
Improved Variational Inference with Inverse Autoregressive Flow

Diederik P. Kingma Tim Salimans Rafal Jozefowicz Xi Chen
dpmkingma@openai.com tim@openai.com rafal@openai.com peter@openai.com

Ilya Sutskever
ilya@openai.com

Max Welling*
M.Welling@uva.nl

Abstract

The framework of normalizing flows provides a general strategy for flexible variational inference of posteriors over latent variables. We propose a new type of normalizing flow, inverse autoregressive flow (IAF), that, in contrast to earlier published flows, scales well to high-dimensional latent spaces. The proposed flow consists of a chain of invertible transformations, where each transformation is based on an autoregressive neural network. In experiments, we show that IAF significantly improves upon diagonal Gaussian approximate posteriors. In addition, we demonstrate that a novel type of variational autoencoder, coupled with IAF, is competitive with neural autoregressive models in terms of attained log-likelihood on natural images, while allowing significantly faster synthesis.

1 Introduction

Stochastic variational inference (Blei et al., 2012; Hoffman et al., 2013) is a method for scalable posterior inference with large datasets using stochastic gradient ascent. It can be made especially efficient for continuous latent variables through latent-variable reparameterization and inference networks, amortizing the cost, resulting in a highly scalable learning procedure (Kingma and Welling, 2013; Rezende et al., 2014; Salimans et al., 2014). When using neural networks for both the inference network and generative model, this results in class of models called variational autoencoders (Kingma and Welling, 2013) (VAEs). A general strategy for building flexible inference networks, is the framework of *normalizing flows* (Rezende and Mohamed, 2015). In this paper we propose a new type of flow, *inverse autoregressive flow* (IAF), which scales well to high-dimensional latent space.

At the core of our proposed method lie Gaussian autoregressive functions that are normally used for density estimation: functions that take as input a variable with some specified ordering such as multidimensional tensors, and output a mean and standard deviation for each element of the input variable conditioned on the previous elements. Examples of such functions are autoregressive neural density estimators such as RNNs, MADE (Germain et al., 2015), PixelCNN (van den Oord et al., 2016b) or WaveNet (van den Oord et al., 2016a) models. We show that such functions can often be turned into invertible nonlinear transformations of the input, with a simple Jacobian determinant. Since the transformation is flexible and the determinant known, it can be used as a normalizing flow, transforming a tensor with relatively simple known density, into a new tensor with more complicated density that is still cheaply computable. In contrast with most previous work on

*University of Amsterdam, University of California Irvine, and the Canadian Institute for Advanced Research (CIFAR).

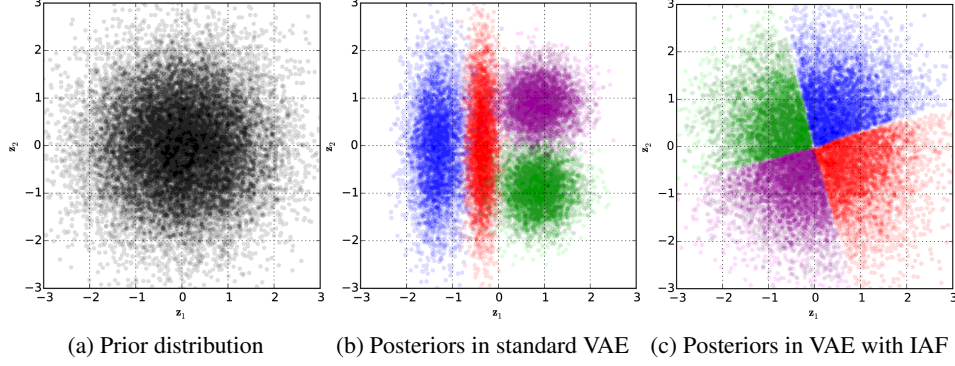


Figure 1: Best viewed in color. We fitted a variational auto-encoder (VAE) with a spherical Gaussian prior, and with factorized Gaussian posteriors **(b)** or inverse autoregressive flow (IAF) posteriors **(c)** to a toy dataset with four datapoints. Each colored cluster corresponds to the posterior distribution of one datapoint. IAF greatly improves the flexibility of the posterior distributions, and allows for a much better fit between the posteriors and the prior.

improving inference models including previously used normalizing flows, this transformation is well suited to high-dimensional tensor variables, such as spatio-temporally organized variables.

We demonstrate this method by improving inference networks of deep variational auto-encoders. In particular, we train deep variational auto-encoders with latent variables at multiple levels of the hierarchy, where each stochastic variable is a three-dimensional tensor (a stack of featuremaps), and demonstrate improved performance.

2 Variational Inference and Learning

Let \mathbf{x} be a (set of) observed variable(s), \mathbf{z} a (set of) latent variable(s) and let $p(\mathbf{x}, \mathbf{z})$ be the parametric model of their joint distribution, called the *generative model* defined over the variables. Given a dataset $\mathbf{X} = \{\mathbf{x}^1, \dots, \mathbf{x}^N\}$ we typically wish to perform maximum marginal likelihood learning of its parameters, i.e. to maximize

$$\log p(\mathbf{X}) = \sum_{i=1}^N \log p(\mathbf{x}^{(i)}), \quad (1)$$

but in general this marginal likelihood is intractable to compute or differentiate directly for flexible generative models, e.g. when components of the generative model are parameterized by neural networks. A solution is to introduce $q(\mathbf{z}|\mathbf{x})$, a parametric *inference model* defined over the latent variables, and optimize the *variational lower bound* on the marginal log-likelihood of each observation \mathbf{x} :

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z}|\mathbf{x})] = \mathcal{L}(\mathbf{x}; \boldsymbol{\theta}) \quad (2)$$

where $\boldsymbol{\theta}$ indicates the parameters of p and q models. Keeping in mind that Kullback-Leibler divergences $D_{KL}(\cdot)$ are non-negative, it's clear that $\mathcal{L}(\mathbf{x}; \boldsymbol{\theta})$ is a lower bound on $\log p(\mathbf{x})$ since it can be written as follows):

$$\mathcal{L}(\mathbf{x}; \boldsymbol{\theta}) = \log p(\mathbf{x}) - D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})) \quad (3)$$

There are various ways to optimize the lower bound $\mathcal{L}(\mathbf{x}; \boldsymbol{\theta})$; for continuous \mathbf{z} it can be done efficiently through a re-parameterization of $q(\mathbf{z}|\mathbf{x})$, see e.g. (Kingma and Welling, 2013; Rezende et al., 2014).

As can be seen from equation (3), maximizing $\mathcal{L}(\mathbf{x}; \boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$ will concurrently maximize $\log p(\mathbf{x})$ and minimize $D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$. The closer $D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$ is to 0, the closer $\mathcal{L}(\mathbf{x}; \boldsymbol{\theta})$ will be to $\log p(\mathbf{x})$, and the better an approximation our optimization objective $\mathcal{L}(\mathbf{x}; \boldsymbol{\theta})$ is to our true objective $\log p(\mathbf{x})$. Also, minimization of $D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$ can be a goal in itself, if we're interested in using $q(\mathbf{z}|\mathbf{x})$ for inference after optimization. In any case, the divergence $D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$ is a function of our parameters through both the inference model and the generative model, and increasing the flexibility of either is generally helpful towards our objective.

Note that in models with multiple latent variables, the inference model is typically factorized into partial inference models with some ordering; e.g. $q(\mathbf{z}_a, \mathbf{z}_b | \mathbf{x}) = q(\mathbf{z}_a | \mathbf{x})q(\mathbf{z}_b | \mathbf{z}_a, \mathbf{x})$. We'll write $q(\mathbf{z} | \mathbf{x}, \mathbf{c})$ to denote such partial inference models, conditioned on both the data \mathbf{x} and a further context \mathbf{c} which includes the previous latent variables according to the ordering.

2.1 Requirements for Computational Tractability

Requirements for the inference model, in order to be able to efficiently optimize the bound, are that it is (1) computationally efficient to compute and differentiate its probability density $q(\mathbf{z} | \mathbf{x})$, and (2) computationally efficient to sample from, since both these operations need to be performed for each datapoint in a minibatch at every iteration of optimization. If \mathbf{z} is high-dimensional and we want to make efficient use of parallel computational resources like GPUs, then parallelizability of these operations across dimensions of \mathbf{z} is a large factor towards efficiency. This requirement restricts the class of approximate posteriors $q(\mathbf{z} | \mathbf{x})$ that are practical to use. In practice this often leads to the use of diagonal posteriors, e.g. $q(\mathbf{z} | \mathbf{x}) \sim \mathcal{N}(\boldsymbol{\mu}(\mathbf{x}), \boldsymbol{\sigma}^2(\mathbf{x}))$, where $\boldsymbol{\mu}(\mathbf{x})$ and $\boldsymbol{\sigma}(\mathbf{x})$ are often nonlinear functions parameterized by neural networks. However, as explained above, we also need the density $q(\mathbf{z} | \mathbf{x})$ to be sufficiently flexible to match the true posterior $p(\mathbf{z} | \mathbf{x})$.

2.2 Normalizing Flow

Normalizing Flow (NF), introduced by (Rezende and Mohamed, 2015) in the context of stochastic gradient variational inference, is a powerful framework for building flexible posterior distributions through an iterative procedure. The general idea is to start off with an initial random variable with a relatively simple distribution with known (and computationally cheap) probability density function, and then apply a chain of invertible parameterized transformations \mathbf{f}_t , such that the last iterate \mathbf{z}_T has a more flexible distribution²:

$$\mathbf{z}_0 \sim q(\mathbf{z}_0 | \mathbf{x}), \quad \mathbf{z}_t = \mathbf{f}_t(\mathbf{z}_{t-1}, \mathbf{x}) \quad \forall t = 1 \dots T \quad (4)$$

As long as the Jacobian determinant of each of the transformations \mathbf{f}_t can be computed, we can still compute the probability density function of the last iterate:

$$\log q(\mathbf{z}_T | \mathbf{x}) = \log q(\mathbf{z}_0 | \mathbf{x}) - \sum_{t=1}^T \log \det \left| \frac{d\mathbf{z}_t}{d\mathbf{z}_{t-1}} \right| \quad (5)$$

However, (Rezende and Mohamed, 2015) experiment with only a very limited family of such invertible transformation with known Jacobian determinant, namely:

$$\mathbf{f}_t(\mathbf{z}_{t-1}) = \mathbf{z}_{t-1} + \mathbf{u}h(\mathbf{w}^T \mathbf{z}_{t-1} + b) \quad (6)$$

where \mathbf{u} and \mathbf{w} are vectors, \mathbf{w}^T is \mathbf{w} transposed, b is a scalar and $h(\cdot)$ is a nonlinearity, such that $\mathbf{u}h(\mathbf{w}^T \mathbf{z}_{t-1} + b)$ can be interpreted as a MLP with a bottleneck hidden layer with a single unit. Since information goes through the single bottleneck, a long chain of transformations is required to capture high-dimensional dependencies.

3 Inverse Autoregressive Transformations

In order to find a type of normalizing flow that scales well to high-dimensional space, we consider Gaussian versions of autoregressive autoencoders such as MADE (Germain et al., 2015) and the PixelCNN (van den Oord et al., 2016b). Let \mathbf{y} be a variable modeled by such a model, with some chosen ordering on its elements $\mathbf{y} = \{y_i\}_{i=1}^D$. We will use $[\boldsymbol{\mu}(\mathbf{y}), \boldsymbol{\sigma}(\mathbf{y})]$ to denote the function of the vector \mathbf{y} , to the vectors $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$. Due to the autoregressive structure, the Jacobian is lower triangular with zeros on the diagonal: $\partial[\mu_i, \sigma_i] / \partial \mathbf{y}_j = [0, 0]$ for $j \geq i$. The elements $[\mu_i(\mathbf{y}_{1:i-1}), \sigma_i(\mathbf{y}_{1:i-1})]$ are the predicted mean and standard deviation of the i -th element of \mathbf{y} , which are functions of only the previous elements in \mathbf{y} .

Sampling from such a model is a sequential transformation from a noise vector $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$ to the corresponding vector \mathbf{y} : $y_0 = \mu_0 + \sigma_0 \odot \boldsymbol{\epsilon}_0$, and for $i > 0$, $y_i = \mu_i(\mathbf{y}_{1:i-1}) + \sigma_i(\mathbf{y}_{1:i-1}) \cdot \epsilon_i$. The

²where \mathbf{x} is the context, such as the value of the datapoint. In case of models with multiple levels of latent variables, the context also includes the value of the previously sampled latent variables.

Algorithm 1: Pseudo-code of an approximate posterior with Inverse Autoregressive Flow (IAF)

Data:

\mathbf{x} : a datapoint, and optionally other conditioning information
 θ : neural network parameters
EncoderNN($\mathbf{x}; \theta$): encoder neural network, with additional output \mathbf{h}
AutoregressiveNN[*]($\mathbf{z}, \mathbf{h}; \theta$): autoregressive neural networks, with additional input \mathbf{h}
sum(.): sum over vector elements
sigmoid(.): element-wise sigmoid function

Result:

\mathbf{z} : a random sample from $q(\mathbf{z}|\mathbf{x})$, the approximate posterior distribution
 l : the scalar value of $\log q(\mathbf{z}|\mathbf{x})$, evaluated at sample ' \mathbf{z} '

```
[ $\mu, \sigma, \mathbf{h}$ ]  $\leftarrow$  EncoderNN( $\mathbf{x}; \theta$ )
 $\epsilon \sim \mathcal{N}(0, I)$ 
 $\mathbf{z} \leftarrow \sigma \odot \epsilon + \mu$ 
 $l \leftarrow -\text{sum}(\log \sigma + \frac{1}{2}\epsilon^2 + \frac{1}{2}\log(2\pi))$ 
for  $t \leftarrow 1$  to  $T$  do
  [ $\mathbf{m}, \mathbf{s}$ ]  $\leftarrow$  AutoregressiveNN[ $t$ ]( $\mathbf{z}, \mathbf{h}; \theta$ )
   $\sigma \leftarrow \text{sigmoid}(\mathbf{s})$ 
   $\mathbf{z} \leftarrow \sigma \odot \mathbf{z} + (1 - \sigma) \odot \mathbf{m}$ 
   $l \leftarrow l - \text{sum}(\log \sigma)$ 
end
```

computation involved in this transformation is clearly proportional to the dimensionality D . Since variational inference requires sampling from the posterior, such models are not interesting for direct use in such applications. However, the inverse transformation is interesting for normalizing flows, as we will show. As long as we have $\sigma_i > 0$ for all i , the sampling transformation above is a one-to-one transformation, and can be inverted: $\epsilon_i = \frac{y_i - \mu_i(\mathbf{y}_{1:i-1})}{\sigma_i(\mathbf{y}_{1:i-1})}$.

We make two key observations, important for normalizing flows. The first is that this inverse transformation can be parallelized, since (in case of autoregressive autoencoders) computations of the individual elements ϵ_i do not depend on each other. The vectorized transformation is:

$$\epsilon = (\mathbf{y} - \mu(\mathbf{y})) / \sigma(\mathbf{y}) \quad (7)$$

where the subtraction and division are elementwise.

The second key observation, is that this inverse autoregressive operation has a simple Jacobian determinant. Note that due to the autoregressive structure, $\partial[\mu_i, \sigma_i] / \partial y_j = [0, 0]$ for $j \geq i$. As a result, the transformation has a lower triangular Jacobian ($\partial \epsilon_i / \partial y_j = 0$ for $j > i$), with a simple diagonal: $\partial \epsilon_i / \partial y_i = \sigma_i$. The determinant of a lower triangular matrix equals the product of the diagonal terms. As a result, the log-determinant of the Jacobian of the transformation is remarkably simple and straightforward to compute:

$$\log \det \left| \frac{d\epsilon}{d\mathbf{y}} \right| = \sum_{i=1}^D -\log \sigma_i(\mathbf{y}) \quad (8)$$

The combination of model flexibility, parallelizability across dimensions, and simple log-determinant, make this transformation interesting for use as a normalizing flow over high-dimensional latent space.

4 Inverse Autoregressive Flow (IAF)

We propose a new type normalizing flow (eq. (5)), based on transformations that are equivalent to the inverse autoregressive transformation of eq. (7) up to reparameterization. See algorithm 1 for pseudo-code of an approximate posterior with the proposed flow. We let an initial encoder neural network output μ_0 and σ_0 , in addition to an extra output \mathbf{h} , which serves as an additional input to each subsequent step in the flow. We draw a random sample $\epsilon \sim \mathcal{N}(0, I)$, and initialize the chain with:

$$\mathbf{z}_0 = \mu_0 + \sigma_0 \odot \epsilon \quad (9)$$

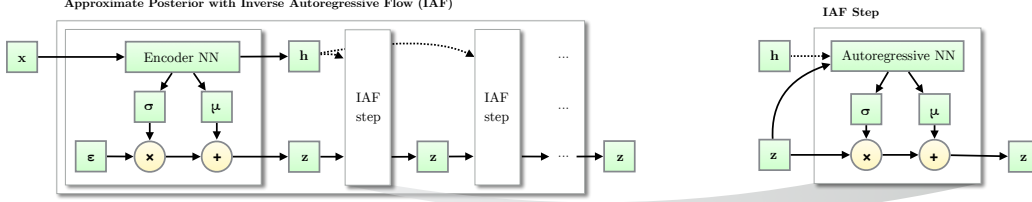


Figure 2: Like other normalizing flows, drawing samples from an approximate posterior with Inverse Autoregressive Flow (IAF) consists of an initial sample \mathbf{z} drawn from a simple distribution, such as a Gaussian with diagonal covariance, followed by a chain of nonlinear invertible transformations of \mathbf{z} , each with a simple Jacobian determinants.

The flow consists of a chain of T of the following transformations:

$$\mathbf{z}_t = \boldsymbol{\mu}_t + \boldsymbol{\sigma}_t \odot \mathbf{z}_{t-1} \quad (10)$$

where at the t -th step of the flow, we use a different autoregressive neural network with inputs \mathbf{z}_{t-1} and \mathbf{h} , and outputs $\boldsymbol{\mu}_t$ and $\boldsymbol{\sigma}_t$. The neural network is structured to be autoregressive w.r.t. \mathbf{z}_{t-1} , such that for any choice of its parameters, the Jacobians $\frac{d\boldsymbol{\mu}_t}{d\mathbf{z}_{t-1}}$ and $\frac{d\boldsymbol{\sigma}_t}{d\mathbf{z}_{t-1}}$ are triangular with zeros on the diagonal. As a result, $\frac{d\mathbf{z}_t}{d\mathbf{z}_{t-1}}$ is triangular with $\boldsymbol{\sigma}_t$ on the diagonal, with determinant $\prod_{i=1}^D \sigma_{t,i}$. (Note that the Jacobian w.r.t. \mathbf{h} does not have constraints.) Following eq. (5), the density under the final iterate is:

$$\log q(\mathbf{z}_T | \mathbf{x}) = - \sum_{i=1}^D \left(\frac{1}{2} \epsilon_i^2 + \frac{1}{2} \log(2\pi) + \sum_{t=0}^T \log \sigma_{t,i} \right) \quad (11)$$

The flexibility of the distribution of the final iterate \mathbf{z}_T , and its ability to closely fit to the true posterior, increases with the expressivity of the autoregressive models and the depth of the chain. See figure 2 for an illustration.

A numerically stable version, inspired by the LSTM-type update, is where we let the autoregressive network output $[\mathbf{m}_t, \mathbf{s}_t]$, two unconstrained real-valued vectors:

$$[\mathbf{m}_t, \mathbf{s}_t] \leftarrow \text{AutoregressiveNN}[t](\mathbf{z}_t, \mathbf{h}; \boldsymbol{\theta}) \quad (12)$$

and compute \mathbf{z}_t as:

$$\boldsymbol{\sigma}_t = \text{sigmoid}(\mathbf{s}_t) \quad (13)$$

$$\mathbf{z}_t = \boldsymbol{\sigma}_t \odot \mathbf{z}_{t-1} + (1 - \boldsymbol{\sigma}_t) \odot \mathbf{m}_t \quad (14)$$

This version is shown in algorithm 1. Note that this is just a particular version of the update of eq. (10), so the simple computation of the final log-density of eq. (11) still applies.

We found it beneficial for results to parameterize or initialize the parameters of each $\text{AutoregressiveNN}[t]$ such that its outputs \mathbf{s}_t are, before optimization, sufficiently positive, such as close to +1 or +2. This leads to an initial behaviour that updates \mathbf{z} only slightly with each step of IAF. Such a parameterization is known as a 'forget gate bias' in LSTMs, as investigated by Jozefowicz et al. (2015).

Perhaps the simplest special version of IAF is one with a simple step, and a linear autoregressive model. This transforms a Gaussian variable with diagonal covariance, to one with linear dependencies, i.e. a Gaussian distribution with full covariance. See appendix A for an explanation.

Autoregressive neural networks form a rich family of nonlinear transformations for IAF. For non-convolutional models, we use the family of masked autoregressive networks introduced in (Germain et al., 2015) for the autoregressive neural networks. For CIFAR-10 experiments, which benefits more from scaling to high dimensional latent space, we use the family of convolutional autoregressive autoencoders introduced by (van den Oord et al., 2016b,c).

We found that results improved when reversing the ordering of the variables after each step in the IAF chain. This is a volume-preserving transformation, so the simple form of eq. (11) remains unchanged.

5 Related work

Inverse autoregressive flow (IAF) is a member of the family of normalizing flows, first discussed in (Rezende and Mohamed, 2015) in the context of stochastic variational inference. In (Rezende and Mohamed, 2015) two specific types of flows are introduced: planar flows and radial flows. These flows are shown to be effective to problems relatively low-dimensional latent space (at most a few hundred dimensions). It is not clear, however, how to scale such flows to much higher-dimensional latent spaces, such as latent spaces of generative models of larger images, and how planar and radial flows can leverage the topology of latent space, as is possible with IAF. Volume-conserving neural architectures were first presented in (Deco and Brauer, 1995), as a form of nonlinear independent component analysis.

Another type of normalizing flow, introduced by (Dinh et al., 2014) (*NICE*), uses similar transformations as IAF. In contrast with IAF, this type of transformations updates only half of the latent variables $\mathbf{z}_{1:D/2}$ per step, adding a vector $f(\mathbf{z}_{D/2+1:D})$ which is a neural network based function of the remaining latent variables $\mathbf{z}_{D/2+1:D}$. Such large blocks have the advantage of computationally cheap inverse transformation, and the disadvantage of typically requiring longer chains. In experiments, (Rezende and Mohamed, 2015) found that this type of transformation is generally less powerful than other types of normalizing flow, in experiments with a low-dimensional latent space. Concurrently to our work, NICE was extended to high-dimensional spaces in (Dinh et al., 2016) (*Real NVP*). An empirical comparison would be an interesting subject of future research.

A potentially powerful transformation is the *Hamiltonian flow* used in Hamiltonian Variational Inference (Salimans et al., 2014). Here, a transformation is generated by simulating the flow of a Hamiltonian system consisting of the latent variables \mathbf{z} , and a set of auxiliary momentum variables. This type of transformation has the additional benefit that it is guided by the exact posterior distribution, and that it leaves this distribution invariant for small step sizes. Such a transformation could thus take us arbitrarily close to the exact posterior distribution if we can apply it for a sufficient number of times. In practice, however, Hamiltonian Variational Inference is very demanding computationally. Also, it requires an auxiliary variational bound to account for the auxiliary variables, which can impede progress if the bound is not sufficiently tight.

An alternative method for increasing the flexibility of the variational inference, is the introduction of auxiliary latent variables (Salimans et al., 2014; Ranganath et al., 2015; Tran et al., 2015) and corresponding auxiliary inference models. Latent variable models with multiple layers of stochastic variables, such as the one used in our experiments, are often equivalent to such auxiliary-variable methods. We combine deep latent variable models with IAF in our experiments, benefiting from both techniques.

6 Experiments

We empirically evaluate IAF by applying the idea to improve variational autoencoders. Please see appendix C for details on the architectures of the generative model and inference models. Code for reproducing key empirical results is available online³.

6.1 MNIST

In this experiment we follow a similar implementation of the convolutional VAE as in (Salimans et al., 2014) with ResNet (He et al., 2015) blocks. A single layer of Gaussian stochastic units of dimension 32 is used. To investigate how the expressiveness of approximate posterior affects performance, we report results of different IAF posteriors with varying degrees of expressiveness. We use a 2-layer MADE (Germain et al., 2015) to implement one IAF transformation, and we stack multiple IAF transformations with ordering reversed between every other transformation.

Results: Table 1 shows results on MNIST for these types of posteriors. Results indicate that as approximate posterior becomes more expressive, generative modeling performance becomes better. Also worth noting is that an expressive approximate posterior also tightens variational lower bounds as expected, making the gap between variational lower bounds and marginal likelihoods smaller. By making IAF deep and wide enough, we can achieve best published log-likelihood on dynamically

³<https://github.com/openai/iaf>

Table 1: Generative modeling results on the dynamically sampled binarized MNIST version used in previous publications (Burda et al., 2015). Shown are averages; the number between brackets are standard deviations across 5 optimization runs. The right column shows an importance sampled estimate of the marginal likelihood for each model with 128 samples. Best previous results are reproduced in the first segment: [1]: (Salimans et al., 2014) [2]: (Burda et al., 2015) [3]: (Kaae Sønderby et al., 2016) [4]: (Tran et al., 2015)

Model	VLB	$\log p(\mathbf{x}) \approx$
Convolutional VAE + HVI [1]	-83.49	-81.94
DLGM 2hl + IWAE [2]		-82.90
LVAE [3]		-81.74
DRAW + VGP [4]	-79.88	
Diagonal covariance	-84.08 (± 0.10)	-81.08 (± 0.08)
IAF (Depth = 2, Width = 320)	-82.02 (± 0.08)	-79.77 (± 0.06)
IAF (Depth = 2, Width = 1920)	-81.17 (± 0.08)	-79.30 (± 0.08)
IAF (Depth = 4, Width = 1920)	-80.93 (± 0.09)	-79.17 (± 0.08)
IAF (Depth = 8, Width = 1920)	-80.80 (± 0.07)	-79.10 (± 0.07)

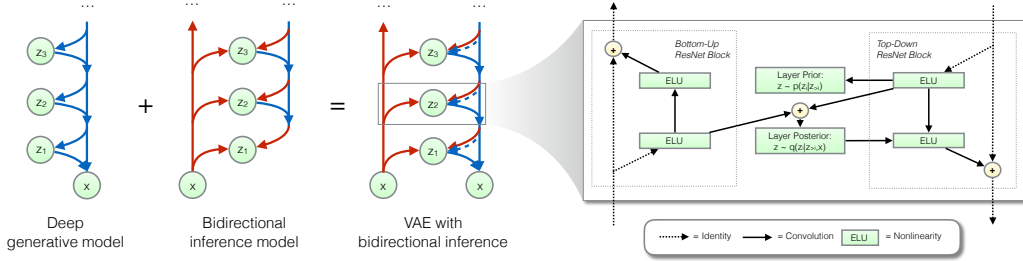


Figure 3: Overview of our ResNet VAE with bidirectional inference. The posterior of each layer is parameterized by its own IAF.

binarized MNIST: **-79.10**. On Hugo Larochelle’s statically binarized MNIST, our VAE with deep IAF achieves a log-likelihood of **-79.88**, which is slightly worse than the best reported result, **-79.2**, using the PixelCNN (van den Oord et al., 2016b).

6.2 CIFAR-10

We also evaluated IAF on the CIFAR-10 dataset of natural images. Natural images contain a much greater variety of patterns and structure than MNIST images; in order to capture this structure well, we experiment with a novel architecture, ResNet VAE, with many layers of stochastic variables, and based on residual convolutional networks (ResNets) (He et al., 2015, 2016). Please see our appendix for details.

Log-likelihood. See table 2 for a comparison to previously reported results. Our architecture with IAF achieves **3.11 bits per dimension**, which is better than other published latent-variable models, and almost on par with the best reported result using the PixelCNN. See the appendix for more experimental results. We suspect that the results can be further improved with more steps of flow, which we leave to future work.

Synthesis speed. Sampling took about **0.05 seconds/image** with the ResNet VAE model, versus **52.0 seconds/image** with the PixelCNN model, on a NVIDIA Titan X GPU. We sampled from the PixelCNN naively by sequentially generating a pixel at a time, using the full generative model at each iteration. With custom code that only evaluates the relevant part of the network, PixelCNN sampling could be sped up significantly; however the speedup will be limited on parallel hardware due to the

Table 2: Our results with ResNet VAEs on CIFAR-10 images, compared to earlier results, in *average number of bits per data dimension* on the test set. The number for convolutional DRAW is an upper bound, while the ResNet VAE log-likelihood was estimated using importance sampling.

Method	bits/dim \leq
<i>Results with tractable likelihood models:</i>	
Uniform distribution (van den Oord et al., 2016b)	8.00
Multivariate Gaussian (van den Oord et al., 2016b)	4.70
NICE (Dinh et al., 2014)	4.48
Deep GMMs (van den Oord and Schrauwen, 2014)	4.00
Real NVP (Dinh et al., 2016)	3.49
PixelRNN (van den Oord et al., 2016b)	3.00
Gated PixelCNN (van den Oord et al., 2016c)	3.03
<i>Results with variationally trained latent-variable models:</i>	
Deep Diffusion (Sohl-Dickstein et al., 2015)	5.40
Convolutional DRAW (Gregor et al., 2016)	3.58
ResNet VAE with IAF (Ours)	3.11

sequential nature of the sampling operation. Efficient sampling from the ResNet VAE is a parallel computation that does not require custom code.

7 Conclusion

We presented *inverse autoregressive flow* (IAF), a new type of normalizing flow that scales well to high-dimensional latent space. In experiments we demonstrated that autoregressive flow leads to significant performance gains compared to similar models with factorized Gaussian approximate posteriors, and we report close to state-of-the-art log-likelihood results on CIFAR-10, for a model that allows much faster sampling.

Acknowledgements

We thank Jascha Sohl-Dickstein, Karol Gregor, and many others at Google Deepmind for interesting discussions. We thank Harri Valpola for referring us to Gustavo Deco’s relevant pioneering work on a form of inverse autoregressive flow applied to nonlinear independent component analysis.

References

- Blei, D. M., Jordan, M. I., and Paisley, J. W. (2012). Variational Bayesian inference with Stochastic Search. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 1367–1374.
- Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., and Bengio, S. (2015). Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*.
- Burda, Y., Grosse, R., and Salakhutdinov, R. (2015). Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*.
- Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by Exponential Linear Units (ELUs). *arXiv preprint arXiv:1511.07289*.
- Deco, G. and Brauer, W. (1995). Higher order statistical decorrelation without information loss. *Advances in Neural Information Processing Systems*, pages 247–254.
- Dinh, L., Krueger, D., and Bengio, Y. (2014). Nice: non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*.
- Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016). Density estimation using Real NVP. *arXiv preprint arXiv:1605.08803*.

- Germain, M., Gregor, K., Murray, I., and Larochelle, H. (2015). Made: Masked autoencoder for distribution estimation. *arXiv preprint arXiv:1502.03509*.
- Gregor, K., Besse, F., Rezende, D. J., Danihelka, I., and Wierstra, D. (2016). Towards conceptual compression. *arXiv preprint arXiv:1604.08772*.
- Gregor, K., Mnih, A., and Wierstra, D. (2013). Deep AutoRegressive Networks. *arXiv preprint arXiv:1310.8499*.
- He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016). Identity mappings in deep residual networks. *arXiv preprint arXiv:1603.05027*.
- Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347.
- Jozefowicz, R., Zaremba, W., and Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2342–2350.
- Kaae Sønderby, C., Raiko, T., Maaløe, L., Kaae Sønderby, S., and Winther, O. (2016). How to train deep variational autoencoders and probabilistic ladder networks. *arXiv preprint arXiv:1602.02282*.
- Kingma, D. P. and Welling, M. (2013). Auto-encoding variational Bayes. *Proceedings of the 2nd International Conference on Learning Representations*.
- Ranganath, R., Tran, D., and Blei, D. M. (2015). Hierarchical variational models. *arXiv preprint arXiv:1511.02386*.
- Rezende, D. and Mohamed, S. (2015). Variational inference with normalizing flows. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 1530–1538.
- Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1278–1286.
- Salimans, T. (2016). A structured variational auto-encoder for learning deep hierarchies of sparse features. *arXiv preprint arXiv:1602.08734*.
- Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv preprint arXiv:1602.07868*.
- Salimans, T., Kingma, D. P., and Welling, M. (2014). Markov chain Monte Carlo and variational inference: Bridging the gap. *arXiv preprint arXiv:1410.6460*.
- Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. *arXiv preprint arXiv:1503.03585*.
- Tran, D., Ranganath, R., and Blei, D. M. (2015). Variational gaussian process. *arXiv preprint arXiv:1511.06499*.
- van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016a). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.
- van den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. (2016b). Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*.
- van den Oord, A., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., and Kavukcuoglu, K. (2016c). Conditional image generation with pixelcnn decoders. *arXiv preprint arXiv:1606.05328*.
- van den Oord, A. and Schrauwen, B. (2014). Factoring variations in natural images with deep gaussian mixture models. In *Advances in Neural Information Processing Systems*, pages 3518–3526.
- Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. *arXiv preprint arXiv:1605.07146*.
- Zeiler, M. D., Krishnan, D., Taylor, G. W., and Fergus, R. (2010). Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2528–2535. IEEE.

A Linear IAF

Perhaps the simplest special case of IAF is the transformation of a Gaussian variable with diagonal covariance to one with linear dependencies.

Any full-covariance multivariate Gaussian distribution with mean \mathbf{m} and covariance matrix \mathbf{C} can be expressed as an autoregressive model with

$$\begin{aligned} y_i &= \mu_i(\mathbf{y}_{1:i-1}) + \sigma_i(\mathbf{y}_{1:i-1}) \cdot \epsilon_i, \text{ with} \\ \mu_i(\mathbf{y}_{1:i-1}) &= m_i + \mathbf{C}[i, 1:i-1] \mathbf{C}[1:i-1, 1:i-1]^{-1} (\mathbf{y}_{1:i-1} - \mathbf{m}_{1:i-1}), \text{ and} \\ \sigma_i(\mathbf{y}_{1:i-1}) &= \mathbf{C}[i, i] - \mathbf{C}[i, 1:i-1] \mathbf{C}[1:i-1, 1:i-1]^{-1} \mathbf{C}[1:i-1, i]. \end{aligned}$$

Inverting the autoregressive model then gives $\epsilon = (\mathbf{y} - \boldsymbol{\mu}(\mathbf{y})) / \boldsymbol{\sigma}(\mathbf{y}) = \mathbf{L}(\mathbf{y} - \mathbf{m})$ with \mathbf{L} the inverse Cholesky factorization of the covariance matrix \mathbf{C} .

By making $\mathbf{L}(\mathbf{x})$ and $\mathbf{m}(\mathbf{x})$ part of our variational encoder we can then use this inverse flow to form a posterior approximation. In experiments, we do this by starting our with a fully-factorized Gaussian approximate posterior as in e.g. (Kingma and Welling, 2013): $\mathbf{y} = \boldsymbol{\mu}(\mathbf{x}) + \boldsymbol{\sigma}(\mathbf{x}) \odot \epsilon$ where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$, and where $\boldsymbol{\mu}(\mathbf{x})$ and $\boldsymbol{\sigma}(\mathbf{x})$ are vectors produced by our inference network. We then let the inference network produce an extra output $\mathbf{L}(\mathbf{x})$, the lower triangular inverse Cholesky matrix, which we then use to update the approximation. With this setup, the problem is overparameterized, so we define the mean vector \mathbf{m} above to be the zero vector, and we restrict \mathbf{L} to have ones on the diagonal. One step of linear IAF then turns the fully-factorized distribution of \mathbf{y} into an arbitrary multivariate Gaussian distribution: $\mathbf{z} = \mathbf{L}(\mathbf{x}) \cdot \mathbf{y}$. This results in a simple and computationally efficient posterior approximation, with scalar density function given by $q(\mathbf{z}|\mathbf{x}) = q(\mathbf{y}|\mathbf{x})$. By optimizing the variational lower bound we then fit this conditional multivariate Gaussian approximation to the true posterior distribution.

B MNIST

In MNIST experiment we follow a similar implementation of the convolutional VAE as in (Salimans et al., 2014) with ResNet (He et al., 2015) blocks. A single layer of Gaussian stochastic units of dimension 32 is used. The inference network has three 2-strided resnet blocks with 3x3 filters and [16,32,32] feature maps. Between every other strided convolution, there is another resnet block with stride 1 and same number feature maps. There is one more fully-connected layer after convolutional layers with 450 hidden units. The generation network has a symmetric structure with strided convolution replaced by transposed convolution (Zeiler et al., 2010). When there is a change in dimensionality, we use strided convolution or transposed convolution to replace the identity connection in ResNet blocks. Exponential Linear Units (Clevert et al., 2015) are used as activation functions. The whole network is parametrized according to Weight Normalization (Salimans and Kingma, 2016) and data-dependent initialization is used.

C ResNet VAE

For details of the ResNet VAE architecture used for CIFAR-10, please see figure 3 and our code. The main benefits of this architecture is that it forms a flexible autoregressive prior latent space, while still being straightforward to sample from.

For CIFAR-10, we used a novel neural variational autoencoder (VAE) architecture with ResNet (He et al., 2015, 2016) units and multiple stochastic layers. Our architecture consists of L stacked blocks, where each block ($l = 1..L$) is a combination of a bottom-up residual unit for inference, producing a series of bottom-up activations $\mathbf{h}_l^{(q)}$, and a top-down residual unit used for both inference and generation, producing a series of top-down activations $\mathbf{h}_l^{(p)}$.

The hidden layer of each residual function in the generative model contains a combination of the usual deterministic hidden units and a relatively small number of stochastic hidden units with a heteroscedastic diagonal Gaussian distribution $p(\mathbf{z}_l | \mathbf{h}_l^{(p)})$ given the unit's input $\mathbf{h}_l^{(p)}$, followed by a nonlinearity. We utilize wide (Zagoruyko and Komodakis, 2016) *pre-activation residual units* (He et al., 2015) with single-hidden-layer residual functions.

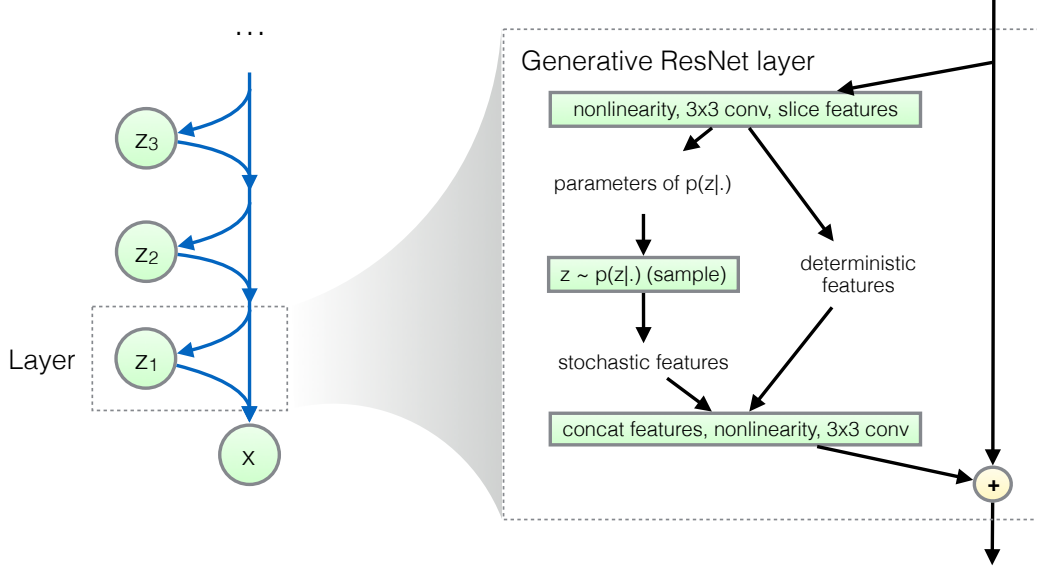


Figure 4: Generative ResNet and detail of layer. This is the generative component of our ResNet VAE.

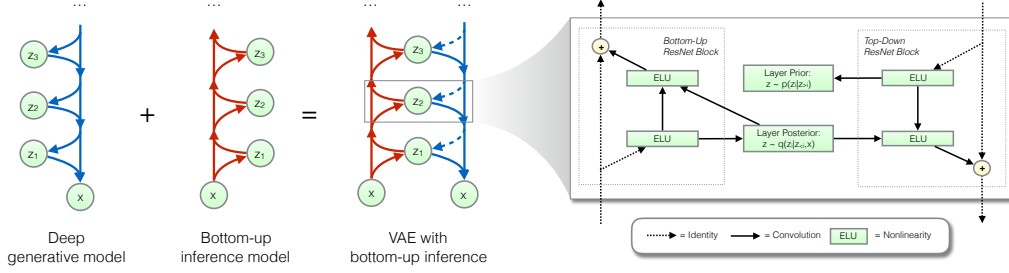
See figure 5 for an illustration of the generative ResNet. Assuming L layers of latent variables, the generative model's density function is factorized as $p(\mathbf{x}, \mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3, \dots) = p(\mathbf{x}, \mathbf{z}_{1:L}) = p(\mathbf{x}|\mathbf{z}_{1:L})p(\mathbf{z}_{1:L})$. The second part of this density, the prior over the latent variable, is autoregressive: $p(\mathbf{z}_{1:L}) = p(\mathbf{z}_L) \prod_{l=1}^{L-1} p(\mathbf{z}_l|\mathbf{z}_{l+1:L})$. This autoregressive nature of the prior increases the flexibility of the true posterior, leading to improved empirical results. This improved performance is easily explained: as the true posterior is largely a function of this prior, a flexible prior improves the flexibility of the true posterior, making it easier for the VAE to match the approximate and true posteriors, leading to a tighter bound, without sacrificing the flexibility of the generative model itself.

C.1 Bottom-Up versus Bidirectional inference

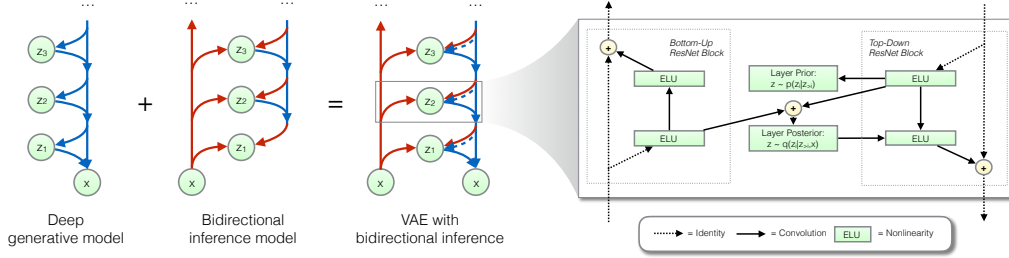
Figure 5 illustrates the difference between a *bidirectional* inference network, whose topological ordering over the latent variables equals that of the generative model, and a *bottom-up* inference network, whose topological ordering is reversed.

The top left shows a generative model with three levels of latent variables, with topological ordering $\mathbf{z}_1 \rightarrow \mathbf{z}_2 \rightarrow \mathbf{z}_3 \rightarrow \mathbf{x}$, and corresponding joint distribution that factorizes as $p(\mathbf{x}, \mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3) = p(\mathbf{x}|\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3)p(\mathbf{z}_1|\mathbf{z}_2, \mathbf{z}_3)p(\mathbf{z}_2|\mathbf{z}_3)p(\mathbf{z}_3)$. The top middle shows a corresponding inference model with reversed topological ordering, and corresponding factorization $q(\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3|\mathbf{x}) = q(\mathbf{z}_1|\mathbf{x})q(\mathbf{z}_2|\mathbf{z}_1, \mathbf{x})q(\mathbf{z}_3|\mathbf{z}_2, \mathbf{z}_1, \mathbf{x})$. We call this a bottom-up inference model. Right: the resulting variational autoencoder (VAE). In the VAE, the log-densities of the inference model and generative model, $\log p(\mathbf{x}, \mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3) - \log q(\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3|\mathbf{x})$ respectively, are evaluated under a sample from the inference model, to produce an estimate of the variational bound. Computation of the density of the samples $(\mathbf{x}, \mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3)$ under the generative model, requires computation of the conditional distributions of the generative model, which requires bidirectional computation. Hence, evaluation of the model requires both bottom-up inference (for sampling \mathbf{z} s, and evaluating the posterior density) and top-down generation.

In case of bidirectional inference (see also (Salimans, 2016; Kaae Sønderby et al., 2016)), we first perform a fully deterministic bottom-up pass, before sampling from the posterior in top-down order in the topological ordering of the generative models.



(a) Schematic overview of a ResNet VAE with bottom-up inference.



(b) Schematic overview of a ResNet VAE with bidirectional inference.

Figure 5: Schematic overview of topological orderings of bottom-up (a) versus bidirectional (b) inference networks, and their corresponding variational autoencoders (VAEs). See section C.1

C.2 Inference with stochastic ResNet

Like our generative Resnet, the both our bottom-up and bidirectional inference models are implemented through ResNet blocks. See figure 6. As we explained, in case of bottom-up inference, latent variables are sampled in bottom-up order, i.e. the reverse of the topological ordering of the generative model. The residual functions in the bottom-up inference network compute a conditional approximate posterior, $q(\mathbf{z}_l | \mathbf{h}_{l+1}^{(p)})$, conditioned on the bottom-up residual unit's input. The sample from this distribution is then, after application of the nonlinearity, treated as part of the hidden layer of the bottom-up residual function and thus used upstream.

In the bidirectional inference case, the approximate posterior for each layer is conditioned on both the bottom-up input and top-down input: $q(\mathbf{z}_l | \mathbf{h}_l^{(q)}, \mathbf{h}_l^{(p)})$. The sample from this distribution is, again after application of the nonlinearity, treated as part of the hidden layer of the top-down residual function and thus used downstream.

C.3 Approximate posterior

The approximate posteriors $q(\mathbf{z}_l | \cdot)$ are defined either through a diagonal Gaussian, or through an IAF posterior. In case of IAF, the context \mathbf{c} is provided by either $\mathbf{h}_l^{(q)}$, or $\mathbf{h}_l^{(q)}$ and $\mathbf{h}_l^{(p)}$, dependent on inference direction.

We use either diagonal Gaussian posteriors, or IAF with a single step of masked-based PixelCNN (van den Oord et al., 2016b) with zero, one or two layers of hidden layers with ELU nonlinearities (Clevert et al., 2015). Note that IAF with zero hidden layers corresponds to a linear transformation; i.e. a Gaussian with off-diagonal covariance.

We investigate the importance of a full IAF transformation with learned (dynamic) $\sigma_t(\cdot)$ rescaling term (in eq.(10)), versus a fixed $\sigma_t(\cdot) = 1$ term. See table 3 for a comparison of resulting test-set bpp (bits per pixel) performance, on the CIFAR-10 dataset. We found the difference in performance to be almost negligible.

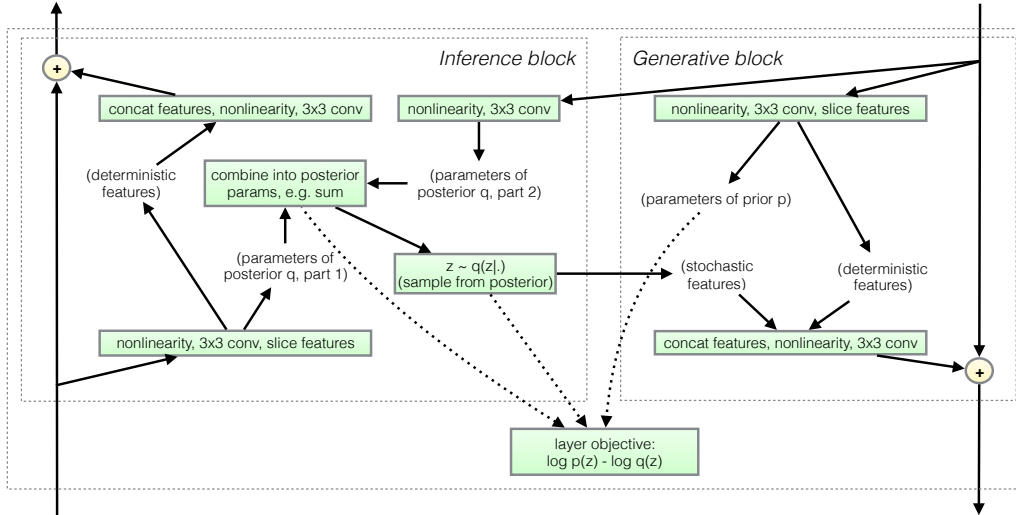
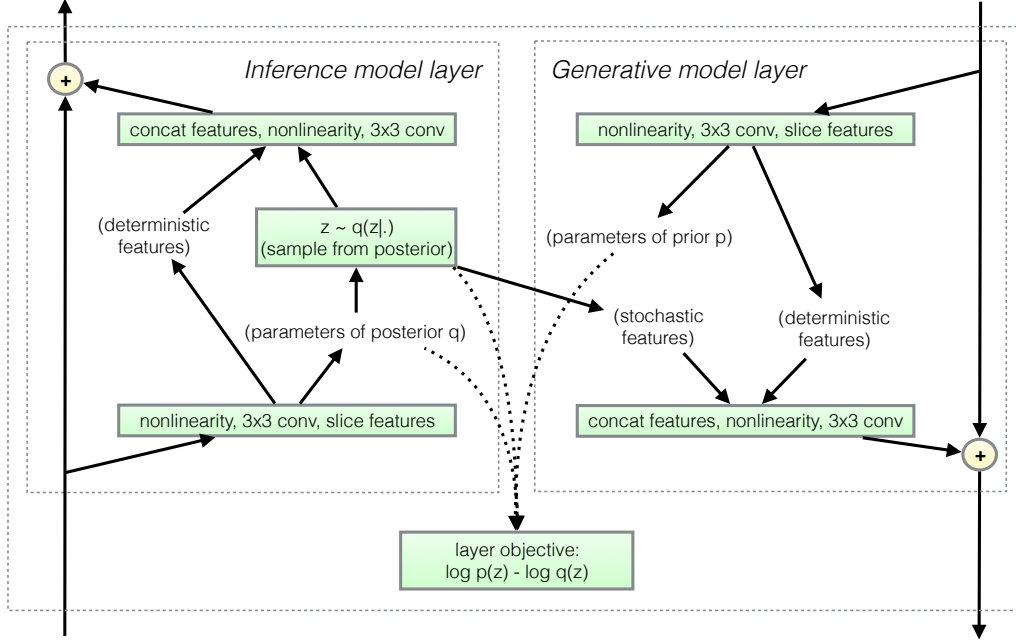


Figure 6: Detail of a single layer of the ResNet VAE, with the bottom-up inference (top) and bidirectional inference (bottom).

C.4 Bottom layer

The first layer of the encoder is a convolutional layer with 2×2 spatial subsampling; the last layer of the decoder has a matching convolutional layer with 2×2 spatial upsampling. The ResNet layers could perform further up- and downsampling, but we found that this did not improve empirical results.

Table 3: CIFAR-10 test-set bpp (bits per pixel), when training IAF with location-only perturbation versus full (location+scale) perturbation.

ResNet depth	Inference direction	Posterior	Location-only	Location+scale
4	Bottom-up	IAF with 0 hidden layers	3.67	3.68
4	Bottom-up	IAF with 1 hidden layers	3.61	3.61
4	Bidirectional	IAF with 0 hidden layers	3.66	3.67
4	Bidirectional	IAF with 1 hidden layers	3.58	3.56
8	Bottom-up	IAF with 0 hidden layers	3.54	3.55
8	Bottom-up	IAF with 1 hidden layers	3.48	3.49
8	Bidirectional	IAF with 0 hidden layers	3.51	3.52
8	Bidirectional	IAF with 1 hidden layers	3.45	3.42

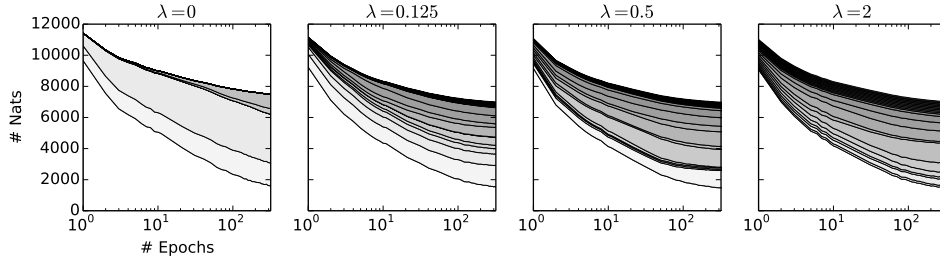


Figure 7: Shown are stack plots of the number of nats required to encode the CIFAR-10 set images, per stochastic layer of the 24-layer network, as a function of the number of training epochs, for different choices of minimum information constraint λ (see section C.8). Enabling the constraint ($\lambda > 0$) results in avoidance of undesirable stable equilibria, and fuller use of the stochastic layers by the model. The bottom-most (white) area corresponds to the bottom-most (reconstruction) layer, the second area from the bottom denotes the first stochastic layer, the third area denotes the second stochastic layer, etc.

C.5 Discretized Logistic Likelihood

The first layer of the encoder, and the last layer of the decoder, consist of convolutions that project from/to input space. The pixel data is scaled to the range $[0, 1]$, and the data likelihood of pixel values in the generative model is the probability mass of the pixel value under the logistic distribution. Noting that the CDF of the standard logistic distribution is simply the sigmoid function, we simply compute the probability mass per input pixel using $p(x_i|\mu_i, s_i) = \text{CDF}(x_i + \frac{1}{256}|\mu_i, s_i) - \text{CDF}(x_i|\mu_i, s_i)$, where the locations μ_i are output of the decoder, and the log-scales $\log s_i$ are learned scalar parameter per input channel.

C.6 Weight initialisation and normalization

We also found that the noise introduced by batch normalization hurts performance; instead we use weight normalization (Salimans and Kingma, 2016) method. We initialized the parameters using the data-dependent technique described in (Salimans and Kingma, 2016).

C.7 Nonlinearity

We compared ReLU, softplus, and ELU (Clevert et al., 2015) nonlinearities; we found that ELU resulted in significantly better empirical results, and used the ELU nonlinearity for all reported experiments and both the inference model and the generative model.

C.8 Objective with Free Bits

To accelerate optimization and reach better optima, we optimize the bound using a slightly modified objective with *free bits*: a constraint on the minimum amount of information per group of latent

Table 4: Our results in *average number of bits per data dimension* on the test set with ResNet VAEs, for various choices of posterior ResNet depth, and IAF depth.

Posterior:	ResNet Depth:	4	8	12
Bottom-up, factorized Gaussians		3.71	3.55	3.44
Bottom-up, IAF, linear, 1 step)		3.68	3.55	3.41
Bottom-up, IAF, 1 hidden layer, 1 step)		3.61	3.49	3.38
Bidirectional, factorized Gaussians		3.74	3.60	3.46
Bidirectional, IAF, linear, 1 step)		3.67	3.52	3.40
Bidirectional, IAF, 1 hidden layer, 1 step)		3.56	3.42	3.28
Bidirectional, IAF, 2 hidden layers, 1 steps)		3.54	3.39	3.27
Bidirectional, IAF, 1 hidden layer, 2 steps)		3.53	3.36	3.26

variables. Consistent with findings in (Bowman et al., 2015) and (Kaae Sønderby et al., 2016), we found that stochastic optimization with the unmodified lower bound objective often gets stuck in an undesirable stable equilibrium. At the start of training, the likelihood term $\log p(\mathbf{x}|\mathbf{z})$ is relatively weak, such that an initially attractive state is where $q(\mathbf{z}|\mathbf{x}) \approx p(\mathbf{z})$. In this state, encoder gradients have a relatively low signal-to-noise ratio, resulting in a stable equilibrium from which it is difficult to escape. The solution proposed in (Bowman et al., 2015) and (Kaae Sønderby et al., 2016) is to use an optimization schedule where the weight of the latent cost $D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$ is slowly annealed from 0 to 1 over many epochs.

We propose a different solution that does not depend on an annealing schedule, but uses a modified objective function that is constant throughout training instead. We divide the latent dimensions into the K groups/subsets within which parameters are shared (e.g. the latent feature maps, or individual dimensions if no parameter are shared across dimensions). We then use the following objective, which ensures that using less than λ nats of information per subset j (on average per minibatch \mathcal{M}) is not advantageous:

$$\tilde{\mathcal{L}}_\lambda = \mathbb{E}_{\mathbf{x} \sim \mathcal{M}} [\mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{x}|\mathbf{z})]] - \sum_{j=1}^K \text{maximum}(\lambda, \mathbb{E}_{\mathbf{x} \sim \mathcal{M}} [D_{KL}(q(\mathbf{z}_j|\mathbf{x})||p(\mathbf{z}_j))]) \quad (15)$$

Since increasing the latent information is generally advantageous for the first (unaffected) term of the objective (often called the *negative reconstruction error*), this results in $\mathbb{E}_{\mathbf{x} \sim \mathcal{M}} [D_{KL}(q(\mathbf{z}_j|\mathbf{x})||p(\mathbf{z}_j))] \geq \lambda$ for all j , in practice.

We experimented with $\lambda \in [0, 0.125, 0.25, 0.5, 1, 2, 4, 8]$ and found that values in the range $\lambda \in [0.125, 0.25, 0.5, 1, 2]$ resulted in more than 0.1 nats improvement in bits/pixel on the CIFAR-10 benchmark.

C.9 IAF architectural comparison

We performed an empirical evaluation on CIFAR-10, comparing various choices of ResNet VAE combined with various approximate posteriors and model depths, and inference directions. See table 4 for results. Our best CIFAR-10 result, 3.11 bits/dim in table 2 was produced by a ResNet VAE with depth 20, bidirectional inference, nonlinear IAF with 2 hidden layers and 1 step. Please see our code for further details.

D Equivalence with Autoregressive Priors

Earlier work on improving variational auto-encoders has often focused on improving the prior $p(\mathbf{z})$ of the latent variables in our generative model. For example, (Gregor et al., 2013) use a variational auto-encoder where both the prior and inference network have recursion. It is therefore worth noting that our method of improving the fully-factorized posterior approximation with inverse autoregressive flow, in combination with a factorized prior $p(\mathbf{z})$, is equivalent to estimating a model where the prior $p(\mathbf{z})$ is autoregressive and our posterior approximation is factorized. This result follows directly from the analysis of section 3: we can consider the latent variables \mathbf{y} to be our target for inference, in

which case our prior is autoregressive. Equivalently, we can consider the whitened representation \mathbf{z} to be the variables of interest, in which case our prior is fully-factorized and our posterior approximation is formed through the inverse autoregressive flow that whitens the data (equation 7).