# Quasi-Autoregressive Residual (QuAR) Flows

**Achintya Gopal** [1]

## Abstract

Normalizing Flows are a powerful technique for learning and modeling probability distributions given samples from those distributions. The current state of the art results are built upon residual flows as these can model a larger hypothesis space than coupling layers. However, residual flows are extremely computationally expensive both to train and to use, which limits their applicability in practice. In this paper, we introduce a simplification to residual flows using a Quasi-Autoregressive (QuAR) approach. Compared to the standard residual flow approach, this simplification retains many of the benefits of residual flows while dramatically reducing the compute time and memory requirements, thus making flow-based modeling approaches far more tractable and broadening their potential applicability.

## 1. Introduction

Learning a probability distribution from some available data is a core problem within machine learning. Fitting distributions can be simple for some low-dimensional datasets, but fitting distributions to high-dimensional data with complex correlations requires a more systematic solution. Normalizing Flows are a family of deep generative models for designing large, complex distributions that capture the essential relationships among the data points. For instance, Normalizing Flows are capable of generating realistic images and achieve close to state of the art performance in density estimation (Chen et al., 2019).

Early implementations of Normalizing Flows were coupling layers (Dinh et al., 2014; 2016; Kingma & Dhariwal, 2018) and autoregressive flows (Papamakarios et al., 2017; Kingma et al., 2016). These have convenient mathematical properties (easy to compute inverses and log-determinants) but use non-standard architectures and optimizers. The newer technique of residual flows (Chen et al., 2019) allows for models that are built on standard components, are more expressive, and have inductive biases that favor simpler functions. However, these models are much more computationally expensive, making it difficult to use these in practical settings.

We propose a variant on residual flows that simplifies the architecture to be Quasi-Autoregressive (QuAR: to be defined in Section 4.1). This provides the benefits of both autoregressive flows and residual flows: computationally tractable mathematical properties (inverse and log-determinant) and more expressive models, respectively. Additionally, we show that the Lipschitz constant (a key constraint in QuAR Flows) can be made more flexible, further increasing the modeling power of quasi-autoregressive flows.

## 2. Background

### 2.1. Normalizing Flows

The generative process for flows is defined as:

$$z \sim p_z(z)$$
$$x = g(z)$$

where $p_z$ is often a Normal distribution and $g$ is an invertible function. Notationally, we will use $f = g^{-1}$. Using change of variables, the log likelihood of $x$ is

$$\log p_x(x) = \log p_z\left(f(x)\right) + \log \left| \det \left( \frac{\partial f(x)}{\partial x} \right) \right|$$

To train flows (i.e., maximize the log likelihood of data points), we need to be able to compute the logarithm of the absolute value of the determinant of the Jacobian of $f$, also called the *log-determinant*. To construct large normalizing flows, we can compose smaller ones as this is still invertible and the log-determinant of this composition is the sum of the individual log-determinants.

### 2.2. Autoregressive Flows

For a multivariate distribution, the probability density of a data point can be computed using the chain rule:

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i | x_{<i})$$

[1]Bloomberg Quant Research, New York, NY, USA. Correspondence to: Achintya Gopal <agopal6@bloomberg.net>.

Second workshop on *Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models* (ICML 2020), Virtual Conference. 

**Algorithm 1** Inverse of Residual Flow via Fixed Point Iteration

**Input:** data $y$, residual block $g$, number of iterations $n$
Initialize $x_0 = y$.
**for** $i = 1$ **to** $n$ **do**
  $x_i = y - g(x_{i-1})$
**end for**

By using a univariate normalizing flow $f_\theta(x_i|x_{<i})$ such as an affine transformation for each univariate density, we get an autoregressive flow (Papamakarios et al., 2017). Given that its Jacobian is triangular, the determinant is easy to compute as it is the product of the diagonal of the Jacobian. These models have a tradeoff where the log-likelihood is parallelizable but the sampling process is sequential, or vice versa depending on parameterization (Kingma et al., 2016).

### 2.3. Residual Flows

A residual flow is a residual network ($f(x) = x + \mathcal{F}(x)$) where the Lipschitz constant of $\mathcal{F}$ is strictly less than one. This constraint on the Lipschitz constant ensures invertibility; the transform is invertible using Banach's fixed point algorithm (Algorithm 1) where the convergence rate is exponential in the number of iterations and is faster for smaller Lipschitz constants (Behrmann et al., 2019).

The log-determinant is computed by estimating the Taylor series:

$$\ln |J_f(x)| = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{\text{tr}(J_{\mathcal{F}}^k)}{k}$$

The Skilling-Hutchinson estimator (Skilling, 1989) is used to estimate the trace in the power series; the infinite series is estimated using the Russian Roulette estimator (Kahn, 1955) which randomizes the number of terms evaluated leading to an unbiased estimator of the series.

This architecture can be trained with standard optimizers and higher learning rates without the loss diverging, and it achieves state of the art results in density estimation, sometimes with fewer parameters than coupling layers. All of this indicates that this architecture has a beneficial inductive bias for density estimation.

## 3. Related Work

(Rezende & Mohamed, 2015) introduced planar flows and radial flows to use in the context of variational inference. Sylvester flows created by (van den Berg et al., 2018) enhanced planar flows by increasing the capacity of each individual transform. Both planar flows and Sylvester flows can be seen as special cases of residual flows.

(Germain et al., 2015) introduced large autoregressive models which were then used in flows by Inverse Autoregressive Flows (IAF) (Kingma et al., 2016) and Masked Autoregressive flows (MAF) (Papamakarios et al., 2017). Larger autoregressive flows were created by Neural Autoregressive Flows (NAF) (Huang et al., 2018) and Block NAFs (BNAF) (Cao et al., 2019), both of which have better log-likelihoods but cannot be sampled from.

Before residual flows, state of the art performance with flows was achieved by coupling layers such as NICE (Dinh et al., 2014), RealNVP (Dinh et al., 2016), Glow (Kingma & Dhariwal, 2018) and Flow++ (Ho et al., 2019).

Whereas autoregressive methods and coupling layers have block structures in their Jacobian, Residual Flows (Chen et al., 2019) have a dense Jacobian; FFJORD (Grathwohl et al., 2018) is a continuous normalizing flow based on Neural ODEs (Chen et al., 2018) and can be viewed as a continuous version of Residual Flows.

## 4. Quasi-Autoregressive (QuAR) Flows

### 4.1. Quasi-Autoregressive

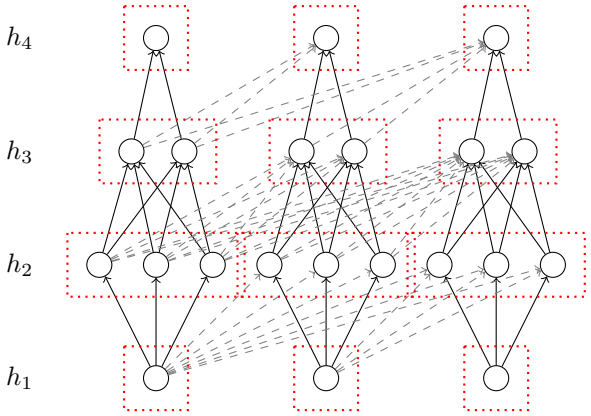


*Figure 1.* Feedforward Diagram for Quasi-Autoregressive Residual Connection. In the above plot, there are $L = 4$ layers, the number of input dimensions is $D = 3$; $k = 3$ for $h_2$ and $k = 2$ for $h_3$ which is why there are three neurons per dotted box in layer $h_2$ and two neurons per dotted box in layer $h_3$. The solid lines represent connections in the network that contribute to the diagonal of the Jacobian, i.e., solid lines are used to compute $\partial_{x_1} x_{i,d,j}$. Also, the solid lines in the first layer are new connections not included in MADE.

**Definition 4.1.** A function $f : \mathbb{R}^D \to \mathbb{R}^D$ is *quasi-autoregressive* if the Jacobian of $f(x)$ is upper triangular everywhere or lower triangular everywhere. In the special case where all of the values along the diagonal are zero, $f$ is *autoregressive*.

Within residual flows (defined as $f(x) = x + \mathcal{F}(x)$), if $\mathcal{F}$ is autoregressive, then the flow is volume preserving, i.e., the log-determinant is zero. This is equivalent to autoregressive

flows where the univariate normalizing flow is simply a shift. On the other hand, if $\mathcal{F}$ is quasi-autoregressive, the flow can change the volume of the space allowing for higher density regions around data points.

MADE (Germain et al., 2015) introduced a method to create large autoregressive networks by creating masks per layer that ensure the network is autoregressive. To simplify their masking mechanism and remove the randomness in it, the size of every hidden layer within the network is set to a multiple $k$ of the input dimension $D$. We define a network with $L$ layers where we use the notation $h_1$ to denote the input layer and $h_L$ to denote the output layer. We can then partition hidden layer $h_i$ into $D$ sets $\{H_{i,d}\}_{d=1}^{D}$ where each set $H_{i,d}$ is of size $k$. For $i = 1$ and $i = L$, $k = 1$. We define the mask such that the neurons in $H_{i,d_1}$ and $H_{i+1,d_2}$ are connected if $d_1 < d_2$ for $i = 1$ and $d_1 \leq d_2$ for $i > 1$.

To modify the above construction to be quasi-autoregressive, we change the rule constructing the mask to allow for a connection when $d_1 = d_2$ in the first layer. To compute the log-determinant of this network, we need to compute the gradients along the diagonal of the Jacobian during the forward pass. Using the notation $x_{i,d,j}$ as the output of the $j$-th neuron in the set $H_{i,d}$ and the notation $\partial_{x_1} x_{i,d,j} = \frac{\partial x_{i,d,j}}{\partial x_{1,d,1}}$ (where we use the first neuron in the denominator since $|H_{1,d}| = 1$):

$$\partial_{x_1} x_{1,d,j} = 1$$
$$\partial_{x_1} x_{i,d,j} = \sum_k \frac{\partial x_{i,d,j}}{\partial x_{i-1,d,k}} \partial_{x_1} x_{i-1,d,k} \quad i > 1$$

where the above is simply an application of the chain rule to compute $\frac{\partial x_{L,d,1}}{\partial x_{1,d,1}}$. Instead of having to construct the full Jacobian of size $n^2$, we create vectors $\partial_{x_1} x_{i,d,j}$, each with the same size as $x_i$. Compared to a similar step in standard residual flows, this is an exact computation with the downside that the gradients must be implemented without easy support from an autograd engine. Figure 1 illustrates a four layer example of a quasi-autoregressive network.

PixelCNN (van den Oord et al., 2016) extended the masking in MADEs to Convolutional Neural Networks. We utilize this extension to extend QuAR flows to the convolutional case to model images; the results of this extension are shown in Section 5.2. The details of the math are in Appendix C.

### 4.2. Lipschitz Constraint

For a function $\mathcal{F} = f_L \circ \cdots \circ f_1$, an upper bound on $\text{Lip}(\mathcal{F})$ is

$$\text{Lip}(\mathcal{F}) \leq \prod_{i=1}^{L} \text{Lip}(f_i)$$

and so to normalize $\text{Lip}(\mathcal{F})$, each function $f_i$ is usually normalized independently. We remove this independence

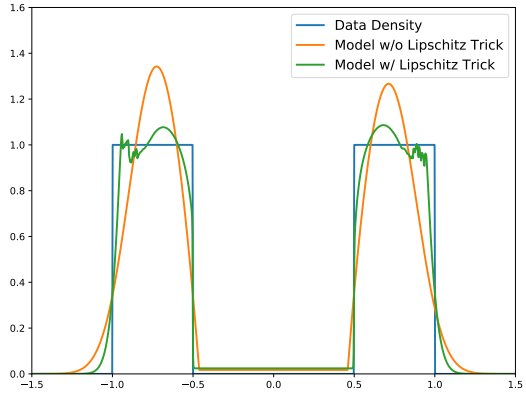


*Figure 2.* Plot of Likelihood for the data distribution, and models trained with and without Lipschitz Trick.

to increase the flexibility of $\mathcal{F}$, which we later reference as a Lipschitz Trick. We compose the function $f_{L+1}$ with $\mathcal{F}$ where

$$f_{L+1}(x;\theta) = \sigma x \left(\theta + \prod_{i=1}^{L} \text{Lip}(f_i)\right)^{-1}$$

where $\theta \in \mathbb{R}^D_{\geq 0}$ is a learnable parameter and $\sigma \in [0, 1)$ is a constant set at initialization that determines the maximum Lipschitz constant $\mathcal{F}$ can attain, controlling the convergence rate of Algorithm 1. If $\theta$ were not a learnable parameter and identically zero, then the regularization would be similar to hard spectral normalization which does not allow the neural network to have the flexibility to learn its Lipschitz constant. To ensure that the Lipschitz constant is upper bounded by $\sigma$, $\theta$ must be non-negative.

Similar to (Chen et al., 2019), to reliably compute the spectral norm of the weight matrices within the feedforward and convolutional layers, we use the power iteration method (Miyato et al., 2018) with a variable number of iterations.

## 5. Experiments

### 5.1. Synthetic Data

#### 5.1.1. Lipschitz Constraint

To test the flexibility of a single QuAR flow, we create a simple network with one overparameterized QuAR flow containing more than 100K parameters sandwiched between affine transforms and try to fit a mixture of two uniform distributions. While the Lipschitz property of a QuAR flow limits its ability to change the volume of the space, the inclusion of affine transforms relaxes this limitation as an affine transform can change the volume without restriction. From Figure 2, it can be seen that the same model with only one additional parameter and flexible Lipschitz constants is better able to model the complex function.

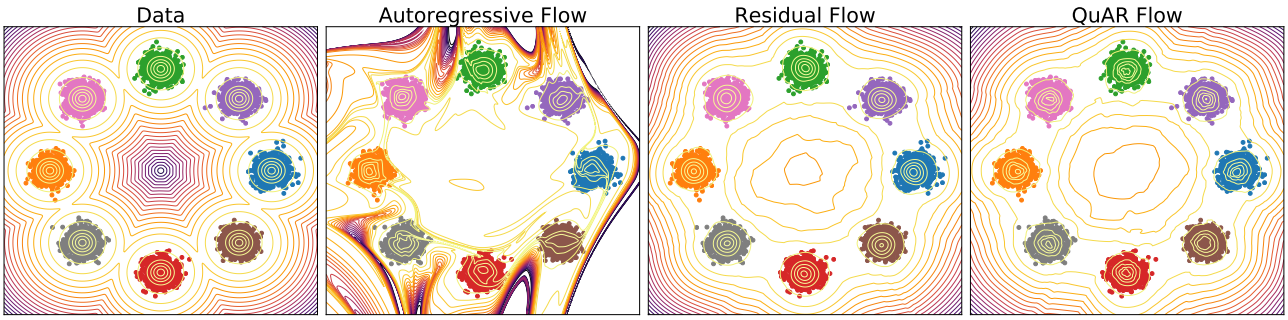


*Figure 3.* The contour plots in log space of mixture of Gaussians for, from left to right, the data distribution and the learned distributions from autoregressive flows, residual flows, and QuAR flows.

#### 5.1.2. Inductive Bias

To compare the inductive biases of autoregressive flows, residual flows and QuAR flows, we train all three on a two dimensional mixture of eight Gaussians. More detailed descriptions of the architectures are in Appendix A.1.

Rather than analyzing the likelihoods, which show how well the distribution is modeled in dense regions, we show the contour plot of the log-likelihood in Figure 3 to analyze the sparse regions. Though the autoregressive model correctly added significant density to the modes, its likelihood in sparse regions has unintuitive contours. QuAR flows and residual flows learned contours that better match the original distribution in sparse regions, suggesting that these flows have better inductive biases for learning simpler functions.

### 5.2. Experiments on Image Data

To evaluate the performance of QuAR flows on images, we train convolutional QuAR flows similar to the architecture used in residual flows on MNIST and CIFAR-10. We compare QuAR flows against Residual Flows, Glow (Kingma & Dhariwal, 2018) and Flow++ (Ho et al., 2019). Glow and Flow++ are used in place of autoregressive flows as these models use coupling layers which can be viewed as block-autoregressive.

Though the architecture and the number of parameters in the QuAR flows and residual flows are the same, the number of *learnable* parameters is much fewer since approximately half of the values in the weight matrices are unused. Table 1 shows that QuAR Flows have an improved bits per dimension (bpd) (Appendix A.2) for MNIST despite having fewer learnable parameters than the other models; the improvement might be due to the Lipschitz Trick. Unfortunately, QuAR flows have a worse bpd than residual flows for CIFAR-10. Nevertheless, they perform comparably to Glow while having half the number of parameters, and even fewer learnable parameters.

Whereas two forward evaluations are needed to evaluate QuAR flows, residual flows require "one plus number of Taylor series terms evaluated" evaluations. To get accurate log-likelihoods from residual flows during test time, twenty terms of the Taylor series of the log-determinant are computed which leads to QuAR flows being over an order of magnitude faster (Table 2). The difference in training time per batch is much smaller since (Chen et al., 2019) trained their model by evaluating on average four terms. QuAR flows also require less memory, most notably during training as the computation graph is much simpler for QuAR flows, so much less data is cached to compute the gradient.

Although QuAR flows' log likelihoods on CIFAR-10 are more comparable to those of Glow than of residual flows, there are two indications that QuAR flows are better suited to density estimation than Glow is. First, QuAR flows can be optimized with Adam (Kingma & Ba, 2015) rather than Adamax without diverging. Second, QuAR flows can be optimized effectively with a batch size of 64 with a learning rate of 1e-3 (versus 512 for Glow with a learning rate of 2e-5), suggesting the gradients are less noisy and the model can be trained faster as a function of the number of updates.

| Data set | Glow | Flow++ | Residual | QuAR |
|---|---|---|---|---|
| MNIST | 1.05 | - | 0.970 | **0.963** |
| CIFAR10 | 3.35 | 3.29 | **3.280** | 3.378 |

*Table 1.* Bits Per Dimension for Image Datasets. Though Flow++ (Ho et al., 2019) also used variational dequantization, we do not compare against these numbers.

| | | Residual | QuAR |
|---|---|---|---|
| Train | Memory | 27.8 GB | 17.8 GB |
| | Time per Batch | 6.64 s | 2.85 s |
| Test | Memory | 25.3 GB | 21.0 GB |
| | Time per Batch | 39.4 s | 1.38 s |

*Table 2.* Time and Space Usage on CIFAR-10: Batch Size of 96 for Train and Batch Size of 256 for Test.

## 6. Conclusion and Future Work

In this paper, we showed that Quasi-Autoregressive Residual Flows retain the beneficial inductive bias towards simpler functions of residual flows while greatly reducing the computational cost. We also showed a way to increase the modeling power of Lipschitz-constrained models; this suggests ways to increase capacity in other models that have this requirement, such as WGANs (Arjovsky et al., 2017).

## References


Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein generative adversarial networks. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/arjovsky17a.html.

Behrmann, J., Grathwohl, W., Chen, R. T. Q., Duvenaud, D., and Jacobsen, J.-H. Invertible residual networks. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 573–582, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/behrmann19a.html.

Cao, N. D., Aziz, W., and Titov, I. Block neural autoregressive flow. In *Conference on Uncertainty in Artificial Intelligence*, volume 3, pp. 1723–1735. UAI, 2019. URL http://auai.org/uai2019/proceedings/papers/511.pdf.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 6571–6583. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7892-neural-ordinary-differential-equations.pdf.

Chen, R. T. Q., Behrmann, J., Duvenaud, D. K., and Jacobsen, J.-H. Residual flows for invertible generative modeling. In Wallach, H., Larochelle, H., Beygelzimer, A., dÁlché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 9916–9926. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9183-residual-flows-for-invertible-generative-modeling.pdf.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *arXiv e-prints*, art. arXiv:1511.07289, November 2015.

Dinh, L., Krueger, D., and Bengio, Y. NICE: Non-linear Independent Components Estimation. *arXiv e-prints*, art. arXiv:1410.8516, October 2014.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using Real NVP. *arXiv e-prints*, art. arXiv:1605.08803, May 2016.

Germain, M., Gregor, K., Murray, I., and Larochelle, H. Made: Masked autoencoder for distribution estimation. In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 881–889, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/germain15.html.

Gouk, H., Frank, E., Pfahringer, B., and Cree, M. Regularisation of Neural Networks by Enforcing Lipschitz Continuity. *arXiv e-prints*, art. arXiv:1804.04368, April 2018.

Grathwohl, W., Chen, R. T. Q., Bettencourt, J., Sutskever, I., and Duvenaud, D. FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models. *arXiv e-prints*, art. arXiv:1810.01367, October 2018.

Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2722–2730, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/ho19a.html.

Huang, C.-W., Krueger, D., Lacoste, A., and Courville, A. Neural autoregressive flows. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2078–2087, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/huang18d.html.

Kahn, H. *Use of Different Monte Carlo Sampling Techniques*. RAND Corporation, 1955.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Machine Learning*, 2015.

Kingma, D. P. and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 10215–10224. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/8224-glow-generative-flow-with-invertible-1x1-convolutions.pdf.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. Improved variational inference with inverse autoregressive flow. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 4743–4751. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6581-improved-variational-inference-with-inverse-autoregressive-flow.pdf.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral Normalization for Generative Adversarial Networks. *arXiv e-prints*, art. arXiv:1802.05957, February 2018.

Papamakarios, G., Pavlakou, T., and Murray, I. Masked autoregressive flow for density estimation. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 2338–2347. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6828-masked-autoregressive-flow-for-density-estimation.pdf.

Polyak, B. and Juditsky, A. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30:838–855, 07 1992. doi: 10.1137/0330046.

Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1530–1538, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/rezende15.html.

Skilling, J. The eigenvalues of mega-dimensional matrices. In *Maximum Entropy and Bayesian Methods*, pp. 455–466. Springer, 1989.

van den Berg, R., Hasenclever, L., Tomczak, J. M., and Welling, M. Sylvester Normalizing Flows for Variational Inference. *arXiv e-prints*, art. arXiv:1803.05649, March 2018.

van den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. Pixel Recurrent Neural Networks. *arXiv e-prints*, art. arXiv:1601.06759, January 2016.

# A. Experiment Setup

## A.1. Architecture

For Section 5.1.1, the residual connection used within the QuAR flow is a feedforward neural network with sizes $1 - 128 - 128 - 128 - 1$ with ELU nonlinearities (Clevert et al., 2015).

For Section 5.1.2 (and Appendix B), the architectures were designed such that the number of parameters in all three models were comparable. All three were composed of 16 flows with affine transforms between each flow. The residual flow and QuAR flow used feedforward neural networks with sizes $2 - 128 - 128 - 2$ with ELU nonlinearities and the autoregressive model used feedforward neural networks with sizes $1 - 128 - 128 - 2$. The autoregressive model has an output size of 2 to parameterize an affine transformation similar to Glow.

For Section 5.2, the architecture used for MNIST and CIFAR-10 were similar to those used in residual flows. Actnorm (Kingma & Dhariwal, 2018) was used before and after every QuAR flow. Each residual connection was a convolutional network:

ELU $\rightarrow$ 3x3 Conv $\rightarrow$ ELU $\rightarrow$ 1x1 Conv $\rightarrow$ ELU $\rightarrow$ 3x3 Conv

For MNIST, a hidden size of 512 is used; for CIFAR-10, a hidden size of 528 is used because the hidden size is set to a multiple of the number of channels and the number of channels are 3, 12, and 48 because of the squeeze operation. Whereas (Chen et al., 2019) added 4 fully connected residual blocks at the end of the network, for simplicity, we did not add these layers. For the comparison in time and memory usage, we also removed these layers from residual flows. We also used the memory reduction trick and Backward-in-Forward from (Chen et al., 2019).

Similar to other flow papers, we normalize the images to be within the range [0, 1] and add uniform noise $\mathcal{U}(0, 1/256)$ to dequantize the discrete variables. Also similar to other papers, the first transform applied is a Logit Transform.

All Residual Flows and QuAR Flows were optimized with Adam whereas the autoregressive model was optimized with Adamax. For the toy data experiments, the models were trained for 20,000 updates; for the image experiments, the models were trained for 300 epochs. Additionally for image experiments, we use Polyak averaging (Polyak & Juditsky, 1992) for evaluation with a decay of 0.999.

## A.2. Bits Per Dimension

The performance of log-likelihood models for images is often defined using bits per dimension. Given a dequantization distribution $q(x)$ for $x \in \mathbb{R}^d$, the bits per dimension is defined as

$$\frac{\log p(x) - \log q(x)}{d \log 2}$$

# B. Additional Results on Synthetic Data

For additional analysis of the inductive bias in flow models, we analyze where the eight modes are mapped into the latent space and how they are transformed to those locations. Whereas in residual flows (Figure 6) and QuAR flows (Figure 5) the modes are transformed "smoothly" to their location in the latent space, autoregressive flows (Figure 4) take an unintuitive path to the latent space with some "extreme" transformations. It's possible that these types of transformations are why using optimizers like Adam with autoregressive flows and Glow can cause the loss to diverge.

From the plots, it seems that although autoregressive flows can change the volume of the space, the technique typically moves the space around to normalize the data instead. On the other hand, QuAR flows and residual flows both appear to preferentially reduce the volume such that the space between data points is removed. It is unclear whether this behavior extends to higher dimensions, but it is possible that this is the inductive bias added to these flows.

# C. Convolution QuAR Flow

Similar to the fully connected version of autoregressive models, the convolutional version defines a specific ordering in the inputs. The ordering used in QuAR flows is the same as (van den Oord et al., 2016). The masking is also the same to ensure that the operation is autoregressive. However, the same way two different types of masks were used in MADE, PixelRNN also uses two different masks. To change the model to be quasi-autoregressive, we only use one mask so that $\frac{\partial f(x)_i}{\partial x_i} \neq 0$.

(Gouk et al., 2018) defined a way to apply spectral norm to convolutions; however, due to the simplification to the convolutional operator by (van den Oord et al., 2016), a simpler power iteration method can be used. The key observation is that for a block matrix

$$W = \left(\begin{array}{c|c} \mathbf{A} & \mathbf{B} \\ \hline \mathbf{0} & \mathbf{D} \end{array}\right)$$

the spectral norm is the same as for

$$\left(\begin{array}{c|c} \mathbf{A} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{D} \end{array}\right)$$

Because $W$ is a convolution, $W$ can further be written as

$$W = \left(\begin{array}{c|c} \mathbf{A} & \mathbf{B} \\ \hline \mathbf{0} & \mathbf{A} \end{array}\right)$$

Thus, when computing the spectral norm of the convolutional weights, the spectral norm of the 1x1 convolution applied along the channels that affect the Jacobian of the block can be computed using the power iteration method of (Miyato et al., 2018). This further reduces the memory requirements on QuAR flows.

This observation could be applied to the weight matrices in fully connected layers even though this does not help memory or time. However, it can cause problems. Say we have a matrix

$$W = \begin{pmatrix} 2 & 3 \\ 0 & 1 \end{pmatrix}$$

and we choose to run the power iteration algorithm on

$$\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$$

$u$ found by the power iteration method would be $[1, 0]$ and the spectral norm is 2. Now, say that after gradient descent, the matrix we are analyzing changes

$$\begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}$$

In most implementations of the power iteration method, $u$ is initialized to the vector found in the previous iteration. However, in this case, after applying power iteration, $u$ and the spectral norm found by the algorithm will not change even though the spectral norm is now 3. This can be mitigated by either not using the block form with the additional zeros or adding random noise to $u$ before applying the power iteration method.

## D. Image Generations

We generated samples from our model with best bits per dimensions for MNIST (Figure 7) and CIFAR-10 (Figure 8).

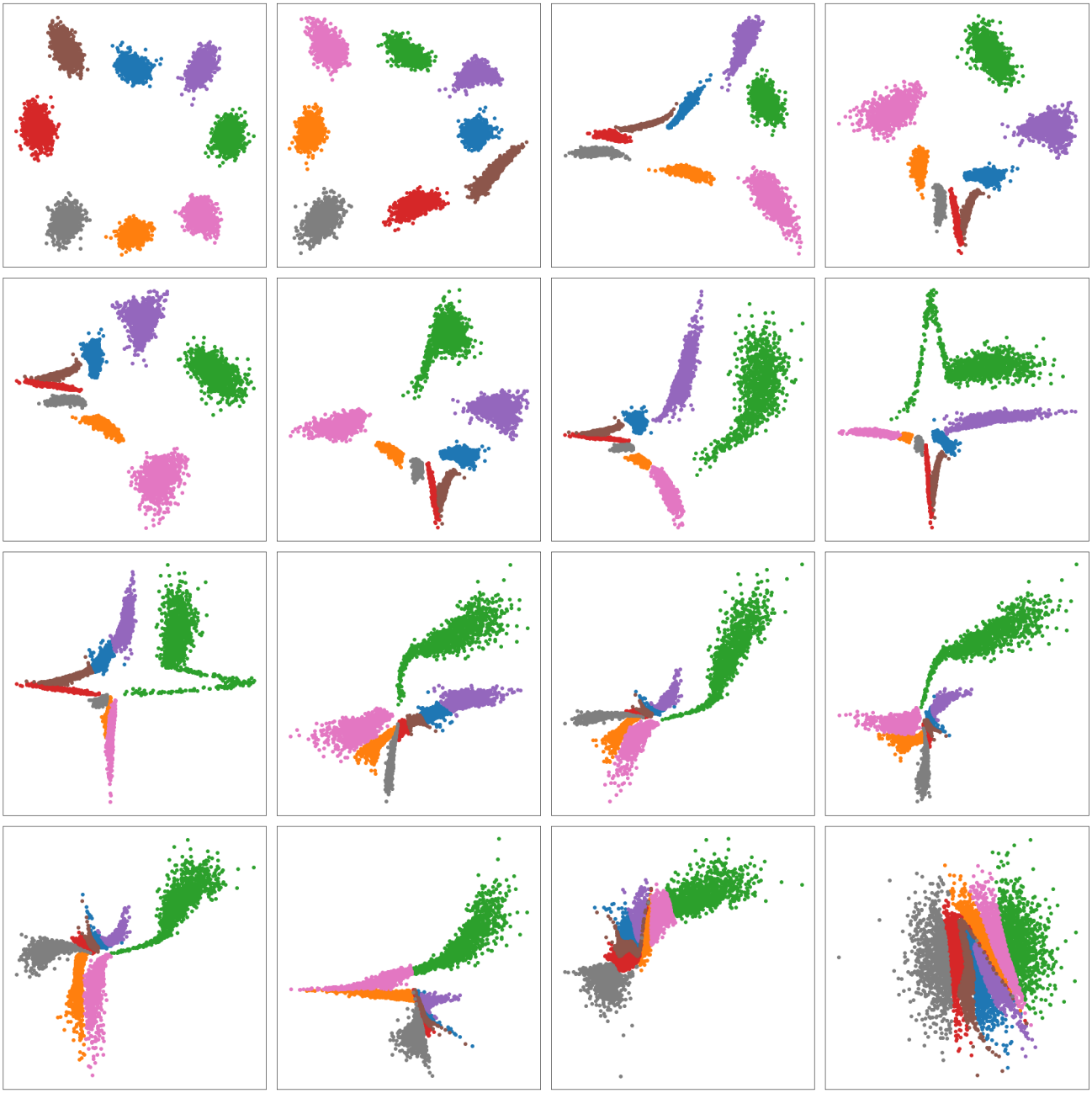

*Figure 4.* Plot of the eight Gaussians and their location after every transformation using Autoregressive Flows.

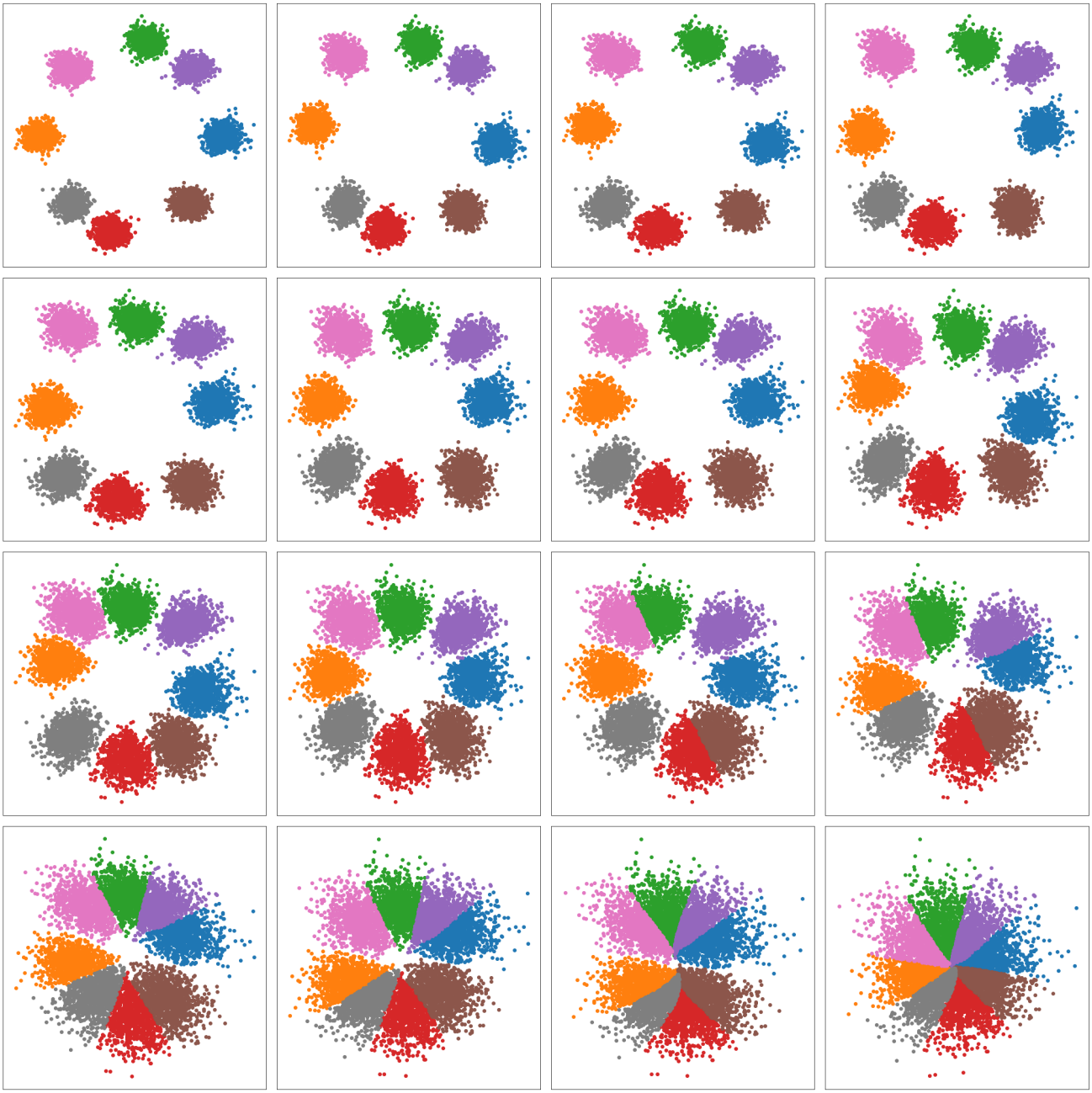

*Figure 5.* Plot of the eight Gaussians and their location after every transformation using QuAR Flows.

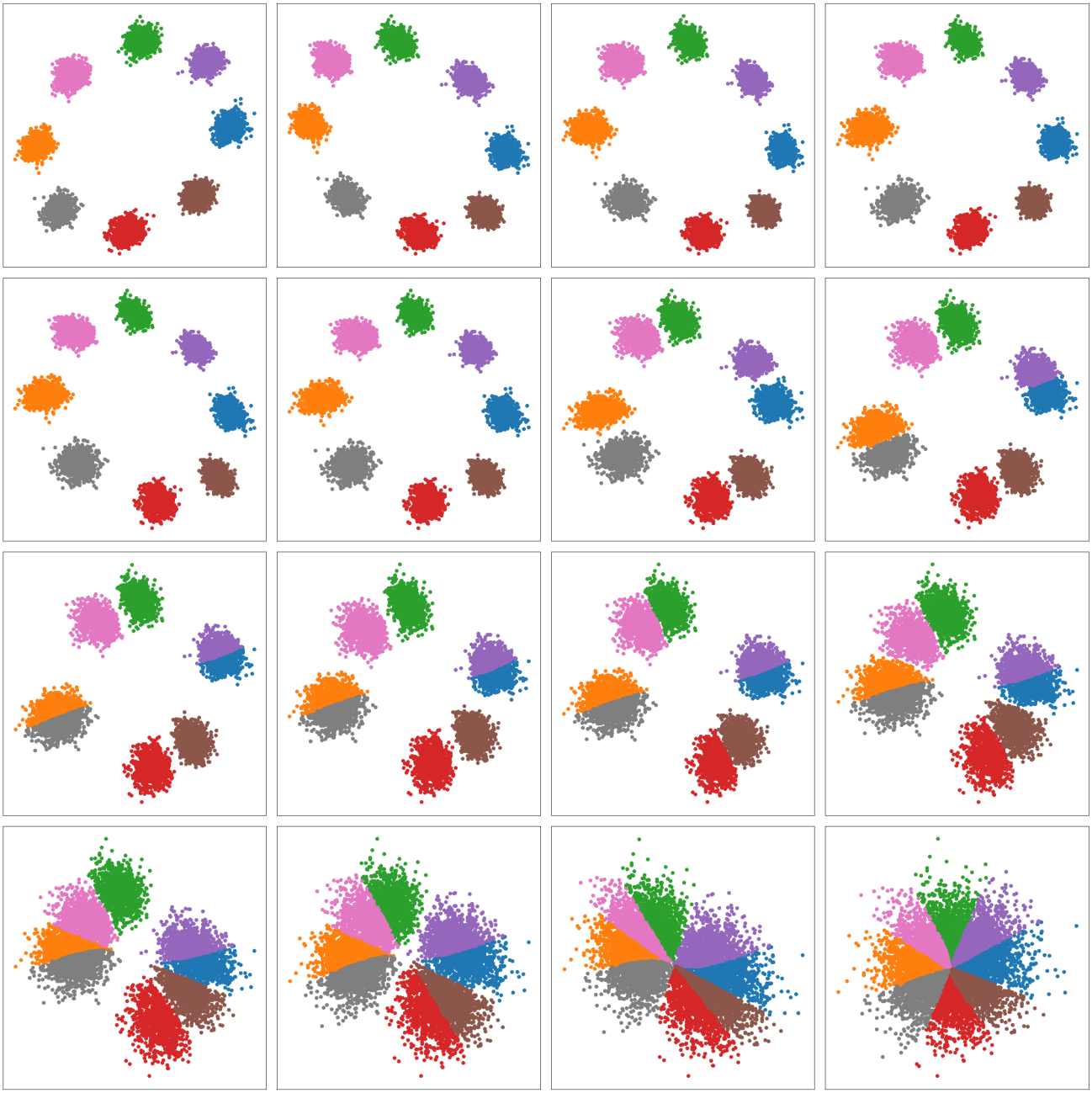

*Figure 6.* Plot of the eight Gaussians and their location after every transformation using Residual Flows.

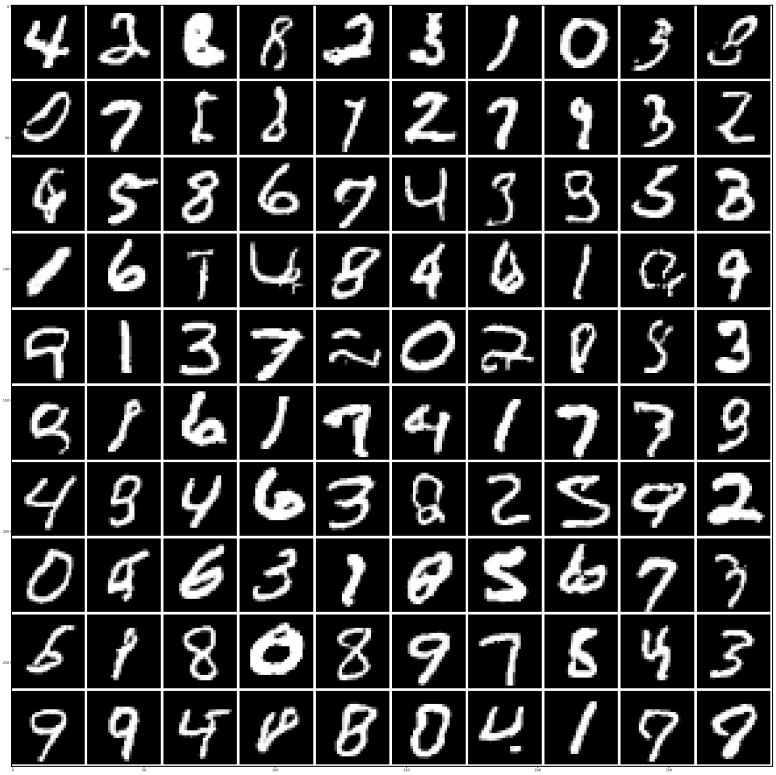

*Figure 7.* Random Samples from MNIST

*Figure 8.* Random Samples from CIFAR-10