1st Symposium on Advances in Approximate Bayesian Inference, 2018 1–14

Scalable Reversible Generative Models with Free-form Continuous Dynamics

Will Grathwohl[*†] WGRATHWOHL@CS.TORONTO.EDU
Ricky T. Q. Chen[†] RTQICHEN@CS.TORONTO.EDU
Jesse Bettencourt JESSEBETT@CS.TORONTO.EDU
Ilya Sutskever ILYASU@OPENAI.COM
David Duvenaud DUVENAUD@CS.TORONTO.EDU
University of Toronto, Ontario, Canada

Abstract

A promising class of generative models maps points from a simple distribution to a complex distribution through an invertible neural network. Likelihood-based training of these models requires restricting their architectures to allow cheap computation of Jacobian determinants. Alternatively, the Jacobian trace can be used if the transformation is specified by an ordinary differential equation. In this paper, we use Hutchinson's trace estimator to give a scalable unbiased estimate of the log-density. The result is a continuous-time invertible generative model with unbiased density estimation and one-pass sampling, while allowing unrestricted neural network architectures. We demonstrate our approach on high-dimensional density estimation, image generation, and variational inference, achieving the state-of-the-art among exact likelihood methods with efficient sampling.

Keywords: Density Estimation, Variational Inference, Neural ODEs

1. Introduction

Reversible generative models use cheaply invertible neural networks to transform samples from a fixed base distribution. Examples include NICE (Dinh et al., 2014), Real NVP (Dinh et al., 2017), and Glow (Kingma and Dhariwal, 2018). These models are easy to sample from, and can be trained by maximum likelihood using the change of variables formula. However, this requires placing awkward restrictions on their architectures, such as partitioning dimensions or using rank one weight matrices, in order to avoid an $\mathcal{O}(D^3)$ cost determinant computation.

Recently, Chen et al. (2018) introduced a continuous-time analog of normalizing flows, defining the mapping from latent variables to data using ordinary differential equations (ODEs). In their model, the likelihood can be computed using relatively cheap trace operations. A more flexible, but still restricted, family of network architectures can be used to avoid this $\mathcal{O}(D^2)$ time cost.

Extending this work, we introduce an unbiased stochastic estimator of the likelihood that has $\mathcal{O}(D)$ time cost, allowing completely unrestricted architectures. We call our approach Free-form Jacobian of Reversible Dynamics (FFJORD).

[*] Work done while at OpenAI.
[†] Equal contribution. Order determined by coin toss.

2. Background: Generative Models and Change of Variables

The change of variables formula allows one to specify a complex normalized distribution implicitly by warping a base distribution $p_{\mathbf{z}}(\mathbf{z})$ through an invertible function $f : \mathbb{R}^D \to \mathbb{R}^D$. Given a random variable $\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})$ the log-density of $\mathbf{x} = f(\mathbf{z})$ follows

$$\log p_{\mathbf{x}}(\mathbf{x}) = \log p_{\mathbf{z}}(\mathbf{z}) - \log \det \left| \frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} \right| \tag{1}$$

where $\partial f(\mathbf{z})/\partial \mathbf{z}$ is the Jacobian of f. In general, computation of the log determinant has a time cost of $\mathcal{O}(D^3)$. Much work has gone into using restricted neural network architectures to make computing the Jacobian determinant more tractable (Dinh et al., 2017; Kingma and Dhariwal, 2018; Papamakarios et al., 2017; Kingma et al., 2016; Dinh et al., 2014).

Chen et al. (2018) propose an alternative generative model which replaces the warping function with an integral of continuous-time dynamics. The generative process works by first sampling from a base distribution $\mathbf{z}_0 \sim p_{z_0}(\mathbf{z}_0)$. Then, given an ODE defined by the parametric function $f(\mathbf{z}(t), t; \theta)$, we solve the initial value problem $\mathbf{z}(t_0) = \mathbf{z}_0, \partial \mathbf{z}(t)/\partial t = f(\mathbf{z}(t), t; \theta)$ to obtain $\mathbf{z}(t_1)$ which constitutes our data. These models are called Continous Normalizing Flows (CNF). The log-density of the data under this model takes the form:

$$\log p(\mathbf{z}(t_1)) = \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \operatorname{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right) dt. \tag{2}$$

A major contribution of Chen et al. (2018) is a method to backpropagate through the solution of an ODE, known as the Adjoint-sensitivity method. We use this approach to train our proposed model.

3. Scalable Log-Density Evaluation with Unrestricted Architectures

Switching from discrete-time dynamics to continuous-time dynamics reduces the primary computational bottleneck of normalizing flows from $\mathcal{O}(D^3)$ to $\mathcal{O}(D^2)$, at the cost of introducing a numerical ODE solver. This allows the use of more expressive architectures. For example, each layer of the original normalizing flows model of Rezende and Mohamed (2015) is a one-layer neural network with only a single hidden unit. In contrast, the transformation used in continuous normalizing flows (Chen et al., 2018) is a one-layer neural network with many hidden units. In this work, we construct an unbiased estimate of the log-density with $\mathcal{O}(D)$ cost, allowing completely unrestricted neural network architectures to be used.

3.1. Unbiased Linear-time Log-Density Estimation

In general, computing $\operatorname{Tr}\left(\partial f/\partial \mathbf{z}(t)\right)$ exactly costs $\mathcal{O}(D^2)$, or approximately the same cost as D evaluations of f, since each entry of the diagonal of the Jacobian requires computing a separate derivative of f. However, there are two tricks that can help. First, vector-Jacobian products $\boldsymbol{v}^T \frac{\partial f}{\partial \mathbf{z}}$ can be computed for approximately the same cost as evaluating f, using reverse-mode automatic differentiation. Second, we can get an unbiased estimate of the trace of a matrix by taking a double product of that matrix with a noise vector:

$$\operatorname{Tr}(A) = E_{p(\boldsymbol{\epsilon})}[\boldsymbol{\epsilon}^T A \boldsymbol{\epsilon}]. \tag{3}$$

The above equation holds for any D-by-D matrix A and distribution $p(\boldsymbol{\epsilon})$ over D-dimensional vectors such that $\mathbb{E}[\boldsymbol{\epsilon}] = 0$ and $\mathrm{Cov}(\boldsymbol{\epsilon}) = I$. The Monte Carlo estimator derived from (3) is known as the Hutchinson's trace estimator (Hutchinson, 1989; Adams et al., 2018).

To keep the dynamics deterministic within each call to the ODE solver, we can use a fixed noise vector $\boldsymbol{\epsilon}$ for the duration of each solve without introducing bias:

$$\log p(\mathbf{z}(t_1)) = \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \mathrm{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right) dt = \log p(\mathbf{z}(t_0)) - \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\int_{t_0}^{t_1} \boldsymbol{\epsilon}^T \frac{\partial f}{\partial \mathbf{z}(t)} \boldsymbol{\epsilon} dt\right] \tag{4}$$

3.2. FFJORD: A Continuous-time Reversible Generative Model

Our complete method uses the efficient log-likelihood estimator of (4) to produce the first scalable and reversible generative model with an unconstrained Jacobian, leading to the name Free-Form Jacobian of Reversible Dyanamics (FFJORD). Pseudo-code of our method is given in Algorithm 1 in the Appendix.

Assuming the cost of evaluating f is on the order of $\mathcal{O}(DH)$ where D is the dimensionality of the data and H is the size of the largest hidden dimension in f, then the cost of computing the likelihood in models which stack transformations that exploit (1) is $\mathcal{O}((DH + D^3)L)$ where L is the number of transformations used. For CNF, this reduces to $\mathcal{O}((DH + D^2)\hat{L})$ for CNFs, where $\hat{L}$ is the number of evaluations of f used by the ODE solver. With FFJORD, this reduces further to $\mathcal{O}((DH + D)\hat{L})$.

4. Experiments

We demonstrate the power of FFJORD on a variety of density estimation tasks as well as approximate inference within variational autoencoders (Kingma and Welling, 2014). Experiments were conducted using a suite of GPU-based ODE-solvers and an implementation of the adjoint method for backpropagation [1]. In all experiments the Runge-Kutta 4(5) algorithm with the tableau from Shampine (1986) was used to solve the ODEs. We ensure tolerance is set low enough so numerical error is negligible; see Appendix F.

Our results are summarized in Tables 1 and 2. On density estimation FFJORD performs better than other reversible generative models on all datasets except for CIFAR10 and even outperforms state-of-the-art autoregressive models (which cannot be efficiently sampled from) on some datasets. In variational inference FFJORD outperforms all other competing normalizing flows. Full details of our experiments can be found in Appendix D.

5. Conclusion

We have presented FFJORD, a reversible generative model for high dimensional data which can compute exact log-likelihoods and can be sampled from efficiently. Our model uses continuous-time dynamics to produce a generative model which is parameterized by an unrestricted neural network. All required quantities for training and sampling can be computed using automatic-differentiation, Hutchinson's trace estimator, and black-box ODE

1. We have released our implementation of FFJORD and the adjoint method which can be found at https://github.com/rtqichen/ffjord and https://github.com/rtqichen/torchdiffeq.

	POWER	GAS	HEPMASS	MINIBOONE	BSDS300	MNIST	CIFAR10
Real NVP	-0.17	-8.33	18.71	13.55	-153.28	1.06*	3.49*
Glow	-0.17	-8.15	18.92	11.35	-155.07	1.05*	**3.35***
FFJORD	**-0.46**	**-8.59**	**15.26**	**10.43**	**-157.67**	**0.99*** (1.05†)	3.40*
MADE	3.08	-3.56	20.98	15.59	-148.85	2.04	5.67
MAF	-0.24	-10.08	17.70	11.75	-155.69	1.89	4.31
TAN	-0.48	-11.19	15.12	11.01	-157.03	-	-
MAF-DDSF	-0.62	-11.96	15.09	8.86	-157.73	-	-

Table 1: Negative log-likehood on test data for density estimation models; **lower is better**. In nats for tabular data and bits/dim for MNIST and CIFAR10. *Results use multi-scale convolutional architectures. †Results use a single flow with a convolutional encoder-decoder architecture.

	MNIST	Omniglot	Frey Faces	Caltech Silhouettes
No Flow	$86.55 \pm .06$	$104.28 \pm .39$	$4.53 \pm .02$	$110.80 \pm .46$
Planar	$86.06 \pm .31$	$102.65 \pm .42$	$4.40 \pm .06$	$109.66 \pm .42$
IAF	$84.20 \pm .17$	$102.41 \pm .04$	$4.47 \pm .05$	$111.58 \pm .38$
Sylvester	$83.32 \pm .06$	$99.00 \pm .04$	$4.45 \pm .04$	$104.62 \pm .29$
FFJORD	$\mathbf{82.82 \pm .01}$	$\mathbf{98.33 \pm .09}$	$\mathbf{4.39 \pm .01}$	$\mathbf{104.03 \pm .43}$

Table 2: Negative ELBO in nats on test data for VAE models; **lower is better**. Frey Faces is presented in bits per dimension. Mean/stdev are estimated over 3 runs.

solvers. Our model stands in contrast to other methods with similar properties which rely on restricted, hand-engineered neural network architectures. We have demonstrated that this additional flexibility allows our approach to achieve improved performance on density estimation and variational inference. We also demonstrate FFJORD's ability to model distributions which comparable methods such as Glow and Real NVP cannot model.

We believe there is much room for further work exploring and improving this method. We are interested specifically in ways to reduce the number of function evaluations used by the ODE-solver without hurting predictive performance. Advancements like these will be crucial in scaling this method to even higher-dimensional datasets.

References

R. P. Adams, J. Pennington, M. J. Johnson, J. Smith, Y. Ovadia, B. Patton, and J. Saunderson. Estimating the Spectral Density of Large Implicit Matrices. *ArXiv e-prints*, February 2018.

Joel Andersson. *A general-purpose software framework for dynamic optimization*. PhD thesis, 2013.

Rianne van den Berg, Leonard Hasenclever, Jakub M Tomczak, and Max Welling. Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649*, 2018.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018.

Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear independent components estimation. *International Conference on Learning Representations Workshop*, 2014.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. *International Conference on Learning Representations*, 2017.

M.F. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. 18:1059–1076, 01 1989.

H.K. Khalil. *Nonlinear Systems*. Pearson Education. Prentice Hall, 2002.

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

Diederik P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *arXiv preprint arXiv:1807.03039*, 2018.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *International Conference on Learning Representations*, 2014.

Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, pages 4743–4751, 2016.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *International Conference on Learning Representations*, 2018.

George Papamakarios, Iain Murray, and Theo Pavlakou. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347, 2017.

Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. Routledge, 1962.

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *International Conference on Machine Learning*, 2015.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

Lawrence F Shampine. Some practical Runge-Kutta formulas. *Mathematics of Computation*, 46(173):135–150, 1986.

Appendix A. Figures

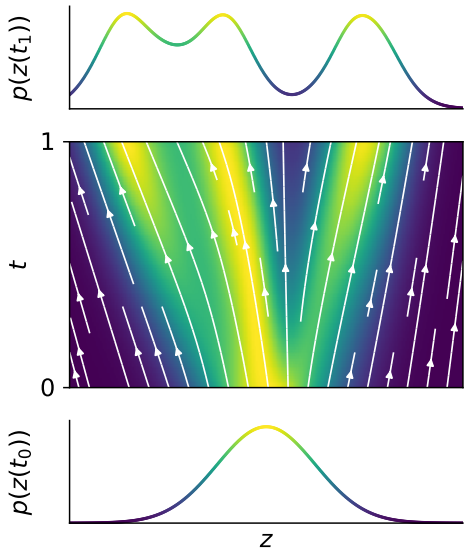

Figure 1: FFJORD transforms a simple base distribution at t_0 into the target distribution at t_1 by integrating over learned continuous dynamics.

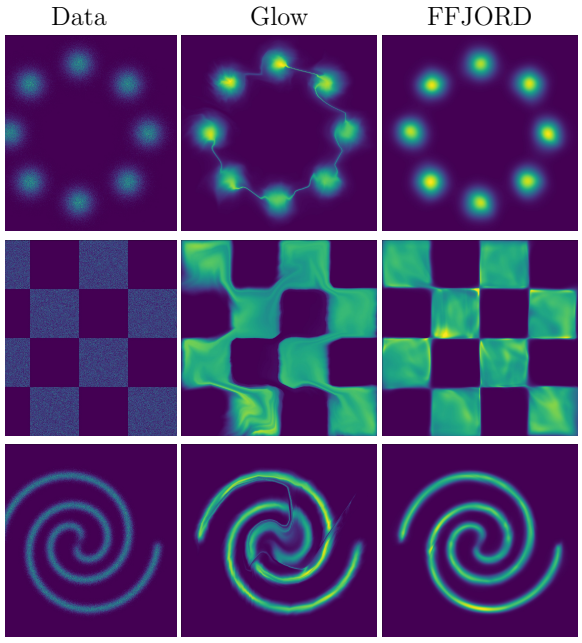

Figure 2: Comparison of trained FFJORD and Glow models on 2-dimensional distributions including multi-modal and discontinuous densities.

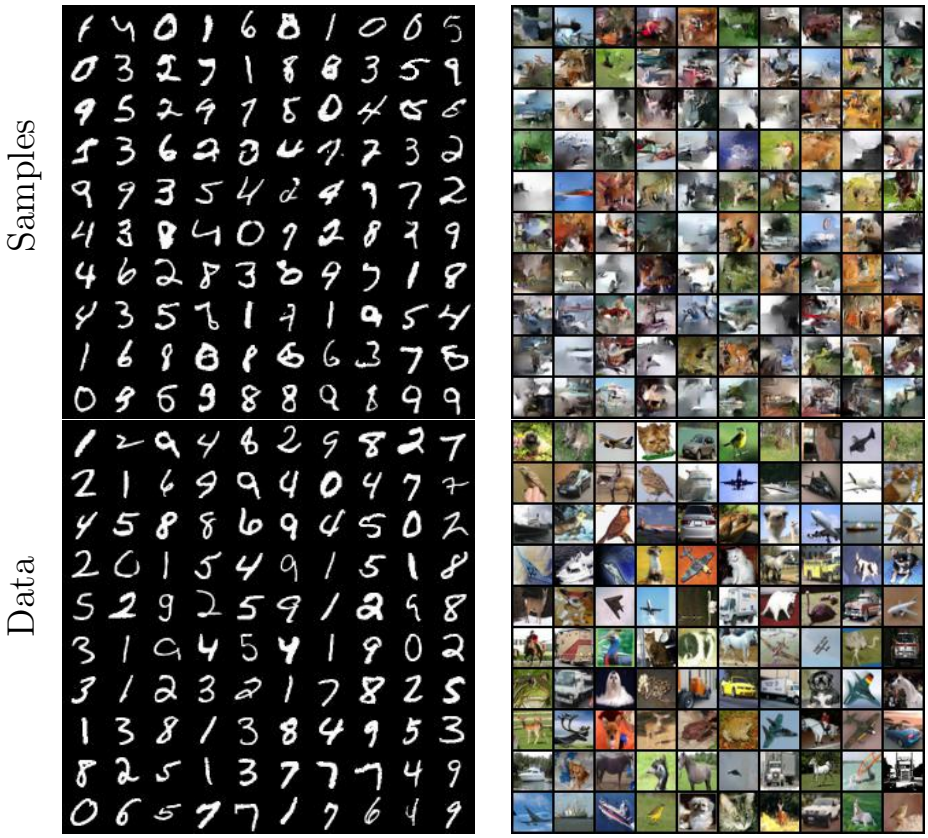

Figure 3: Samples and data from our image models. MNIST on left, CIFAR10 on right.

Appendix B. Backpropagating through ODE Solutions with the Adjoint Method

CNFs are trained to maximize (2). This objective involves the solution to an initial value problem with dynamics parameterized by θ. For any scalar loss function which operates on the solution to an initial value problem

$$L(\mathbf{z}(t_1)) = L\left(\int_{t_0}^{t_1} f(\mathbf{z}(t), t; \theta) dt\right) \tag{5}$$

Pontryagin (1962) shows that its derivative takes the form of another initial value problem

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} \left(\frac{\partial L}{\partial \mathbf{z}(t)}\right)^T \frac{\partial f(\mathbf{z}(t), t; \theta)}{\partial \theta} dt. \tag{6}$$

The quantity $-\partial L / \partial \mathbf{z}(t)$ is known as the adjoint state of the ODE. Chen et al. (2018) use a black-box ODE solver to compute $\mathbf{z}(t_1)$, and then another call to a solver to compute (6) with the initial value $\partial L / \partial \mathbf{z}(t_1)$. This approach is a continuous-time analog to the backpropgation algorithm (Rumelhart et al., 1986; Andersson, 2013) and can be combined with gradient-based optimization methods to fit the parameters θ.

Appendix C. Joint Dynamics for Continuous Normalizing Flows

Given a datapoint $\mathbf{x}$, we can compute both the point $\mathbf{z}_0$ which generates $\mathbf{x}$, as well as $\log p(\mathbf{x})$ under the model by solving the initial value problem:

$$\underbrace{\begin{bmatrix} \mathbf{z}_0 \\ \log p(\mathbf{x}) - \log p_{z_0}(\mathbf{z}_0) \end{bmatrix}}_{\text{solutions}} = \underbrace{\int_{t_1}^{t_0} \begin{bmatrix} f(\mathbf{z}(t), t; \theta) \\ -\operatorname{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right) \end{bmatrix} dt}_{\text{dynamics}}, \qquad \underbrace{\begin{bmatrix} \mathbf{z}(t_1) \\ \log p(\mathbf{x}) - \log p(\mathbf{z}(t_1)) \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ 0 \end{bmatrix}}_{\text{initial values}} \tag{7}$$

which integrates the combined dynamics of $z(t)$ and the log-density of the sample backwards in time from t_1 to t_0. We can then compute $\log p(\mathbf{x})$ using the solution of (7) and adding $\log p_{z_0}(\mathbf{z}_0)$. The existence and uniqueness of (7) require that f and its first derivatives be Lipschitz continuous (Khalil, 2002), which can be satisfied in practice using neural networks with smooth Lipschitz activations.

Appendix D. Experimental Details and Additional Results

D.1. Density Estimation

On the tabular datasets we performed a grid-search over network architectures. We searched over models with 1, 2, 5, or 10 flows with 1, 2, 3, or 4 hidden layers per flow. Since each dataset has a different number of dimensions, we searched over hidden dimensions equal to 5, 10, or 20 times the data dimension (hidden dimension multiplier in Table 4). We tried both the tanh and softplus nonlinearities. The best performing models can be found in the Table 4.

Algorithm 1 Unbiased stochastic log-density estimation using the FFJORD model

Require: dynamics f_θ, start time t_0, stop time t_1, minibatch of samples $\mathbf{x}$.
  $\boldsymbol{\epsilon} \leftarrow$ sample_unit_variance($\mathbf{x}$.shape) ▷ Sample $\boldsymbol{\epsilon}$ outside of the integral
  function $f_{aug}([\mathbf{z}_t, \log p_t], t)$: ▷ Augment f with log-density dynamics.
    $f_t \leftarrow f_\theta(\mathbf{z}(t), t)$ ▷ Evaluate dynamics
    $g \leftarrow \boldsymbol{\epsilon}^T \frac{\partial f}{\partial \mathbf{z}}\big|_{\mathbf{z}(t)}$ ▷ Compute vector-Jacobian product with automatic differentiation
    $\widetilde{\mathrm{Tr}} = \text{matrix\_multiply}(g, \boldsymbol{\epsilon})$ ▷ Unbiased estimate of $\mathrm{Tr}(\frac{\partial f}{\partial \mathbf{z}})$ with $\boldsymbol{\epsilon}^T \frac{\partial f}{\partial \mathbf{z}} \boldsymbol{\epsilon}$
    return $[f_t, -\widetilde{\mathrm{Tr}}]$ ▷ Concatenate dynamics of state and log-density
  end function
  $[\mathbf{z}, \Delta_{logp}] \leftarrow \text{odeint}(f_{aug}, [\mathbf{x}, \mathbf{0}], t_0, t_1)$ ▷ Solve the ODE, ie. $\int_{t_0}^{t_1} f_{aug}([\mathbf{z}(t), \log p(\mathbf{z}(t))], t)\, dt$
  $\log \hat{p}(\mathbf{x}) \leftarrow \log p_{\mathbf{z}_0}(\mathbf{z})$ - Δ_{logp} ▷ Add change in log-density
  return $\log \hat{p}(\mathbf{x})$

On the image datasets we experimented with two different model architectures; a single flow with an encoder-decoder style architecture and a multiscale architecture composed of multiple flows.

While they were able to fit MNIST and obtain competitive performance, the encoder-decoder architectures were unable to fit more complicated image datasets such as CIFAR10 and Street View House Numbers. The architecture for MNIST which obtained the results in Table 1 was composed of four convolutional layers with $64 \to 64 \to 128 \to 128$ filters and down-sampling with strided convolutions by two every other layer. There are then four transpose-convolutional layers who's filters mirror the first four layers and up-sample by two every other layer. The softplus activation function is used in every layer.

The multiscale architectures were inspired by those presented in Dinh et al. (2017). We compose multiple flows together interspersed with "squeeze" operations which down-sample the spatial resolution of the images and increase the number of channels. These operations are stacked into a "scale block" which contains N flows, a squeeze, then N flows. For MNIST we use 3 scale blocks and for CIFAR10 we use 4 scale blocks and let $N = 2$ for both datasets. Each flow is defined by 3 convolutional layers with 64 filters and a kernel size of 3. The softplus nonlinearity is used in all layers.

Both models were trained with the Adam optimizer (Kingma and Ba, 2014). We trained for 500 epochs with a learning rate of .001 which was decayed to .0001 after 250 epochs. Training took place on six GPUs and completed after approximately five days.

D.2. Variational Autoencoder

In VAEs it is common for the encoder network to also output the parameters of the flow as a function of the input $\mathbf{x}$. With FFJORD, we found this led to differential equations which were too difficult to integrate numerically. Instead, the encoder network outputs a low-rank update to a global weight matrix and an input-dependent bias vector. Neural

	POWER	GAS	HEPMASS	MINIBOONE	BSDS300	MNIST	CIFAR10
Real NVP	-0.17	-8.33	18.71	13.55	-153.28	1.06*	3.49*
Glow	-0.17	-8.15	18.92	11.35	-155.07	1.05*	**3.35***
FFJORD	**-0.46**	**-8.59**	**15.26**	**10.43**	**-157.67**	**0.99*** (1.05†)	3.40*
MADE	3.08	-3.56	20.98	15.59	-148.85	2.04	5.67
MAF	-0.24	-10.08	17.70	11.75	-155.69	1.89	4.31
TAN	-0.48	-11.19	15.12	11.01	-157.03	-	-
MAF-DDSF	-0.62	-11.96	15.09	8.86	-157.73	-	-

Table 3: Negative log-likehood on test data for density estimation models; **lower is better**. In nats for tabular data and bits/dim for MNIST and CIFAR10. *Results use multi-scale convolutional architectures. †Results use a single flow with a convolutional encoder-decoder architecture.

network layers inside of FFJORD take the form

$$\text{layer}(h; \mathbf{x}, W, b) = \sigma\left(\left(\underbrace{W}_{D_{out}\times D_{in}} + \underbrace{\hat{U}(\mathbf{x})}_{D_{out}\times k}\underbrace{\hat{V}(\mathbf{x})^T}_{D_{in}\times k}\right)h + \underbrace{b}_{D_{out}\times 1} + \underbrace{\hat{b}(\mathbf{x})}_{D_{out}\times 1}\right) \tag{8}$$

where h is the input to the layer, σ is an element-wise activation function, D_{in} and D_{out} are the input and output dimensionality of this layer, and $\hat{U}(\mathbf{x})$, $\hat{V}(\mathbf{x})$, $\hat{b}(\mathbf{x})$ are data-dependent parameters returned from the encoder networks. A full description of the model architectures used and our experimental setup can be found in Appendix D.2.

Our experimental procedure exactly mirrors that of Berg et al. (2018). We use the same 7-layer encoder and decoder, learning rate (.001), optimizer (Adam Kingma and Ba (2014)), batch size (100), and early stopping procedure (stop after 100 epochs of no validaiton improvment). The only difference was in the nomralizing flow used in the approximate posterior.

We performed a grid-search over neural network architectures for the dynamics of FFJORD. We searched over networks with 1 and 2 hidden layers and hidden dimension 512, 1024, and 2048. We used flows with 1, 2, or 5 steps and wight matrix updates of rank 1, 20, and 64. We use the softplus activation function for all datasets except for Caltech Silhouettes where we used tanh. The best performing models can be found in the Table 5. Models were trained on a single GPU and training took between four hours and three days depending on the dataset.

Dataset	nonlinearity	# layers	hidden dim multiplier	# flow steps	batchsize
POWER	tanh	3	10	5	10000
GAS	tanh	3	20	5	1000
HEPMASS	softplus	2	10	10	10000
MINIBOONE	softplus	2	20	1	1000
BSDS300	softplus	3	20	2	10000

Table 4: Best performing model architectures for density estimation on tabular data with FFJORD.

Dataset	nonlinearity	# layers	hidden dimension	# flow steps	rank
MNIST	softplus	2	1024	2	64
Omniglot	softplus	2	512	5	20
Frey Faces	softplus	2	512	2	20
Caltech	tanh	1	2048	1	20

Table 5: Best performing model architectures for VAEs with FFJORD.

D.3. Standard Deviations for Tabular Density Estimation

	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
Real NVP	-0.17 ± 0.01	-8.33 ± 0.14	18.71 ± 0.02	13.55 ± 0.49	-153.28 ± 1.78
Glow	-0.17 ± 0.01	-8.15 ± 0.40	18.92 ± 0.08	11.35 ± 0.07	-155.07 ± 0.03
FFJORD	-0.46 ± 0.01	-8.59 ± 0.12	15.26	10.43 ± 0.04	-157.67
MADE	3.08 ± 0.03	-3.56 ± 0.04	20.98 ± 0.02	15.59 ± 0.50	-148.85 ± 0.28
MAF	-0.24 ± 0.01	-10.08 ± 0.02	17.70 ± 0.02	11.75 ± 0.44	-155.69 ± 0.28
TAN	-0.48 ± 0.01	-11.19 ± 0.02	15.12 ± 0.02	11.01 ± 0.48	-157.03 ± 0.07
MAF-DDSF	-0.62 ± 0.01	-11.96 ± 0.33	15.09 ± 0.40	8.86 ± 0.15	-157.73 ± 0.04

Table 6: Negative log-likehood on test data for density estimation models. Means/stdev over 3 runs.

Appendix E. Reducing Variance with Bottleneck Capacity

Often, there exist bottlenecks in the architecture of the dynamics network, i.e. hidden layers whose width H is smaller than the dimensions of the input D. In such cases, we can reduce the variance of Hutchinson's estimator by using the cyclic property of trace. Since the variance of the estimator for $\mathrm{Tr}(A)$ grows asymptotic to $||A||_F^2$ (Hutchinson, 1989), we

suspect that having fewer dimensions should help reduce variance. If we view view the dynamics as a composition of two functions $f = g \circ h(\mathbf{z})$ then we observe

$$\text{Tr}\underbrace{\left(\frac{\partial f}{\partial \mathbf{z}}\right)}_{D\times D} = \text{Tr}\underbrace{\left(\frac{\partial g}{\partial h}\frac{\partial h}{\partial \mathbf{z}}\right)}_{D\times D} = \text{Tr}\underbrace{\left(\frac{\partial h}{\partial \mathbf{z}}\frac{\partial g}{\partial h}\right)}_{H\times H} = \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\boldsymbol{\epsilon}^T\frac{\partial h}{\partial \mathbf{z}}\frac{\partial g}{\partial h}\boldsymbol{\epsilon}\right]. \tag{9}$$

When f has multiple hidden layers, we choose H to be the smallest dimension. This bottleneck trick can reduce the norm of the matrix which may also help reduce the variance of the trace estimator.

Appendix F. Numerical Error from the ODE Solver

ODE solvers are numerical integration methods so there is error inherent in their outputs. Adaptive solvers (like those used in all of our experiments) attempt to predict the errors that they accrue and modify their step-size to reduce their error below a user set tolerance. It is important to be aware of this error when we use these solvers for density estimation as the solver outputs the density that we report and compare with other methods. When tolerance is too low, we run into machine precision errors. Similarly when tolerance is too high, errors are large, our training objective becomes biased and we can run into divergent training dynamics.

Since a valid probability density function integrates to one, we take a model trained on Figure 1 and numerically find the area under the curve using Riemann sum and a very fine grid. We do this for a range of tolerance values and show the resulting error in Figure 4. We set both `atol` and `rtol` to the same tolerance.

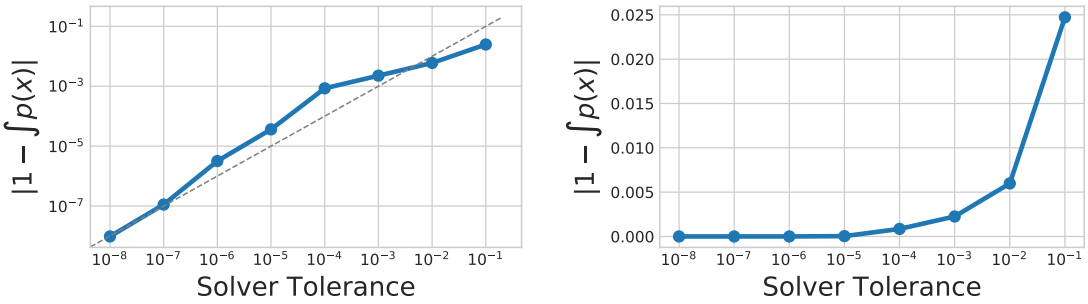

Figure 4: Numerical integration shows that the density under the model does integrate to one given sufficiently low tolerance. Both log and non-log plots are shown.

The numerical error follows the same order as the tolerance, as expected. During training, we find that the error becomes non-negligible when using tolerance values higher than 10^{-5}. For most of our experiments, we set tolerance to 10^{-5} as that gives reasonable performance while requiring few number of evaluations. For the tabular experiments, we use `atol`=10^{-8} and `rtol`=10^{-6}.

Appendix G. Analysis and Discussion

We perform a series of ablation experiments to gain a better understanding of the proposed model.

G.1. Faster Training with Bottleneck Trick

We plot the training losses on MNIST using an encoder-decoder architecture (see Appendix D.1 for details). Loss during training is plotted in Figure 5, where we use the trace estimator directly on the $D \times D$ Jacobian or we use the bottleneck trick to reduce the dimension to $H \times H$. Interestingly, we find that while the bottleneck trick (9) can lead to faster convergence when the trace is estimated using a Gaussian-distributed $\boldsymbol{\epsilon}$, we did not observe faster con distributed $\boldsymbol{\epsilon}$.

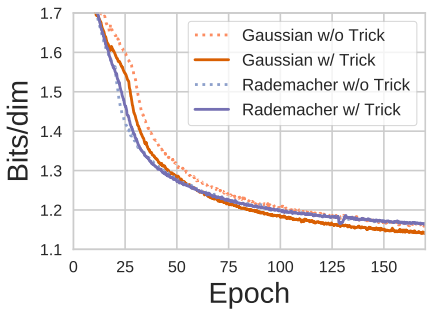

Figure 5: The variance of our model's log-density estimator can be reduced using neural network architectures with a bottleneck layer, speeding up training.

G.2. Number of Function Evaluations vs. Data Dimension

The full computational cost of integrating the instantaneous change of variables (**??**) is $\mathcal{O}(DH\widehat{L})$ where D is dimensionality of the data, H is the size of the hidden state, and $\widehat{L}$ is the number of function evaluations (NFE) that the adaptive solver uses to integrate the ODE. In general, each evaluation of the model is $\mathcal{O}(DH)$ and in practice, H is typically chosen to be close to D. Since the general form of the discrete change of variables equation (1) requires $\mathcal{O}(D^3)$-cost, one may wonder whether the number of evaluations $\widehat{L}$ depends on D.

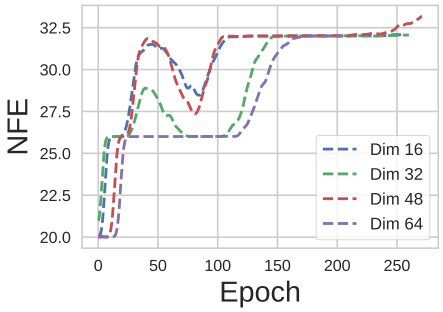

Figure 6: Number of function evaluates used by the adaptive ODE solver (NFE) is approximately independent of data-dimension.

We train VAEs using FFJORD flows with increasing latent dimension D. The NFE throughout training is shown in Figure 6. In all models, we find that the NFE increases throughout training, but converges to the same value, independent of D. This phenomenon can be verified with a simple thought experiment. Take an $\mathbb{R}^D$ isotropic Gaussian distribution as the data distribution and set the base distribution of our model to be an isotropic Gaussian. Then the optimal differential equation is zero for any D, and the number evaluations is zero. We can conclude that the number of evaluations is not dependent on the dimensionality of the data but the complexity of its distribution, or more specifically, how difficult it is to transform its density into the base distribution.

G.3. Single-scale vs. Multi-scale FFJORD

Crucial to the scalability of Real NVP and Glow is the multiscale architecture originally proposed in Dinh et al. (2017). We compare an single-scale encoder-decoder style FFJORD with a multiscale FFJORD on the MNIST dataset where both models have a comparable number of parameters and plot the total NFE–in both forward and backward passes–against the loss achieved in Figure 7. We find that while the single-scale model uses approximately one half as many function evaluations as the multiscale model, it is not able to achieve the same performance as the multiscale model.

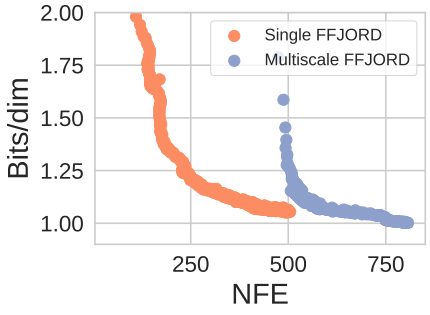

Figure 7: For image data, multiscale architectures require the ODE solver to use a greater number of function evaluations (NFE), but these models achieve better performance.

Appendix H. Scope and Limitations

Number of function evaluations can be prohibitive. The number of function evaluations required to integrate the dynamics is not fixed ahead of time and is a function of the data, model architecture, and model parameters. We find that this tends to grow as the models trains and can become prohibitively large, even when memory stays constant due to the adjoint method. Various forms of regularization such as weight decay and spectral normalization (Miyato et al., 2018) can be used to reduce the this quantity but their use tends to hurt performance slightly.

Limitations of general-purpose ODE solvers. In theory, we can model any differential equation (given mild assumptions based on existence and uniqueness of the solution), but in practice our reliance on general-purpose ODE solvers restricts us to non-stiff differential equations that can be efficiently solved. ODE solvers for stiff dynamics exist, but they evaluate f many more times to achieve the same error. We find that using a small amount of weight decay sufficiently constrains the ODE to be non-stiff.