Neural PCA for Flow-Based Representation Learning

Shen Li* , **Bryan Hooi**
Institute of Data Science, National University of Singapore
shen.li@u.nus.edu , bhooi@comp.nus.edu.sg

Abstract

Of particular interest is to discover useful representations solely from observations in an unsupervised generative manner. However, the question of whether existing normalizing flows provide effective representations for downstream tasks remains mostly unanswered despite their strong ability for sample generation and density estimation. This paper investigates this problem for such a family of generative models that admits exact invertibility. We propose Neural Principal Component Analysis (Neural-PCA) that operates in full dimensionality while capturing principal components in *descending* order. Without exploiting any label information, the principal components recovered store the most informative elements in their *leading* dimensions and leave the negligible in the *trailing* ones, allowing for clear performance improvements of 5%-10% in downstream tasks. Such improvements are empirically found consistent irrespective of the number of latent trailing dimensions dropped. Our work suggests that necessary inductive bias be introduced into generative modelling when representation quality is of interest.

1 Introduction

One of the core objectives of generative models is to deliver the promise of learning useful representations of observations in an *unsupervised* manner (without exploiting any label information) [Bengio *et al.*, 2013]. The usefulness of a representation is often measured by the representation's quality for generic downstream tasks that are discriminative in nature [Chapelle *et al.*, 2009][Dempster *et al.*, 1977], e.g. classification tasks, and informative in describing the observations [Hjelm *et al.*, 2018].

This paper investigates the usefulness of representations given by Normalizing Flows (NF) [Rezende and Mohamed, 2015][Dinh *et al.*, 2014][Dinh *et al.*, 2016][Kingma and Dhariwal, 2018][Chen *et al.*, 2019], a family of likelihood-based models that admits exact invertibility and inference. We empirically find that state-of-the-art NFs yield poor representations in terms of discriminativeness and informativeness. We attribute it to a mathematical fact that an NF preserves dimensionality throughout the transform such that invertibility is guaranteed. This, apparently, violates the manifold hypothesis that data resides in a lower dimensional manifold embedded in a full Euclidean space [Fefferman *et al.*, 2016]. Consequently, dimensionality preservation leads to redundant representations, which undermines representations desired as shown in our empirical studies.

To alleviate the conflict with the manifold hypothesis, we propose Neural-PCA, a flow model that operates in a full-dimensional space while capturing principal components in descending order. Without exploiting any label information, the principal components recovered in an unsupervised way store the most informative elements in leading dimensions, allowing for clear improvements in downstream tasks (e.g. classification and mutual information estimation). Empirically, we find that such improvements are consistent irrespective of the number of trailing dimensions of latent codes dropped, which acts as evidence of the manifold hypothesis. At the same time, dropping leading dimensions results in significant decline in classification accuracy and mutual information estimation, which indicates that leading dimensions capture the most dominant variations of data whereas trailing dimensions store less informative ones. Neural-PCA on one hand preserves exact invertibility, while respecting the manifold hypothesis for interpretability and better representations for downstream tasks on the other.

Specifically, a Neural-PCA can be constructed by appending the proposed PCA block to any regular normalizing flow, continuous or discrete. A PCA block is a bijection that allows for an easy inverse and an efficient evaluation of its Jacobian determinant. We further show that, to encourage principal component learning, a non-isotropic base density is a desirable choice along with Neural-PCA. Moreover, in terms of sample generation and inference, we propose an efficient approach for learning orthogonal statistics in $SO(n)$ so that Neural-PCA can be properly evaluated and inversed. Experimental results suggest that Neural-PCA captures principal components in an unsupervised yet generative manner, improving performance in downstream tasks while maintaining the ability for generating visually authentic images and for density estimation.

*Shen Li is sponsored by Google PhD fellowship 2021.

2 Background

A Normalizing Flow is an invertible transformation from a complex distribution to a simple probability distribution, or vice versa. Formally, let $\mathbf{z} \in \mathcal{Z} \subseteq \mathbb{R}^n$ be a random latent variable with a known and tractable probability density (known as base density). In the context of generative models, a normalizing flow defines an invertible differentiable function $\mathbf{f}$ such that $\mathbf{x} = \mathbf{f}(\mathbf{z})$. By using the change of variable formula [Papamakarios *et al.*, 2019], the probability density function (pdf) of $\mathbf{x}$ is given by

$$p_X(\mathbf{x}) = p_Z(\mathbf{g}(\mathbf{x})) \left|\det\left(\frac{\partial \mathbf{g}}{\partial \mathbf{x}}\right)\right| = p_Z(\mathbf{z})\left|\det\left(\frac{\partial \mathbf{f}}{\partial \mathbf{z}}\right)\right|^{-1} \tag{1}$$

where $\mathbf{g}$ is the inverse of $\mathbf{f}$. The density $p_X(\mathbf{x})$ is called a pushforward of the base density $p_Z(\mathbf{z})$ by the function $\mathbf{f}$. This movement from base density to a complex density is the *generative direction*. Its inverse $\mathbf{g}$ acts in the opposite direction called *normalizing direction*, which pushes a complex irregular data distribution towards a simpler or more "normal" form. We refer readers to [Kobyzev *et al.*, 2020] for a complete review of normalizing flows.

Throughout the entire treatment of this paper, we define a normalizing flow in the *normalizing direction*: for any bijection $\mathbf{g}$, $\mathbf{x}$ denotes its input and $\mathbf{z}$ denotes the output, i.e. $\mathbf{z} = \mathbf{g}(\mathbf{x})$. Here, $\mathbf{x}$ and $\mathbf{z}$ can be, for example, 4D tensors for image data or 2D tensors for tabular data.

3 Neural PCA

Manifold hypothesis [Narayanan and Mitter, 2010] is a commonly-held assumption that data are extremely concentrated on a low-dimensional manifold embedded in the ambient space with a few noisy ones residing off the manifold in various directions. The manifold is often twisted and coupled, manifesting a complex landscape. Besides merely modeling the data distribution, we expect an NF to capture major data variances into the subspace $\mathcal{Z}$ spanned by leading dimensions and leave data noise in the trailing dimensions. This inspires us to derive an operator that expands/contracts and rotates the manifold in a variance-dependent (PCA-like) manner. Experiments suggest that the proposed module will affect all the preceding transforms in terms of back-propagation, thereby leading to the desired representations. The training algorithm of Neural-PCA is summarized in Algorithm 1.

3.1 PCA Block

Our proposed method, Neural-PCA, can be constructed by appending a PCA block to any regular normalizing flow $\mathbf{h}$ (defined in the normalizing direction). A PCA block consists of a batch normalization layer (BatchNorm) [Dinh *et al.*, 2016] and a PCA layer, which defines two invertible transforms: Φ and Ψ, respectively.

Formally, Φ is defined as: $\mathbf{x} \xrightarrow{\Phi} \mathbf{z} := \alpha \frac{\mathbf{x}-\tilde{\boldsymbol{\mu}}}{\sqrt{\tilde{\boldsymbol{\sigma}}^2+\epsilon}} + \beta$ where $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\sigma}}$ are the estimated batch mean and variance, α is a rescaling factor and β is an offset. Both can be learned end-to-end. However, in an endeavor to perform principal component analysis, we fix $\beta = 0$ to enforce zero mean.

Algorithm 1 Training algorithm of Neural-PCA

Input: Batched training data $\{X^{(1)}, ..., X^{(M)}\}$; Baseline flow $\mathbf{h}_\theta$; PCA-block $\Phi_\alpha \circ \Psi$; learnable parameters $\Theta := \{\theta, \alpha\}$.
Output: The optimal Θ^*; the BatchNorm statistics $\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\sigma}}$; and $\tilde{V}$.

for all epoch **do**
  for all m in $1, ..., M$ **do**
    $Z^{(m)}, \log|\det(J_h)| \leftarrow \mathbf{h}(X^{(m)})$;
    $Z^{(m)}, \log|\det(J_\Phi)| \leftarrow \Phi(Z^{(m)})$; ▷ BN Layer
    $U^{(m)}, \Sigma^{(m)}, V^{(m)T} \leftarrow \text{SVD}(Z^{(m)})$;
    $Z^{(m)} \leftarrow Z^{(m)} V^{(m)}$; ▷ PCA Layer
    $J \leftarrow \log p(Z^{(m)}) + \log|\det(J_g)|$;
    $\Theta \leftarrow \Theta - \eta \nabla_\Theta J$; ▷ η is the learning rate
$\bar{\boldsymbol{\mu}} \leftarrow \mathbf{0}, \bar{\boldsymbol{\sigma}} \leftarrow \mathbf{0}, \overline{V} \leftarrow \mathbf{O}$;
for all m in $1, ..., M$ **do** ▷ Computing training statistics
  $Z^{(m)}, \log|\det(J_h)| \leftarrow \mathbf{h}(X^{(m)})$;
  $Z^{(m)}, \log|\det(J_\Phi)| \leftarrow \Phi(Z^{(m)})$;
  $U^{(m)}, \Sigma^{(m)}, V^{(m)T} \leftarrow \text{SVD}(Z^{(m)})$;
  $\bar{\boldsymbol{\mu}} \leftarrow \bar{\boldsymbol{\mu}} + \text{mean}(Z^{(m)})$, $\bar{\boldsymbol{\sigma}} \leftarrow \bar{\boldsymbol{\sigma}} + \text{Var}(Z^{(m)})$;
  $\overline{V} \leftarrow \overline{V} + V^{(m)}$;
$\bar{\boldsymbol{\mu}} \leftarrow \bar{\boldsymbol{\mu}}/M$; $\bar{\boldsymbol{\sigma}} \leftarrow \bar{\boldsymbol{\sigma}}/M$; $\overline{V} \leftarrow \overline{V}/M$;
$\tilde{V} \leftarrow \arg\min_{V \in SO(n)} \|\overline{V} - V\|_F$; ▷ Mean rotation

After BatchNorm, we have normalized batched data X as input for the PCA layer Ψ. Here, X is a matrix of shape (B, C), where B is the batch size. Each row of X represents one sample of dimensionality C. Then, the PCA layer performs full SVD on X: $X = U\Sigma V^T$, yielding a C-by-C orthogonal rotation matrix V. In the forward pass, the PCA layer acts as a rotation of the data X by post-multiplying V such that the 1st column (dimension) contains the largest variance, followed by the 2nd, etc. Finally, the PCA layer outputs the resultant matrix $Z = XV$ of shape (B, C). In aggregate, Neural-PCA in the *normalizing direction* can be constructed by composition: $\mathbf{g} := \Psi \circ \Phi \circ \mathbf{h}$.

3.2 Choices of Base Density

The optimization objective of Neural-PCA is in the same spirit of a regular NF, i.e. $\max_\Theta \mathbb{E}[\log p_\Theta(\mathbf{x})]$ (hereinafter w.l.o.g. we consider one sample log-likelihood and drop $\mathbb{E}[\cdot]$ to avoid notation clusters). A closed form objective can be derived by applying the change-of-variable formula (1). Oftentimes, prior works choose an isotropic Gaussian (IG, e.g. normal distribution) as the base density for simplicity. In contrast, we show that this choice discourages the ordering of principal components. Instead, we assign different Gaussian variances to different dimensions. This induces a non-isotropic Gaussian as base density. Without loss of generality, each dimension i is assigned with a Gaussian variance σ_i, for $i = 1, ..., n$ with the relation $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$ (equality holds when IG is employed). Then, the optimization objective (denoted by $\mathcal{J}(\Theta; \sigma_1, ..., \sigma_n)$) can be written as

$$\max_\Theta \mathcal{J}(\Theta; \sigma_1, ..., \sigma_n) := \log p_\Theta(\mathbf{x})$$
$$= -\frac{1}{2}\sum_{i=1}^{n} \frac{z_i^2}{\sigma_i^2} - \sum_{i=1}^{n} \log \sigma_i - \frac{n}{2}\log 2\pi + \log\left|\det\left(\frac{\partial \mathbf{g}_\Theta}{\partial \mathbf{x}}\right)\right| \tag{2}$$

where $\mathbf{z} = \mathbf{g}_{\mathbf{\Theta}}(\mathbf{x})$ and Θ is the learnable parameters of $\mathbf{g}$. Note that σ_i's are predefined hyperparameters and that the optimization is performed over Θ only. Then, the objective can be simplified as

$$\begin{aligned} &\max_{\Theta} \mathcal{J}(\Theta; \sigma_1, ..., \sigma_n) \\ &= -\frac{1}{2}\sum_{i=1}^{n}\frac{z_i^2}{\sigma_i^2} - \frac{n}{2}\log 2\pi + \log\left|\det\left(\frac{\partial \mathbf{g}_\Theta}{\partial \mathbf{x}}\right)\right| \end{aligned} \tag{3}$$

Alternative to such a hard assignment of Gaussian variance σ_i's, a hierarchical model shown in Figure 1(b) can relax the statistical dependency. In experiments, we find that using the model illustrated in Figure 1(a) yields sufficiently good performance. The satisfactory performance can be explained by Theorem 1, which reveals its relation to the hierarchical Bayesian generative model with $\tilde{\sigma}_i$'s as random variables obeying uniform distributions, i.e. $\tilde{\sigma}_i \sim \mathcal{U}[\alpha_i, \beta_i]$.

Theorem 1. *Maximizing $\mathcal{J}(\Theta; \sigma_1, ..., \sigma_n)$ with respect to Θ is equivalent to maximizing a lower bound of the log-likelihood of a hierarchical Bayesian generative model with $\tilde{\sigma}_i$'s as random variables obeying uniform distributions $\mathcal{U}[\alpha_i, \beta_i]$, where α_i and β_i are subject to $\sigma_i^2 = \alpha_i\beta_i$, $0 < \alpha_i < \beta_i \leq 1$ for all i's; that is,*

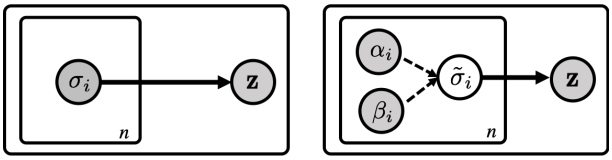
$$\begin{aligned} &\max_{\Theta} \mathcal{J}(\Theta; \sigma_1, ..., \sigma_n) \\ &\leq \log p(\mathbf{z}|\alpha_i, \beta_i, i = 1, ..., n) + \log\left|\det\left(\frac{\partial \mathbf{g}_\Theta}{\partial \mathbf{x}}\right)\right| \end{aligned} \tag{4}$$

Proof. A full proof can be found in Appendix A. □

Remark 1. Suppose all the uniform distributions have equal variance, which indicates that $\beta_i - \alpha_i = \tau$ (τ is a constant), for all i's. Note that $\sigma_i = \sqrt{\alpha_i\beta_i}$. Then these two equations yield that $\sigma_i = \sqrt{\alpha_i^2 + \tau\alpha_i}$, and further that α_i is an increasing function of σ_i for $\sigma_i > 0$ and so is β_i. This implies that the predefined hyperparameters σ_i's with the ordering $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$ lead to the maximization of the log-likelihood lower bound of the hierarchical Bayesian generative models with $\tilde{\sigma}_i$'s locating at a non-decreasing sequence of intervals $[\alpha_i, \beta_i]$, separately for all i's. As shown in Figure 1, the sub-figure (a) illustrates the graphical model to fit practically and (b) demonstrates the model that (a) induces by maximizing its log-likelihood lowerbound.

Remark 2. Theorem 1 suggests that $\mathcal{J}(\Theta; \sigma_1, ..., \sigma_n)$ is a lower bound of log-likelihood of the hierarchical Bayesian model induced (see Figure 1(b)) when $\beta_i \leq 1$. In experiments, to achieve this theoretical bound, we enforce all σ_i's within the open interval $(0, 1)$ evenly spaced in descending order. Since data intrinsic dimension is inaccessible and cannot be known *a priori* in most cases, such an "even-spacing" assignment is a reasonable choice for principal component learning.

Remark 3. Evaluating the log-determinant term in Eq. (3) is prohibitively expensive. We propose an efficient training procedure of Neural-PCA, which requires a gradient-stopping trick. The specific treatment is relegated to Appendix B.



(a) The model to fit (b) The model induced

Figure 1: Graphical models to fit and induced.

3.3 Learning Mean Rotation as Training Statistics

In terms of sampling and inference, a regular normalizing flow with BatchNorm layers utilizes exponential moving averages of mean and variance as training statistics when the forward or inverse operation proceeds in the test mode. However, such a moving-averaging strategy cannot be directly applied to the PCA layer in Neural-PCA, since all V's are in the special orthogonal group $SO(n)$ and arithmetic averaging is not closed in $SO(n)$.

Theorem 2. *For a finite positive integer $M < \infty$, the mean rotation $\mathfrak{M}(V^{(1)}, ..., V^{(M)})$ of $V^{(1)}, ..., V^{(M)} \in SO(n)$ is the orthogonal projection of the arithmetic mean of all rotation matrices $\overline{V} = \sum_{m=1}^{M} V^{(m)}/M$ in the linear space $\mathbb{R}^n$ onto the special orthogonal group $SO(n)$.*

Proof. The proof is relegated to Appendix C. □

Fortunately, according to Theorem 2, we can leverage the relation between the mean rotation and the usual arithmetic mean, which allows for learning training statistics in a PCA block (including the mean and variance of BatchNorm and the rotation matrix of PCA layer):

In the last training epoch, we freeze all learnable parameters Θ of Neural-PCA, and perform an extra pass over the training data to obtain arithmetic means of statistics in BatchNorm (i.e. $\bar{\boldsymbol{\mu}}$ and $\bar{\boldsymbol{\sigma}}$) and the arithmetic mean of $V^{(m)}$ for all $m = 1, ..., M$, where M is the total number of batches in the last training epoch. The arithmetic means of the statistics, $\bar{\boldsymbol{\mu}}$ and $\bar{\boldsymbol{\sigma}}$, are used in place of the exponential moving averaged ones for the BatchNorm layer. As for the PCA layer, by virtue of Theorem 2, we solve an additional optimization problem which finds a matrix $\tilde{V}$ that is closest to the arithmetic mean $\overline{V}$ in the $SO(n)$ group: $\tilde{V} = \arg\min_{V \in SO(n)} \|\overline{V} - V\|_F$, where $\overline{V} = \sum_i V^{(i)}/M$ and $\tilde{V}$ denotes the mean rotation $\mathfrak{M}(V^{(1)}, \cdots, V^{(M)})$. We show that this optimization problem has a closed form solution: $\tilde{V} = QP^T$ with the proof relegated to Appendix D. Here, Q and P are left- and right-singular vectors of $\overline{V}$. Note that now the rotation matrix $\tilde{V}$ does not depend on any test batch statistics when Neural-PCA is evaluated in the test mode. Hence, the PCA layer's forward (or inverse) operation can proceed by post-multiplying $\tilde{V}$ (or $\tilde{V}^T$) with zero log-determinant of Jacobian matrix. See Algorithm 1 for the entire training procedure.

3.4 Measuring the Usefulness of Representations

Given a flow-based representation $\mathbf{z} = \mathbf{g}(\mathbf{x}) \in \mathbb{R}^n$, we consider its corrupted representations with the leading or trailing

κ dimensions removed, denoted by $\mathbf{z}_{\downarrow\kappa^+}$ and $\mathbf{z}_{\downarrow\kappa^-}$, respectively, for $\kappa = 0, ..., n-1$. Note that $\mathbf{z}_{\downarrow 0^+} = \mathbf{z}_{\downarrow 0^-} = \mathbf{z} := \mathbf{z}_{\downarrow 0}$ which denotes the original representation $\mathbf{z}$ with zero dimensions removed. Since Neural-PCA aligns dimensions in descending order, removing trailing dimensions amounts to retaining leading principal components. We are interested in the usefulness of corrupted representations in terms of discriminativeness and informativeness, which can be measured by the following downstream metrics:

Nonlinear classification accuracy. For each κ, we train a classifier using $\mathbf{z}_{\downarrow\kappa^+}$ or $\mathbf{z}_{\downarrow\kappa^-}$ of the training split of a dataset, choose the best model using the validation split and finally evaluate classification accuracy using the test split. For evaluation in nonlinear classification, we implement an MLP with one hidden layer (200 units) in the following network architecture: $(n-\kappa)\ -200-\mathrm{ReLU}-\mathrm{Dropout}(0.2)-\mathrm{n_C}$, where $(n-\kappa)$ is the number of remaining dimensions and $\mathrm{n_C}$ is the number of classes of a dataset in question. Note that such a narrow architecture (with 200 units) limits the expressiveness of the classifier in order to better show the class separability of the learned representations.

Linear classification accuracy. In similar fashion, for evaluation in linear classification, the classifier is replaced by a support vector machine (SVM) with linear kernel.

Mutual information estimate. Mutual information (MI) measures the statistical dependencies between two random variables. Going a step further, recent work [Tsai *et al.*, 2020] proposed neural methods for point-wise dependency (PD) estimation, characterizing the instance-level dependency between a pair of events taken by random variables. Unlike prior methods [Belghazi *et al.*, 2018][Poole *et al.*, 2019], this fine-grained characterization is suitable for deterministic normalizing flows (when an instance $\mathbf{x}$ is given, $\mathbf{z}_{\downarrow\kappa^+}$ or $\mathbf{z}_{\downarrow\kappa^-}$ is completely determined, but not the other way around for $\kappa \neq 0$). To estimate MI between $\mathbf{x}$ and $\mathbf{z}_\kappa$ ($\kappa \neq 0$), we employ the density-ratio fitting method [Tsai *et al.*, 2020], of which the objective is given by

$$\sup_{\varphi} \mathbb{E}_{P_{X,Z_{\downarrow\kappa}}}\left[\hat{r}_\varphi(\mathbf{x}, \mathbf{z}_{\downarrow\kappa})\right] - \frac{1}{2}\mathbb{E}_{P_X P_{Z_{\downarrow\kappa}}}\left[\hat{r}^2_\varphi(\mathbf{x}, \mathbf{z}_{\downarrow\kappa})\right] \quad (5)$$

Here, we instantiate the density-ratio estimator as the inner-product in the embedding space followed by softplus activation to ensure positiveness, i.e. $\hat{r}_\varphi(\mathbf{x}, \mathbf{z}_{\downarrow\kappa}) := 1 + \mathtt{softplus}(\langle\phi(\mathbf{f}(\mathbf{x})), \phi(\mathbf{z}_{\downarrow\kappa})\rangle)$, where $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^{n-\kappa}$ is a feature map, and $\phi : \mathbb{R}^{n-\kappa} \mapsto \mathbb{R}^{d_e}$ is a nonlinear mapping with d_e the dimensionality of the embedding space. Both $\mathbf{f}$ and ϕ are parameterized by neural networks. Then, mutual information can be evaluated by taking the expectation of the estimated log-density-ratio: $I(X; Z_{\downarrow\kappa}) \approx \mathbb{E}_{P_{X,Z_{\downarrow\kappa}}}[\log \hat{r}_\varphi(\mathbf{x}, \mathbf{z}_{\downarrow\kappa})]$. Mathematically, higher MI, $I(X; Z_{\downarrow\kappa})$, implies lower conditional entropy $H(X|Z_{\downarrow\kappa})$, suggesting that partial latent representation (corrupted) can be used to infer the original data $\mathbf{x}$.

4 Related Work

Finding a good rotation matrix for ICA or PCA has been explored in literature. [Meng *et al.*, 2020] proposed a trainable rotation layer, whereby the rotation matrix is parameterized with multiple trainable Householder reflections. Specifically, an $n \times n$ orthogonal matrix R is represented as a product of n Householder reflections: $R = H_1 H_2 \cdots H_n$, where $H_i = I - 2\mathbf{v}_i\mathbf{v}_i^T / \|\mathbf{v}_i\|_2^2$ for any vector $\mathbf{v}_i \in \mathbb{R}^n$, $i = 1, ..., n$.

However, full parameterization of a rotation matrix costs $\mathcal{O}(n^2)$ space, which precludes its application for high dimensional data. A potential remedy is to utilize patch-based parameterization of rotation matrices [Meng *et al.*, 2020]. This amounts to learning rotation in each subspace separately, as the resulting rotation matrix is block-diagonal as a whole. This significantly reduces the number of parameters overall. However, we empirically show that the trainable rotation layer is agnostic to intrinsic data variations and thus fails to learn discriminative representations. In contrast, our mean rotation learning method is parameter-free and variation-aware, which provides useful representations for downstream tasks.

Model Name	#Extra Params	BatchNorm	PCA Layer	Base
Baseline	0	–	–	IG
Baseline-NIG	0	–	–	NIG
Baseline-BN	$\mathcal{O}(n)$	✓	–	NIG
Baseline-R	$\mathcal{O}(n^2)$	–	–	NIG
Baseline-BN-R	$\mathcal{O}(n^2)$	✓	–	NIG
Neural-PCA-IG	$\mathcal{O}(n)$	✓	✓	IG
Neural-PCA	$\mathcal{O}(n)$	✓	✓	NIG

Table 1: Model variants for comparison. IG denotes isotropic Gaussian, and NIG denotes non-isotropic Gaussian. Base denotes Base Density. Here, n denotes the full dimensionality.

5 Experiments

5.1 Implementation Details

To demonstrate the efficacy of Neural-PCA in terms of its contributing components, throughout experiments, we consider several model variants shown in Table **??** for comparison. A baseline model (Baseline) is a regular normalizing flow with an isotropic Gaussian $\mathcal{N}(\mathbf{0}, I)$ as its base density. "Baseline-NIG" is a baseline model that uses a non-isotropic Gaussian as base density. This is to show the efficacy of the PCA block of Neural-PCA. "Baseline-BN" is a baseline model with a BatchNorm layer at the end. This is to show the necessity of PCA layer in a PCA block. "Baseline-R" is a baseline model with a trainable rotation layer. "Neural-PCA-IG" is the same as Neural-PCA except for the choice of an isotropic Gaussian $\mathcal{N}(\mathbf{0}, I)$ as base density. We show in experiments that to learn useful representations, the PCA block and non-isotropy of base density are two essential building blocks complementary to one another.

5.2 Evaluation on Toy Datasets

For intuitive understanding, we conduct experiments on a two-dimensional toy dataset, Two-Spiral. As shown in Figure 2, each of the two colors represents one class. However, this label information is not given during training, as the goal is to show improvements on downstream tasks with latent codes that are learned in an unsupervised manner.

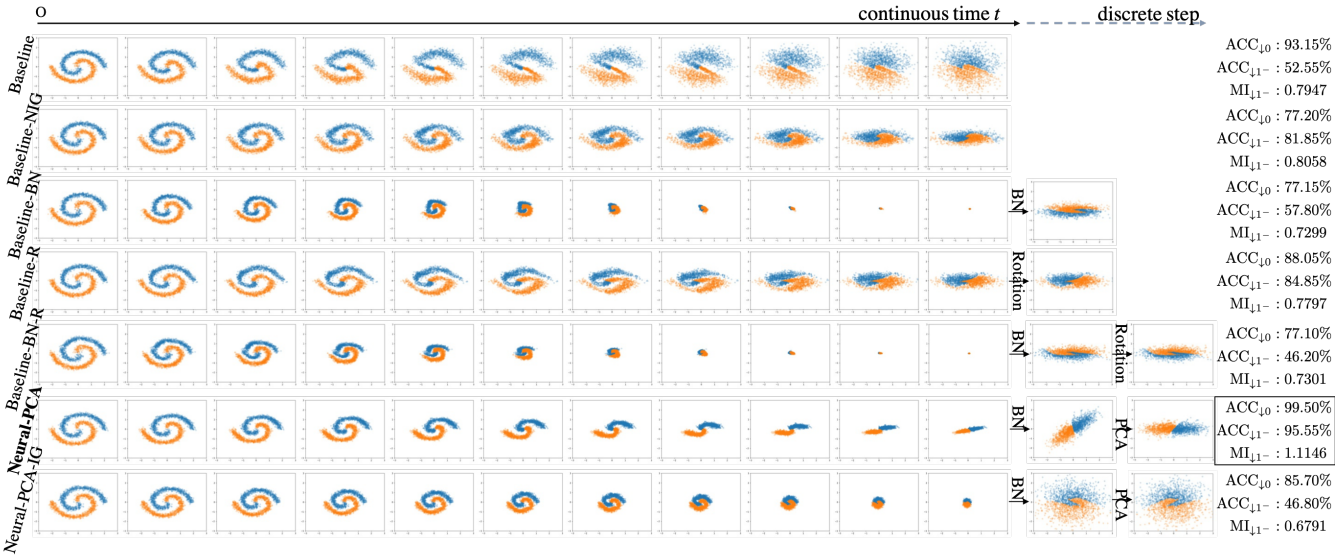


Figure 2: Evolution of latents through time on 2D Two-Spiral in different model variants. The first column is the target density. Here, $\text{ACC}_{\downarrow 0}$ denotes test classification accuracy with full latent codes, $\text{ACC}_{\downarrow 1-}$ and $\text{MI}_{\downarrow 1-}$ denote test accuracy and mutual information, respectively with the last dimension removed (that is, projecting data to the horizontal axis). Neural-PCA outperforms others considerably in classification accuracy and mutual information estimate by learning linearly-separable latent representations, acting in a way of unrolling the coupled spiral.

We use the state-of-the-art flow, FFJORD [Grathwohl *et al.*, 2018], as the baseline flow for modeling distribution of tabular data. FFJORD is a continuous flow that allows unrestricted architectures as compared to Neural ODE [Chen *et al.*, 2018]. For an instantiation of Neural-PCA, a PCA block is appended to a baseline flow. To encourage the latent ordering, a non-isotropic Gaussian density $\mathcal{N}(\mathbf{0}, \text{diag}(1, 0.1))$ is used as the base density. The batch size for training is set to 100 for all models. For Neural-PCA, the proposed projection method (cf. Section 3.3) is utilized to aggregate all rotation matrices computed from different batches. After training, models are evaluated on a linear classification task using the learned representations of 2000 test points. To measure linear separability, we use SVM with linear kernel as the binary classifier.

Ideally, as a diffeomorphism, a well-fitted flow model should unroll a spiral into a finite line segment lying in the latent space. The length of the spiral should be proportional to the largest variance and the thickness to the least. Further, to encourage axis-aligned representations, a non-isotropic Gaussian density with a diagonal covariance matrix is chosen as the base density. The ordering of the diagonals encodes our prior belief and encourages the desired unrolling behavior. This hypothesis verifies our choices for the base density, a non-isotropic Gaussian $\mathcal{N}(\mathbf{0}, \text{diag}(1, 0.1))$. As shown in Figure 2, Neural-PCA is capable of unrolling the spiral, whereas other models contract or expand the spiral, disrespecting its intrinsic manifold. Intriguingly, this unrolling behavior results in far clearer linear separability in the latent space (99.50% accuracy). Also noting that, for Neural-PCA after PCA layer, the horizontal dimension contains the largest variation and the vertical has the least, we observe that the classification boundary is almost perpendicular to the first major axis. This suggests that these two classes can still be well-separated even after dimensionality reduction (projecting points to the horizontal axis achieves 95.55% accuracy). The other methods apparently fail to do so. This visually reveals why Neural-PCA outperforms the others in learning better representations of data. In mutual information estimation, Neural-PCA also attains a higher MI estimate than other model variants. Detailed comparison and analyses can be found in Appendix E.

5.3 Evaluation on High-Dimensional Datasets

To demonstrate the effectiveness of Neural-PCA on high-dimensional data, we conduct evaluations on image datasets: MNIST and FashionMNIST, which both give $1024 (= 32 \times 32 \times 1)$-dimensional spaces. Note that the original spatial sizes of MNIST and FashionMNIST data are $(28 \times 28 \times 1)$. Preprocessing includes padding zeros on boundaries for such data. Dequantization is also performed by adding uniform noise to the images [Uria *et al.*, 2013][Theis *et al.*, 2015].

We choose the state-of-the-art flow, RQ-NSF(C) [Durkan *et al.*, 2019], as the baseline flow for modeling image data. RQ-NSF(C) is an analytically invertible model based on monotonic rational-quadratic splines that improves the flexibility of coupling transforms. Similarly as in toy data, to encourage the ordering, we set σ_i's evenly from 1.0 down to 0.005, respectively for $i = 1, \cdots, n$. All models are trained for 100,000 iterations with Adam optimizer. The batch size is set to 1800. Learning rates are annealed from 5×10^{-4} down to zero following a cosine learning policy. Similarly in the Two-Spiral setting, mean rotations are calculated using the projection method proposed in Section 3.3.

Nonlinear Classification. As shown in Figure 3(a), Neural-PCA outperforms all the other model variants by

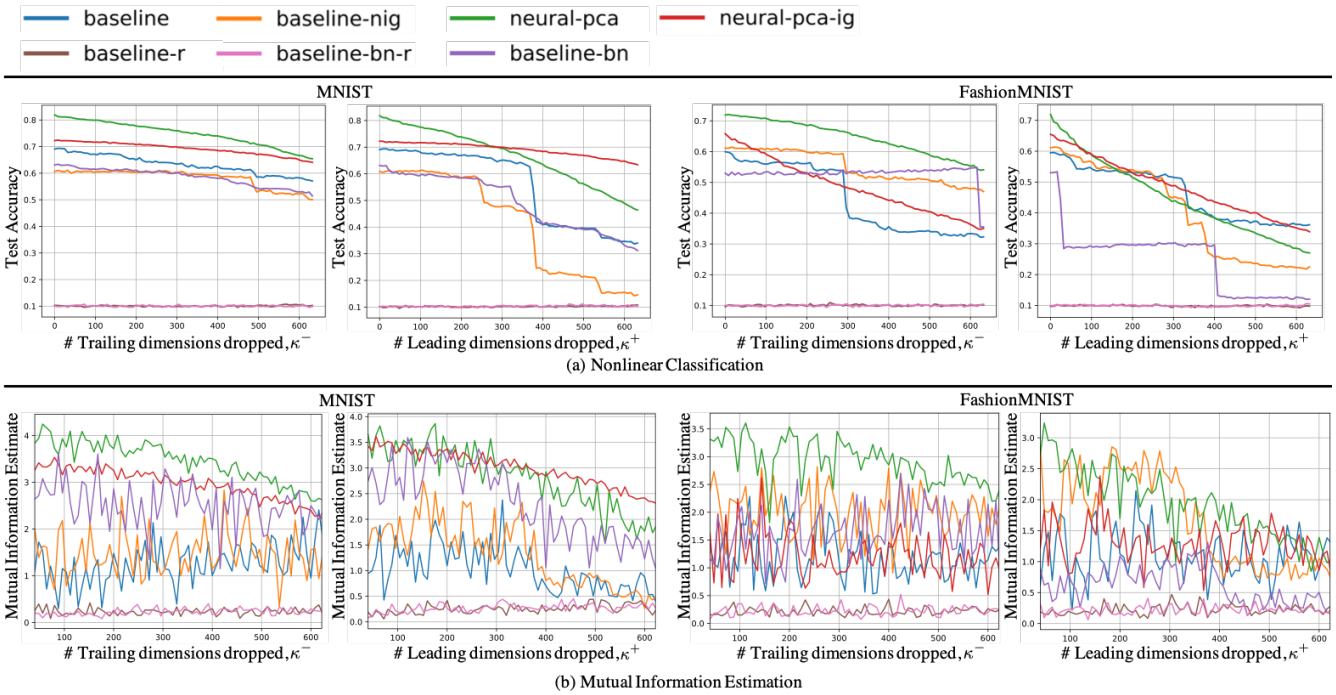


Figure 3: Comparison of the usefulness of representations learned by different models in terms of (a) nonlinear classification and (b) mutual information estimation. Similar behavior can be observed for linear classification by SVM (cf. Appendix).

large margins when zero dimensions are removed as well as when a large fraction of trailing dimensions are removed ($624 > 1024/2$). Removing leading dimensions, on the other hand, gives rise to rapid decline in test accuracy of Neural-PCA, which suggests that the principal components recovered in our proposed unsupervised manner manifests strong discriminativeness of the data. We observe that Baseline-R and Baseline-BN-R learn poor representations for classification with test accuracy $\sim 10\%$; as the number of categories is 10 for all datasets, these two models amount to random guess.

Linear Classification by SVM. For linear classification, we use SVM with linear kernel in place of the MLP as in nonlinear classification. Similar behaviors can be observed (cf. Appendix): substantial accuracy improvements of $\gtrsim 10\%$ on MNIST and FashionMNIST when zero dimensions are removed. The improvements remain clear even after removing a large proportion of trailing dimensions.

Mutual Information Estimation. In terms of mutual information estimation, similar and consistent behaviors of Neural-PCA as in classification metrics can be observed, as suggested in Figure 3(b).

5.4 Comparison with Post-PCA models.

This section demonstrates the necessity of training flow with PCA-Block end-to-end jointly. As in the toy data setting, we consider post-PCA models based on baseline features learned by Baseline, Baseline-NIG, Baseline-BN, Baseline-BN-R and Neural-PCA-IG. Our results suggest that the representations learned by these models have already been axis-aligned, and therefore post-PCA on them would not result in non-trivial rotation further. Comparison results and detailed analysis can be found in Appendix F.

5.5 Miscellaneous

Other interesting properties of Neural-PCA, including generation quality, density estimation, mean rotation analysis and latent interpolation, are relegated to Appendix G, H and I, respectively. The implementation details for experiments can be found in Appendix J. Code is publicly available at `https://github.com/MathsShen/Neural-PCA`.

6 Conclusions

We have proposed Neural-PCA that operates in full dimensionality while capturing principal components in descending order. Consequently, the principal components recovered store the most informative elements in leading dimensions and leave the negligible in trailing ones, allowing for considerable improvements in downstream tasks. Echoing the manifold hypothesis, such improvements are empirically found consistent irrespective of the number of trailing dimensions dropped. Our work suggests that necessary inductive biases be introduced into generative modelling when representation quality is of interest.

Acknowledgements

We thank all anonymous reviewers for constructive suggestions on the manuscript of this paper. This work is in memoriam of Shen Li's Grandmother, Baoling Hong (1939-2021).

References

[Belghazi *et al.*, 2018] Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeswar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and R Devon Hjelm. Mine: mutual information neural estimation. *arXiv preprint arXiv:1801.04062*, 2018.

[Bengio *et al.*, 2013] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[Chapelle *et al.*, 2009] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

[Chen *et al.*, 2018] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

[Chen *et al.*, 2019] Ricky T. Q. Chen, Jens Behrmann, David K Duvenaud, and Joern-Henrik Jacobsen. Residual flows for invertible generative modeling. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 9916–9926. Curran Associates, Inc., 2019.

[Dempster *et al.*, 1977] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[Dinh *et al.*, 2014] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[Dinh *et al.*, 2016] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *International Conference on Learning Representations*, 2016.

[Durkan *et al.*, 2019] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In *Advances in Neural Information Processing Systems*, pages 7509–7520, 2019.

[Fefferman *et al.*, 2016] Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. Testing the manifold hypothesis. *Journal of the American Mathematical Society*, 29(4):983–1049, 2016.

[Grathwohl *et al.*, 2018] Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. In *International Conference on Learning Representations*, 2018.

[Hjelm *et al.*, 2018] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2018.

[Kingma and Dhariwal, 2018] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224, 2018.

[Kobyzev *et al.*, 2020] Ivan Kobyzev, Simon Prince, and Marcus Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 1–1, 2020.

[Meng *et al.*, 2020] Chenlin Meng, Yang Song, Jiaming Song, and Stefano Ermon. Gaussianization flows. In *International Conference on Artificial Intelligence and Statistics*, 2020.

[Narayanan and Mitter, 2010] Hariharan Narayanan and Sanjoy Mitter. Sample complexity of testing the manifold hypothesis. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems-Volume 2*, pages 1786–1794, 2010.

[Papamakarios *et al.*, 2019] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019.

[Poole *et al.*, 2019] Ben Poole, Sherjil Ozair, Aaron Van Den Oord, Alex Alemi, and George Tucker. On variational bounds of mutual information. In *International Conference on Machine Learning*, pages 5171–5180. PMLR, 2019.

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

[Theis *et al.*, 2015] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.

[Tsai *et al.*, 2020] Yao-Hung Hubert Tsai, Han Zhao, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. Neural methods for point-wise dependency estimation. *arXiv preprint arXiv:2006.05553*, 2020.

[Uria *et al.*, 2013] Benigno Uria, Iain Murray, and Hugo Larochelle. Rnade: the real-valued neural autoregressive density-estimator. In *Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 2*, pages 2175–2183, 2013.