Neural Inverse Transform Sampler

Henry Li [1] **Yuval Kluger** [1]

Abstract

Any explicit functional representation f of a density is hampered by two main obstacles when we wish to use it as a generative model: designing f so that sampling is fast, and estimating $Z = \int f$ so that $Z^{-1}f$ integrates to 1. This becomes increasingly complicated as f itself becomes complicated. In this paper, we show that when modeling one-dimensional conditional densities with a neural network, Z can be exactly and efficiently computed by letting the network represent the cumulative distribution function of a target density, and applying a generalized fundamental theorem of calculus. We also derive a fast algorithm for sampling from the resulting representation by the inverse transform method. By extending these principles to higher dimensions, we introduce the **Neural Inverse Transform Sampler (NITS)**, a novel deep learning framework for modeling and sampling from general, multidimensional, compactly-supported probability densities. NITS is a highly expressive density estimator that boasts end-to-end differentiability, fast sampling, and exact and cheap likelihood evaluation. We demonstrate the applicability of NITS by applying it to realistic, high-dimensional density estimation tasks: likelihood-based generative modeling on the CIFAR-10 dataset, and density estimation on the UCI suite of benchmark datasets, where NITS produces compelling results rivaling or surpassing the state of the art.

1. Introduction

Building efficient, highly expressive generative density estimators has been a long-standing challenge in machine learning. A sufficiently powerful estimator will have far reaching effects on a bevy of inference tasks, ranging from classification to generative modeling to missing value imputation.

Density estimation of continuous random variables revolves around a fundamental trade off between expressivity and tractability, which is most concretely related to the computation of the *partition function* $Z_\theta = \int_\Omega f_\theta$ for a positive function $f_\theta : \Omega \to \mathbb{R}_+$. The general rule of thumb: the more flexible the underlying f_θ, the more difficult the estimation of Z_θ.

For instance, the most general model is perhaps the energy-based model (EBM), where the desired density ν is approximated via an energy landscape — for example, the Boltzmann (or Gibbs) distribution

$$\nu_\theta = \frac{e^{-f_\theta(x)}}{\int_\Omega e^{-f_\theta(x)} dx}, \tag{1}$$

where f_θ may be an arbitrary function. The problem with this formulation is that analytic integration of functions of general form is unknown, and the cost of numerical integration techniques scales exponentially with the dimension of Ω. Thus there are few, if any, approaches that represent an arbitrary f_θ in high dimension, though some methods may estimate f_θ via a variational lower bound (Sohl-Dickstein et al., 2015; Du & Mordatch, 2019).

On the other hand, one may obtain Z_θ through direct analytic derivation — but this is only applicable for a small handful of distributions. (Theis et al., 2012) and (Uria et al., 2016), for example, perform density estimation via Gaussian mixture and Gaussian scale mixture models, which both have well-known partition functions.

Ultimately, most approaches avoid explicit estimation of the density. Normalizing flows (Dinh et al., 2014; Rezende & Mohamed, 2015) learn the transform between the desired density and a reference density (usually Gaussian), and compute ν via the probabilistic change-of-variables formula, which does not require Z, while score-based (Song et al., 2020) models work with the gradient of the log-likelihood, which again does not depend on the partition function. These techniques for sidestepping the computation of Z each have drawbacks, which we will explore in Section 4.

Directly approximating a pdf ν with a parametric function

[1]Department of Applied Mathematics, Yale University, New Haven, CT. Correspondence to: Henry Li <henry.li@yale.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

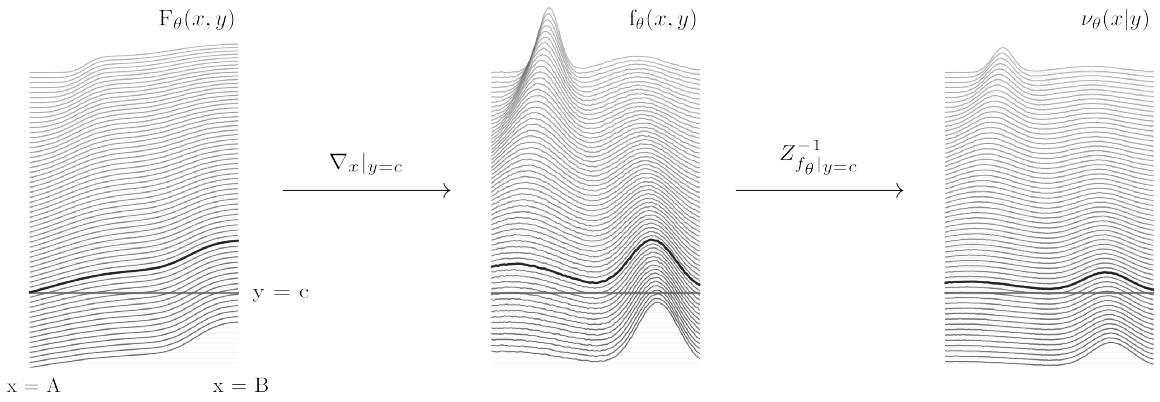

Figure 1. The *integration trick*, used here to enable direct representation of a conditional density $\nu_\theta(x|y = c)$. The original neural network (left) is differentiated w.r.t. x (middle), then rescaled by the partition function (right), which we compute directly by evaluating F_θ at the boundaries of $[A, B]$. All operations are restricted to the line $y = c$ (for more details, see Sections 2.1 and 2.3).

f_θ has remained an elusive task. This is also the approach we take on with the proposed Neural Inverse Transform Sampler (NITS). NITS is a deep neural network augmented by two basic ideas: fast integration via the **gradient theorem**, and fast sampling via the **inverse transform method**.

In the one-dimensional case, we leverage the fundamental theorem of calculus to normalize a neural network's output $F_\theta(x)$ so that it is the cumulative distribution function (cdf) of a 1D probability distribution over a bounded interval $[A, B]$. This also enforces the neural network derivative $\nabla_x F_\theta(x)$ to be the corresponding probability density function (pdf). Fast evaluation of the cdf means fast sampling from the induced distribution using the inverse transform method. Furthermore, the parameters of the distribution can be trained via gradient descent, as the framework is end-to-end differentiable.

This idea is straightforwardly extended to higher dimensions, where a multi-dimensional probabilistic model is composed of multiple one-dimensional models whose statistics are computed either in parallel or in sequence, according to assumptions on the correlation structure of the random variables. See Figure 1 for a graphical breakdown. In Section 2 we explore these ideas in greater detail.

Our Contributions In this work, we demonstrate two novel computational ideas: a method for obtaining cheap and exact integrals of 1D neural network functions which we call the *integration trick*, and an architecture for density estimation that allows for fast and accurate sampling via the inverse transform method. These techniques enable the design of a deep learning framework for generalized density estimation of multi dimensional, compactly supported, continuous valued random variables, which we call the Neural Inverse Transform Sampler[1]. NITS is (to our knowledge) the first framework to provide explicit[2] representations for densities with non-analytical partition functions.

We apply the resulting framework in two scenarios. First, in a generative autoregressive density model on natural images. And second on the UCI suite of density estimation benchmark datasets. We report competitive results in both settings.

Finally, we corroborate our empirical results by demonstrating that our model can universally approximate compactly supported densities of continuous random variables.

2. Probabilistically Normalized Networks

We consider the task of representing a parametric family of compactly supported probability densities on $[A, B]^n \subset \mathbb{R}^n$, $A, B \in \mathbb{R}$. An ideal model should possess the following properties: (1) high expressiveness, (2) fast sampling, (3) fast computation of ν_θ, and (4) end-to-end differentiability.

In this section we propose the **probabilistically normalized network** (PNN), which induces such a family. We will start from the one dimensional case, introducing the modeling framework in Section 2.1, and the sampling framework in section 2.2. We will then generalize to the higher dimensional case in Section 2.3 in two ways: first, assuming that $X = (X_1, \ldots, X_n)^T$ are independent, and second, that they are autoregressively correlated.

[1]Code available at github.com/lihenryhfl/NITS.

[2]Explicit, in the sense that the density is directly represented by a neural network.

2.1. Modeling a Compactly Supported 1D Distribution

We begin with a continuous compactly supported one dimensional random variable $X \sim \nu_\theta$. We would like to express $\nu_\theta : [A, B] \to \mathbb{R}$ as some neural network function parameterized by θ. To be a valid density, ν_θ must satisfy two conditions.

Condition 1: Integration to Unity i.e.,

$$\int_A^B \nu_\theta(x)dx = 1. \tag{2}$$

For this to hold, we choose to form the decomposition $\nu_\theta = f_\theta/Z_\theta$, where f_θ can now be an arbitrary neural network, and $Z_\theta = \int_{\mathbb{R}} f_\theta(x)dx$ is the partition function that normalizes f_θ so that the resulting ν_θ will integrate to 1.

Unfortunately, when f_θ is a simple neural network, any integral including Z_θ can only be computed via numerical quadrature (e.g., neural ODE frameworks (Chen et al., 2018a; Wehenkel & Louppe, 2019)), which is expensive in low dimensions and virtually intractable in higher dimensions.

However, by letting f_θ be the *derivative* of a neural network, we can compute ν in $\mathcal{O}(1)$ time (with respect to the size of the integration region). To do this, we leverage the crucial observation that, in one dimension, whenever there exists another function $F_\theta : X \to \mathbb{R}$ such that

$$F_\theta(x) = \frac{d}{dx} f_\theta(x) \quad \text{for all } x \text{ in } \mathbb{R}, \tag{3}$$

the integral of f_θ over any bounded interval $[a, b]$ can be evaluated via the formula

$$\int_a^b f_\theta(t)dt = F_\theta(b) - F_\theta(a), \tag{4}$$

due to the **fundamental theorem of calculus**.

Consequently, when $F_\theta(x)$ is a neural network and $f_\theta(x) := \nabla_x F_\theta(x)$ its derivative, the required assumption Eq. (3) is automatically satisfied, and the pdf ν_θ becomes computable via

$$\nu_\theta(x) = \frac{f_\theta(x)}{F_\theta(B) - F_\theta(A)}. \tag{5}$$

As the denominator is $Z_\theta = \int_{\mathbb{R}} f_\theta(x)dx = \int_A^B f_\theta(x)dx$, Condition 1 (i.e. Eq. 2) must hold. This technique, which we term the *integration trick*, is cheap, exact, and preserves gradients of F_θ and f_θ with respect to both θ and x.

Moreover, it is now clear that F_θ divided by the same denominator as in Eq. 5 must be the cdf corresponding to the density ν_θ:

$$N_\theta(x) = \frac{F_\theta(x)}{F_\theta(B) - F_\theta(A)} - F_\theta(A), \tag{6}$$

where we subtract by $F_\theta(A)$ so that $N_\theta(A) = 0$. Additionally, now that we have defined $N_\theta(x)$, we note that we can obtain ν_θ directly via $\nu_\theta(x) = \nabla_x N_\theta(x)$ (we previously obtained ν_θ by scaling the gradient of F_θ). This is because the differentiation and rescaling operators commute.

Condition 2: Positivity i.e.,

$$\nu_\theta(x) > 0 \text{ for all } x \text{ in } [A, B]. \tag{7}$$

This is equivalent to requiring that

$$\nabla_x F_\theta \geq 0 \quad \text{for all } x \text{ in } [A, B], \tag{8}$$

as ν_θ is proportional to $\nabla_x F_\theta$ up to a factor Z_θ, which is always positive. There are many ways of enforcing Eq. 8 — we take the simple approach of using positive monotonic activations (e.g. sigmoid, tanh, ReLU) and enforcing the positivity of the weights of the neural network F_θ. This is sufficient for Condition 2, given the following lemma:

Lemma 1. *A fully connected neural network $F_\theta : \mathbb{R}^n \to \mathbb{R}$ with positive weights and positive monotonic activations has directional derivative $f_\theta := \nabla_{x_i} F_\theta > 0$ for all $i = 1, \ldots n$.*

Probabilistically Normalized Network As both ν_θ and $N_\theta(x)$ describe a rescaled neural network (scaled by the same constant $Z_\theta^{-1} = (F_\theta(B) - F_\theta(A))^{-1}$), we call the resulting function a probabilistically normalized network, or PNN.

2.2. Sampling from a Compactly Supported 1D Distribution

Sampling from the PNN-induced distribution is now straightforward given our cheap access to N_θ. We can use the well-known inverse transform theorem, which we state below for completeness.

Lemma 2. ***(Inverse Transform Theorem)*** *Let N_θ be as defined above, and $N_\theta^{-1}(y)$, $y \in [0, 1]$ denote its inverse, i.e.*

$$N_\theta^{-1}(y) = \min\{x : N_\theta(x) \geq y\} \quad y \in [0, 1]. \tag{9}$$

If we let μ be the Lebesgue density on $[0, 1]$, then ν_θ is its pushforward density via the transform N_θ^{-1}, i.e.

$$\nu_\theta = N_{\theta*}^{-1}(\mu). \tag{10}$$

Thus, sampling from $x \sim \nu_\theta$ is a simple two step process:

1. draw $z \sim \text{Unif}[0, 1]$,
2. compute $x = N_\theta^{-1}(z)$,

where N_θ^{-1} can be accurately and efficiently calculated via bisection search, due to the monotonic nature of the cdf N_θ.

2.3. Modeling Multi-Dimensional Distributions

Let us now consider the case of modeling $X = (X_1, ..., X_n)^T$, an $n > 1$ dimensional, continuous, compactly supported random variable. By making various further assumptions on the correlation structure of the coordinates in X, we can straightforwardly extend the 1D formulation in the previous subsection to higher dimensions. Here we consider two cases: a coordinate-wise independent correlation structure, and an autoregressive correlation structure.

Coordinate-wise Independent Model Suppose that all elements of X are independently but perhaps not identically distributed. Note that, while simple, this correlation structure is not supported by normalizing flow-based approaches (Dinh et al., 2014; Rezende & Mohamed, 2015) due to their invertibility constraint, which by design *requires* dependence between coordinates. Let $N_\theta : \mathbb{R}^n \to [0,1]^n$ be the concatenation of n PNNs (and θ be the concatenation of their parameters), i.e.,

$$N_\theta = (N_{\theta_1}, \dots, N_{\theta_n})^T \quad \text{and} \quad \theta = (\theta_1, \dots, \theta_n)^T. \tag{11}$$

Then the density of the resulting distribution is simply the trace of its Jacobian

$$\nu_\theta(x_1, \dots, x_n) = \text{tr}(J_{N_\theta}) = \prod_{i=1}^{n} \nu_{\theta_i}(x_i),$$

where J_f denotes the Jacobian of f. And since the per-dimension cdf is a 1D PNN, sampling only requires a bisection search as before, and can be performed over all dimensions in parallel.

Autoregressive Model Retaining the previous definition of X, we consider the case where X_i depends on X_j for all $j < i$. This requires the PNN F_{θ_i} to take multi-dimensional inputs, i.e. $F_{\theta_i} : \mathbb{R}^i \to [0,1]$, as we are now modeling the conditional likelihood $P(X_i|X_{<i})$. Note that the parameters θ_i can be different for each i.

To obtain the conditional variants of N_θ and ν_θ, we apply the following theorem, which is a multi-dimensional generalization of the fundamental theorem of calculus, reproduced below for completeness.

Lemma 3. ***(Gradient Theorem)*** *Let $F : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function and $\varphi : [a,b] \to \mathbb{R}^n$ be a curve in $\mathbb{R}^n$, where $a, b \in \mathbb{R}$ and $\varphi(a), \varphi(b)$ are the endpoints of the curve. Then*

$$\int_{\varphi[a,b]} \nabla F \cdot dr = F(\varphi(b)) - F(\varphi(a)). \tag{12}$$

Let $F_{\theta_i} : \mathbb{R}^i \to \mathbb{R}$ be a neural network and $\varphi_{x_{<i}}(x_i)$ be the line along the i-th coordinate direction passing through $x_{<i}$, i.e.,

$$\varphi_{x_{<i}}(x_i) = x_i e_i + x_{<i} \odot (1 - e_i), \tag{13}$$

where $\odot$ is the Hadamard product and e_i is the ith basis vector. Then a conditional cdf of x_i, conditioned on the first $i-1$ coordinates of x, may be written as

$$N_{\theta_i}(x_i|x_{<i}) = \frac{F_{\theta_i}(x_i, x_{<i})}{F_{\theta_i}(B, x_{<i}) - F_{\theta_i}(A, x_{<i})} - F_{\theta_i}(A, x_{<i}), \tag{14}$$

where $[A, B]$ contains the compact support of x_i. Again, $\nu_{\theta_i}(x_i|x_{<i})$ can simply be computed from Eq. 14 by differentiating w.r.t. x_i, i.e.,

$$\nu_{\theta_i}(x_i|x_{<i}) = \frac{d}{dx_i} N_{\theta_i}(x_i|x_{<i})$$

See Figure 1 for a graphical representation of the rescaling process.

As in the 1D case, we enforce F_{θ_i} to have positive weights and monotonic activation functions. Letting $\theta = (\theta_1, \dots, \theta_n)^T$, the above definitions produce the joint density

$$\nu_\theta(x_1, \dots, x_n) = \prod_{i=1}^{n} \nu_{\theta_i}(x_i|x_{<i}),$$

and we refer to the underlying rescaled neural network N_θ as an **autoregressive PNN**.

To draw points from the induced distribution ν_θ, we use ancestral sampling, and draw each coordinate via the inverse transform method. Namely, to obtain $x_i \sim \nu_\theta$ given previously sampled $x_{<i}$, we

1. sample $z \sim \text{Unif}[0,1]$,
2. compute $x_i = N_\theta^{-1}(z|x_{<i})$,

where $N_\theta^{-1}(z|x_{<i})$ is the inverse of $N_\theta(x_i|x_{<i})$ with respect to the input x_i (the conditional case of Eq. 9). When $i = 1$, x_1 is sampled in the same manner as the 1D case, since $N_\theta(x_1)$ is not conditionally dependent on any other coordinates.

3. Neural Inverse Transform Sampler

Our proposed Neural Inverse Transform Sampler for a general, compactly supported n-dimensional random variable X consists of two components:

1. a set of concatenated PNN functions $N_\theta = (N_{\theta_1}, \dots, N_{\theta_n})^T$
2. a corresponding weight model $W_\phi(x)$ that supplies the parameters θ.

Algorithm 1 Sampling from P_θ

Input: PNN $N_\theta(x)$, parameter network $W_\phi(x)$
Initialize $x[i] = 0$ for $i = 1, \dots, n$.
Sample $\mathbf{z}[i] \sim \texttt{Uniform}[0, 1]$ for $i = 1, \dots, n$.
for $i = 1$ **to** n **do**
  $\theta_i = W_\phi(x)$
  $x[i] = N_{\theta_i}^{-1}(z_i|x)$[3][2]
end for

In this paper, we focus on maximum likelihood estimation (MLE), and perform density estimation by

$$\max_{\phi} \sum_{i=1}^{n} \log \nu_{W_\phi(x_i)}(x_i), \tag{15}$$

where $\nu_\theta(x)$ is the PNN density parameterized by θ obtained from the weight model W_ϕ. However, this is not the only way to train a NITS model, as any likelihood-based loss can be used.

NITS has a fast sampling scheme, supports efficient and exact likelihood evaluation, and is expressive and end-to-end differentiable. Algorithm 1 describes the process for sampling from P_θ in the autoregressive (and most general) variant of NITS[3]. In the following subsections, we discuss important properties and implementational details.

3.1. Architecture

For a neural network of n layers, our proposed architecture for F_θ can be specified by a recursive definition. Letting a_ℓ denote the activations at the ℓth layer of the PNN (and $a_0 = x$),

$$a_\ell = \sigma(h_A(A_\ell)^T a_{\ell-1} + h_b(b_\ell, A_\ell)) \tag{16}$$

for $\ell = 1, 2, \dots, n-1$, where σ is the sigmoid activation function, and $\{A_\ell, b_\ell\}$ denote the weights and biases of the ℓ-th layer. The final layer of the PNN is defined differently:

$$F_\theta(x) = a_n = h_s(A_n)^T a_{n-1}. \tag{17}$$

h_A, h_b, and h_s all denote parameter transformation functions, which we will subsequently define.

As discussed in Section 2.1, to obtain the density $\nu_\theta \propto \nabla_x F_\theta$, we required that 1) $\int \nu_\theta(x)dx = 1$, and 2) $\nu_\theta(x) \geq 0$ for all x. We enforced condition 1) by dividing the output of F_θ by its integral over the bounds $[A, B]$, as in Eqs. (5, 6, and 14). This is cheap and exact by our framework, via application of Lemma 3. Condition 2) was enforced by judicious choice of activation function σ and weight parameter transformation functions and Lemma 1:

$$\sigma(x) = \texttt{sigmoid}(x) \tag{18}$$
$$h_A(A) = exp(-A), \tag{19}$$

The final two parameter transformation functions are notable deviations from a regular fully connected neural network:

$$h_b(b, A) = -exp(-A) \odot b \tag{20}$$
$$h_s(A) = \texttt{softmax}(A). \tag{21}$$

In words, we apply a parameter transformation function h_b to the bias b, and a softmax function in the final layer to A (the weights, the activations).

Note that this parameterization is equivalent to that of a convex sum of standard multilayer perceptrons [4], and these modifications are not necessary to enforce monotonicity or positivity of the weights. Instead, they serve to bridge the gap between NITS and another parametric family of distributions, the **mixture of logistics distribution**, which we will explore in the proceeding subsection.

3.2. NITS is a Deep Mixture of Logistics

The mixture of logistics is a common distribution in statistics and probability, with density and cdf

$$f(x) = \sum_{i=1}^{n} \alpha_i \frac{e^{(x-\mu_i)/s_i}}{s_i(1 + e^{(x-\mu_i)/s_i})^2} \tag{22}$$
$$F(x) = \sum_{i=1}^{n} \alpha_i \sigma((x - \mu_i)/s_i), \tag{23}$$

respectively, for a mixture of size n and parameters $(\{\mu_i\}_{i=1}^n, \{s_i\}_{i=1}^n)$.

On the other hand, letting $a_{n-2} = F_\theta^{n-2}(x)$ denote the input to the penultimate layer of F_θ, we observe that the final two layers of F_θ can be written as:

$$\begin{aligned} F_\theta(x) &= \texttt{softmax}(A_n)^T \sigma((a_{n-2} - b_{n-1})/\exp(A_{n-1})) \\ &= \sum_{i=1}^{k} \beta_i \sigma((a_{n-2} - b_i)/s_i) \end{aligned} \tag{24}$$

where k is the width of the final hidden layer of F_θ, $\beta_i = \texttt{softmax}(A_n)_i$, and $s_i = \exp(A_{n-1})_i$.

Comparing Eqs. 23 and 24, it is easy to see that the cdf N_θ of a two-layer PNN would be nearly identical to a mixture

[3] In Algorithm 1, $N_\theta^{-1} = \texttt{monotonic\_inverse}(z, \texttt{f}(\cdot), \epsilon)$ is a function that takes a scalar z, tolerance ϵ, and monotonic function N_θ and returns x such that $|N_\theta(x) - z| < \epsilon$, i.e., it inverts N_θ. The monotonicity of N_θ permits the use of bisection search, which has runtime $\mathcal{O}(\log(1/\epsilon))$.

[4] To see this, examine Eq. 16, and note that choosing $C_\ell = h_A(A_\ell)$ and $d_\ell = h_b(b_\ell, A_\ell)$ produces the recurrence $a_\ell = C_\ell^T a_{\ell-1} + d_\ell$, which is a standard perceptron formulation. Then Eq. 17 is a convex combination of such networks.

of logistics distribution, with the exception of the normalization by Z_θ^{-1} applied to F_θ, which arises from the compact support assumption of our probabilistic model. This also means the density function ν_θ is quite similar to a mixture of logistics density function f.

Thus we make the following statement:

Lemma 4. *Let F_θ be a two-layer PNN with hidden dimension k and support bounds $[A, B]$, and F be the cdf of a k-mixture of logistics distribution. Then, as $B \to \infty$, there exists a sequence of parameters θ for F_θ so that*

$$|F_\theta(x) - F(x)| < \epsilon \tag{25}$$

for all $\epsilon > 0$.

Given this connection, we can think of the PNN as parameterizing a "deep" mixture of logistics distribution: the first $n-2$ layers of the neural network transform the input x into a deep hidden representation $a_{n-2} = F_\theta^{n-2}(x)$, and the final two layers of the neural network define a mixture of logistics distribution over this representation a_{n-2}. This formulation may explain the representational power of NITS, which we will next explore.

3.3. NITS is a Universal Density Estimator

In this section, we show that NITS can be used to approximate any continuous, real-valued autoregressive random variable with compact support, given sufficient width and depth of the underlying PNNs.

We start from the one-dimensional case, and show that each conditional distribution ν_θ is a universal density estimator.

Theorem 1. *(PNN is a universal approximator for positive 1D densities with bounded support) Let N_θ be a PNN with weights θ. Suppose $\nu : \mathbb{R} \times \mathbb{R}^{d-1} \to \mathbb{R}$ is a differentiable one-dimensional conditional density with bounded support, i.e. supp$(\nu) = [A, B]^d$ for real A, B. Then there exists a set of weights θ such that*

$$|\nu_\theta(x|y) - \nu(x|y)| < \epsilon \tag{26}$$

for any $\epsilon > 0$, where $\nu_\theta(x|y) := \nabla_x|_{y=c} N_\theta(x, y)$.

Now, noting that any target density ν of an autoregressive random variable $X = (X_1, \dots X_d)$ can be factored as

$$\nu(x) = \nu(x_d|x_{d-1}, \dots, x_1) \dots \nu(x_1) \tag{27}$$

we observe that Theorem 1 is easily applicable to estimation of ν.

Corollary 1. *(PNN is a universal density estimator for general continuous autoregressive random variables) Let $\mu(x)$ be a general joint density for a d-dimensional autoregressive random variable, i.e. takes on the form in Eq. 27. Then there exists a set of PNNs $\{N_{\theta_i}\}_{i=1}^d$ that induce a ν_θ (just as Eq. 27) such that for any $\epsilon > 0$,*

$$||\nu_\theta(x) - \nu(x)||_1 < \epsilon. \tag{28}$$

Furthermore, the case where X is coordinate-wise independent, i.e. $\mu(x) = \mu(x_d) \dots \mu(x_1)$, is also immediately true as a special case of Corollary 1, where we assume that $\mu(x_i|x_{i-1}, \dots, x_1) = \mu(x_i)$.

4. Related Work

While there are many works that take on the task of general density estimation, nearly all approaches avoid direct representation of the pdf itself. The two exceptions are the Real-valued Neural Autoregressive Distribution Estimator (RNADE) (Uria et al., 2013) and Mixtures of Conditional Gaussian Scale Mixtures (MCGSM) (Theis et al., 2012; Theis & Bethge, 2015), which, like our approach, form an autoregressive assumption on the modeled variable and compute one-dimensional conditional densities over each coordinate of the variable. Notably, like our NITS framework, both RNADE (Uria et al., 2013) and MCGSM (Theis & Bethge, 2015) employ a network W_ϕ to produce parameters for the conditional densities. However, unlike our approach, which approximates the conditional density via a deep neural network, both RNADE and MCGSM use much simpler distributions—mixtures of Gaussians and mixtures of scale Gaussians, respectively.

At large, the general landscape of density estimation models could be understood by how they tackle the partition function Z_θ. In this sense, modern approaches can be categorized into **two main thrusts**: those that avoid calculating Z_θ entirely and those that calculate Z_θ analytically.

Avoiding Z_θ Techniques that sidestep calculation of Z_θ form the brunt of contemporary approaches, and do so via mathematical or architectural constraints.

Normalizing flows, which were first proposed by (Dinh et al., 2014) for density estimation and extended by many later works, including (Rezende & Mohamed, 2015; Chen et al., 2018a; Kingma & Dhariwal, 2018), define a distribution as the pushforward measure of an invertible transformation T, and compute the density via a probabilistic change of variables formula, involving T^{-1} and dT/dx, which may be highly expensive, thus requiring the careful design of T.

Recently, score-based models (Song & Ermon, 2019; Song et al., 2020) have been shown to be highly effective for image generation, and involve estimation of the score function $\nabla_x \log p(x)$ of the model. Crucially, as Z_θ does not depend on x, the gradient with respect to x removes the dependence of the score function on Z_θ. However, this also removes the ability for the model to directly represent the

likelihood during training. During inference, (Song et al., 2020) show that score-based models can *approximate* $p(x)$ via the pushforward of a normalizing flow. However, the resulting likelihood is not exact, and inherits all the pitfalls of a normalizing flow-based model.

Perhaps the most well-known approach is variational inference. Proposed by (Jordan et al., 1999), adapted with stochastic gradient descent by (Hoffman et al., 2013), and applied to deep learning by (Kingma & Welling, 2013; Rezende et al., 2014), the variational inference framework has achieved remarkable traction in the machine learning community, and involves maximization of a lower bound on the likelihood, called Evidence Lower Bound Optimization (ELBO). As the name suggests, these techniques bound the likelihood rather than directly estimating it.

Deriving Z_θ Much of classical maximum likelihood estimation (MLE) relies on forms of f_θ for which Z_θ can be computed analytically. This includes MLE with the exponential family of distributions (e.g. Normal, Gamma, Dirichlet, etc.), the logistic distribution, the von Mises Fisher distribution for hyperspheres, and many more. Of the contemporary approaches, only RNADE, MCGSM, and related works (Uria et al., 2013; 2014; Germain et al., 2015) use analytically tractable partition functions. However, as previously stated, this limits the approaches to simple distributions.

In contrast, our approach can be considered to be in a third category: numerical computation of Z_θ. Due to the computational difficulty of computing high dimensional integrals, we know of no other works in this vein. Note that even our approach has significant drawbacks: our proposed *integration trick* for computing integrals of parametric functions can only compute one-dimensional integrals at a time, which limits our model to an autoregressive architecture.

We also discuss related work involving approximate density models, i.e. probabilistic models that do not support the computation of *exact* model likelihoods, but rather *approximate* model likelihoods. As previously discussed, score-based models can be considered one such approach: while the score function induces a continuous normalizing flow (CNF) whose prior is provably Gaussian when the diffusion time T is infinitely long, the diffusion model is only tractable for finite T, for which the prior is intractable. However, (Song et al., 2020) still use a Gaussian prior to compute the pushforward measure in the induced CNF. Therefore the pushforward density predicted by the CNF is never exact. Another such approximate density model is the Autoregressive Energy Machine (AEM) proposed in (Nash & Durkan, 2019), which (like NITS) also estimate Z_θ numerically by factorizing the high-dimensional density into autoregressive conditional 1D densities, and estimate the Z_θ of each 1D density individually. However, while NITS computes Z_θ exactly, AEMs estimate it via a biased importance sampling scheme. Since both score-based diffusion models and AEMs do not compute exact densities, we do not consider them in the following density estimation benchmarks.

Figure 2. Randomly generated images from DISCRETE NITS-CONV (top left) and NITS-CONV (top right). Compare with competing discretized and continuous density models, Pixel CNN (bottom left) and Flow++ (bottom right), respectively.

On the broader topic of numerical integration, Neural ODEs (Chen et al., 2018b) and many subsequent works (Grathwohl et al., 2018; Song et al., 2020) tackle the related task of integrating an arbitrary 1D function over some interval $[t_0, t_1]$. However, rather than estimating the desired function as a derivative of another differentiable function, these works model the desired function directly, and use a differentiable ODE solver to perform integrations.

Additionally, though the idea of representing monotone functions using neural networks has been thoroughly explored (Archer & Wang, 1993; Sill, 1998; Daniels & Velikova, 2010; Gupta et al., 2016), our application to *direct* density estimation is novel. (Chen et al., 2018a) explore the use of a monotone neural network function to augment the autoregressive transformation in Inverse Autoregressive Flows (IAF) (Kingma et al., 2016), which in the original formulation is a simple linear transform. This was further extended by (Wehenkel & Louppe, 2019), who compute monotone functions by integrating positive neural network functions with an ODE solver.

5. Experiments

In this section, we extensively evaluate the capabilities of our proposed approach via density estimation on two vastly

Table 1. Negative log likelihood (in bits/dim) for CIFAR-10. The table is split into halves, with discretized density models above and continuous density models below. We obtain competitive results among both types of models.

MODEL	CIFAR-10
PIXEL CNN	3.14
GATED PIXEL CNN	3.03
ROW PIXEL RNN	3.00
PIXEL CNN++	2.92
IMAGE TRANSFORMER	2.90
PIXELSNAIL	2.85
DISCRETE NITS-CONV (OURS)	2.94
REALNVP	3.49
GLOW	3.35
FLOW++	3.08
NITS-CONV (OURS)	2.97

different modalities. In our first experiment, we propose two NITS-based image density models, based on the causal convolution architecture proposed in (Van Oord et al., 2016) and refined in (Salimans et al., 2017; Chen et al., 2018c). We evaluate our two models on the CIFAR-10 dataset, where we demonstrate the state-of-the-art performance among continuous density estimators (Section 5.1). In the second experiment, we benchmark NITS against competing density estimation models with a suite of UCI datasets (Section 5.2).

5.1. Pixel-wise Density Estimation with PixelCNN++

First, we evaluate our approach against other generative image density estimation models on the CIFAR-10 dataset. To improve the capacity of our model, we use a convolutional neural architecture in our weight model W_ϕ, employing causally masked (Van Oord et al., 2016) atrous (Sermanet et al., 2013) convolutions with residual (He et al., 2016) and short-cut (Salimans et al., 2017) connections. As the usage of a convolutional weight model still follows the original formulation of NITS, we refer to it simply by the weight model architecture, i.e. NITS-CONV.

Additionally, we explore the performance of NITS-CONV when exploiting the 8-bit quantization of the CIFAR-10 dataset. For this approach, we apply a similar discretization procedure as in (Salimans et al., 2017), where the likelihood of a given intensity value of a given pixel x_i is given by

$$pmf(x_i = a) = \int_A^{\lceil a \rceil} pdf(t)dt - \int_A^{\lfloor a \rfloor} pdf(t)dt,$$

for $a \in \{0, 1, 2, 3, \dots, 255\}$. The integral computation on the right hand side is quite simple for our NITS density model, due to fast integration via the *integration trick*. We differentiate this variant of our model from the original with the *discretized* modifier, i.e. DISCRETE NITS-CONV.

One notable novelty of NITS-CONV is that it is the first modern autoregressive image density model with a continuous density model. Thus the parameterization of our model does not rely on the quantization factor of the modeled images. This bridges the gap between autoregressive density models and flow-based models, which are all continuous density models.

In Table 1, we compare our model to both discretized density models (i.e., PixelCNN (Van Oord et al., 2016) and its variants (Salimans et al., 2017; Chen et al., 2018c; Oord et al., 2016), and Image Transformer (Parmar et al., 2018)) and continuous density models (i.e. RealNVP (Dinh et al., 2016), Glow(Kingma & Dhariwal, 2018), and Flow++ (Ho et al., 2019)). See Figure 2 for examples of generated images.

5.2. Density Estimation on UCI Datasets

We additionally evaluate our method against the UCI suite of density estimation benchmarks. We compare our approach against Masked Autoregressive Flows (MAF) (Papamakarios et al., 2017), Transformation Autoregressive Networks (TAN) (Oliva et al., 2018), Neural and Block-Neural Autoregressive Flows (NAF and B-NAF) (Chen et al., 2018a; De Cao et al., 2020), Free-form Jacobian of Reversible Dynamics (FFJORD) (Grathwohl et al., 2018), Sum of Squares Polynomial Flow (SOS) (Jaini et al., 2019), RealNVP (Dinh et al., 2016), and Masked Autoencoder Density Estimation (Germain et al., 2015). The results are shown in Table 2.

We report state of the art density estimation on four out of five benchmarks in the UCI and BSDS300 datasets. Furthermore, our continuous density autoregressive model outperforms the only other competing continuous density autoregressive model that has been historically applied to the UCI datasets, MADE (Germain et al., 2015), on all datasets.

6. Conclusion

In this work we introduced two novel computational ideas: fast integration over arbitrary densities via the *integration trick*, and fast sampling from arbitrary densities via the inverse transform method. We combine these ideas in the design of the Neural Inverse Transform Sampler (NITS), which enjoys high expressiveness, fast density estimation, fast sampling, and end-to-end differentiability. Notably, NITS is (to our knowledge) the first model to provide explicit density approximation for densities with non-analytical partition functions. We benchmark NITS against existing density estimators, and show that NITS exhibits state of the art performance on several datasets.

Table 2. Test log likelihood for UCI datasets and BSDS300, with error bars corresponding to two standard deviations. The table is split into two halves: the upper half denotes flow-based models, and the lower half denotes autoregressive continuous density models. NITS-CONV is only applied to BSDS300, as the convolutional architecture is only readily applicable to images.

MODEL	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
MAF	0.30 ± 0.01	9.59 ± 0.02	-17.39 ± 0.02	-11.68 ± 0.44	156.36 ± 0.28
TAN	0.48 ± 0.01	11.19 ± 0.02	-15.12 ± 0.02	-11.01 ± 0.48	157.03 ± 0.07
NAF	0.62 ± 0.02	11.91 ± 0.13	-15.09 ± 0.40	$\mathbf{-8.86 \pm 0.15}$	157.73 ± 0.04
B-NAF	0.61 ± 0.01	12.06 ± 0.02	-14.71 ± 0.02	-8.95 ± 0.07	157.36 ± 0.03
FFJORD	0.46 ± 0.01	8.59 ± 0.12	-14.92 ± 0.08	-10.43 ± 0.04	157.40 ± 0.19
SOS	0.60 ± 0.01	11.99 ± 0.41	-15.15 ± 0.10	-8.90 ± 0.11	157.48 ± 0.41
NSF	$\mathbf{0.66 \pm 0.01}$	13.09 ± 0.02	-14.01 ± 0.03	-9.22 ± 0.48	157.31 ± 0.28
REALNVP	0.17 ± 0.01	8.33 ± 0.14	-18.71 ± 0.02	-13.84 ± 0.52	153.28 ± 1.78
MADE MOG	0.40 ± 0.01	8.47 ± 0.02	-15.15 ± 0.02	-12.27 ± 0.47	153.71 ± 0.28
NITS-MLP (OURS)	$\mathbf{0.66 \pm 0.01}$	$\mathbf{13.20 \pm 0.01}$	$\mathbf{-12.93 \pm 0.02}$	-10.85 ± 0.02	155.91 ± 0.21
NITS-CONV (OURS)	-	-	-	-	$\mathbf{163.35 \pm 0.22}$

References

Archer, N. P. and Wang, S. Application of the back propagation neural network algorithm with monotonicity constraints for two-group classification problems. *Decision Sciences*, 24(1):60–75, 1993.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018a.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018b.

Chen, X., Mishra, N., Rohaninejad, M., and Abbeel, P. Pixelsnail: An improved autoregressive generative model. In *International Conference on Machine Learning*, pp. 864–872. PMLR, 2018c.

Daniels, H. and Velikova, M. Monotone and partially monotone neural networks. *IEEE Transactions on Neural Networks*, 21(6):906–917, 2010.

De Cao, N., Aziz, W., and Titov, I. Block neural autoregressive flow. In *Uncertainty in Artificial Intelligence*, pp. 1263–1273. PMLR, 2020.

Dinh, L., Krueger, D., and Bengio, Y. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Du, Y. and Mordatch, I. Implicit generation and modeling with energy based models. 2019.

Germain, M., Gregor, K., Murray, I., and Larochelle, H. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pp. 881–889. PMLR, 2015.

Grathwohl, W., Chen, R. T., Bettencourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Gupta, M., Cotter, A., Pfeifer, J., Voevodski, K., Canini, K., Mangylov, A., Moczydlowski, W., and Van Esbroeck, A. Monotonic calibrated interpolated look-up tables. *The Journal of Machine Learning Research*, 17(1):3790–3836, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *International Conference on Machine Learning*, pp. 2722–2730. PMLR, 2019.

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. Stochastic variational inference. *Journal of Machine Learning Research*, 14(5), 2013.

Jaini, P., Selby, K. A., and Yu, Y. Sum-of-squares polynomial flow. In *International Conference on Machine Learning*, pp. 3009–3018. PMLR, 2019.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

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

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29:4743–4751, 2016.

Nash, C. and Durkan, C. Autoregressive energy machines. 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. 1735–1744. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/durkan19a.html.

Oliva, J., Dubey, A., Zaheer, M., Poczos, B., Salakhutdinov, R., Xing, E., and Schneider, J. Transformation autoregressive networks. In *International Conference on Machine Learning*, pp. 3898–3907. PMLR, 2018.

Oord, A. v. d., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., and Kavukcuoglu, K. Conditional image generation with pixelcnn decoders. *arXiv preprint arXiv:1606.05328*, 2016.

Papamakarios, G., Pavlakou, T., and Murray, I. Masked autoregressive flow for density estimation. *arXiv preprint arXiv:1705.07057*, 2017.

Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Shazeer, N., Ku, A., and Tran, D. Image transformer. In *International Conference on Machine Learning*, pp. 4055–4064. PMLR, 2018.

Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In *International conference on machine learning*, pp. 1530–1538. PMLR, 2015.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pp. 1278–1286. PMLR, 2014.

Salimans, T., Karpathy, A., Chen, X., and Kingma, D. P. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

Sill, J. Monotonic networks. 1998.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pp. 2256–2265. PMLR, 2015.

Song, Y. and Ermon, S. Generative modeling by estimating gradients of the data distribution. *arXiv preprint arXiv:1907.05600*, 2019.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

Theis, L. and Bethge, M. Generative image modeling using spatial lstms. *Advances in Neural Information Processing Systems*, 28:1927–1935, 2015.

Theis, L., Hosseini, R., and Bethge, M. Mixtures of conditional gaussian scale mixtures applied to multiscale image representations. 2012.

Uria, B., Murray, I., and Larochelle, H. Rnade: The real-valued neural autoregressive density-estimator. *arXiv preprint arXiv:1306.0186*, 2013.

Uria, B., Murray, I., and Larochelle, H. A deep and tractable density estimator. In *International Conference on Machine Learning*, pp. 467–475. PMLR, 2014.

Uria, B., Côté, M.-A., Gregor, K., Murray, I., and Larochelle, H. Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, 17 (1):7184–7220, 2016.

Van Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. Pixel recurrent neural networks. In *International Conference on Machine Learning*, pp. 1747–1756. PMLR, 2016.

Wehenkel, A. and Louppe, G. Unconstrained monotonic neural networks. *Advances in Neural Information Processing Systems*, 32:1545–1555, 2019.

A. Proofs

A.1. PNNs Compute Valid Density Functions

Proof. (Lemma 1) Let F_θ be an ℓ-layer neural network with positive weights W and positive monotonic activations σ of the general form defined by the recursive equation

$$a_i = \sigma(W_i^T a_{i-1} + b_i) \text{ for } i = 1, \dots, n,$$

where a_i is the activation of the i-th layer of F_θ, $a_\ell = F_\theta(x)$, and $a_0(x) = x$. We note that

$$\frac{da_i}{da_{i-1}} = (W_i \sigma'(W_i^T a_{i-1} + b_i))^T,$$

where, since $(W_i)_{jk} > 0$ for all j, k and $\sigma'(W_i^T a_{i-1} + b_i)_j > 0$ for all j by assumption, $\frac{da_i}{da_{i-1}} > 0$.

Thus, by the chain rule of differentiation,

$$\begin{aligned}\nabla_x F_\theta(x) &= \frac{da_\ell}{da_{\ell-1}} \frac{da_{\ell-1}}{da_{\ell-2}} \cdots \frac{da_2}{da_1} \\ &> 0.\end{aligned}$$

Since this argument is independent of x, we have shown the statement for all x. □

A.2. NITS is a Mixture of Logistics Distribution

Proof. (Lemma 4) According to Eq. 24, the final two layers of the PNN can be written as

$$N_\theta(x) = Z_\theta \sum_{i=1}^{k} \beta_i \sigma((x - b_i)/s_i)$$

where $Z_\theta = \int \nabla_x F_\theta(x) dx$ is the partition function. Further note that since N_θ is the scaled convex combination of sigmoid functions, $Z_\theta \to 1$ as A and B tend toward positive and negative infinity, respectively. Therefore this function is equivalent to the cdf of a mixture of logistics function (i.e. Eq. 23) in the limit. □

A.3. NITS is a Universal Density Estimator

We first show the following lemma:

Lemma 5. *Let N_θ be a PNN with compact boundary $[A^*, B^*]$ and N be an arbitrary positive monotonic cdf with bounded density support $[A, B]$. Then for any $\epsilon > 0$ there exists an A^*, B^* and set of parameters θ such that*

$$||N_\theta(x) - N(x)||_{L_\infty([A,B])} < \epsilon. \tag{29}$$

Proof. (Lemma 5) Choose ϵ_1, ϵ_2 so that $\epsilon = \epsilon_1 + \epsilon_2$. Define the cumulative distribution function (cdf) corresponding to the density ν as

$$N(x) := \int_{\mathbb{R}} \nu(x) dx = \int_A^B \nu(x) dx. \tag{30}$$

We shall use Lemma 2 in (Chen et al., 2018a), where it is shown that, given $\epsilon_1 > 0$ and an arbitrary positive monotonic function $N : \mathbb{R} \to [0, 1]$, there exists a $\theta = (\{\mu_i\}_{i=1}^n, \{s_i\}_{i=1}^n)$ such that a function of the form

$$F(x) = \sum_{i=1}^{n} \alpha_i \sigma\left(\frac{x - \mu_i}{s_i}\right) \tag{31}$$

will be ϵ_1-close to N in $L_\infty([A, B])$, i.e.

$$|F(x) - N(x)| < \epsilon_1 \quad \text{for all } x \in [A, B] \tag{32}$$

A one layer PNN with bounds $[A', B']$ can be written in the form

$$N_\theta(x) = \frac{F(x)}{F(B') - F(A')}. \tag{33}$$

Letting $C_{A',B'} = \frac{1}{F(B') - F(A')}$, and noting that $F(x) < 1$ for all $x \in \mathbb{R}$, we have

$$\begin{aligned}|F(x)(1 - C_{A',B'})| &\le |1 - C_{A',B'}| \\ &= \frac{1}{F(B') - F(A')} - 1.\end{aligned}$$

Now we pick A^*, B^* so that

$$F(B^*) - F(A^*) > \frac{1}{\epsilon_2 + 1}. \tag{34}$$

The existence of such a $A^*, B^* \in \mathbb{R}$ is guaranteed by the fact that $\frac{1}{\epsilon_2+1} < 1$ and $\lim_{x\to\infty} F(x) = 1$ and $\lim_{x\to-\infty} F(x) = 0$. Therefore,

$$|F(x)(1 - C_{A^*,B^*})| < \epsilon_2. \tag{35}$$

And so by choosing the bounds of the PNN N_θ as (A^*, B^*),

$$\begin{aligned}|N_\theta(x) - N(x)| &= |CF(x) - N(x) + F(x)(1 - \\ &\qquad C_{A^*,B^*}) - F(x)(1 - C_{A^*,B^*})| \\ &\le |F(x) - N(X)| + |F(x)(1 - \\ &\qquad C_{A^*,B^*})| \\ &\le \epsilon_1 + \epsilon_2 = \epsilon.\end{aligned}$$

□

Now we show the main theorem.

Proof. (Theorem 1) Choose ϵ_3, ϵ_4 so that $\epsilon = \epsilon_3 + \epsilon_4$. As N_θ and ν are both differentiable, we define

$$A_h = \frac{N_\theta(x + h) - N_\theta(x)}{h} \tag{36}$$

$$B_h = \frac{N(x + h) - N(x)}{h}, \tag{37}$$

and observe that for any ϵ_3, there exists h such that

$$|A_h - \nabla_x N_\theta(x)| < \epsilon_3/2 |B_h - \nabla_x N(x)| < \epsilon_3/2.$$

Moreover, from Lemma 5, we observe that there exists some θ and bounds $[A^*, B^*]$ such that

$$\begin{aligned}
|A_h - B_h| &= \left| \frac{N_\theta(x+h) - N_\theta(x)}{h} - \frac{N(x+h) - N(x)}{h} \right| \\
&= \frac{|N_\theta(x+h) - N(x+h) - N_\theta(x) - N(x)|}{h} \\
&\leq \frac{|N_\theta(x+h) - N(x+h)| + |N_\theta(x) - N(x)|}{h} \\
&\leq \epsilon_4.
\end{aligned}$$

Thus,

$$\begin{aligned}
&|\nabla_x N_\theta(x) - \nabla_x N(x)| \\
&= |\nabla_x N_\theta(x) - \nabla_x N(x) + (A_h - B_h) - (A_h - B_h)| \\
&\leq |\nabla_x N_\theta(x) - A_h| + |\nabla_x N(x) - B_h| + |A_h - B_h| \\
&\leq \epsilon_3 + \epsilon_4 = \epsilon
\end{aligned}$$

□

Finally, we show the autoregressive corollary.

Proof. (Corollary 1) Briefly, the sketch of the proof involves bounding each joint density

$$||\nu_\theta(x_i, \ldots, x_1) - \nu_\theta(x_i, \ldots, x_1)|| < \epsilon_i,$$

by some ϵ_i for all i, such that some recursive inequality of $||\nu_\theta(x) - \nu(x)||_1$, subsequently defined, is ultimately bounded by ϵ.

We first observe the following inequality:

$$\begin{aligned}
||ab - cd|| &= \frac{1}{2} ||(a+c)(b-d) + (a-c)(b+d)|| \\
&\leq \frac{1}{2} \left(||a+c||\,||b-d|| + ||a-c||\,||b+d|| \right).
\end{aligned}$$

Now, we note that for each $i = 1, \ldots, n$, we may choose

$$\begin{aligned}
a &= \nu_\theta(x_i|x_{<i}), & b &= \nu_\theta(x_{<i}), \\
c &= \nu(x_i|x_{<i}), & d &= \nu(x_{<i}),
\end{aligned}$$

where we define $\nu_\theta(x_0) = \nu(x_0) = 1$.

Applying the above inequality, we can see that

$$\begin{aligned}
&||\nu_\theta(x_i|x_{<i})\nu_\theta(x_{<i}) - \nu(x_2|x_{<i})\nu(x_{<i})||_1 \\
&\leq \frac{1}{2} \left(C_{i1}||a-c||_1 + C_{i2}||b-d||_1 \right), \qquad (38)
\end{aligned}$$

where $C_1 = \nu_\theta(x_i|x_{<i}) + \nu(x_i|x_{<i})$ and $C_2 = \nu_\theta(x_{<i}) + \nu(x_{<i})$ are bounded quantities due to the boundedness of ν_θ and ν, and $||a-c||_1 = ||\nu_\theta(x_i|x_{<i}) - \nu(x_i|x_{<i})||_1$ and $||b-d||_1 = ||\nu_\theta(x_{<i}) - \nu(x_{<i})||$. Notice that this is a recursive definition, as $||b-d||_1$ is simply Eq. 38 at $i-1$.

Therefore, if we choose

$$\epsilon_i = \frac{\epsilon}{\prod_{j=i+1}^{n} C_{j2} C_{i1} 2^{i-2}},$$

we can see that

$$\begin{aligned}
||\nu_\theta(x) - \nu(x)||_1 &\leq \sum_{i=1}^{n} \frac{\epsilon}{2^i} \\
&< \epsilon
\end{aligned}$$

□

B. Experimental Hyperparameters

The architecture used by NITS-MLP, which has a fully connected weight model with residual connections as used in (Nash & Durkan, 2019). More details are included in Table 3. Models were trained with early stopping with a patience of 5 epochs, and a learning rate of $2e-4$.

For NITS-CONV, which has a convolutional weight model, the architecture involves a block ResNet architecture involving causally masked atrous convolutional layers, as described in 5.1. The model applied to the CIFAR-10 dataset is comprised of 5 ResNet blocks, and contains a total of 53M parameters. The model applied to the BSDS300 dataset is comprised of a single ResNet block, and contains a total of 3.8M parameters. Models were trained for 150 epochs with a learning rate of $2e-4$

All PNN architectures involved two hidden layers with 16 hidden units. All models were trained on a Nvidia RTX 2080 Ti GPU.

Table 3. Information and hyperparameters for NITS-MLP on UCI Datasets.

	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
DIMENSION	6	8	21	43	63
TRAIN SET SIZE	1,615,917	852,174	315,123	29,556	1,000,000
BATCH SIZE	1024	1024	1024	128	1024
HIDDEN DIM	1024	1024	512	128	1024
DROPOUT	0.2	0.2	0.3	0.1	0.2
RESIDUAL BLOCKS	4	4	4	8	4

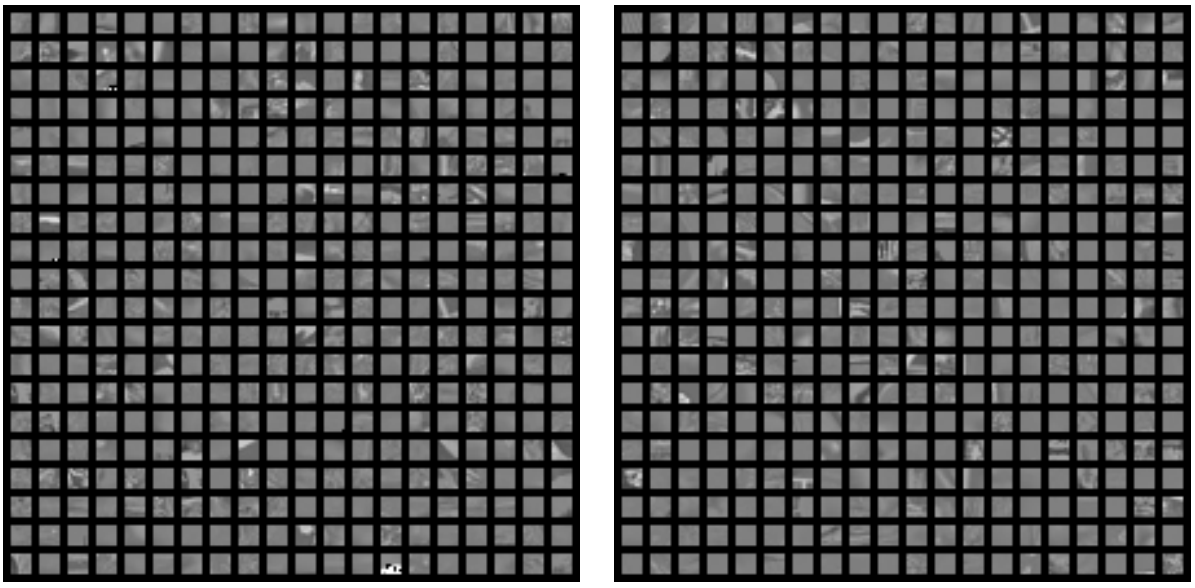

Figure 3. Additional generated images from NITS-CONV trained on the BSDS300 dataset (left) and true images from the dataset (right).