GENERATIVE MODELLING WITH OPERATOR VALUED KERNELS: A DEEP LEARNING PERSPECTIVE

WARREN PARIS-MOE, PATRICK WYROD

Department of Applied Mathematics, University of Washington, Seattle, WA
parismoe@uw.edu, pwyrod@uw.edu

Abstract. This report documents the methodology and subsequent observations for exploring neural network-based generative models through the framework of reproducing kernel Hilbert spaces (RKHS). We begin by introducing the goals of generative modelling as a transport problem parameterized within an operator-valued RKHS. Deep generative models, the focus of our exploration, are then introduced within this context beginning with variational autoencoders (VAEs). We increase the complexity of the architecture considered –still within the aforementioned framework– to flow-based approaches, culminating in generative flow (Glow): we deem this a sufficient representative of the current state-of-the-art.

1. Introduction and Overview

One goal of unsupervised machine learning is to model the complex probability distribution present within a set data samples when its structure is unknown. Learning the underlying probability distribution enables one to generate new samples from that distribution that resemble the original set. This coins the term used to describe the class of models examined in this work, *generative modelling*. The large potential of generative models comes from their capacity to learn sophisticated representations of input data with little to no human supervision [7]. The ability to learn from large unlabelled datasets has applications to areas such as density estimation, outlier detection, prior construction, latent variable inference, and dataset summarization [9].

Throughout this report, consider:
$H = \{h_1, h_2, ...\}$**:** an observable (often referred to as just "random") variable
$X = \{x_1, x_2, ...\}$**:** a target variable
Given H and X, machine learning models can be broadly classified into 2 categories based on their goals: a *discriminative model* such as regression aims to learn a predictor through the conditional probability $\mathbb{P}(X|H = h)$, whereas a *generative model* constructs $\mathbb{P}(H|X = x)$ to model the joint distribution $p(H, X)$. As the focus of this project, henceforce, the term "model" refers to a generative model unless specified otherwise.

2. Theoretical Background

2.1. **Transport-based generative modelling.** Consider the induced probability measure $\mu \in \mathbb{P}(H)$ and a map $T : H \to X$ defining a pushforward measure $T_{\#}(\mu) \in \mathbb{P}(X)$ for some $H \subset \mathbb{R}^d$. By

Date: June 10, 2023.

fixing $\mu = \mathcal{N}(0, I)$ and considering targets $\{x_1, x_2, ...\} \overset{\text{i.i.d.}}{\sim} \nu \in \mathbb{P}(\mathbb{R}^d)$ where $X \subset \mathbb{R}^d$, the problem can be viewed as finding the transport map:

$$T : \mathbb{R}^d \to \mathbb{R}^d \quad \text{s.t.} \quad T_{\#}(\mu) \approx \nu \tag{1}$$

In order for this construction to be computationally tractable, we consider an empirical approximate measure and its corresponding empirical pushforward:

$$\mu^N = \frac{1}{N}\sum_{j=1}^{N} \delta_{h_j} \quad \Longrightarrow \quad T_{\#}(\mu^N) = \frac{1}{N}\sum_{j=1}^{N} \delta_{T(h_j)} \tag{2}$$

where $\{h_1, ..., h_N\} \overset{\text{i.i.d.}}{\sim} \mu$ are sampled observations and δ_{h_j} is the pointwise evaluation functional; ν^N is defined analogously over the target space for a set $\{x_1, ..., x_N\}$.

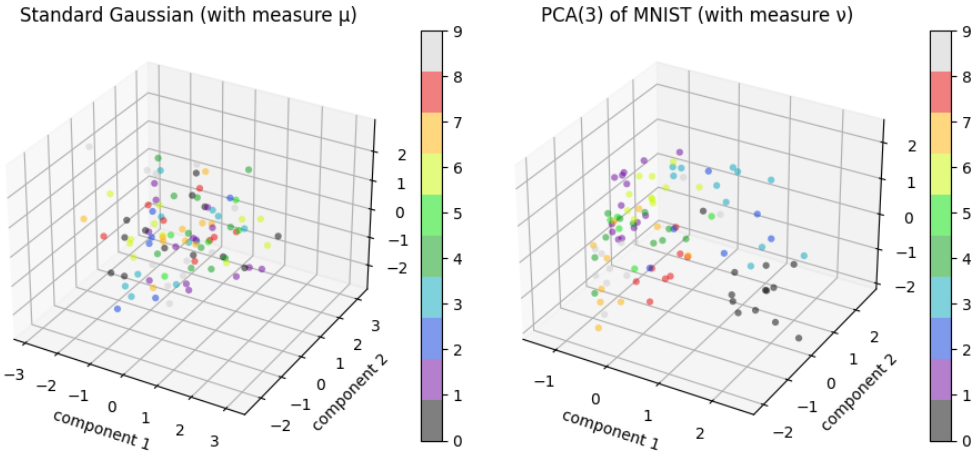


FIGURE 1. Example of empirical approximation for $\mu^{100} \approx \mathcal{N}(0, I_3)$ (Left), $\nu^{100} \approx$ PCA(3, MNIST) (Right)

Thus, $T : \mathbb{R}^d \to \mathbb{R}^d$ s.t. $T_{\#}(\mu^N) \approx \nu^N$ is a tractable approximation to the true generative model transport problem defined in eq. (1) which can be expressed as the solution to optimization problem:

$$T = \underset{S:\mathbb{R}^d \to \mathbb{R}^d}{\arg\min} D(S_{\#}\mu^N, \nu^N) \tag{3}$$

While most models optimize over a different objective function D (alongside other modifications, such as regularization), having this objective at its core defines transport-based generative modelling. D is often taken to be a statistical divergence since it serves to quantify the probabilistic "distance" from the target measure. However, other metrics such as Wasserstein distance can also fulfil this role (as show by Arjovsky and his collaborators in *Wasserstein GAN*, 2017 [1]); the flexibility of this formulation allows it to be adapted to a wide range of distributional similarity relationships and hence effective mappings.

2.2. **Deep generative modelling.** The goal of generative models is to learn the true underlying distribution of a dataset, allowing for the generation of new data points from this learned distribution. They have been approached from various angles, with the development of various neural network-based generative models such as variational autoencoders (VAEs) [8], generative adversarial networks (GANs) [4], and normalizing flows (NFs) [9] being among the most prominent of last decade.

Neural networks are the defining component of deep learning models. Their fundamental unit is a layer $\sigma(hw)$ which applies an activation function σ to a weighting w of its input h; the resultant network NN from a composition of layers is the primary learning mechanism of deep models:

$$NN : \mathbb{R}^d \to \mathbb{R}^d : h \mapsto \sigma_1(...(\sigma_{L-1}(\sigma_L(hw^{(L)})w^{(L-1)})...)w^{(1)})$$

where $w^{(l)}$ is the weight applied at layer $l \in \{1, ..., L\}$.

To paramaterize our transport problem within an RKHS, we begin with an example illustrating the correspondence to a radial basis function (RBF) kernel. First, consider the trivial case $d = 1, l = 1$:

$$NN(h) = \sigma_1(hw) = \exp\left(-\frac{\|h - w\|^2}{2\gamma^2}\right) \tag{4}$$

where γ is the kernel width and w parameterizes a reference point (center) in the network's input space. This is simply the RBF kernel in $\mathbb{R}$.

Extending this single-layer "network" to $\mathbf{h} \in \mathbb{R}^d$, $\mathbf{w} \in \mathbb{R}^d$:

$$NN(\mathbf{h}) = \sigma_1(\mathbf{hw}) = \exp\left(-\frac{\|\mathbf{h} - \mathbf{w}\|^2}{2\gamma^2}\right) \tag{5}$$

manifests as the notion of an elementwise *operator-valued* kernel as an *RBF network*, with each node being an independent instance of the 1-dimensional precursor from eq. (4). The RKHS is correspondingly defined elementwise.

If we instead fix $d = 1$ and extend the trivial network's (from eq. (4)) depth to an arbitrary $l = L$:

$$NN(h) = \sigma_1(...(\sigma_{L-1}(\sigma_L(hw^{(L)})w^{(L-1)})...)w^{(1)}) = k(...(k(k(h, w^{(L)}), w^{(L-1)})...), w^{(1)}) \tag{6}$$

The final output of this composition bears little semblance to that of the RBF kernel at the first layer, $\sigma_L(hw^{(L)})$, as the input space for each successive mapping is an RBF feature space rather than the observed input space H. However, denoting the input to layer l as $\hat{h}_{l-1}$, we see that

$$\sigma_l(\hat{h}_{l-1}w^{(l)}) = \exp\left(-\frac{\|\hat{h}_{l-1} - w^{(l)}\|^2}{2\gamma^2}\right)$$

remains a valid kernel with respect to its input space $\forall l \in \{1, ..., L\}$, and in particular for $l = 1$ (the final output).

Extending the RBF kernel to a neural network of both arbitrary dimension and depth is a substantially more complex problem than each independently. If we add a second layer to the RBF network from eq. (5):

$$NN(\mathbf{h}) = \sigma_1(\sigma_2(\mathbf{h}\mathbf{W}^{(2)})\mathbf{W}^{(1)}) = k \circ \left(k(\mathbf{h}, \mathbf{W}^{(2)}),\ \mathbf{W}^{(1)}\right)$$

we lose the simplification of elementwise kernels and must define $\mathbf{W}^1, \mathbf{W}^2 \in \mathbb{R}^{d\times d}$ if we wish to account for all connections and generalize to a fully-connected network. If we restrict the network to have uniform dimension across all layers by enforcing a diagonal $\mathbf{W}^{(1)}$ prior to the second activation, we are effectively severing all but 1 connection from each node in the input layer to 1 other node in the subsequent layer. In doing so this network may be extended in a similar manner as with the scalar kernel in eq. (6), albeit with a substantially more complicated input space for the final layer:

$$NN(\mathbf{h}) = \sigma_1(\hat{\mathbf{h}}_2\mathbf{w}^{(1)})$$

Alternatively, since the RBF kernel generalizes to vector-valued inputs, we can express each layer elementwise as:

$$(\sigma_1(\hat{\mathbf{h}}_2))_j = \exp\left(-\frac{\|\hat{\mathbf{h}}_2 - \mathbf{W}_{j,:}\|^2}{2\gamma^2}\right), \quad j = 1, ...d$$

This preserves the fully-connected structure and therefore generalizes the notion of a kernel (and corresponding RKHS) to the broad class of feed-forward networks. The only assumption for this scalar RBF kernel example is its validity as a kernel for vector-valued inputs. Thus, this extension can be made for any function (or combination of functions) satisfying the definition of a kernel over $\mathbb{R}^d$. So long as weighting severs connections accordingly to enforce the definition of an operator-valued kernel, this generalization can be extended to include kernels only valid over $\mathbb{R}^{d'}, d' < d$, as well as changes in dimension between layers. Consequently, the resultant hypothesis space is unfathomably intricate while encapsulating the much simple hypothesis spaces of each component kernels.

3. Algorithm Implementation and Development

3.1. **Variational autoencoders.**

As we transition from an arbitrary neural network to the specific architectures underlying deep generative models, consider the following notation:

$\tilde{p}_\theta(x, h)$**:** generative model parameterized by θ.

$\tilde{q}_\phi(h|z)$**:** recognition model parameterized by ϕ. This supports the generative model within a cohesive model structure.

With the relationship between kernel functions and neural networks established, our first example of a generative model utilizing this architecture to construct its hypothesis space is the VAE. This is a 2-phase model composed of two neural networks:

Encoder: $NN : X \to H$; learns an effective latent representation of the *target variable*
Decoder: $NN : H \to X$; learns an effective input space representation of the *observed variable*

An inferential model is obtained through a backpropagation-optimized training process on the evidence lower bound (ELBO) loss, commonly expressed as:

$$L^{\text{ELBO}}_{\theta,\phi}(x) = \ln \tilde{p}_\theta(x) - D_{KL}(\tilde{q}_\phi(\cdot|x) \parallel \tilde{p}_\theta(\cdot|x))$$

Notably, joint training of the encoder and decoder is enforced through the Kullback-Leibler (KL) divergence term; the mapping is each direction is learned simultaneously.

Framed in the context of the transport problem defined in eq. (1), the latent space learned by the encoder is the distribution $\mu = \mathcal{N}(0, I)$ and the input space is our target distribution $\nu \in \mathbb{P}(\mathbb{R}^d)$. At inference, the decoder aligns with the approximation derived in eq. (2) for any number of samples N. However, it must be considered jointly with the encoder for training. We reconcile this by considering the decoder as our transport map and the encoder as supporting structure, allowing us to express the canonical form of its training procedure from section 3.1 in the form of the optimization problem defined in eq. (3):

$$T^{\text{VAE}} = \underset{S:\mathbb{R}^d\to\mathbb{R}^d}{\arg\min} \Big(D_{KL}(S(\mu^N)||\nu^N) - \mathbb{E}_{x\sim S(\mu^N)}[\log \tilde{p}(x)]\Big)$$

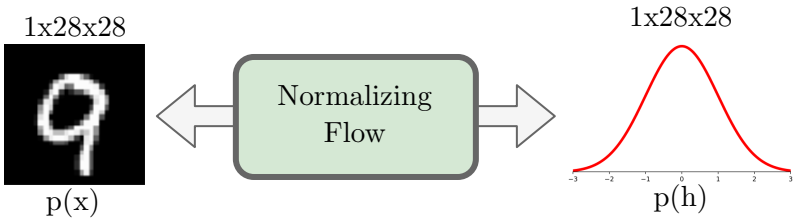


FIGURE 2. Directions of "flow" between the unknown MNIST distribution $p(x)$ (left) and the base distribution $p(h$ (right)

3.2. **Normalizing Flows.** Flow-based models are a subclass of deep generative modelling that can transform a simple base distribution (i.e., Gaussian, logistic, etc.) into a complex one via a sequence of invertible transformations. Distinct from other types of generative modelling, flow-based models can explicitly learn an unknown data distribution $\mathbb{P}(X)$.

3.2.0.1. Basics. Given a random variable h with a known probability density function $h \sim p_H(h)$, the goal is to find a bijective mapping $h = f(x)$ where f is invertible and differentiable while h has the same dimensions as x. Applying the multivariate change of variables theorem, we can calculate the probability density function of the target variable X:

$$p_X(x) = p_H(f(x))|\det \frac{\partial f(x)}{\partial x}|. \tag{7}$$

Here, $\frac{\partial f(x)}{\partial x}$ is the Jacobian matrix of f at x. If we choose our function f such that the determinant of its Jacobian and its inverse f^{-1} can be trivially obtained, then we can easily sample from $p_X(x)$ with

$$\begin{aligned} h &\sim p_H(h), \\ x &= f^{-1}(h) \end{aligned} \tag{8}$$

which forms a map from $H \to X$. The inverse function f^{-1} acts as a generating function, pushing the the base prior density p_H towards the more complex one. This is called moving in the *generative direction*.

The function f "flows" in the *normalizing direction* opposite to the generative direction (i.e., from a complex distribution to a simpler, more "normal" distribution of the prior p_H). This gives rise to the name "normalizing flows", especially if using a *normal* prior distribution [9]. Then, with an arbitrarily complex function f, we can theoretically model and generate any distribution p_X from any prior p_H assuming reasonable conditions on the two distributions are met [9, 7, 3].

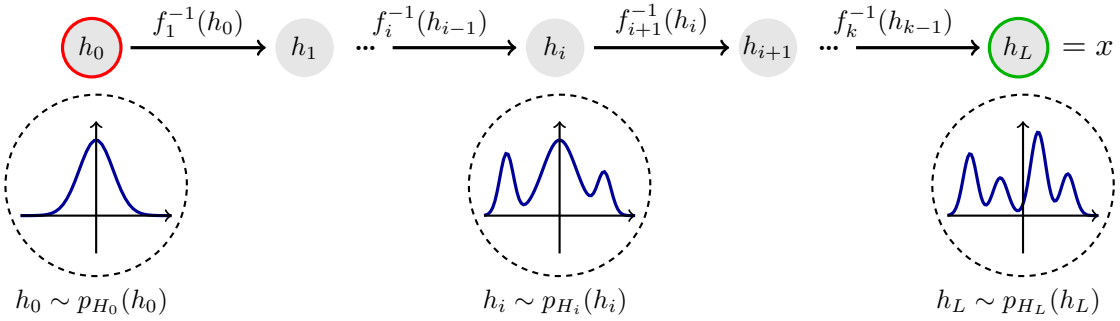


FIGURE 3. Normalizing flow structure for producing new samples.

Building arbitrarily complex functions which are likely nonlinear and are also bijections proves to be a difficult task. This is especially true as we need their inverses and Jacobian determinants

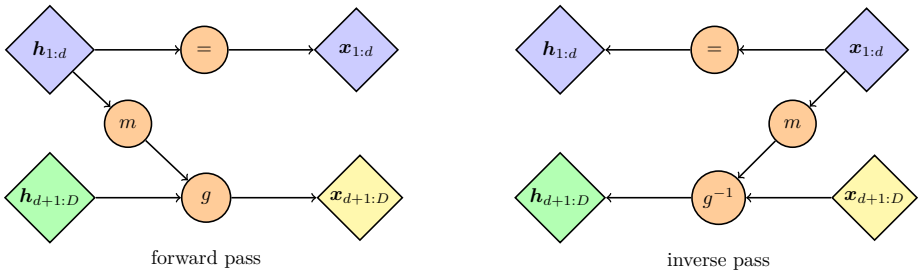


FIGURE 4. General architecture of a coupling layer in NFs.

to be easy to compute. A solution that normalizing flows use for this problem is the idea that the nested composition of a set of invertible function is itself invertible and its Jacobian has a known structure. If $f_1, ..., f_L$ are a set of L bijective functions such that their composition is given by $f = f_L \circ f_{L-1} \circ \cdots \circ f_1$, then f is also bijective with inverse

$$f = f_1^{-1} \circ \cdots \circ f_{L-1}^{-1} \circ f_L^{-1}.$$

The transformations are designed to be easily invertible and have easily computable Jacobian determinants, which makes it possible to compute the likelihood of the data under the model.

$$NF : \mathbb{R}^d \to \mathbb{R}^d : h \mapsto f_1^{-1} \circ \cdots \circ f_{L-1}^{-1} \circ f_L^{-1}(h).$$

A key question arises from the previous section: How does one select a transformation f such that the determinant of the Jacobian and the inverse are "easily" obtained while still allowing for complex behaviors of data to be discovered? The component used by most normalizing flow models to achieve such a feat is known as a coupling layer. The technique works by splitting the input x into two blocks (x_1, x_2) and apply a transformation from (x_1, x_2) to our layer output (y_1, y_2) as follows:

$$y_1 = x_1$$
$$y_2 = g\left(x_2; m\left(x_1\right)\right)$$

where $g : \mathbb{R}^{d/2} \times m\left(\mathbb{R}^{d/2}\right) \to \mathbb{R}^{d/2}$ is the coupling law, an invertible map with respect to its first argument given the second. The resulting structure is shown in Figure 4. Techniques used to permute the split indices of x_1 and x_2 vary across NF implementations [3, 7]. Regardless, the advantage from leaving half the inputs untouched is shape of the Jacobian produced:

$$\frac{\partial y}{\partial x} = \begin{bmatrix} I_{d/2} & 0 \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{bmatrix}$$

Note that $I_{d/2}$ is the identity matrix of size $d/2$. By the calculating the determinant of this matrix, we arrive at $\det \frac{\partial y}{\partial x} = \det \frac{\partial y_2}{\partial x_2}$. The triangular nature of our Jacobian makes the computation of its inverse and determinant significantly cheaper. Taking the inverse of our coupled transformations, we have our *coupling layer* with *coupling function* m:

$$x_1 = y_1$$
$$x_2 = g^{-1}\left(y_2; m\left(y_1\right)\right)$$

The factorized prior p_{H_i} pushes the NF model to find features of importance over a set training samples. The determinant of the Jacobian term penalizes contradiction and supports the expansion of high density regions present within the dataset [3].

A key aspect of normalizing flows is that the exact log-likelihood of input data becomes tractable. Say our model is made up of a flow f_θ parameterized by θ. Additionally, assume we know the base measure p_H which is parameterized by ϕ. Given our target variable with some complicated

distribution $X \subset \mathbb{R}^d$, we conduct likelihood-based estimation of our parameters $\Theta = (\theta, \phi)$ by taking the logarithm of eq. (7) with our density functions relive to Θ:

$$\log p(X|\Theta) = \sum_{x \in X} \log p_X(x|\Theta) = \sum_{x \in X} [\log p_H(f(x|\theta)|\phi) + \log |\det \frac{\partial f(x)}{\partial x}|].$$

Maximizing the exact log-likelihood is a more difficult problem, so the training objective of flow-based generative models is simply the negative log-likelihood (NLL) over the input training dataset (either a sum or an average can be used):

$$\mathcal{L}_\theta^{\text{NLL}}(X) = \frac{1}{d} \sum_{x \in X} -\log p_X(x|\Theta).$$

We then can formulate this in terms of eq. (3):

$$T^{\text{NLL}} = \arg\min_{\Theta=(\theta,\phi)} \mathcal{L}_\theta^{\text{NLL}}(X)$$

3.2.0.2. *NICE: Nonlinear Independent Component Analysis*. Dinh et al. tackle the permutation and coupling problems with one of the simpler methods present in the literature on normalizing flows [3]. The permutation is made easy by splitting an input x into odd x_1 and even x_2 indices. Then when coupling layers are chained together, each subsequent layer flips the subset that is passed with the identity function. Additive coupling layers are employed by using an additive coupling law $g(a; b) = a + b$ which makes it so that calculating the inverse transformation is only as costly as the original transformation. For NICE, this gives us two operations of importance:

$$x_1 = y_1$$
$$x_2 = g^{-1}(y_2; m(y_1))$$

where m is a small ReLU network with $d/2$ input units and $d/2$ output units. Finally, a diagonal scaling matrix S is used to multiply the i^{th} output by S_{ii} which helps preserve a unit Jacobian and gives more weight to dimensions of importance.

3.2.0.3. *Glow: Generative Flow with Invertible* 1×1 *Convolutions*. Kingma and Dhariwal approach the permutation problem by using invertible 1×1 2d convolutions to reverse/rotate the ordering of indices [7]. This uses the fact that a 1×1 convolution with the same channel dimension for both input and outputs can function as a permutation operation [6].

The input must be structured slightly differently to apply convolutional layers than previous models discussed. Rather, we add a channels dimension c to our input resulting in tensors of size $c \times h \times w$. Then for an input $\mathbf{h}$ with $c \times c$ weight matrix $\mathbf{W}$, the determinant of the Jacobian for the invertible 1×1 convolution simplifies to

$$\log \left| \det \left(\frac{d\,\text{conv2D}(\mathbf{h}; \mathbf{W})}{d\mathbf{h}} \right) \right| = h \cdot w \cdot \log |\det(\mathbf{W})|.$$

This allows for Glow to add another dimension to the data for extracting information without imposing too much complexity on the model. Activation normalization layers are used to perform an affine transformation of the activation functions using an additional scale and bias parameter for each channel. This functions similar to batch normalization except the performance/memory overhead is reduced since, especially for large images. Glow uses an affine coupling law $g(a; b) = a \odot b_1 + b_2, b_1 \neq 0$ for $m : \mathbb{R}^{d/2} \to \mathbb{R}^{d/2} \times \mathbb{R}^{d/2}$ [7, 3, 6]. Splits are performed along the channel dimension in order to simplify the structure of the neural network.

4. Computational Results

4.1. **Setup and Deployment.** The PyTorch deep learning library [10] contains all necessary functionality to implement, train, and infer from the models introduced in the preceding section 3. Our implementation of each is fully subsistent within this framework and deployed as follows:

Data: All data is sourced from MNIST in the form provided by Bamdad Hosseini in AMATH 563 HW2 [5]: 60,000 samples across 784 dimension with labels. It is ingested into a `Dataset` and processed as a `Dataloader`. We employ standard data handling practices to ensure the integrity of our results, including:
(1) Separate directories for our train, test, and validation sets.
(2) Shuffling of train and test sets.

Instantiation: Each architecture is defined as `nn.Module` subclass

Training: Models are trained using standard optimizers from `torch.optim` with differentiation handled by `torch.autograd`

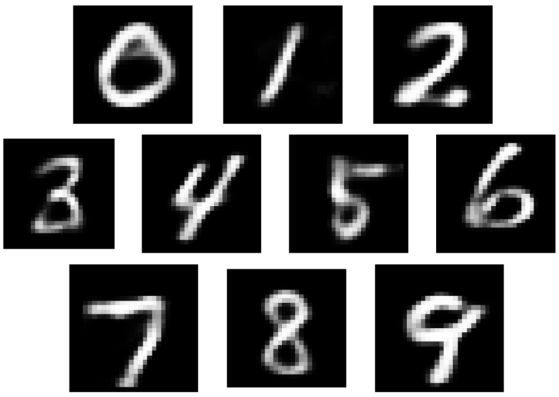
4.2. **Evaluation.** Unlike their discriminative counterparts, generative models do not necessarily allow for straightforward evaluation due to the unsupervised nature of the task. Since training loss is a function of both the model and data and we are considering models with completely different transport optimization objectives, little meaningful insight can be gained from its analysis. While several quantitative metrics have been proposed for generative model evaluation –such as inception score and Frechet inception distance– their adoption is not universal. A study conducted on the aforementioned 2 by Bezalel et. al. (2022) [2] demonstrates a lack of consistency among them in addition to highlighting the meaninglessness of their values. Thus, considering the simplicity and human-interpretability of our data (MNIST) in conjunction with the criticisms of quantitative metrics, we opted for the ubiquitous evaluation technique of visual inspection.

The prior distribution chosen for a normalizing flow model is typically selected such that it is factorial with independent components. NICE was tested with two of the most common distributions that fall in this class: standard Gaussian distribution and logistic distribution. The Gaussian distribution produced qualitative results of less quality to visual inspection with worse performance than when using the logistic prior. This likely follows from the fact that the logistic distribution tends to provide better behaved gradients. A set of 100 randomly generated samples are shown in Figures 6a and 6b after 10 and 111 training epochs, respectively. Similarly, 100 generated new samples from the Glow model are shown in Figures 7a and 7b after 10 and 42 training epochs.

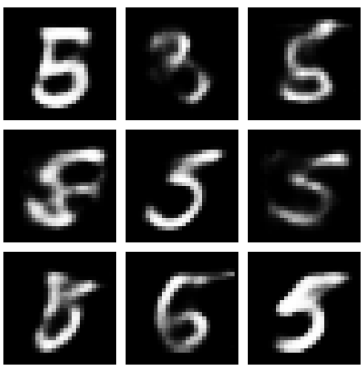
5. Discussions

Due to the complexity of these models and the lack of meaningful comparative basis between their configuration-specific parameters, we only mention basic configuration details as additional

Figure 5. **VAE**
Trained over 10 epochs with one-hot label encoding (a.k.a. "conditional VAE")
Layer structure: $[784 + 10] \to [400] \to [20 + 10]$

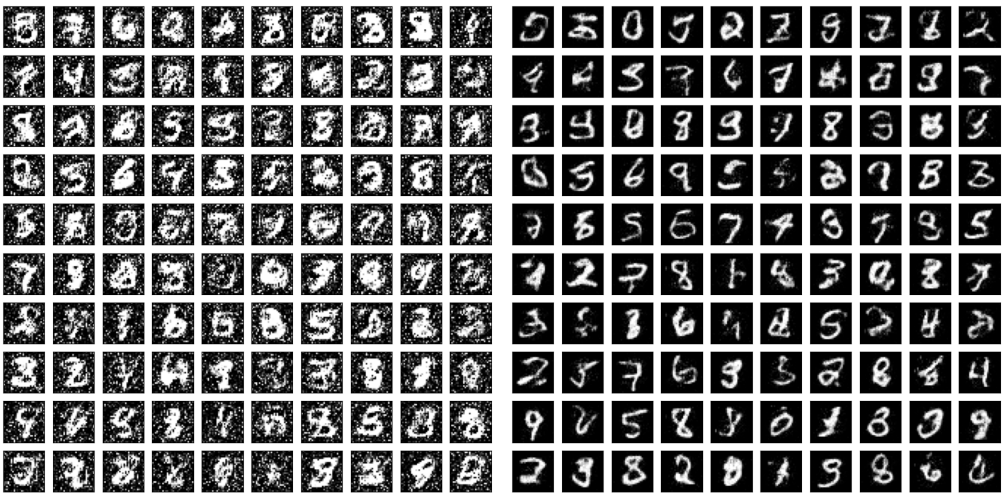


(a) First sample generated for each label



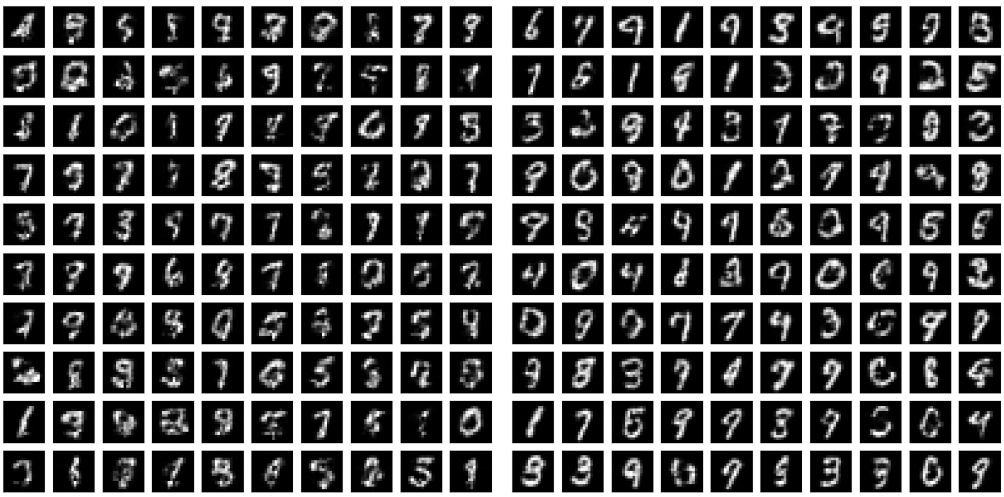
(b) First 9 samples generated for label=5

FIGURE 6. **NICE**: Model trained with a batch size of 32 and learning rate of 1e-3 using the AdaM optimizer. MNIST samples generated after n training epochs:



(A) 10 training epochs. (B) 111 training epochs.

FIGURE 7. **Glow**: Model trained with a batch size of 64 and learning rate of 1e-4 using the AdaM optimizer. MNIST samples generated after n epochs: Generated MNIST samples after:



(A) 10 training epochs. (B) 42 training epochs.

context to complement the two pertinent bases of comparison: visual evaluation and input data (as epochs over the same trainset). We emphasize the following observations:

(1) Considering the results from kernel regression in AMATH 563 HW2 [5] as a baseline for the digit class' relative difficulty, it appears the simplest deep model considered –the VAE– retains the relative class difficulties to a greater extent than the more intricate NICE model; the center entries of the 1st and 3rd row of 5b appear to show boundaries between 2 classes a human would deem most similar to 5. This suggests its hypothesis space more uniformly encapsulates the distribution of MNIST as a whole.

(2) The apparent relative differences between each models' hypothesis space can be inferred by observing patterns in the weakest samples of each. The VAE is not shy to display pixels to display pixels it has confidence in as bold white and black, with a blur visible at the more difficult features (such as the sharp edge-to-curve transitions with 5 and 3). The NICE model displays a unique distinct pattern across all samples compared to the VAE. Its greater difficulty may suggest a less-humanlike learning mechanism. The Glow samples appear more simiar to the VAE due to their bold black background, however the regions surrounding sharp transitions display less-humanlike representation than the blurring pattern of the VAE.
(3) The observed decrease in visual quality between NICE and Glow for 10 epochs in significant. This likely has to do with the convolution layers adding another dimension to the data allowing for additional information to be discover. Also, Glow was able to produce decent looking results after just 42 epochs while NICE took 111 epochs. We also that the MNIST images fed to Glow needed to be scaled down from 28×28 to 16×16, further showing the impressiveness relative to NICE.
(4) While time constraints made it difficult to conduct rigorous comparisons of computation time, the resulting visuals with respect to training epochs show that the attains competitive performance over fewer VAE runs of the data. This, combined with training times per epoch separated by 1-2 orders of magnitude and the results above demonstrate that a greater capacity does not scale linearly with computational demand.

Overall, exploring modern deep generative models within the constraints of a transport map parameterized within an RKHS was challenging due to such models being founded primarily on empirical study, but our understanding and appreciation of the theory behind them was greatly enchanced as a result. Implementing these models –particularly Glow– was also challenging due to its novelty and intricacy.

Acknowledgements

We would like to extend our sincere thanks and gratitude to Dr. Bamdad Hosseini (University of Washington) and Dr. Martha White (University of Alberta), whose lectures and unpublished lecture notes from Spring 2023 and Fall 2021, respectively, proved invaluable to developing the intuition necessary to pursue this project. We also wish to acknowledge our use of OpenAI's ChatGPT –the inference interface of a generative model– for the LaTeX formatting of this document.

References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan, 2017.
[2] E. Betzalel, C. Penso, A. Navon, and E. Fetaya. A study on the evaluation of generative models, 2022.
[3] L. Dinh, D. Krueger, and Y. Bengio. NICE: Non-linear Independent Components Estimation, Apr. 2015.
[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. pages 2672–2680, 2014.
[5] B. Hosseini. *AMATH 563 Lecture Notes*. 2023.
[6] A. Kallappa, S. Nagar, and G. Varma. FInC Flow: Fast and Invertible $k \times k$ Convolutions for Normalizing Flows, Jan. 2023.
[7] D. P. Kingma and P. Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions, July 2018.
[8] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
[9] I. Kobyzev, S. J. Prince, and M. A. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE transactions on pattern analysis and machine intelligence*, 43(11):3964–3979, 2020.
[10] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.