

A A Connection Between VAEs and Flows

Variational Autoencoders (VAEs) can be seen as a composable stochastic transformations. From this viewpoint, the log-likelihood resulting from a single transformation can be written as

$$\log p(\mathbf{x}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{z})] + \underbrace{\mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \left[\log \frac{p(\mathbf{x}|\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \right]}_{\text{Lik. contrib. } \mathcal{V}(\mathbf{x}, \mathbf{z})} + \underbrace{\mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \left[\log \frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})} \right]}_{\text{Bound looseness } \mathcal{E}(\mathbf{x}, \mathbf{z})}, \quad (4)$$

which consists of 1) the log-likelihood of $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x})$ under the remaining layers $p(\mathbf{z})$, 2) the likelihood contribution term $\mathcal{V}(\mathbf{x}, \mathbf{z})$ and 3) the looseness of the bound $\mathcal{E}(\mathbf{x}, \mathbf{z})$.

Normalizing flows, on the other hand, make use of deterministic transformations. Specifically, using a diffeomorphism $f : \mathcal{Z} \rightarrow \mathcal{X}$, the log-likelihood can be computed as

$$\log p(\mathbf{x}) = \log p(\mathbf{z}) + \underbrace{\log \left| \det \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right|}_{\text{Lik. contrib. } \mathcal{V}(\mathbf{x}, \mathbf{z})}, \quad \mathbf{z} = f^{-1}(\mathbf{x}), \quad (5)$$

which consists 1) the log-likelihood of $\mathbf{z} = f^{-1}(\mathbf{x})$ under $p(\mathbf{z})$ (possibly another flow) and 2) the likelihood contribution term, which here corresponds to the log Jacobian determinant. Notice that for normalizing flows, the likelihood is exact and hence the *bound looseness* term $\mathcal{E}(\mathbf{x}, \mathbf{z}) = 0$.

In the remainder of this section we show that the change-of-variables formula (Eq. 5) can be obtained from the ELBO (Eq. 4).

Proof. We can use a composition of a function g with a Dirac δ -function:

$$\int \delta(g(\mathbf{z})) f(g(\mathbf{z})) \left| \det \frac{\partial g(\mathbf{z})}{\partial \mathbf{z}} \right| d\mathbf{z} = \int \delta(\mathbf{u}) f(\mathbf{u}) d\mathbf{u} \quad (6)$$

to conclude that

$$\delta(g(\mathbf{z})) = \left| \det \frac{\partial g(\mathbf{z})}{\partial \mathbf{z}} \right|_{\mathbf{z}=\mathbf{z}_0}^{-1} \delta(\mathbf{z} - \mathbf{z}_0) \quad (7)$$

with \mathbf{z}_0 being the root of $g(\mathbf{z})$. This result assumes that g is smooth (derivative exists), f has compact support, the root is unique and the Jacobian is non-singular.

Let $f : \mathcal{Z} \rightarrow \mathcal{X}$ be a diffeomorphism and define a pair of deterministic conditionals

$$p(\mathbf{x}|\mathbf{z}) = \delta(\mathbf{x} - f(\mathbf{z})) \quad (8)$$

$$p(\mathbf{z}|\mathbf{x}) = \delta(\mathbf{z} - f^{-1}(\mathbf{x})). \quad (9)$$

Applying the above result to $p(\mathbf{x}|\mathbf{z})$, we set $g(\mathbf{z}) = \mathbf{x} - f(\mathbf{z})$ and find $\mathbf{z}_0 = f^{-1}(\mathbf{x})$ and

$$p(\mathbf{x}|\mathbf{z}) = \delta(\mathbf{z} - f^{-1}(\mathbf{x})) |\det \mathbf{J}| = p(\mathbf{z}|\mathbf{x}) |\det \mathbf{J}|, \quad (10)$$

where

$$\mathbf{J}^{-1} = \left. \frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} \right|_{\mathbf{z}=f^{-1}(\mathbf{x})}.$$

Let further $q(\mathbf{z}|\mathbf{x}) = p(\mathbf{z}|\mathbf{x}) = \delta(\mathbf{z} - f^{-1}(\mathbf{x}))$. The resulting ELBO gives rise to the change-of-variables formula,

$$\log p(\mathbf{x}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \left[\log p(\mathbf{z}) + \log \frac{p(\mathbf{x}|\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} + \log \frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})} \right] \quad (11)$$

$$= \log p(\mathbf{z}) + \log |\det \mathbf{J}|, \quad \text{for } \mathbf{z} = f^{-1}(\mathbf{x}), \quad (12)$$

where the likelihood contribution $\mathcal{V}(\mathbf{x}, \mathbf{z}) = \log \frac{p(\mathbf{x}|\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} = \log |\det \mathbf{J}|$, while the bound looseness term $\mathcal{E}(\mathbf{x}, \mathbf{z}) = \log \frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})} = 0$, trivially.

B The Bound Looseness for Inference Surjections

For inference surjections $f : \mathcal{X} \rightarrow \mathcal{Z}$, the bound looseness term $\mathcal{E}(\mathbf{x}, \mathbf{z}) = 0$, given that the *stochastic right inverse condition* is satisfied. The stochastic right inverse condition requires that $p(\mathbf{x}|\mathbf{z})$ defines a distribution over the possible right inverses of the surjection f .

A right inverse function $g : \mathcal{Z} \rightarrow \mathcal{X}$ to a function $f : \mathcal{X} \rightarrow \mathcal{Z}$ satisfies $f \circ g = \text{id}_{\mathcal{Z}}$, but not necessarily $g \circ f = \text{id}_{\mathcal{X}}$. Here $\text{id}_{\mathcal{S}}$ denotes an identity map defined on the space \mathcal{S} .

We satisfy the stochastic right inverse condition by requiring that $p(\mathbf{x}|\mathbf{z})$ only has support over the *fiber* of \mathbf{z} , i.e. the set of elements $\mathcal{B}(\mathbf{z})$ in the domain \mathcal{X} that are mapped to \mathbf{z} , $\mathcal{B}(\mathbf{z}) := \{\mathbf{x}|\mathbf{z} = f(\mathbf{x})\}$. A simple check for stochastic right invertibility is thus: For any \mathbf{z} , computing $\mathbf{z} = f(\mathbf{x})$, for $\mathbf{x} \sim p(\mathbf{x}|\mathbf{z})$ should return the original \mathbf{z} .

Given that the distribution $p(\mathbf{z})$ has full support over \mathcal{Z} and the stochastic right inverse condition is satisfied, we have that, for any observed \mathbf{x} , only one \mathbf{z} could have given rise to the observation \mathbf{x} . Consequently, the posterior distribution $p(\mathbf{z}|\mathbf{x}) = \delta(\mathbf{z} - f(\mathbf{x}))$ is deterministic. By defining $q(\mathbf{z}|\mathbf{x}) = p(\mathbf{z}|\mathbf{x})$, the bound looseness is thus $\mathcal{E}(\mathbf{x}, \mathbf{z}) = 0$.

C List of SurVAE Layers

See Table 6 and Table 7 for lists of generative and inference surjection layers, respectively.

Table 6: Summary of some generative surjection layers.

Surjection	Forward	Inverse	$\mathcal{V}(\mathbf{x}, \mathbf{z})$
Rounding	$x = \lfloor z \rfloor$	$z \sim q(z x)$ where $z \in [x, x + 1)$	$-\log q(z x)$
Slicing	$\mathbf{x} = \mathbf{z}_1$	$\mathbf{z}_1 = \mathbf{x}, \mathbf{z}_2 \sim q(\mathbf{z}_2 \mathbf{x})$	$-\log q(\mathbf{z}_2 \mathbf{x})$
Abs	$s = \text{sign } z$ $x = z $	$s \sim \text{Bern}(\pi(x))$ $z = s \cdot x, s \in \{1, -1\}$	$-\log q(s x)$
Max	$k = \arg \max \mathbf{z}$ $x = \max \mathbf{z}$	$k \sim \text{Cat}(\boldsymbol{\pi}(x))$ $\mathbf{z}_k = x, \mathbf{z}_{-k} \sim q(\mathbf{z}_{-k} x, k)$	$-\log q(k x) - \log q(\mathbf{z}_{-k} x, k)$
Sort	$\mathcal{I} = \text{argsort } \mathbf{z}$ $\mathbf{x} = \text{sort } \mathbf{z}$	$\mathcal{I} \sim \text{Cat}(\boldsymbol{\pi}(\mathbf{x}))$ $\mathbf{z} = \mathbf{x}_{\mathcal{I}}$	$-\log q(\mathcal{I} \mathbf{x})$
ReLU	$x = \max(z, 0)$	if $x = 0 : z \sim q(z)$, else : $z = x$	$\mathbb{I}(x = 0)[- \log q(z)]$

Table 7: Summary of some inference surjection layers.

Surjection	Forward	Inverse	$\mathcal{V}(\mathbf{x}, \mathbf{z})$
Rounding	$x \sim p(x z)$ where $x \in [z, z + 1)$	$z = \lfloor x \rfloor$	$\log p(z x)$
Slicing	$\mathbf{x}_1 = \mathbf{z}, \mathbf{x}_2 \sim p(\mathbf{x}_2 \mathbf{z})$	$\mathbf{z} = \mathbf{x}_1$	$\log p(\mathbf{x}_2 \mathbf{z})$
Abs	$s \sim \text{Bern}(\pi(z))$ $x = s \cdot z, s \in \{-1, 1\}$	$s = \text{sign } x$ $z = x $	$\log p(s z)$
Max	$k \sim \text{Cat}(\boldsymbol{\pi}(z))$ $\mathbf{x}_k = z, \mathbf{x}_{-k} \sim p(\mathbf{x}_{-k} z, k)$	$k = \arg \max \mathbf{x}$ $z = \max \mathbf{x}$	$\log p(k z) + \log p(\mathbf{x}_{-k} z, k)$
Sort	$\mathcal{I} \sim \text{Cat}(\boldsymbol{\pi}(z))$ $\mathbf{x} = \mathbf{z}_{\mathcal{I}}$	$\mathcal{I} = \text{argsort } \mathbf{x}$ $\mathbf{z} = \text{sort } \mathbf{x}$	$\log p(\mathcal{I} z)$
ReLU	if $z = 0 : x \sim p(x)$, else : $x = z$	$z = \max(x, 0)$	$\mathbb{I}(z = 0) \log p(x)$

D The Absolute Value Surjection

We here develop the absolute value surjections, both in the generative direction $x = |z|$ and in the inference direction $z = |x|$. We will make use of Dirac delta functions to develop the likelihood contributions, but we could equivalently develop them using Gaussian distributions where $\sigma \rightarrow 0$.

D.1 Generative Direction

Forward and Inverse. We define the forward and inverse transformations as

$$p(x|z) = \sum_{s \in \{-1,1\}} p(x|z, s)p(s|z) = \sum_{s \in \{-1,1\}} \delta(x - sz)\delta_{s, \text{sign}(z)}, \quad (13)$$

$$q(z|x) = \sum_{s \in \{-1,1\}} q(z|x, s)q(s|x) = \sum_{s \in \{-1,1\}} \delta(z - sx)q(s|x), \quad (14)$$

where the forward transformation $p(x|z)$ is fully deterministic and corresponds to $x = |z|$. The inference direction involves two steps, 1) sample the sign s of z conditioned of x , and 2) deterministically map x to $z = sx$. Note that $q(s|x)$ may either be trained as a classifier or fixed to e.g. $q(s|x) = 1/2$. The last choice especially makes sense when $p(z)$ is symmetric.

Likelihood Contribution. We may develop the likelihood contribution by computing

$$\mathcal{V} = \mathbb{E}_{q(z|x, s)q(s|x)} \left[\log \frac{p(x|z, s)p(s|z)}{q(z|x, s)q(s|x)} \right] \quad (15)$$

$$= \mathbb{E}_{\delta(z - sx)q(s|x)} \left[\log \frac{\delta(x - sz)\delta_{s, \text{sign}(z)}}{\delta(z - sx)q(s|x)} \right] \quad (16)$$

$$\approx -\log q(s|x), \quad \text{where } z = sx, \quad s \sim q(s|x). \quad (17)$$

Here, $\delta(x - sz)$ and $\delta(z - sx)$ cancel since $\delta(x - sz) = \delta(z - x/s)|1/s| = \delta(z - sx)$.

D.2 Inference Direction

Forward and Inverse. We define the forward and inverse transformations as

$$p(x|z) = \sum_{s \in \{-1,1\}} p(x|z, s)p(s|z) = \sum_{s \in \{-1,1\}} \delta(x - sz)p(s|z), \quad (18)$$

$$q(z|x) = \sum_{s \in \{-1,1\}} q(z|x, s)q(s|x) = \sum_{s \in \{-1,1\}} \delta(z - sx)\delta_{s, \text{sign}(x)}, \quad (19)$$

where the inverse transformation $q(z|x)$ is fully deterministic and corresponds to $z = |x|$. The generative direction involves two steps, 1) sample the sign of x conditioned of z , and 2) deterministically map z to $x = sz$. Note that $p(s|z)$ may either be trained as a classifier or fixed to e.g. $p(s|z) = 1/2$. The last choice gives rise to an absolute value surjection which may be used to enforce exact symmetry across the origin.

Likelihood Contribution. We may develop the likelihood contribution by computing

$$\mathcal{V} = \mathbb{E}_{q(z|x, s)q(s|x)} \left[\log \frac{p(x|z, s)p(s|z)}{q(z|x, s)q(s|x)} \right] \quad (20)$$

$$= \mathbb{E}_{\delta(z - sx)\delta_{s, \text{sign}(x)}} \left[\log \frac{\delta(x - sz)p(s|z)}{\delta(z - sx)\delta_{s, \text{sign}(x)}} \right] \quad (21)$$

$$= \log p(s|z), \quad \text{where } z = sx = |x|, \quad s = \text{sign}(x). \quad (22)$$

Here, $\delta(x - sz)$ and $\delta(z - sx)$ cancel since $\delta(x - sz) = \delta(z - x/s)|1/s| = \delta(z - sx)$.

E The Maximum Value Surjection

We here develop the maximum value surjections, both in the generative direction $x = \max z$ and in the inference direction $z = \max x$. We will make use of Dirac delta functions to develop the likelihood contributions, but we could equivalently develop them using Gaussian distributions where $\sigma \rightarrow 0$.

E.1 Generative Direction

Forward and Inverse. We define the forward and inverse transformations as

$$p(x|z) = \sum_{k=1}^K p(x|z, k)p(k|z) = \sum_{k=1}^K \delta(x - z_k) \delta_{k, \arg \max(z)}, \quad (23)$$

$$q(z|x) = \sum_{k=1}^K q(z|x, k)q(k|x) = \sum_{k=1}^K \delta(z_k - x) q(z_{-k}|x, k)q(k|x), \quad (24)$$

where k refers to the indices of z , K is the number of elements in z and z_{-k} is z excluding element k . The forward transformation $p(x|z)$ is fully deterministic and corresponds to $x = \max z$. The inference direction involves three steps, 1) sample the index k for the argmax of z conditioned of x , 2) deterministically map x to $z_k = x$, and 3) infer the remaining elements z_{-k} of z . Note that $q(k|x)$ may either be trained as a classifier or fixed to e.g. $q(k|x) = 1/K$.

For q to define a right-inverse of p , we require that $q(z_{-k}|x, k)$ only has support in $(-\infty, x)^{K-1}$ such that z_k will be the maximum value.

Likelihood Contribution. We may develop the likelihood contribution by computing

$$\mathcal{V} = \mathbb{E}_{q(z|x, k)q(k|x)} \left[\log \frac{p(x|z, k)p(k|z)}{q(z|x, k)q(k|x)} \right] \quad (25)$$

$$= \mathbb{E}_{\delta(z_k - x)q(z_{-k}|x, k)q(k|x)} \left[\log \frac{\delta(x - z_k) \delta_{k, \arg \max(z)}}{\delta(z_k - x) q(z_{-k}|x, k) q(k|x)} \right] \quad (26)$$

$$\approx -\log q(k|x) - \log q(z_{-k}|x, k), \quad \text{where } z_k = x, \quad z_{-k} \sim q(z_{-k}|x, k), \quad k \sim q(k|x). \quad (27)$$

E.2 Inference Direction

Forward and Inverse. We define the forward and inverse transformations as

$$p(x|z) = \sum_{k=1}^K p(x|z, k)p(k|z) = \sum_{k=1}^K \delta(x_k - z) p(x_{-k}|z, k)p(k|z), \quad (28)$$

$$q(z|x) = \sum_{k=1}^K q(z|x, k)q(k|x) = \sum_{k=1}^K \delta(z - x_k) \delta_{k, \arg \max(x)}, \quad (29)$$

where k refers to the indices of x , K is the number of elements in x and x_{-k} is x excluding element k . The inverse transformation $q(z|x)$ is fully deterministic and corresponds to $z = \max x$. The inference direction involves three steps, 1) sample the index k for the argmax of x conditioned of z , 2) deterministically map z to $x_k = z$, and 3) infer the remaining elements x_{-k} of x . Note that $p(k|z)$ may either be trained as a classifier or fixed to e.g. $p(k|z) = 1/K$.

For p to define a right-inverse of q , we require that $p(x_{-k}|z, k)$ only has support in $(-\infty, z)^{K-1}$ such that x_k will be the maximum value.

Likelihood Contribution. We may develop the likelihood contribution by computing

$$\mathcal{V} = \mathbb{E}_{q(z|x, k)q(k|x)} \left[\log \frac{p(x|z, k)p(k|z)}{q(z|x, k)q(k|x)} \right] \quad (30)$$

$$= \mathbb{E}_{\delta(z - x_k) \delta_{k, \arg \max(x)}} \left[\log \frac{\delta(x_k - z) p(x_{-k}|z, k) p(k|z)}{\delta(z - x_k) \delta_{k, \arg \max(x)}} \right] \quad (31)$$

$$= \log p(k|z) + \log p(x_{-k}|z, k), \quad \text{where } z = x_k = \max x, \quad k = \arg \max x. \quad (32)$$

F The Sort Surjection

We here develop the sorting surjections, both in the generative direction $x = \text{sort } z$ and in the inference direction $z = \text{sort } x$. We will make use of Dirac delta functions to develop the likelihood contributions, but we could equivalently develop them using Gaussian distributions where $\sigma \rightarrow 0$.

F.1 Generative Direction

Forward and Inverse. We define the forward and inverse transformations as

$$p(\mathbf{x}|\mathbf{z}) = \sum_{\mathcal{I}} p(\mathbf{x}|\mathbf{z}, \mathcal{I})p(\mathcal{I}|\mathbf{z}) = \sum_{\mathcal{I}} \delta(\mathbf{x} - \mathbf{z}_{\mathcal{I}})\delta_{\mathcal{I}, \text{argsort}(\mathbf{z})}, \quad (33)$$

$$q(\mathbf{z}|\mathbf{x}) = \sum_{\mathcal{I}} q(\mathbf{z}|\mathbf{x}, \mathcal{I})q(\mathcal{I}|\mathbf{x}) = \sum_{\mathcal{I}} \delta(\mathbf{z} - \mathbf{x}_{\mathcal{I}^{-1}})q(\mathcal{I}|\mathbf{x}), \quad (34)$$

where \mathcal{I} refers to a set of permutation indices, \mathcal{I}^{-1} refers to the inverse permutation indices and $\mathbf{z}_{\mathcal{I}}$ refers to the elements of \mathbf{z} permuted according to the indices \mathcal{I} . Note that there are $D!$ possible permutations.

The forward transformation $p(\mathbf{x}|\mathbf{z})$ is fully deterministic and corresponds to $\mathbf{x} = \text{sort } \mathbf{z}$. The inference direction involves two steps, 1) sample permutation indices \mathcal{I} conditioned of \mathbf{x} , and 2) deterministically permute \mathbf{x} according to the inverse permutation \mathcal{I}^{-1} to obtain $\mathbf{z} = \mathbf{x}_{\mathcal{I}^{-1}}$. Note that $q(\mathcal{I}|\mathbf{x})$ may either be trained as a classifier or fixed to e.g. $q(\mathcal{I}|\mathbf{x}) = 1/D!$.

Likelihood Contribution. We may develop the likelihood contribution by computing

$$\mathcal{V} = \mathbb{E}_{q(\mathbf{z}|\mathbf{x}, \mathcal{I})q(\mathcal{I}|\mathbf{x})} \left[\log \frac{p(\mathbf{x}|\mathbf{z}, \mathcal{I})p(\mathcal{I}|\mathbf{z})}{q(\mathbf{z}|\mathbf{x}, \mathcal{I})q(\mathcal{I}|\mathbf{x})} \right] \quad (35)$$

$$= \mathbb{E}_{\delta(\mathbf{z} - \mathbf{x}_{\mathcal{I}^{-1}})q(\mathcal{I}|\mathbf{x})} \left[\log \frac{\delta(\mathbf{x} - \mathbf{z}_{\mathcal{I}})\delta_{\mathcal{I}, \text{argsort}(\mathbf{z})}}{\delta(\mathbf{z} - \mathbf{x}_{\mathcal{I}^{-1}})q(\mathcal{I}|\mathbf{x})} \right] \quad (36)$$

$$\approx -\log q(\mathcal{I}|\mathbf{x}), \quad \text{where } \mathcal{I} \sim q(\mathcal{I}|\mathbf{x}). \quad (37)$$

F.2 Inference Direction

Forward and Inverse. We define the forward and inverse transformations as

$$p(\mathbf{x}|\mathbf{z}) = \sum_{\mathcal{I}} p(\mathbf{x}|\mathbf{z}, \mathcal{I})p(\mathcal{I}|\mathbf{z}) = \sum_{\mathcal{I}} \delta(\mathbf{x} - \mathbf{z}_{\mathcal{I}^{-1}})p(\mathcal{I}|\mathbf{z}), \quad (38)$$

$$q(\mathbf{z}|\mathbf{x}) = \sum_{\mathcal{I}} q(\mathbf{z}|\mathbf{x}, \mathcal{I})q(\mathcal{I}|\mathbf{x}) = \sum_{\mathcal{I}} \delta(\mathbf{z} - \mathbf{x}_{\mathcal{I}})\delta_{\mathcal{I}, \text{argsort}(\mathbf{x})}, \quad (39)$$

where \mathcal{I} refers to a set of permutation indices, \mathcal{I}^{-1} refers to the inverse permutation indices and $\mathbf{x}_{\mathcal{I}}$ refers to the elements of \mathbf{x} permuted according to the indices \mathcal{I} . Note that there are $D!$ possible permutations.

The inverse transformation $q(\mathbf{z}|\mathbf{x})$ is fully deterministic and corresponds to $\mathbf{z} = \text{sort } \mathbf{x}$. The generative direction involves two steps, 1) sample permutation indices \mathcal{I} conditioned of \mathbf{z} , and 2) deterministically permute \mathbf{z} according to the inverse permutation \mathcal{I}^{-1} to obtain $\mathbf{x} = \mathbf{z}_{\mathcal{I}^{-1}}$. Note that $p(\mathcal{I}|\mathbf{z})$ may either be trained as a classifier or fixed to e.g. $p(\mathcal{I}|\mathbf{z}) = 1/D!$.

Likelihood Contribution. We may develop the likelihood contribution by computing

$$\mathcal{V} = \mathbb{E}_{q(\mathbf{z}|\mathbf{x}, \mathcal{I})q(\mathcal{I}|\mathbf{x})} \left[\log \frac{p(\mathbf{x}|\mathbf{z}, \mathcal{I})p(\mathcal{I}|\mathbf{z})}{q(\mathbf{z}|\mathbf{x}, \mathcal{I})q(\mathcal{I}|\mathbf{x})} \right] \quad (40)$$

$$= \mathbb{E}_{\delta(\mathbf{z} - \mathbf{x}_{\mathcal{I}})\delta_{\mathcal{I}, \text{argsort}(\mathbf{x})}} \left[\log \frac{\delta(\mathbf{x} - \mathbf{z}_{\mathcal{I}^{-1}})p(\mathcal{I}|\mathbf{z})}{\delta(\mathbf{z} - \mathbf{x}_{\mathcal{I}})\delta_{\mathcal{I}, \text{argsort}(\mathbf{x})}} \right] \quad (41)$$

$$= \log p(\mathcal{I}|\mathbf{z}), \quad \text{where } \mathbf{z} = \mathbf{x}_{\mathcal{I}} = \text{sort } \mathbf{x}, \mathcal{I} = \text{argsort } \mathbf{x}. \quad (42)$$

G The Stochastic Permutation

We here develop the stochastic permutation layer which randomly permutes its input. The inverse pass mirrors the forward pass. Note that stochastic permutation is *not* a surjection, but rather a stochastic transform. We will make use of Dirac delta functions to develop the likelihood contributions, but we could equivalently develop them using Gaussian distributions where $\sigma \rightarrow 0$.

Forward and Inverse. We define the forward and inverse transformations as

$$p(\mathbf{x}|\mathbf{z}) = \sum_{\mathcal{I}} p(\mathbf{x}|\mathbf{z}, \mathcal{I})p(\mathcal{I}) = \sum_{\mathcal{I}} \delta(\mathbf{x} - \mathbf{z}_{\mathcal{I}}) \text{Unif}(\mathcal{I}), \quad (43)$$

$$q(\mathbf{z}|\mathbf{x}) = \sum_{\mathcal{I}} q(\mathbf{z}|\mathbf{x}, \mathcal{I})q(\mathcal{I}) = \sum_{\mathcal{I}} \delta(\mathbf{z} - \mathbf{x}_{\mathcal{I}^{-1}}) \text{Unif}(\mathcal{I}), \quad (44)$$

where \mathcal{I} refers to a set of permutation indices, \mathcal{I}^{-1} refers to the inverse permutation indices and $\mathbf{z}_{\mathcal{I}}$ refers to the elements of \mathbf{z} permuted according to the indices \mathcal{I} . Note that there are $D!$ possible permutations.

The transformation is stochastic and involves the same two steps in both directions: 1) Sample permutation indices \mathcal{I} uniformly at random, and 2) deterministically permute the input according to the samples indices \mathcal{I} .

Likelihood Contribution. We may develop the likelihood contribution by computing

$$\mathcal{V} = \mathbb{E}_{q(\mathbf{z}|\mathbf{x}, \mathcal{I})q(\mathcal{I})} \left[\log \frac{p(\mathbf{x}|\mathbf{z}, \mathcal{I})p(\mathcal{I})}{q(\mathbf{z}|\mathbf{x}, \mathcal{I})q(\mathcal{I})} \right] \quad (45)$$

$$= \mathbb{E}_{\delta(\mathbf{z} - \mathbf{x}_{\mathcal{I}^{-1}}) \text{Unif}(\mathcal{I})} \left[\log \frac{\delta(\mathbf{x} - \mathbf{z}_{\mathcal{I}}) \text{Unif}(\mathcal{I})}{\delta(\mathbf{z} - \mathbf{x}_{\mathcal{I}^{-1}}) \text{Unif}(\mathcal{I})} \right] \quad (46)$$

$$= 0. \quad (47)$$

This layer thus takes the simple form: Both during the forward and inverse passes, shuffle the input uniformly at random. The resulting likelihood contribution is zero.

H The Software Perspective

Normalizing flows provide a powerful modular framework where flexible densities may be specified using a composition of bijective transformations. Each bijection may be implemented as a module contained 3 important components: 1) A forward transformation $x = f(z)$, 2) an inverse transformation $z = f^{-1}(x)$, and 3) a Jacobian determinant $\log |\det \mathbf{J}|$. Several software libraries for normalizing flows have been built using this modular design principle (Dillon et al., 2017; Bingham et al., 2018).

SurVAE flows suggest that such software frameworks may be directly extended since the modules follow the exact same design principles – each module has 3 important components:

1. A forward transformation $\mathcal{Z} \rightarrow \mathcal{X}$.
2. An inverse transformation $\mathcal{X} \rightarrow \mathcal{Z}$.
3. A likelihood contribution $\mathcal{V}(x, z)$.

SurVAE flows allow compositions of not only bijective transformations, but also surjective and stochastic transformations. This allows us to obtain methods such as dequantization (Uria et al., 2014; Theis et al., 2016; Ho et al., 2019), variational data augmentation (Huang et al., 2020; Chen et al., 2020), multi-scale architectures (Dinh et al., 2017) as composable surjective transformations and VAEs (Kingma and Welling, 2014; Rezende et al., 2014) as composable stochastic transformations.

In our code³, we provide a library of SurVAE flows that may serve as a prototype for a more extensive library. In the next subsections, we show some selected code snippets from our library. The code is based on PyTorch (Paszke et al., 2019), but can easily be ported to other frameworks. Note that in the implementation, the `forward` method implements the inverse transformation $\mathcal{X} \rightarrow \mathcal{Z}$ and the likelihood contribution $\mathcal{V}(x, z)$, since this is what is needed during the forward pass of backpropagation used for training.

In Sec. H.1 we show an implementation of a VAE as a stochastic transformation, while in Sec. H.2 and Sec. H.3 we show implementations of dequantization and variational data augmentation as surjective transformations. Finally, in Sec. H.4, we show an example of how to construct an augmented normalizing flow through composition of SurVAE layers.

H.1 VAE

We implement VAEs as a composable stochastic transformation.

```
class VAE(StochasticTransform):
    '''A variational autoencoder layer.'''

    def __init__(self, decoder, encoder):
        super(VAE, self).__init__()
        self.decoder = decoder
        self.encoder = encoder

    def forward(self, x):
        z, log_qz = self.encoder.sample_with_log_prob(context=x)
        log_px = self.decoder.log_prob(x, context=z)
        ldj = log_px - log_qz
        return z, ldj

    def inverse(self, z):
        x = self.decoder.sample(context=z)
        return x
```

³The code is available at https://github.com/didriknielsen/survae_flows

H.2 Dequantization

We implement UniformDequantization, which may be used to convert between discrete and continuous variables, as a generative rounding surjection.

```
class UniformDequantization(Surjection):
    '''A uniform dequantization layer.'''

    def forward(self, x):
        z = x.float() + torch.rand_like(x)
        ldj = torch.zeros(x.shape[0])
        return z, ldj

    def inverse(self, z):
        x = z.floor().long()
        return x
```

H.3 Augmentation

We implement Augment, a generative tensor slicing surjection, which may be used to construct e.g. augmented normalizing flows (Huang et al., 2020; Chen et al., 2020).

```
class Augment(Surjection):
    '''An augmentation layer.'''

    def __init__(self, encoder, split_size):
        super(Augment, self).__init__()
        self.encoder = encoder
        self.split_size = split_size

    def forward(self, x):
        z2, log_qz2 = self.encoder.sample_with_log_prob(context=x)
        z = torch.cat([x, z2], dim=1)
        ldj = -log_qz2
        return z, ldj

    def inverse(self, z):
        x, z2 = torch.split(z, self.split_size, dim=1)
        return x
```

H.4 Example: Augmented Normalizing Flows

We showcase here the simplicity of implementing an augmented normalizing flow using the SurVAE flow framework. In Listing 1, a simple normalizing flow consisting of 2 coupling layers is constructed. In Listing 2, this is extended by adding an Augment surjection, resulting in an augmented flow.

Listing 1: A basic flow.	Listing 2: An augmented flow.
<pre>Flow(base_dist=Normal((2,)), transforms=[CouplingBijection(), Reverse(), CouplingBijection(),])</pre>	<pre>Flow(base_dist=Normal((4,)), transforms=[Augment(Normal((2,)), (2,2)), CouplingBijection(), Reverse(), CouplingBijection(),])</pre>

Using the models in Listing 1 and Listing 2, we compare a standard coupling flow with a simple extension using an additional Augment layer. We use 4 coupling layers instead of 2 and train models both using identical setups 10000 iterations each. Augmented flows have improved capabilities of modelling data with disconnected components. In Fig. 8, we observe that the augmented flows tend to place their mass more out in a more "clean" fashion and thus demonstrate improved ability to model complicated 2D densities.

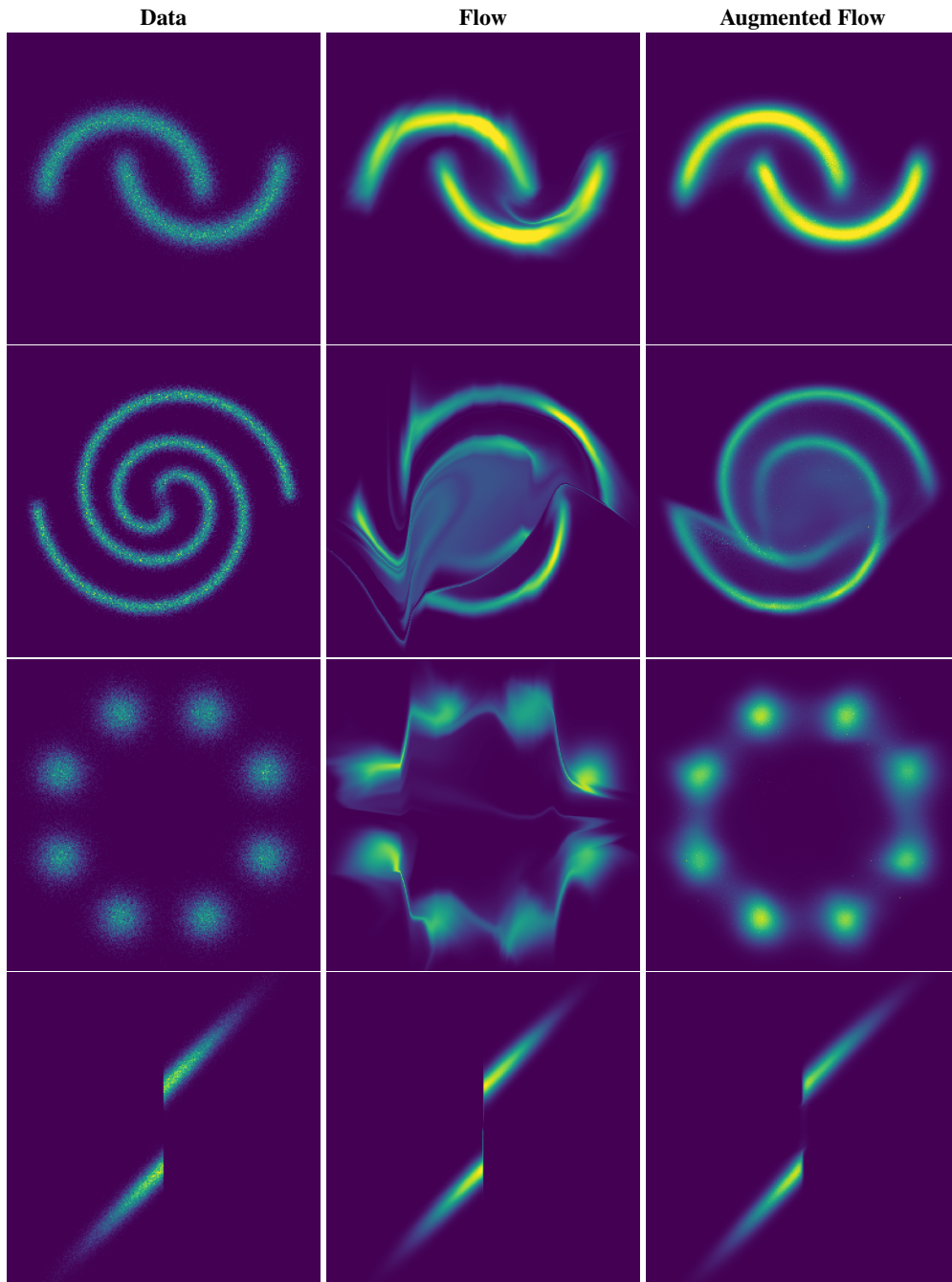


Figure 8: Augmented flows show improved capabilities over flows at modelling 2D densities, especially where there are disconnected components. With SurVAE flows, augmented flows are implemented by adding a single surjective augmentation layer to the flow as shown in Listing 2.

I Experimental Details

We here give more details on the experiments. For further details, see our open-source code⁴.

I.1 Synthetic Data

Data. We used 4 synthetic datasets, checkerboard, corners, gaussians and circles. For each synthetic dataset, 128000 samples were used as a training set and 128000 more samples as a test set. The checkerboard dataset is anti-symmetric, while the 3 others are symmetric.

Training. We used the Adam optimizer (Kingma and Ba, 2015) with a learning rate of 10^{-3} . All models were trained for 10000 iterations (10 epochs) using a batch size of 128.

Baseline. The baseline flow is a composition of 4 affine coupling bijections with the ordering reversed in-between. The coupling layers are parameterized by MLPs with hidden units (200,100) and ReLU activations. The base distribution is a standard Gaussian.

Symmetric AbsFlow. For the symmetric datasets, AbsFlow uses all the same layers as the baseline. In addition, an abs surjection is added, followed by and inverse softplus (gaussians) or logit (checkerboard and corners). In the generative direction, the abs surjection randomly samples the sign with equal probabilities. The extra layers contain no parameters, and the AbsFlow thus have the exact same number of parameters as the baseline.

Anti-Symmetric AbsFlow. For the anti-symmetric dataset, AbsFlow uses only a single abs surjection and a uniform base distribution. In the generative direction, a classifier network learns the probabilities of sampling the sign conditioned on z . This classifier network is, like the coupling layer networks, an MLP with (200,100) hidden units and ReLU activations. In this case, the AbsFlow thus has $\sim 1/4$ the number of parameters.

I.2 Point Cloud Data

Data. We used the SpatialMNIST dataset (Edwards and Storkey, 2017). This dataset was constructed by, for each digit in the MNIST dataset, sampling 50 points according to the normalized pixel intensities. We used the official code⁵ to construct the dataset. We split the dataset into parts of 50000-10000-10000 for training, validation and test (without shuffling). Each data example is a set of 50 2D points which we represent as a tensor of shape (2, 50).

Training. Both models were trained for 500 epochs using a batch size of 128. We used the Adam optimizer (Kingma and Ba, 2015) with an initial learning rate of 10^{-3} . The learning rate was warmed up linearly for 2000 iterations and then decayed by 0.995 every epoch. All models were trained using a single GPU for about 40 hours.

Evaluation. SortFlow allows exact computation of the likelihood, while the PermuteFlow only allows computation of lower bounds. We evaluated PermuteFlow using the IWBO (importance weighted bound) (Burda et al., 2016) using $k = 1000$ importance samples. PermuteFlow obtains an ELBO of -5.32 PPLL and an IWBO of -5.30 PPLL, while SortFlow obtains an exact log-likelihood of -5.53 PPLL.

Hyperparameters. We tuned the dropout rate using the validation set. We considered dropout rates of $\{0.0, 0.05, 0.1, 0.2, 0.3\}$ for both models. We found 0.1 to work best for PermuteFlow, while 0.2 worked best for SortFlow.

PermuteFlow. We used a flow of an initial stochastic permutation layer followed by 32 steps with ActNorm layers (Kingma and Dhariwal, 2018) in-between. Each step consisted of 1) an affine coupling bijection which transforms the first half tensor (1, 50) conditioned on the other half (1, 50), 2) reversing the order along the spatial dimension, 3) an affine coupling bijection which transforms the first half tensor (2, 25) conditioned on the other half (2, 25), 4) a stochastic permutation along the point dimension. Each of the coupling bijections are parameterized by Transformer networks (Vaswani et al., 2017) without positional encoding. The Transformers used 2 blocks, with $d_{\text{model}} = 64$, $d_{\text{ff}} = 256$ and 8 attention heads.

⁴https://github.com/didriknielsen/survae_flows

⁵<https://github.com/conorndurkan/neural-statistician>

SortFlow. This follows the setup of PermuteFlow, with the following changes: 1) The initial stochastic permutation is replaced by a sorting layer. 2) The stochastic permutations in the flow are swapped with fixed permutations (sampled at random once, before training). 3) The Transformers make use of a learned positional encoding, since the sorting layer enforces a canonical ordering of the points.

I.3 Image Data

Data. We used the CIFAR-10, ImageNet 32×32 and ImageNet 64×64 datasets. The CIFAR-10 dataset comes pre-split in 50000 training examples and 10000 test examples. The ImageNet datasets also come pre-split in 1,281,149 training examples and 49,999 validation examples. We use these splits and report results for the test set of CIFAR-10 and the validation sets of ImageNet 32×32 and ImageNet 64×64 .

Training. We used the Adamax optimizer (Kingma and Ba, 2015) with an initial learning rate of 10^{-3} and a batch size of 32. The learning rate was linearly warmed up for 5000 iterations. For CIFAR-10, the models were first trained for 500 epochs with the learning rate decayed by 0.995 every epoch. Next, the models were "cooled down" for an additional 50 epochs with a smaller learning rate of $2 \cdot 10^{-5}$. For the ImageNet datasets, the models were first trained for 25 epochs (ImageNet 32×32) and 20 epochs (ImageNet 64×64) with the learning rate decayed by 0.95 every epoch. Next, the models were "cooled down" for an additional 2 epochs with a smaller learning rate of $5 \cdot 10^{-5}$. The CIFAR-10 and ImageNet 32×32 models were trained on a single GPU for about 2 weeks, while the ImageNet 64×64 models were trained using 4 GPUs for about 3 weeks. We provide pre-trained model checkpoints in our open-source code. Note that data augmentation was applied during training of the CIFAR-10 models, including random flipping and rotations. See code for more details.

Evaluation. The CIFAR-10 models were evaluated using the IWBO (importance weighted bound) (Burda et al., 2016) using $k = 1000$ importance samples. The ImageNet models were evaluated using the ELBO (which corresponds to the IWBO with $k = 1$ importance sample).

Baseline. For CIFAR-10 and ImageNet 32×32 , the flow uses 2 scales with 12 steps/scale. For ImageNet 64×64 , the flow uses 3 scales with 8 steps/scale. Each step consists of an affine coupling bijection (Dinh et al., 2017) and an invertible 1×1 convolution (Kingma and Dhariwal, 2018). All models are trained with variational dequantization (Ho et al., 2019) and an initial squeezing layer (Dinh et al., 2017) to increase the number of channels from 3 to 12. The coupling bijections are parameterized by DenseNets (Huang et al., 2017).

MaxPoolFlow. The MaxPoolFlow uses the exact same setup as the baseline, but replaces the tensor slicing surjection with a max pooling surjection. In the generative direction, we used the simplest possible choice: Each input pixel is equally likely to be copied to any of the pixels in its corresponding 2×2 patch. The remaining 3 elements are sampled such that the copied value remains the largest: They are set equal to this maximum value minus noise from a standard half-normal distribution (i.e. Gaussian distribution with only positive values). We used simple choices containing *no extra parameters* in order to facilitate more fair comparison. Note that the max pooling layer could be potentially be improved by using more sophisticated choices for the distribution for sampling the remaining elements, $p(x_{-k}|z)$, and/or by using a classifier, $p(k|z)$, to predict the indices k .

J Additional Samples

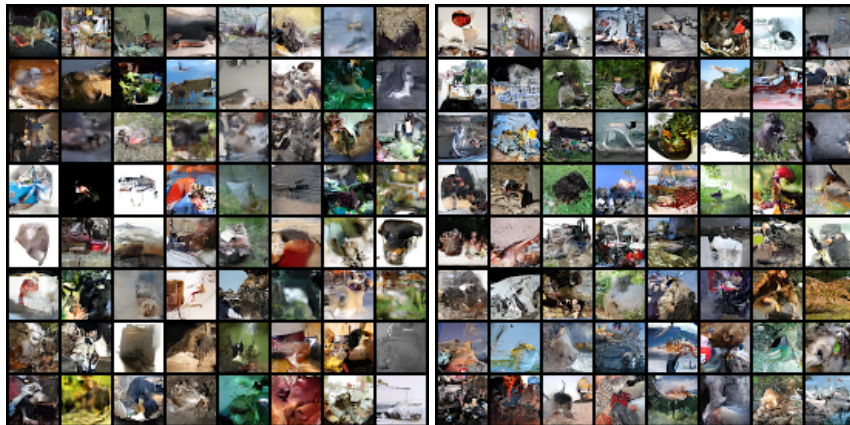
Samples from SurVAE flows trained on CIFAR-10, ImageNet 32×32 and ImageNet 64×64 using either max pooling or tensor slicing for downsampling are shown in Fig. 9, Fig. 10 and Fig. 11, respectively.



(a) Max Pooling.

(b) No Pooling.

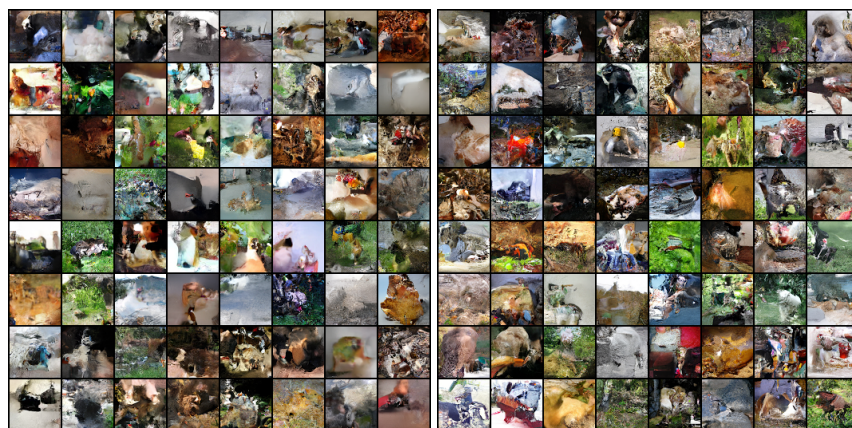
Figure 9: Unconditional samples from SurVAE flows trained on CIFAR-10.



(a) Max Pooling.

(b) No Pooling.

Figure 10: Unconditional samples from SurVAE flows trained on ImageNet 32×32 .



(a) Max Pooling.

(b) No Pooling.

Figure 11: Unconditional samples from SurVAE flows trained on ImageNet 64×64 .