{z_t^j\}$ for the positions of the Lagrangian markers at time t. At each time step, we want to pick a random direction θ and follow a flow in that direction, as discussed in section 2.4. Further, each flow should be based on an ascent of a local log-likelihood, as discussed in section 2.1.

It turns out that it's actually better to pick a random orthogonal matrix U at each time-step and simultaneously calculate flows in each of the corresponding axis directions. Our choice of an isotropic Gaussian means that we can fully factorise the local log-likelihood, and so we can compute each flow in parallel. That is, at each time step, we left multiply each z_t^j by U, and then simultaneously move each coordinate $(z_t^j)_k$ via flows ϕ_t^k in a direction of ascent of the following one-dimensional Lagrangians:

$$\begin{aligned} L_t^k\left[\phi_t^k\right] &= \frac{1}{m}\sum_{j=1}^m \left(\log \left(\left.\frac{d\phi_t^k}{dz_k}\right|_{z_k=(z_t^j)_k} \right) + \lambda \log(p_{\mathcal{N}(0,1)}(\phi_t^k((z_t^j)_k))) \right) \\ &= \frac{1}{m}\sum_{j=1}^m \left(\log \left(\left.\frac{d\phi_t^k}{dz_k}\right|_{z_k=(z_t^j)_k} \right) - \frac{\lambda}{2}|\phi_t^k((z_t^j)_k)|^2 \right) - \frac{1}{2}\log\left(\frac{2\pi}{\lambda}\right). \end{aligned} \tag{23}$$

Provided the maps ϕ_t^k are explicit, have explicitly computable derivatives with respect to z and their parameters (denoted by α_t^k), have sufficient flexibility, and are sufficiently smooth, they should be able to be used as our normalising flows. We pick the parameters of the maps via ascent, that is:

$$\alpha_t^k \propto \nabla_{\alpha_t^k} L_t^k. \tag{24}$$

As an example, the flows in [1] take the form

$$\varphi(x) = (1-\sigma)x + \varphi_0 + \gamma\sqrt{\epsilon^2 + [(1-\sigma)x - x_0]^2}. \tag{25}$$

Note that, when γ, σ, and φ_0 are zero, the map reduces to the identity. Also, the parameter σ quantifies the amount of stretching; φ_0, the displacement; and γ the slope change at x_0.

[4] That is, we can ignore higher-order terms such as $dAdB$, for quantities A and B that are continuously differentiable with respect to t.

3 Parametric normalising flows with deep learning

Neural network models for normalising flows generally have the following form. We discretise the normalising flow into K steps and then take a family of invertible transformations $\{\phi_\zeta\}_{\zeta \in Z}$ and apply a series of K such transformations $\phi_{\zeta_1}, \dots, \phi_{\zeta_K}$ to a latent code drawn from some known (usually Gaussian) distribution. That is, we sample points

$$\begin{aligned} &z^0 \sim p_\psi(z^0) \\ &z^i = \phi_{\zeta_i}(z^{i-1}),\ i = 1, \dots, K, \end{aligned} \tag{26}$$

where p_ψ is a known probability distribution with parameters ψ (which may or may not be learnt), and estimate probability via:

$$p_{Z^K}(z^K) = p_\psi(\phi_{\zeta_1}^{-1} \circ \dots \phi_{\zeta_K}^{-1}(z^K)) \prod_{i=1}^{k} \left| \frac{\partial \phi_{\zeta_i}}{\partial z^{i-1}} \right|^{-1}_{z^{i-1} = \phi_{\zeta_i}^{-1} \circ \cdots \circ \phi_{\zeta_K}^{-1}(z^K)} \tag{27}$$

The parameters ζ_i for each mapping are the output of passing a collection of activations through a neural network with parameters θ_i. These activations could be a datapoint, x, a hidden layer, h, the previous code layer, z^{i-1}, or some combination of these.

In this section, we draw on excellent notes by Adam Kosoriek [7] and Eric Jang [8].

3.1 Choosing a family of transformations

There are a number of factors to consider when choosing a family $\{\phi_\zeta\}_{\zeta \in Z}$ of transformations.

1. Parallel computation of ϕ_ζ. This is important if we want to sample from high-dimensional p_{Z_K}. Because we use GPUs, parallel computation of ϕ_ζ is more important than the actual computation cost.

2. Invertibility. We would like ϕ_ζ to have an explicit inverse ϕ_ζ^{-1}. This is important if we want to use the normalising flow for density estimation.

3. Parallel computation of ϕ_ζ^{-1}. This is important for density estimation in high dimension.

4. Cheaply computable Jacobian determinant. This is important for cheap density estimation. In the worst case, computing a Jacobian determinant can cost up to $O(n^3)$. Ideally, we'd like to calculate the Jacobian with $O(n)$ cost.

5. Sufficiently flexible. We need the maps $\{\phi_\zeta\}_{\zeta \in Z}$ to be flexible enough that we can produce complex probability distributions with a small number of mappings.

Unfortunately, it's difficult to come up with a flow satisfying all of these properties, since having one of these properties may result in the flow not having another of these properties. We'll look at some examples below.

3.2 Simple flows

The application of deep learning to normalising flows was first introduced in [2]. [5] The two simple flows proposed in the paper are described below. Whilst both flows have easily computable determinants, they use maps ϕ_ζ that are only invertible under certain conditions. Furthermore, they are not very expressive, which means that a large K is needed to model complex probability distributions. The two flows can be thought of as picking a plane or a point and moving probability mass away or towards the plane or point (in direction of the plane's normal vector or radially).

3.2.1 Planar flows

Planar flows take the form

$$\phi_{\zeta=(u,w,b)}(z) = z + uh(w^T z + b), \tag{28}$$

[5] At least, to this author's knowledge.

where h is an element-wise non-linearity. The determinant of this transformation is easily computable with linear cost as follows:

$$\left|\frac{\partial \phi_\zeta}{\partial z}\right| = \left|1 + u^T(h'(w^T z + b)w)\right|. \tag{29}$$

This can be seen by noting that higher-order terms vanish in the Levi-Civita expansion of the determinant.

3.2.2 Radial flows

Radial flows take the form

$$\phi_{\zeta=(z_0,\alpha,\beta)} = z + \beta h(\alpha, r)(z - z_0), \tag{30}$$

where $r = \|z - z_0\|$, $\alpha \in \mathbb{R}_{>0}$, $\beta \in \mathbb{R}$, and $h = 1/(\alpha + r)$. As with planar flows, the determinant can be computed with linear cost:

$$\left|\frac{\partial \phi_\zeta}{\partial z}\right| = (1 + \beta h(\alpha, r))^n \left|1 - \frac{\beta r h(\alpha, r)^2}{1 + \beta h(\alpha, r)}\right|. \tag{31}$$

3.3 Autoregressive flows

We can use autoregressive models to obtain more expressive normalising flows that still have an easily computable Jacobian. We consider transformations

$$y = \phi_\zeta = \phi_{\zeta(z)}(z) = \left(\phi^{(1)}_{\zeta_1}(z_1), \phi^{(2)}_{\zeta_2}(z_{1:2}), \dots, \phi^{(n)}_{\zeta_n}(z_{1:n})\right), \tag{32}$$

where $z_{1:i}$ denotes the first i components of some latent space vector z and the parameters $\zeta = (\zeta_1, \dots, \zeta_n)$ satisfy

$$\zeta_i = \zeta_i(z_{1:i}). \tag{33}$$

These flows are more expressive because they allow for probability mass to be moved in more than one direction (unlike planar flows, which vary only one direction, and radial flows, which do not depend on angle). Furthermore, they have a lower-triangular Jacobian J. The determinant can therefore be computed with $O(n)$ cost by taking the product of the Jacobian's diagonal elements:

$$\det J = \prod_{i=1}^{n} J_{ii}. \tag{34}$$

The two things we need to consider when dealing with autoregressive flows are therefore whether the function ϕ_ζ is invertible and whether parallel computation is possible for each of the forward and inverse flows.

3.3.1 Real Non-Volume Preserving Flows (R-NVP)

Real Non-Volume Preserving Flows (R-NVP) were proposed in [3] and take the form

$$\begin{aligned} y_{1:k} &= z_{1:k}, \\ y_{k+1:n} &= z_{k+1:n} \odot \sigma(z_{1:k}) + \mu(z_{1:k}), \end{aligned} \tag{35}$$

where $1 < k < n$ is chosen to partition the latent space, σ and μ are neural networks mapping $\mathbb{R}^k$ to $\mathbb{R}^{n-k}$, and $\odot$ denotes the Hadamard product. Generally, a sequence of flows of this form are applied, with the latent space variables reordered at each step to ensure that the overall flow affects all coordinates.

The corresponding inverse flow and Jacobian determinant are (with the division in the second equation taken elementwise)

$$\begin{aligned} z_{1:k} &= y_{1:k}, \\ z_{k+1:n} &= \frac{y_{k+1:n} - \mu_{z_{1:k}}}{\sigma(z_{1:k})}, \\ \det J &= \prod_{i=1}^{n-k} \sigma_i(z_{1:k}). \end{aligned} \tag{36}$$

The Jacobian determinant is cheaply computable, and the forward and inverse functions consist of operations that can be carried out in parallel. Consequently, whilst R-NVP is perhaps not as expressive as we would like, it is still a very practical normalising flow to use.

3.3.2 Masked Autoregressive Flow (MAF)

We can obtain a more expressive normalising flow, named Masked Autoregressive Flow (MAF) [4], by autoregressively generating the transformed latent variables y_i as follows:

$$\begin{aligned} y_1 &= \mu_1 + \sigma_1 z_1, \\ y_i &= \mu_i(y_{1:i-1}) + \sigma_i(y_{1:i-1}) z_i. \end{aligned} \tag{37}$$

Writing $\mu = (\mu_1, \ldots, \mu_n) \in \mathbb{R}^n$ and $\sigma = (\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^n_{>0}$, the Jacobian determinant and the inverse transformation are given by

$$\begin{aligned} z &= \frac{y - \mu(y)}{\sigma} \\ \det J &= \prod_{i=1}^{n} \sigma_i(y_{1:i-1}). \end{aligned} \tag{38}$$

where the first equation is written in vectorised form and with an element-wise division. Note that the Jacobian determinant is cheaply computable and that the inverse transformation can be computed in parallel. The flow is also more expressive than R-NVP.

Unfortunately, the forward computation is inherently sequential and prevents us from making full use of GPUs. As a result, MAF is fast for density estimation, but very slow for sampling.

3.3.3 Inverse Autoregressive Flow (IAF)

Inverse autoregressive flows (IAF) [5] are defined via an inverse autoregressive mapping,

$$y_i = z_i \ \sigma_i(z_{1:i-1}) + \mu_i(z_{1:i-1}). \tag{39}$$

Each of the σ_i and μ_i can be computed in parallel in a single forward pass. The inverse transformation takes the form

$$\begin{aligned} z_1 &= \tilde{\mu}_1 + \tilde{\sigma}_1 y_1, \\ z_i &= \tilde{\mu}(z_{1:i-1}) + \tilde{\sigma}_i(z_{1:i-1}) y_i, \end{aligned} \tag{40}$$

where we define $\tilde{\mu}$ and $\tilde{\sigma}$ with

$$\begin{aligned} \tilde{\mu}_i &= -\mu_i / \sigma_i, \\ \tilde{\sigma}_i &= 1/\sigma_i. \end{aligned} \tag{41}$$

From equations 37 and 40, we see that IAF actually just corresponds to the inverse of MAF. Consequently, the Jacobian is still cheaply computable. The inverse flow of IAF therefore requires sequential computation, whilst the forward pass can be done with parallel computation. The result is that IAF is very slow for density estimation, but fast for sampling.

3.3.4 WaveNet

Although neither IAF or MAF would be suitable for a model requiring both density estimation and sampling, IAF and MAF can be combined to create models with both fast density estimation (and thus also fast training) and fast sampling. This was used to great effect in Parallel WaveNet [6], Google's 2017 speech synthesis network.

References

[1] Tabak, E. ; Vanden-Eijnden, E.. Density estimation by dual ascent of the log-likelihood. Comm. Math. Sci. 8 (2010), 217-233.

[2] Rezende, D. J. and Mohamed, S. Variational inference with normalizing flows. ICLR, 2015.

[3] Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real nvp. ICLR, 2017.

[4] Papamakarios, G., Murray, I., and Pavlakou, T. Masked autoregressive flow for density estimation. NeurIPS, 2017.

[5] Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. Improved variational inference with inverse autoregressive flow. NeurIPS, 2016.

[6] Oord, A. v. d., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., Driessche, G. v. d., Lockhart, E., Cobo, L. C., Stimberg, F., et al. Parallel wavenet: Fast high-fidelity speech synthesis. ICML, 2018.

[7] Adam Kosoriek. Blog post at `http://akosiorek.github.io/ml/2018/04/03/norm_flows.html`

[8] Eric Jang. Blog post in two parts at `https://blog.evjang.com/2018/01/nf1.html` and `https://blog.evjang.com/2018/01/nf2.html`.