Theoretical & Applied Mechanics Letters 10 (2020) 000-5

Contents lists available at ScienceDirect

Theoretical & Applied Mechanics Letters

journal homepage: www.elsevier.com/locate/taml

Letter

Deep density estimation via invertible block-triangular mapping

Keju Tang[b], Xiaoliang Wan[a,*], Qifeng Liao[b]

[a] Department of Mathematics and Center for Computation and Technology, Louisiana State University, Baton Rouge 70803
[b] School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China

HIGHLIGHTS

- By integrating the Knothe-Rosenblatt (KR) rearrangement into the structure of the flow-based generative model, the performance is significantly improved.
- The affine coupling layer of the real-valued non-volume preserving (real-NVP) model has been reformulated to increase the robustness.
- New bijection layers, including a rotation layer and a component-wise nonlinear invertible layer, are introduced for further improvement.

ARTICLE INFO

Article history:
Received 8 January 2020
Accepted 4 March 2020

Keywords:
Deep learning
Density estimation
Optimal transport
Uncertainty quantification

ABSTRACT

In this work, we develop an invertible transport map, called KRnet, for density estimation by coupling the Knothe-Rosenblatt (KR) rearrangement and the flow-based generative model, which generalizes the real-valued non-volume preserving (real-NVP) model [Dinh, et al., arXiv:1605.08803v3]. The triangular structure of the KR rearrangement breaks the symmetry of the real NVP in terms of the exchange of information between dimensions, which not only accelerates the training process but also improves the accuracy significantly. We have also introduced several new layers into the generative model to improve both robustness and effectiveness, including a reformulated affine coupling layer, a rotation layer and a component-wise nonlinear invertible layer. The KRnet can be used for both density estimation and sample generation especially when the dimensionality is relatively high. Numerical experiments have been presented to demonstrate the performance of KRnet.

Density estimation is a challenging problem for high-dimensional data [1]. Some techniques or models have recently been developed in the framework of deep learning under the term generative modeling. Generative models are usually with likelihood-based methods, such as the autoregressive models [2–5], variational autoencoders (VAE) [6], and flow-based generative models [7–9]. A particular case is the generative adversarial networks (GANs) [10], which requires finding a Nash equilibrium of a game. All generative models rely on the ability of deep nets for the nonlinear approximation of high-dimensional mapping.

We pay particular attention to the flow-based generative models for the following several reasons. First, it can be regarded as construction of a transport map instead of a probabilistic model such as the autoregressive model. Second, it does not enforce a dimension reduction step as what the VAE does. Third, it provides an explicit likelihood in contrast to the GAN. Furthermore, the flow-based generative model maintains explicitly the invertibility of the transport map, which cannot be achieved by numerical discretization of the Monge- A mpére flow [11]. In a nutshell, the flow-based generative model is the only model that defines a transport map with explicit invertibility. The potential of flow-based generative modeling is twofold: First, it works for both density generation and sample generation at the same time. This property may bring efficiency to many problems. For example, it can be coupled with the importance sampling technique [12] or used to approximate the a posterior distribution in Bayesian statistics as an alternative of Markov Chain Monte

* Corresponding author.
E-mail address: xlwan@lsu.edu (X.L. Wan).

http://dx.doi.org/10.1016/j.taml.2020.01.023

Carlo (MCMC) [13]. Second, it can be combined with other techniques such as GAN or VAE to obtain a refined generative model [14, 15].

The goal of flow-based generative modeling is to seek an invertible mapping $\boldsymbol{Z} = f(\boldsymbol{Y}) \in \mathbb{R}^n$ where $f(\cdot)$ is a bijection, and $\boldsymbol{Y}, \boldsymbol{Z} \in \mathbb{R}^n$ are two random variables. Let p_Y and p_Z be the probability density functions (PDFs) of $\boldsymbol{Y}$ and $\boldsymbol{Z}$, respectively. We have

$$p_Y(\boldsymbol{y}) = p_Z(f(\boldsymbol{y}))\left|\det \nabla_y f\right|. \tag{1}$$

To construct $f(\cdot)$, the main difficulties are twofold: (1) $f(\cdot)$ is highly nonlinear since the prior distribution for $\boldsymbol{Z}$ must be simple enough, and (2) the mapping $f(\cdot)$ is a bijection. Flow-based generative models deal with these difficulties by stacking together a sequence of simple bijections, each of which is a shallow neural network, and the overall mapping is a deep net. Mathematically, the mapping $f(\cdot)$ can be written in a composite form:

$$\boldsymbol{z} = f(\boldsymbol{y}) = f_{[L]} \circ \ldots \circ f_{[1]}(\boldsymbol{y}), \tag{2}$$

where $f_{[i]}$ indicates a coupling layer at stage i. The mapping $f_{[i]}(\cdot)$ is expected to be simple enough such that its inverse and Jacobi matrix can be easily computed. One way to define $f_{[i]}$ is given by the real NVP [8]. Consider a partition $\boldsymbol{y} = (\boldsymbol{y}_1, \boldsymbol{y}_2)$ with $\boldsymbol{y}_1 \in \mathbb{R}^m$ and $\boldsymbol{y}_2 \in \mathbb{R}^{n-m}$. A simple bijection $f_{[i]}$ is defined as

$$\boldsymbol{z}_1 = \boldsymbol{y}_1, \tag{3}$$

$$\boldsymbol{z}_2 = \boldsymbol{y}_2 \odot \exp(\log \boldsymbol{s}(\boldsymbol{y}_1)) + \boldsymbol{t}(\boldsymbol{y}_1), \tag{4}$$

where $\boldsymbol{s}$ and $\boldsymbol{t}$ stand for scaling and translation depending only on $\boldsymbol{y}_1$, and $\odot$ indicates the Hadamard product or component-wise product. When $\boldsymbol{s}(\boldsymbol{y}_1) = \boldsymbol{1}$, the algorithm becomes non-linear independent component estimation (NICE) [7]. Note that $\boldsymbol{y}_2$ is updated linearly while the mappings $\boldsymbol{s}(\boldsymbol{y}_1)$ and $\boldsymbol{t}(\boldsymbol{y}_1)$ can be arbitrarily complicated, which are modeled as a neural network (NN),

$$(\log \boldsymbol{s}, \boldsymbol{t}) = \mathrm{NN}(\boldsymbol{y}_1). \tag{5}$$

The simple bijection given by Eqs. (3) and (4) is also referred to as an affine coupling layer [8]. The Jacobian matrix induced by one affine coupling layer is lower triangular:

$$\nabla_y \boldsymbol{z} = \begin{bmatrix} \boldsymbol{I} & 0 \\ \nabla_{\boldsymbol{y}_1} \boldsymbol{z}_2 & \mathrm{diag}(\mathrm{s}(\mathrm{y}_1)) \end{bmatrix}, \tag{6}$$

whose determinant can be easily computed as

$$\log|\det \nabla_y \boldsymbol{z}| = \sum_{i=1}^{n-m} \log|s_i(\boldsymbol{y}_1)|. \tag{7}$$

Since an affine coupling layer only modifies a portion of the components of $\boldsymbol{y}$ to some extent, a number of affine coupling layers need to be stacked together to form an evolution such that the desired distribution can be reached.

In the optimal transport theory, a mapping $T : \boldsymbol{Z} \to \boldsymbol{Y}$ is called a transport map such that $T_{\#}\mu_Z = \mu_Y$, where $T_{\#}\mu_Z$ is the push-forward of the law μ_Z of $\boldsymbol{Z}$ such that $\mu_Y(B) = \mu_Z(T^{-1}(B))$ for every Borel set B [16]. It is seen that $T = f^{-1}$, where $f(\cdot)$ is the invertible mapping for the flow-based generative model. In general, we have $Y_i = T(Z_1, \ldots, Z_n)$ or $Z_i = f(Y_1, \ldots, Y_n)$, i.e., each component of $\boldsymbol{Y}$ or $\boldsymbol{Z}$ depends on all components of the other random variable. The Knothe-Rosenblatt rearrangement says that the transport map T may have a lower-triangular structure such that

$$\boldsymbol{z} = T^{-1}(\boldsymbol{y}) = f(\boldsymbol{y}) = \begin{bmatrix} f_1(y_1) \\ f_2(y_1, y_2) \\ \vdots \\ f_n(y_1, y_2, \ldots, y_n) \end{bmatrix}. \tag{8}$$

It is shown in Ref. [17] that such a mapping can be regarded as a limit of a sequence of optimal transport maps when the quadratic cost degenerates. More specifically, the Rosenblatt transformation is defined as

$$\begin{aligned} z_1 &= P(Y_1 \le y_1) = F_1(y_1), \\ z_2 &= P(Y_2 \le y_2 | Y_1 = y_1) = F_2(y_2|y_1), \\ &\vdots \\ z_n &= P(Y_n \le y_n | Y_{n-1} = y_{n-1}, \ldots, Y_1 = y_1) \\ &= F_n(y_n | y_{n-1}, \ldots, y_1), \end{aligned}$$

where

$$\begin{aligned} &P(Z_i \le z_i; i = 1, \ldots, n) \\ &= \int_{\{\boldsymbol{Z} | Z_i \le z_i\}} \mathrm{d}_{y_n} F_n(y_n | y_{n-1}, \ldots, y_1) \ldots \mathrm{d}_{y_1} F_1(y_1) \\ &= \int_{\{\boldsymbol{Z} | Z_i \le z_i\}} \mathrm{d}z_n \ldots \mathrm{d}z_1 = \prod_{i=1}^{n} z_i, \end{aligned}$$

which implies that Z_i are uniformly and independently distributed on $[0, 1]$. Thus the Rosenblatt transformation provides a lower-triangular mapping to map $\boldsymbol{Z}$, which is uniform on $[0, 1]^n$ and has i.i.d. components, to an arbitrary random variable $\boldsymbol{Y}$.

Motivated by the Knothe-Rosenblatt rearrangement, we propose a block-triangular invertible mapping as a generalization of real NVP. Consider a partition of $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K)$, where $\boldsymbol{y}_i = (y_{i,1}, \ldots, y_{i,m})$ with $1 \le K \le n$ and $1 \le m \le n$, and $\sum_{i=1}^{K} \dim(\mathrm{y_i}) = \mathrm{n}$. We define an invertible bijection, called KRnet,

$$\boldsymbol{z} = f(\boldsymbol{y}) = \begin{bmatrix} \hat{f}_1(\boldsymbol{y}_1) \\ \hat{f}_2(\boldsymbol{y}_1, \boldsymbol{y}_2) \\ \vdots \\ \hat{f}_K(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K) \end{bmatrix}, \tag{9}$$

whose structure is consistent with the Knothe-Rosenblatt rearrangement. The flow chart of KRnet is illustrated in Fig. 1. Before a detailed explanation of each layer, we fist look at the main structure of KRnet, which mainly consists of two loops: outer loop and inner loop, where the outer loop has $K - 1$ stages, corresponding to the K mappings $\hat{f}_i$ in Eq. (9), and the inner loop has L stages, corresponding to the number of affine coupling layers.

• Outer loop. Let $f_{[i]}^{\mathrm{outer}}$ indicate one iteration of the outer loop. We have

$$\boldsymbol{z} = f(\boldsymbol{y}) = L_N \circ f_{[K-1]}^{\mathrm{outer}} \circ \ldots \circ f_{[1]}^{\mathrm{outer}}(\boldsymbol{y}), \tag{10}$$

where $\boldsymbol{y}^{(k)} = f_{[k]}^{\mathrm{outer}}(\boldsymbol{y}^{(k-1)})$ with $\boldsymbol{y}^{(0)} = \boldsymbol{y}$, $k = 1, \ldots, K-1$, indicates each iteration of the outer loop, and L_N is a component-wise

nonlinear invertible layer. Each $\boldsymbol{y}^{(k)} = (\boldsymbol{y}_1^{(k)}, \ldots, \boldsymbol{y}_K^{(k)})$ has the same partition. The ith partition $\boldsymbol{y}_i^{(k)}$ will remain unchanged after stage $K-i+1$, in other words, we always freeze the last partition of active dimensions whenever needed. For example, $\boldsymbol{y}_K^{(k)}$ will be updated only when $k=1$ and $\boldsymbol{y}_{K-1}^{(k)}$ will be fixed when $k>2$. This way, the number of effective dimensions decreases as k increases. It should be noted that the number K of partitions is a hyperparameter that can be tuned. Intuitively, m may be small when n is not large. When $m=1$, we deactivate the dimensions one by one.

• Inner loop. The inner loop mainly consists of a sequence of general coupling layers $f_{[k,i]}^{\text{inner}}$, based on which $f_{[k]}^{\text{outer}}$ can be written as:

$$f_{[k]}^{\text{outer}} = L_S \circ f_{[k,L]}^{\text{inner}} \circ \ldots \circ f_{[k,1]}^{\text{inner}} \circ L_R, \tag{11}$$

where L_R is a rotation layer and L_S is a squeezing layer. The general coupling layer $f_{[k,i]}^{\text{inner}}$ includes a scale and bias layer, which plays a similar role to batch normalization.

For all general coupling layers $f_{[k,i]}^{\text{inner}}$, we usually let neural network Eq. (5) have two fully connected hidden layers of the same number of neurons, say l. Since the number of effective dimensions decreases as k increases in the outer loop, we expect that l decreases accordingly. We define a ratio $r<1$. If $f_{[1,i]}^{\text{inner}}$ has M hidden neurons in total, the number becomes Mr^{k-1} for $f_{[k,i]}^{\text{inner}}$. We now explain each layer in Fig. 1:

Squeezing layer. In the squeezing layer, we simply deactivate some dimensions using a mask

$$\boldsymbol{q} = (\underbrace{1,\ldots,1}_{k},\underbrace{0,\ldots,0}_{n-k}),$$

which means that only the first k components, i.e., $\boldsymbol{q} \odot \boldsymbol{y}$, will be active after the squeezing layer while the other $n-k$ components will remain unchanged.

Rotation layer. We define an orthogonal matrix

$$\hat{\boldsymbol{W}} = \begin{bmatrix} \boldsymbol{W} & 0 \\ 0 & \boldsymbol{I} \end{bmatrix},$$

where $\boldsymbol{W} \in \mathbb{R}^{k\times k}$ with k being the number of 1's in $\boldsymbol{q}$, and $\boldsymbol{I} \in \mathbb{R}^{(n-k)\times(n-k)}$ is an identity matrix. Using $\hat{\boldsymbol{W}}$, we obtain $\hat{\boldsymbol{y}} = \hat{\boldsymbol{W}}\boldsymbol{y}$ subject to a rotation of the coordinate system. The Jacobian matrix between $\hat{\boldsymbol{y}}$ and $\boldsymbol{y}$ is $\hat{\boldsymbol{W}}$ whose determinant is needed. For the sake of computation, we consider in reality:

$$\hat{\boldsymbol{W}} = \begin{bmatrix} \boldsymbol{L} & 0 \\ 0 & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{U} & 0 \\ 0 & \boldsymbol{I} \end{bmatrix}, \tag{12}$$

where $\boldsymbol{W} = \boldsymbol{LU}$ is the LU decomposition of $\boldsymbol{W}$. More specifically, $\boldsymbol{L}$ is a lower-triangular matrix, whose entries on the diagonal line are 1, and $\boldsymbol{U}$ is a upper-triangular matrix. Then we have

$$\det \hat{\boldsymbol{W}} = \det \boldsymbol{L} \det \boldsymbol{U} = \det \boldsymbol{U} = \prod_{i=1}^{n} u_{ii}.$$

One simple choice to initialize $\hat{\boldsymbol{W}}$ is $\hat{\boldsymbol{W}} = \boldsymbol{V}^{\mathrm{T}}$, where the column vectors of $\boldsymbol{V}$ are the eigenvectors of the covariance matrix of the input vector. The eigenvectors are ordered such that the associated eigenvalues decrease since the dimensions to be deactivated are at the end. The entries in $\boldsymbol{L}$ and $\boldsymbol{U}$ are trainable.

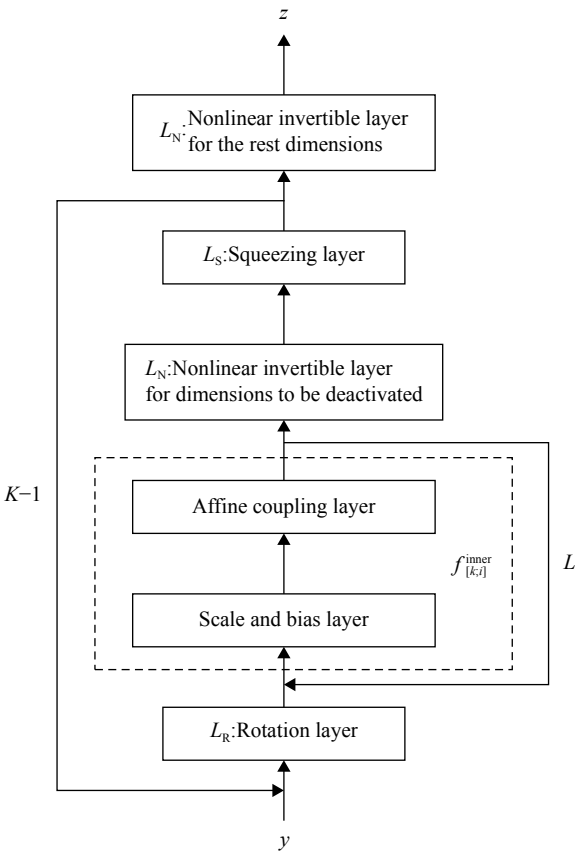

Fig. 1. The flow chart of KRnet

The orthogonality condition may be imposed through a penalty term $\alpha\|\hat{\boldsymbol{W}}^{\mathrm{T}}\hat{\boldsymbol{W}} - \boldsymbol{I}\|_F^2$, where $\|\cdot\|_F$ indicates the Frobenius norm and $\alpha>0$ is a penalty parameter. However, numerical experiments show that a direct training of $\boldsymbol{L}$ and $\boldsymbol{U}$ without the orthogonality condition enforced also works well.

Scale and bias layer. By definition, the KRnet is deep. It is well known that batch normalization can improve the propagation of training signal in a deep net [18]. A simplification of the batch normalization algorithm is

$$\hat{\boldsymbol{y}} = \boldsymbol{a} \odot \boldsymbol{y} + \boldsymbol{b}, \tag{13}$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are trainable [9]. The parameters $\boldsymbol{a}$ and $\boldsymbol{b}$ will be initialized by the mean and standard deviation associated with the initial data. After the initialization, $\boldsymbol{a}$ and $\boldsymbol{b}$ will be treated as regular trainable parameters that are independent of the data.

Reformulated affine coupling layer. We redefine the affine coupling layer of the real NVP as follows:

$$\boldsymbol{z}_1 = \boldsymbol{y}_1, \tag{14}$$

$$\boldsymbol{z}_2 = \boldsymbol{y}_2 \odot [1 + \alpha \tanh(\boldsymbol{s}(\boldsymbol{y}_1)) + e^{\boldsymbol{\beta}} \odot \tanh(\boldsymbol{t}(\boldsymbol{y}_1)], \tag{15}$$

where $\alpha \in (0,1)$ and $\boldsymbol{\beta} \in \mathbb{R}^n$. First of all, the reformulated affine coupling layer adapts the trick of ResNet, where we separate out the identity mapping. Second, we introduce the constant $\alpha \in (0,1)$ to improve the conditioning. It is seen from Eq. (7) that $|\det \nabla_y \boldsymbol{z}| \in (0,+\infty)$ for the original real NVP while

$(1-\alpha)^{n-m} \le |\det \nabla_y \boldsymbol{z}| \le (1+\alpha)^{n-m}$ in our formulation. Our scaling can alleviate the illnesses when the scaling in the original real NVP occasionally become too large or too small. When $\alpha = 1$, $|\det \nabla_y \boldsymbol{z}| \in (0,2)$. This case is actually similar to real NVP in the sense that the scaling can be arbitrarily small, and for well-scaled data by the scaling and bias layer, we do not expect a large scaling will be needed. When $\alpha = 0$, the formulation is the same as NICE, where no scaling is included. Third, we also make the shift bounded by letting it pass a hyperbolic tangent function. The reason for such a modification is similar to that for the scaling. The main difference here is the introduction of the trainable factor $e^{\boldsymbol{\beta}}$. Compared to $\boldsymbol{t}(\boldsymbol{y}_1)$, $e^{\boldsymbol{\beta}}$ depends on the dataset instead of the value of $\boldsymbol{y}_1$, which helps to reduce the number of outliers for sample generation. Numerical experience show that our formulation in general works better than both real NVP and NICE. We usually let $\alpha = 0.6$.

Nonlinear invertible layer. It is seen that the affine coupling layer is linear with respect to the variable to be updated. We introduce component-wise nonlinear invertible mapping to alleviate this limitation. We consider an invertible mapping $y = F(x)$ with $x \in [0,1]$ and $y \in [0,1]$, where $F(x)$ can be regarded as the cumulative distribution function of a random variable defined on $[0,1]$. Then we have

$$F(x) = \int_0^x p(x)\mathrm{d}x, \tag{16}$$

where $p(x)$ is a probability density function. Let $0 = x_0 < x_1 < \ldots < x_{m+1} = 1$ be a partition of $[0,1]$, on which we define $p(x)$ as a piecewise linear polynomial. Then $F(x)$ will be a quadratic function, whose roots can be explicitly computed. When $p(x) \equiv const$, $F(x)$ is an identity mapping. The support of each dimension of $\boldsymbol{Y}$ and $\boldsymbol{Z}$ is $(-\infty,\infty)$. Although a logistic mapping can be used to map $(-\infty,\infty)$ to $(0,1)$, after which $F(x)$ can be implemented, the singularity will make the model not robust when $F^{-1}(x)$ is used to map $(0,1)$ back to $(-\infty,\infty)$. For example, the sigmoid function `tensorflow.math.sigmoid(x)` will be always equal to 1.0 when $x \gtrapprox 17.0$ if x is of type `float32`. Our strategy is simple, we decompose $(-\infty,\infty) = (-\infty,-a) \cup [-a,a] \cup (a,\infty)$ with $a > 0$. On $[-a,a]$, we implement $y = F((x+a)/(2a))$ followed by an affine mapping $2ay - a$. In other words, we maps the domain to $[0,1]$, and then map the range back to $[-a,a]$. On $(-\infty,-a)$ and (a,∞), we just let $y = x$. Since the scaled data will be roughly centered at the origin, we only need to choose a sufficiently large a to cover the data instead of the whole real axis. In summary, we consider a mapping from $\mathbb{R}$ to $\mathbb{R}$, where the mapping is nonlinear on $[-a,a]$ and an identity mapping on $(-\infty,a) \cup (a,\infty)$.

We subsequently present some numerical experiments. For clarity we will turn off the rotation layers and the nonlinear invertible layers to focus on the effect of the triangular structure of KRnet, which provides the main improvement of performance. Let $\boldsymbol{Y} \in \mathbb{R}^n$ have i.i.d. components, where $Y_i \sim \text{Logistic}(0,s)$. Let $\boldsymbol{y}_{[i:(i+k)]} = [y_i,\ldots,y_{i+k}]^{\mathrm{T}}$. We consider the data that satisfy the following criterion:

$$|\mathbf{R}^{\alpha,\Theta_i}\boldsymbol{y}_{[i:i+1]}|_2 \ge C, \quad i = 1,\ldots,n-1,$$

where

$$\mathbf{R}^{\alpha,\Theta_i} = \begin{bmatrix} \alpha & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\Theta_i & -\sin\Theta_i \\ \sin\Theta_i & \cos\Theta_i \end{bmatrix},$$

which is a product of a scaling matrix and a rotation matrix. Simply speaking, we generate an elliptic hole in the data for any two adjacent dimensions such that Y_i become correlated. Let $\Theta_i = \pi/4$, if i is even; $3\pi/4$, otherwise. Let $\alpha = 3$, $s = 2$ and $C = 7.6$.

For the training process we minimize the cross entropy between the model distribution and the data distribution

$$H(\mu_{\text{data}},\mu_{\text{model}}) = -\sum_{i=1}^{N}\log(p_{\boldsymbol{Y}}(\boldsymbol{y}^{(i)};\Theta)), \tag{17}$$

where $\mu_{\text{model}}(\mathrm{d}\boldsymbol{y}) = p_{\boldsymbol{Y}}(\boldsymbol{y})\mathrm{d}\boldsymbol{y}$, N is the size of training dataset and Θ are the parameters to be trained. This is equivalent to minimize the Kullback-Leibler (KL) divergence or to maximize the likelihood. To evaluate the model, we compute the KL divergence

$$D_{\text{KL}}(\mu_{\text{true}}\|\mu_{\text{model}}) = H(\mu_{\text{true}},\mu_{\text{model}}) - H(\mu_{\text{true}}), \tag{18}$$

where μ_{true} is known. First, we generate a validation dataset from μ_{true} which is large enough such that the integration error in terms of μ_{model} is negligible. Second, we compute an approximation of $\mathbb{E}_{\boldsymbol{Y}}[D_{\text{KL}}(\mu_{\text{true}}\|\mu_{\text{model}}(\boldsymbol{Y}))]$ in terms of $\boldsymbol{Y}$, where $\boldsymbol{Y}$ indicates the random variables that correspond to N samples in the training dataset. We take 10 independent training datasets. For each dataset, we train the model for a relatively large number of epochs using the ADAM method [19]. For each epoch, we compute $D_{\text{KL}}(\mu_{\text{true}}\|\mu_{\text{model}})$ using the validation dataset. We pick the minimum KL divergence and compute its average for the 10 runs as an approximation of $\mathbb{E}_{\boldsymbol{Y}}[D_{\text{KL}}(\mu_{\text{true}}\|\mu_{\text{model}}(\boldsymbol{Y}))]$. We choose 10 runs simply based on the problem complexity and our available computational resources.

We first consider four-dimensional data and show the capability of the model by investigating the relation between N, i.e., sample size, and the KL divergence. We let $L = 12$, and $K = 3$, in other words, one dimension will be deactivated every 12 general coupling layers. In $f^{\text{inner}}_{[k,i]}$, $k = 1,2,3$, the neural network Eq. (5) has two hidden layers each of which has mr^{k-1} neurons with $m = 24$ and $r = 0.88$. The ADAM method with 4 mini-batches is used for all the training processes. 8000 epochs are considered for each run and a validation dataset with 1.6e5 samples is used to compute $D_{\text{KL}}(\mu_{\text{true}}\|\mu_{\text{model}})$. The results are plotted in Fig. 2, where the size of training dataset is up to 8e4. Assume that there exist a Θ_0 such that $\mu_{\text{model}}(\Theta_0)$ is very close to μ_{true}. We expect to observe the convergence behavior of maximum likelihood estimator, i.e., $|\hat{\Theta}_N - \Theta_0| \sim N^{-1/2}$, where $\hat{\Theta}_N$ the maximum likelihood estimator. It is seen that the KL divergence between μ_{true} and $\mu_{\text{model}}(\hat{\Theta}_N)$ is indeed dominated by an error of $O(N^{-1/2})$. This implies that the model is good enough to capture the data distribution for all the sample sizes considered.

We subsequently investigate the relation between the KL divergence and the complexity of the model. The results are summarized in Fig. 3, where the degrees of freedom (DOFs) indicate the number of unknown parameters in the model. For comparison, we also include the results given by the real NVP. The configuration of the KRnet is the same as before except that we consider L = 2, 4, 6, 8, 10, and 12. The size of the training dataset is 6.4×10^5 and the size of the validation dataset is 3.2×10^5. We use

a large same size for training dataset such that the error is dominated by the capability of the model. For each run, 8000 epochs are considered except for the two cases indicated by filled squares where 12000 epochs are used because L is large. It is seen both the KRnet and the real NVP demonstrate an algebraic convergence. By curving fitting, we obtain that the KL divergence decays of $O(N_{\text{dof}}^{-1.56})$ for the KRnet and of $O(N_{\text{dof}}^{-0.73})$ for the real NVP, implying the KRnet is much more effective than the real NVP.

We finally test the dependence of the convergence behavior of the KRnet on the dimensionality by considering an eight-dimensional problem. We let $K = 7$, i.e., the random dimensions are deactivated one by one. In $f_{[k,i]}^{\text{inner}}$, $k = 1, \ldots, 7$, the neural network Eq. (5) has two hidden layers each of which has mr^{k-1} neurons with $m = 32$ and $r = 0.9$. For each run, 12000 epochs are considered. All other configurations are the same as the four-dimensional case. The results are plotted in Fig. 4, where we obtain an overall algebraic convergence of $O(N_{\text{dof}}^{-1.63})$ in terms of DOF for $L = 2$, 4, 6, 8, and 10. It appears that the rate is not sensitive to the number of dimensions.

In this work, we have developed a generalization of the real NVP as a technique for density estimation of high-dimensional data. The results are very promising and many questions remain open. For example, the algebraic convergence with respect to the DOFs is only observed numerically. The dependence of accuracy on the sample size is not clear although the convergence rate seems not sensitive to the dimensionality. These questions are being investigated and the results will be reported elsewhere.

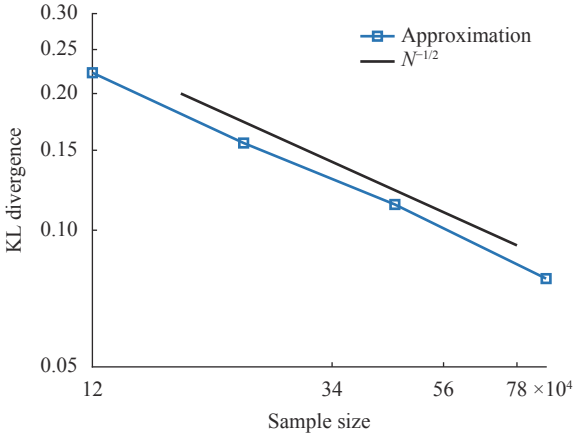

Fig. 2. The KL divergence in terms of sample size for the four-dimensional case

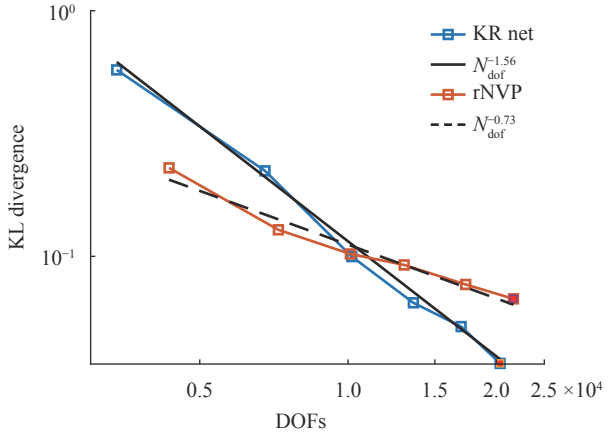

Fig. 3. The KL divergence in terms of degrees of freedom (DOFs) of the model for the four-dimensional case

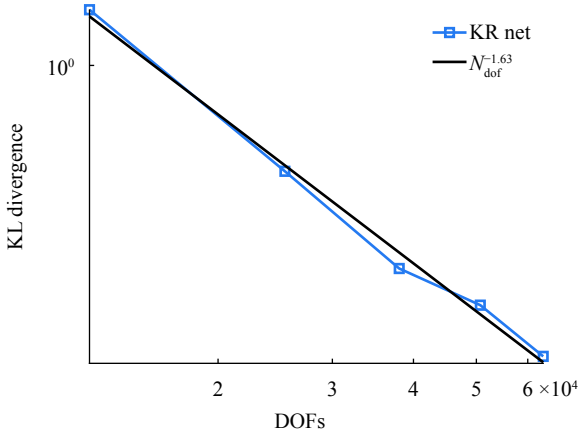

Fig. 4. The KL divergence in terms of degrees of freedom (DOFs) of the model for the eight-dimensional case

Acknowledgement

X. Wan's work was supported by the National Natural Science Foundation of Unite States (DMS-1620026 and DMS-1913163). Q. Liao is supported by the National Natural Science Foundation of China (11601329).

References

[1] D. Scott, Multivariate Density Estimation: Theory, Practice, and Visualization, 2nd Edition, John Wiley & Sons, Inc., 2015.

[2] A. Graves, Generating sequences with recurrent neural networks, (2013), arXiv: 1308.0850.

[3] A. van den Oord, N. Kalchbrenner, K. Kavukcuoglu, Pixel recurrent neural networks, (2016), arXiv: 1601.06759.

[4] A. van den Oord, N. Kalchbrenner, O. Vinyals, et al., Conditional image generation with PixelCNN decoders, (2016), arXiv: 1606.05328.

[5] G. Papamakarios, T. Pavlakou, I. Murray, Masked autoregressive flow for density estimation, (2018), arXiv: 1705.07057v4.

[6] D. P. Kingma, T. Salimans, R. Jozefowicz, et al., Improving variational inference with inverse autoregressive flow, Advances in Neural Information Processing Systems (2016) 4743–4751.

[7] L. Dinh, D. Krueger, S. Bengio, Nice: non-linear independent components estimation, (2014), arXiv: 1410.8516.

[8] L. Dinh, J. Sohl-Dickstein, S. Bengio, Density estimation using real NVP, (2017), arXiv: 1605.08803v3.

[9] D. P. Kingma, P. Dhariwal, Glow: Generative flow with invertible 1x1 convolutions, (2018), arXiv: 1807.03039v2.

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, et al., Generative adversarial nets, Advances in Neural Information Processing Systems (2014) 2672–2680.

[11] L. Zhang, W. E, L. Wang, Monge-Ampére flow for generative

modeling, (2018), arXiv: 1809.10188v1.

[12] X. Wan, S. Wei, Coupling the reduced-order model and the generative model for an importance sampling estimator, J. Compt. Phys 408 (2020) 109281.

[13] A. Spatini, D. Bigoni, Y. Marzouk, Inference via low-dimensional couplings, (2017), arXiv: 1703.06131v4.

[14] A. Grover, M. Dhar, S. Ermon, Flow-GAN: Combining maximum likelihood and adversarial learning in generative models, (2018), arXiv: 1705.08868v2.

[15] J. Zhu, D. Zhao, B. Zhang, LIA: Latently invertible autoencoder with adversarial learning, (2019), arXiv: 1906.08090v1.

[16] F. Santambrogio, Optimal Transport for Applied Mathematicians, Birkhäuser, 2010.

[17] G. Carlier, A. Galichon, F. Santambrogio, From Knothes transport to Breniers map and a continuation method for optimal transport, SIAM J. Math. Anal 41 (2010) 2554–2576.

[18] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariance shift, (2015), arXiv: 1502.03167v3.

[19] D. P. Kingma, J. L. Ba, ADAM: A method for stochastic optimization, (2017), arXiv: 1412.6980v9.