IMPROVING VARIATIONAL INFERENCE WITH INVERSE AUTOREGRESSIVE FLOW

Diederik P. Kingma$^{*\times}$**, Tim Salimans**† **and Max Welling***‡
[*] University of Amsterdam
[×] OpenAI, San Francisco
[†] Aidence, Amsterdam
[‡] University of California, Irvine, and the Canadian Institute for Advanced Research (CIFAR)
D.P.Kingma@uva.nl, salimans.tim@gmail.com, M.Welling@uva.nl

ABSTRACT

We propose a simple and practical method for improving the flexibility of the approximate posterior in variational auto-encoders (VAEs) through a transformation with autoregressive networks.

Autoregressive networks, such as RNNs and RNADE networks, are very powerful models. However, their sequential nature makes them impractical for direct use with VAEs, as sequentially sampling the latent variables is slow when implemented on a GPU. Fortunately, we find that by *inverting* autoregressive networks we can obtain equally powerful data transformations that can be computed in parallel. We call these data transformations *inverse autoregressive flows* (IAF), and we show that they can be used to transform a simple distribution over the latent variables into a much more flexible distribution, while still allowing us to compute the resulting variables' probability density function. The method is computationally cheap, can be made arbitrarily flexible, and (in contrast with previous work) is naturally applicable to latent variables that are organized in multidimensional tensors, such as 2D grids or time series.

The method is applied to a novel deep architecture of variational auto-encoders. In experiments we demonstrate that autoregressive flow leads to significant performance gains when applied to variational autoencoders for natural images.

1 INTRODUCTION

Stochastic gradient variational inference (SGVI), also called stochastic gradient variational Bayes, is a method of of variational inference through a latent-variable reparameterization and stochastic gradient ascent Kingma & Welling (2013); Rezende et al. (2014); Salimans et al. (2014). It can be used in combination with inference networks to amortize the cost of inference in latent-variable models, resulting in a highly scalable learning procedure. When using neural networks for both the inference network and generative model, this results in class of models called variational auto-encoders (VAEs).

At the core of our proposed method lie 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 neural density estimators such as RNNs and RNADE models Uria et al. (2013). We show that such functions can be easily turned into nonlinear transformations of the input, with a very simple Jacobian determinant: the product of standard deviations. Since the transformation is flexible and the determinant known, it can be used to transform a tensor with relatively simple known density, into a new tensor with more complicated density that is still cheaply computable. In contrast with previous work, 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 3D tensor (a stack of featuremaps), and demonstrate that the method greatly improves 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 $p_\theta(\mathbf{x}, \mathbf{z})$ be the model of their joint distribution, called the *generative model* defined over the variables. Given a dataset $\mathbf{X} = \{\mathbf{x}^1, ..., \mathbf{x}^N\}$ we typically wish to perform maximum marginal likelihood learning of the parameters θ, i.e. to maximize

$$\log p_\theta(\mathbf{X}) = \sum_{i=1}^{N} \log p_\theta(\mathbf{x}^{(i)}), \tag{1}$$

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.

2.1 The variational bound

A solution is to introduce $q_\theta(\mathbf{z}|\mathbf{x})$, an *inference model* defined over the latent variables, and optimize the *variational lower bound* on the marginal log-likelihood of each observation $\mathbf{x}$:

$$\log p_\theta(\mathbf{x}) \leq \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\theta(\mathbf{z}|\mathbf{x})\right] = \mathcal{L}(\mathbf{x}; \theta) \tag{2}$$

where it's clear that $\mathcal{L}(\mathbf{x}; \theta)$ is a lower bound on $\log p_\theta(\mathbf{x})$ since it can be written as follows (noting that Kullback-Leibler divergences $D_{KL}(.)$ are non-negative):

$$\mathcal{L}(\mathbf{x}; \theta) = \log p_\theta(\mathbf{x}) - D_{KL}(q_\theta(\mathbf{z}|\mathbf{x}) || p_\theta(\mathbf{z}|\mathbf{x})) \tag{3}$$

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

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

2.2 Requirements for computational tractability

Requirements for the inference model, in order to be able to efficiently optimize the bound, are that it is computationally cheap to (1) compute, (2) differentiate and (3) sample from its probability density $q_\theta(\mathbf{z}|\mathbf{x})$, since these operations need to be performed for each datapoint in a minibatch at every iteration of optimization. If $\mathbf{z}$ is high-dimensional and want to make efficient use of parallel computational resources like GPUs, then (4) parallelizability of these operations across dimensions of $\mathbf{z}$ is a large factor towards efficiency. These four requirements restrict the class of approximate posteriors $q_\theta(\mathbf{z}|\mathbf{x})$ that are practical to use. In practice this often leads to the use of diagonal posteriors, e.g. $q_\theta(\mathbf{z}|\mathbf{x}) \sim \mathcal{N}(\mu_\theta(\mathbf{x}), \sigma_\theta(\mathbf{x}))$, where $\mu_\theta(\mathbf{x})$ and $\sigma_\theta(\mathbf{x})$ are often nonlinear functions parameterized by neural networks. However, as explained above, we also need the density $q_\theta(\mathbf{z}|\mathbf{x})$ to be sufficiently flexible to match the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$.

2.3 Normalizing Flows

Normalizing Flow (NF), investigated by Rezende & 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 simple distribution, and then apply a series of invertible parameterized transformations:

$$\mathbf{z}_0 \sim q_\theta(\mathbf{z}_0|\mathbf{x}) \tag{4}$$

$$\mathbf{z}_K = \mathbf{f}_\theta^K \circ \mathbf{f}_\theta^{K-1} \circ ... \circ \mathbf{f}_\theta^1(\mathbf{z}_0), \tag{5}$$

such that the last iterate $\mathbf{z}_K$ has a more flexible distribution. As long as the Jacobian determinant of each of the transformations $\mathbf{f}_\theta^i$ can be computed, we can still compute the probability density function of the last iterate:

$$\log q_\theta(\mathbf{z}_K|\mathbf{x}) = \log q_\theta(\mathbf{z}_0|\mathbf{x}) - \sum_{i=1}^{K} \log \det \left| \frac{d\mathbf{f}_\theta^i}{d\mathbf{z}_{i-1}} \right| \tag{6}$$

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

$$\mathbf{f}_\theta^i(\mathbf{z}_{i-1}) = \mathbf{z}_{i-1} + \mathbf{u}h(\mathbf{w}^T\mathbf{z}_{i-1} + b) \tag{7}$$

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

Another approach, taken by Dinh et al. (2014), is to transform the data using shearing transformations, which have a fixed Jacobian matrix determinant of one. Typically, this type of transformations updates half of the latent variables $\mathbf{z}^{1,...,D/2}$ per step by adding a vector $\delta_\theta(\mathbf{z}^{D/2+1,...,D})$ which is a function of the remaining latent variables $\mathbf{z}^{D/2+1,...,D}$ which are kept fixed. Such a transformation admits completely general transformations $\delta_\theta()$, but the requirement to partition the set of latent variables is still very limiting. Indeed, Rezende & Mohamed (2015) find that this type of transformation is generally less powerful than the normalizing flow presented above.

A potentially more 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 variables $\mathbf{u}$. 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 as 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 $\mathbf{u}$, which can impede progress if the bound is not sufficiently tight.

So far, no single method has been proposed that is both powerful and computationally cheap, and that satisfies all four of the computational tractibility criteria of section 2.2.

3 WHITENING OF DATA GENERATED BY AUTOREGRESSIVE GAUSSIAN MODELS

In order to find a type of normalizing flow that is both powerful and computationally cheap, we consider a conceptually simple family of autoregressive Gaussian generative models which we will now briefly introduce. Let $\mathbf{y} = \{y_i\}_{i=1}^D$ be some random vector (or tensor) with some ordering on its elements. On this vector we define an autoregressive Gaussian generative model:

$$\begin{aligned} y^0 &= \mu^0 + \sigma^0 \cdot z^0, \\ y^i &= \mu_\theta^i(\mathbf{y}^{1:i-1}) + \sigma_\theta^i(\mathbf{y}^{1:i-1}) \cdot z^i, \\ z^i &\sim \mathcal{N}(0,1) \forall i, \end{aligned} \tag{8}$$

where $\mu_\theta^i(\mathbf{y}^{1:i-1})$ and $\sigma_\theta^i(\mathbf{y}^{1:i-1})$ are general functions, e.g. neural networks with parameters θ, that take the previous elements of $\mathbf{y}$ as input and map them to a predicted mean and standard deviation for each element of $\mathbf{y}$. Models of this type include LSTMs Hochreiter & Schmidhuber (1997) predicting a mean and/or variance over input data, and (locally connected or fully connected) real-valued Gaussian RNADE models Uria et al. (2013). This is a rich class of very powerful models, but the disadvantage of using autoregressive models for variational inference is that the elements y^i have to be generated sequentially, which is slow when implemented on a GPU.

The autoregressive model effectively transforms the vector $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$ to a vector $\mathbf{y}$ with a more complicated distribution. As long as we have $\sigma^i > 0 \forall i$, this type of transformation is one-to-one, and can be inverted using

$$z^i = \frac{y^i - \mu_\theta^i(\mathbf{y}^{1:i-1})}{\sigma_\theta^i(\mathbf{y}^{1:i-1})}. \tag{9}$$

This inverse transformation *whitens* the data, turning the vector $\mathbf{y}$ with complicated distribution back into a vector $\mathbf{z}$ where each element is independently identically distributed according to the standard normal distribution. This inverse transformation is equally powerful as the autoregressive transformation above, but it has the crucial advantage that the individual elements z^i can now be calculated in parallel, as the z^i do not depend on each other given $\mathbf{y}$. The transformation $\mathbf{y} \rightarrow \mathbf{z}$ can thus be completely vectorized:

$$\mathbf{z} = (\mathbf{y} - \mu_\theta(\mathbf{y}))/\sigma_\theta(\mathbf{y}), \tag{10}$$

where the subtraction and division are elementwise. Unlike when using the autoregressive models naively, the *inverse* autoregressive transformation is thus both powerful *and* computationally cheap. Therefore, this type of transformation is a strong condidate for use with variational inference.

4 Inverse Autoregressive Flow (IAF)

A key observation, important for variational inference, is that the autoregressive whitening operation explained above has a lower triangular Jacobian matrix, whose diagonal elements are the elements of $\sigma_\theta(\mathbf{y})$. Therefore, the log-determinant of the transformation is simply minus the sum of the log-standard deviations used by the autoregressive Gaussian generative model:

$$\log \det \left| \frac{d\mathbf{z}}{d\mathbf{y}} \right| = -\sum_{i=1}^{D} \log \sigma_\theta^i(\mathbf{y}) \tag{11}$$

which is computationally cheap to compute, and allows us to evaluate the variational lower bound. As a result, we can use inverse autoregressive transformations (eq. (10)) as a type of normalizing flow (eq. (6)) for variational inference, which we call *inverse autoregressive flow* (IAF). If this transformation is used to match samples from an approximate posterior to a prior $p(\mathbf{z})$ that is standard normal, the transformation will indeed whiten, but otherwise the transformation can produce very general output distributions.

When using inverse autoregressive flow in our posterior approximation, we first map $\mathbf{x} \rightarrow \mu_\theta^0, \sigma_\theta^0, \mathbf{c}$, where $\mathbf{c}$ is an optional *context* variable used as additional input to the autoregressive functions, and we sample the first iterate of the latent variables as $\mathbf{z}_0 \sim q_\theta(\mathbf{z}^0|\mathbf{x})$ through:

$$\mathbf{z}_0 = \mu_\theta^0 + \sigma_\theta^0 \odot \epsilon^0 \tag{12}$$

where $\epsilon^0 \sim \mathcal{N}(0, \mathbf{I})$. Subsequently we apply K steps of IAF:

$$\text{for } i = 1...K: \quad \mathbf{z}_i = (\mathbf{z}_{i-1} - \mu_\theta^i(\mathbf{z}_{i-1}, \mathbf{c}))/\sigma_\theta^i(\mathbf{z}_{i-1}, \mathbf{c}), \tag{13}$$

where at every step we use a differently parameterized autoregressive model $(\mu_\theta^i, \sigma_\theta^i)$. If these models are sufficiently powerful, the final iterate $\mathbf{z}_K$ will have a flexible distribution that can be closely fitted to the true posterior. Some examples are given below.

4.1 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

$$\begin{aligned} y^i &= \mu_\theta^i(\mathbf{y}^{1:i-1}) + \sigma_\theta^i(\mathbf{y}^{1:i-1}) \cdot z^i, \text{ with} \\ \mu_\theta^i(\mathbf{y}^{1:i-1}) &= m^i + \mathbf{C}[i, 1:i-1]\mathbf{C}[1:i-1, 1:i-1]^{-1}(\mathbf{y}^{1:i-1} - \mathbf{m}^{1:i-1}), \text{ and} \\ \sigma_\theta^i(\mathbf{y}^{1:i-1}) &= \mathbf{C}[i,i] - \mathbf{C}[i, 1:i-1]\mathbf{C}[1:i-1, 1:i-1]^{-1}\mathbf{C}[1:i-1, i]. \end{aligned}$$

Inverting the autoregressive model then gives

$$\mathbf{z} = (\mathbf{y} - \mu_\theta(\mathbf{y}))/\sigma_\theta(\mathbf{y}) = \mathbf{L}(\mathbf{y} - \mathbf{m}), \tag{14}$$

with $\mathbf{L}$ the inverse Cholesky factorization of the covariance matrix $\mathbf{C}$.

By making $\mathbf{L}_\theta(\mathbf{x})$ and $\mathbf{m}_\theta(\mathbf{x})$ 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 & Welling (2013): $\mathbf{y} = \mu_\theta(\mathbf{x}) + \sigma_\theta(\mathbf{x}) \odot \epsilon$ where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$, and where $\mu_\theta(\mathbf{x})$ and $\sigma_\theta(\mathbf{x})$ are vectors produced by our inference network. However, in this case we have the inference network produce an extra output $\mathbf{L}_\theta(\mathbf{x})$, the lower triangular inverse Cholesky matrix of (14), which we then use to update the approximation using inverse autoregressive flow. With this setup, the problem is overparameterized, so we define the mean vector $\mathbf{m}$ of (14) to be the zero vector, and we restrict $\mathbf{L}$ to have ones on the diagonal. We then turn the fully-factorized distribution of $\mathbf{y}$ into an arbitrary multivariate Gaussian distribution by applying one iteration of inverse autoregressive flow: $\mathbf{z} = \mathbf{L}_\theta(\mathbf{x}) \cdot \mathbf{y}$. This results in a simple and computationally efficient posterior approximation, with scalar density function given by $q_\theta(\mathbf{z}|\mathbf{x}) = q_\theta(\mathbf{y}|\mathbf{x})$. By optimizing the variational lower bound with respect to θ we then fit this conditional multivariate Gaussian approximation to the true posterior distribution.

4.2 Nonlinear IAF through masked (convolutional) autoencoders

For introducing nonlinear dependencies between the elements of $\mathbf{z}$, we specify the autoregressive Gaussian model $\mu_\theta, \sigma_\theta$ using the family of deep masked autoencoders Germain et al. (2015). These models are arbitrarily flexible neural networks, where masks are applied to the weight matrices in such a way that the output $\mu_\theta(\mathbf{y}), \sigma_\theta(\mathbf{y})$ is autoregressive, i.e. $\partial\mu^i(\mathbf{y})/\partial y_j = 0, \partial\sigma^i(\mathbf{y})/\partial y_j = 0$ for $j \geq i$. Such models can implement very flexible autoregressive Gaussian densities, while still being computionally efficient, and they can be specified in either a fully connected Germain et al. (2015), or convolutional way van den Oord et al. (2016).

In experiments, we use two iterations of IAF ($K = 2$) with masked autoencoders, with reverse variable ordering in each iteration. The initial iterate, $\mathbf{z}_0$, is the sample from a diagonal Gaussian posterior (like in Kingma & Welling (2013)). In each iteration of IAF, we then compute the mean and standard deviation through its masked autoencoder, and apply the transformation of equation (13). The final variable $\mathbf{z}_2$ will have a highly flexible distribution $q(\mathbf{z}_2|\mathbf{x})$, which we use as our variational posterior.

4.3 Nonlinear IAF through Recurrent Neural Networks

Another method of parameterizing the $\mu_\theta, \sigma_\theta$ that define our inverse autoregressive flow are LSTMs and other recurrent neural networks. These are generally more powerful than the masked models, as they have an unbounded context window in computing the conditional means and variances. The downside of this is that these models cause us to lose part of our speed-up: The application of the inverse flow could still be computed in parallel, but the computation is dominated by the last element $\mu_\theta^n, \sigma_\theta^n$ of the recursion, which is much more expensive than the early elements. New methods have recently been developed to ameliorate these difficulties van den Oord et al. (2016), and such methods could also be useful for the application we consider here.

4.4 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 10).

5 Experiments

We empirically demonstrate the usefulness of inverse autoregressive flow for variational inference by training variational auto-encoders for the MNIST data set of handwritten digits and the CIFAR-10 data set of small natural images.

5.1 MNIST

Table 1: Generative modeling results on binarized MNIST. Shown are averages; the number between brackets are standard deviations across different optimization runs. The right column shows an importance sampled estimate of the marginal likelihood for each model.

Posterior	s/epoch	$\log p(\mathbf{x}) \geq$	$\log p(\mathbf{x}) \approx$
Diagonal covariance	25.5	-85.6 ($\pm$ 0.04)	-82.2 ($\pm$ 0.03)
Full covariance	30.0	-84.3 ($\pm$ 0.03)	-81.4 ($\pm$ 0.03)
Autoregressive network	27.8	-84.0 ($\pm$ 0.15)	-81.4 ($\pm$ 0.14)

We follow a similar implementation of the CVAE as in Salimans et al. (2014) with some modifications, mainly that the encoder and decoder are parameterized with ResNet He et al. (2015) blocks. For MNIST, the encoder consists of 3 sequences of two ResNet blocks each, the first sequence acting on 16 feature maps, the others on 32 feature maps. The first two sequences are followed by a 2-times subsampling operation implemented using 2×2 stride, while the third sequence is followed by a fully connected layer with 450 units. The decoder has a similar architecture, but with reversed direction.

For optimization we use Adamax Kingma & Ba (2014) with $\alpha = 0.002$ for optimization, in combination with Polyak averaging Polyak & Juditsky (1992) in the form of an exponential moving average that averages parameters over approximately 10 epochs.

We compared three forms of posteriors: a diagonal Gaussian, a full-covariance Gaussian 4.1, and nonlinear autoregressive flow through masked MLPs; see section 4.2.

Result: Table 1 shows results on MNIST for these types of posteriors. As clear from these results, the full covariance Gaussian greatly improves upon the diagonal Gaussian. The nonlinear autoregressive flow slightly improves upon this result in terms of the lower bound, but not in terms of the marginal likelihood.

5.2 CIFAR-10

For CIFAR-10, we used a neural architecture with ResNet units and multiple stochastic layers. The architecture can be understood as follows. First, take the architecture from MNIST above, and replace the 1D latent space with 3D latent tensor, such that both the encoder and decoder map from a 3D tensor to a distribution over another 3D tensor. Next, 'stack' multiple of such models, each acting on latent space of the layer below instead of the data space. We used 128 featuremaps at every layer encoder and decoder layer, except the input.

In this experiment we compared a diagonal Gaussian posterior with nonlinear autoregressive flow through masked CNNs; see section 4.2.

Result: See table 2 for results.

6 Conclusion

We presented *inverse autoregressive flow* (IAF), a method to make the posterior approximation of a variational auto-encoder more flexible, and thereby improve the variational lower bound. By inverting the sequential data generating process of an autoregressive Gaussian model, inverse autoregressive flow gives us a data transformation that is both very powerful and computationally efficient:

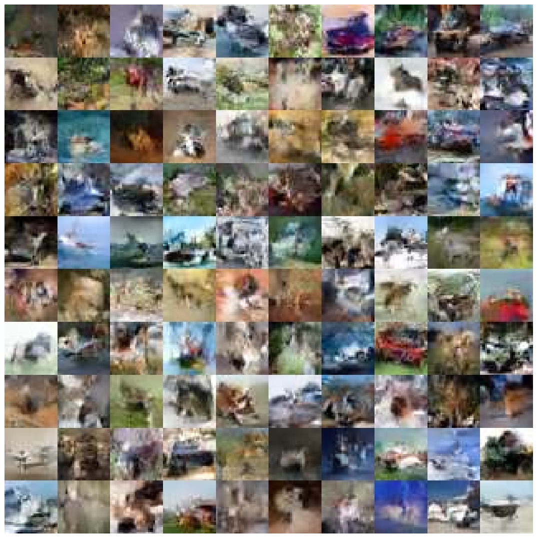

Figure 1: Random samples from learned generative model of CIFAR-10

Table 2: Generative modeling results on CIFAR-10 images.

Method	s/epoch	bits/dim $\leq$
Ours (diagonal covariance)	638	3.86
Ours (nonlinear autoregressive flow)	869	3.57

The transformation has a tractable Jacobian determinant, and it can be vectorized for implementation on a GPU. By applying this transformation to the samples from our approximate posterior we can move their distribution closer to the exact posterior.

We empirically demonstrated the usefulness of inverse autoregressive flow for variational inference by training a novel deep architecture of variational auto-encoders. In experiments we demonstrated that autoregressive flow leads to significant performance gains compared to similar models with diagonal Gaussian approximate posteriors.

REFERENCES

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. *arXiv preprint arXiv:1502.03509*, 2015.

Karol Gregor, Andriy Mnih, and Daan Wierstra. Deep AutoRegressive Networks. *arXiv preprint arXiv:1310.8499*, 2013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural computation*, 9(8): 1735–1780, 1997.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *Proceedings of the 2nd International Conference on Learning Representations*, 2013.

Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of The 32nd International Conference on Machine Learning*, pp. 1530–1538, 2015.

Danilo J Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1278–1286, 2014.

Tim Salimans, Diederip P. Kingma, and Max Welling. Markov chain Monte Carlo and variational inference: Bridging the gap. *arXiv preprint arXiv:1410.6460*, 2014.

Benigno Uria, Iain Murray, and Hugo Larochelle. RNADE: The real-valued neural autoregressive density-estimator. In *Advances in Neural Information Processing Systems*, pp. 2175–2183, 2013.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.