DISCOVERING ORDER IN UNORDERED DATASETS: GENERATIVE MARKOV NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

The assumption that the data samples are independently identically distributed is the backbone of many learning algorithms. Nevertheless, datasets often exhibit rich structures in practice, and thus we argue that there exist some unknown orders within the data instances. Aiming to find such orders, we introduce a novel Generative Markov Network (GMN) which we use to extract the order of data instances automatically. Specifically, we assume that the instances are sampled from a Markov chain. Our goal is to learn the transitional operator of the chain as well as the generation order by maximizing the generation probability under all possible data permutations. One of our key ideas is to use neural networks as the soft lookup table for approximating the possibly huge, but discrete transition matrix. This strategy allows us to amortize the space complexity with a single model and make the transition operator generalizable to unseen instances. To ensure the learned Markov chain is ergodic, we propose a greedy batch-wise permutation scheme that allows fast training. Empirically, we evaluate the learned Markov chain by showing that GMNs are able to discover orders among data instances and also perform comparably well to state-of-the-art methods on the one-shot recognition benchmark task.

1 INTRODUCTION

Recent advances in deep neural networks offer great potentials for machines to learn automatically without humans interventions. For instance, Convolutional Neural Networks (CNNs) (Krizhevsky et al., 2012) provided an automated way for learning image feature representations. Compared to hand-crafted ones such as SIFT and SURF, these hierarchical deep features demonstrate superior performance in recognition problems (Xu et al., 2015; Finn & Levine, 2017) and transfer learning (Glorot et al., 2011). Another example would be *learning to learn* for automatic parameter estimation. Andrychowicz et al. (2016) proposed to update model parameters without any pre-defined update rule such as stochastic gradient descent (SGD) or ADAM (Kingma & Ba, 2014). Surprisingly, this update-rule-free framework showed better performance and faster convergence on both object recognition and image style transformation tasks. In our paper, we investigate the following novel question. Given an unordered dataset where instances may be exhibiting some *i*mplicit order, can we order a dataset automatically according to this order?

We argue that such order often exists even when we are dealing with the data that are naturally thought of as being i.i.d. sampled from a common though complex distribution. For example, let's consider a dataset consisting of the joint locations on the body of the same person taken on different days. The data i.i.d. assumption is justified since postures of a person took on different days are likely unrelated. However, we can arrange the data instances such that the joints follow an articulated motion or set of motions in a way that makes each pose highly predictable giving the previous ones. Although this arrangement depends on the person as ballerinas' poses might obey different dynamics than the poses of tennis players, the simultaneous inference on the pose dynamics can lead to a robust model that explains the correlations among joints. To put it differently, if we reshuffle the frames of a video clip, the data can now be modeled by an i.i.d. model. Nevertheless, reconstructing the order leads to an alternative model where transitions between the frames are easier to fit the links between the latent structures and observations. The ballerina's dancing, if sampled very sparsely, can be thought of as a reshuffled video sequence that needs to be reordered such that a temporal model can generate it.

One naive and obvious way to find the order in a dataset is to perform sorting based on a predefined distance metric; e.g., the Euclidean distance between image pixel values. However, the distance metrics have to be predefined differently and empirically according to distinct types/characteristics of the datasets at hand. A proper distance metric for one domain may not be a good one for other domains. For instance, the $p-$distance is a good measure for DNA/RNA sequences (Nei & Kumar, 2000) while it does not characterize the semantic distances between images. We argue that the key component of the ordering problem lies in the discovery of proper distance metric in an automatic and adaptive way.

To approach this problem, we propose to learn a *distance-metric-free model* to discover the ordering in the dataset. More specifically, we model the data by treating them as if they were generated from a Markov chain. We propose to simultaneously train the transitional operator of the chain and find the best order by a joint optimization over the parameter space as well as all possible permutations. We term our model Generative Markov Networks (GMNs). One of the key ideas in the design of GMNs is to use neural networks as a soft lookup table to approximate the possibly huge, but discrete transition matrix. This strategy allows GMNs to amortize the space complexity using a unified model. Furthermore, due to the differentiable property of neural networks, the transitional operator of GMNs can also generalize on unseen, but similar data instances. As an additional contribution, to ensure the Markov chain learned by GMNs is ergodic, we propose a greedy batch-wise permutation scheme that allows fast training.

One related task is one-shot recognition which has only one labeled data per category in the target domain. Most of the work in this area considers learning a specific distance metric (Koch et al., 2015; Vinyals et al., 2016; Snell et al., 2017) or category-separation metric (Ravi & Larochelle, 2017; Finn et al., 2017) for the data. During the inference phase, they compute either the smallest distance or highest class prediction score between the support and query instances. Alternatively, from a generative modeling perspective, we can first generate the Markov chain for the support instances, then we fit the query instances into the Markov chain and decide the labels with the highest log-likelihood.

Empirically, we evaluate the learned Markov chain by showing that GMNs are able to discover implicit orders among data instances and also perform comparably well to state-of-the-art methods on the one-shot recognition benchmark task.

2 Related Work

The literature on deep generative models and stochastic sampling is abundant. Due to the space limit, we discuss the ones that are most relevant to our work.

Deep Generative Models: We consider two classes of deep generative models based on *ancestral sampling* and *iterative sampling*, respectively. Both Variational Autoencoders (VAEs) (Kingma & Welling, 2013) and Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) can be cast as *ancestral sampling*-based methods. In the inference phase, these approaches generate one sample from the model by performing a single inference pass from the underlying graphical models. As a comparison, methods based on *iterative sampling* perform multiple and iterative passes through all the variables in the corresponding graphical models. Usually, these methods involve simulating a Markov chain in the entire state space, and they aim at improving quality of generated samples by mixing the underlying chain. Recent works on this line of research include (Bengio et al., 2013; 2014; Sohl-Dickstein et al., 2015; Bordes et al., 2017; Song et al., 2017).

At a high level, our approach can be categorized as an *iterative sampling*-based model. However, the proposed model has significant differences in the following three ways. First, all the existing works assumed that training instances are i.i.d. sampled from the stationary distribution of a Markov Chain. This assumption is risky since, often the case, it is hard to measure whether a Markov chain has mixed or not. As a comparison, we only assume that data instances are sampled from the chain, without expecting the chain has mixed. As we will see later, the stationarity assumption in previous works often prevents them from observing the implicit data relationships. Second, prior approaches were proposed based on the notion of denoising models. In other words, their goal is to generate high-quality images; on the other hand, we aim at discovering orders in datasets. Third, to the best of our knowledge, all the existing works are implicit models in the sense that they only admit

efficient sampling schemes. In contrast, the proposed GMN is an explicit model where besides an efficient sampling procedure, the model maintains a tractable likelihood function that can be computed efficiently.

One-Shot Learning: Deep one-shot learning approaches can be divided into two categories as well: *distance-metric-learning* and *categories-separation-metric-learning* approaches. The former aim at either learning a similarity measurement between instance pairs (Koch et al., 2015) or applying specific metric loss based on cosine distance (Vinyals et al., 2016)/ Euclidean distance (Snell et al., 2017). These methods referred to *nonparametric* classifiers and relied heavily upon human design. As a comparison, methods in the second category usually offer more generalities. Typically, this type of methods tackled the problem using a meta-learning framework to train *parametric* classifiers. More precisely, they considered two levels of learning; the first stage is to update base learners' parameters and the second stage is to update parameters for the meta learner. Recent works (Ravi & Larochelle, 2017; Kaiser et al., 2017; Finn et al., 2017) belonged to this category.

The methods mentioned above viewed one-shot recognition as a discriminative task; on the contrary, we hold a generative perspective. Since we consider a Markov chain data generation assumption, we can directly decide the labels for query instances by fitting them into the Markov Chain (or the orders we observe) generated from support instances. This generative nature significantly decreases the difficulty of training as we no longer rely on any designed metric. More details will be covered in Sec. 4.

3 Generative Markov Networks

Let $\{s_i\}_{i=1}^n$ denote our training data which are assumed being generated from an unknown Markov chain. Our goal is to jointly recover the unknown Markov chain as well as the order of generation process. Note that since the generation order is unknown, even if the true Markov chain was given, it would still be computationally intractable to find the optimal order that best fits our data. To get around of this intrinsic difficulty, as we will see in Sec. 3.2 , we propose a greedy algorithm to find an order given the current estimation of the transitional operator.

We denote the underlying data order to be a permutation over $[n]$: $\tau = \{\tau(t)\}_{t=1}^n$, where $\tau(t)$ represents the index of the instance that was generated at the t-th step of the Markov chain. In other words, a Markov chain is formed as follows:

$$s_{\tau(1)} \to s_{\tau(2)} \to \cdots \to s_{\tau(n)}.$$

We consider all the possible permutations τ and the arbitrary distribution over these permutations, which leads to a joint log-likelihood estimation problem:

$$\max_{\theta,\mathcal{P}} \log\big(\sum_{\tau} \mathcal{P}(\tau)\, \mathcal{P}(\{s_i\}_{i=1}^n, \tau; \theta)\big) = \max_{\theta,\mathcal{P}} \log\big(\sum_{\tau} \mathcal{P}(\tau)\ \mathcal{P}^{(1)}(s_{\tau(1)}) \prod_{t=2}^{n} \mathcal{T}(s_{\tau(t)}|s_{\tau(t-1)}; \theta)\big),$$

where $\mathcal{P}^{(1)}(\cdot)$ is the initial distribution of the Markov chain and $\mathcal{T}(s'|s;\theta)$ is the transition operator parametrized by model parameters θ. Note that the effect of the initial distribution $\mathcal{P}^{(1)}(\cdot)$ diminishes with the increase of the data size n. Hence, without loss of generality, we assume $\mathcal{P}^{(1)}(\cdot)$ is uniform over all possible states, leading to the following optimization problem:

$$\max_{\theta,\mathcal{P}} \quad \log\left(\sum_{\tau\in\Pi(n)} \mathcal{P}(\tau) \prod_{t=2}^{n} \mathcal{T}(s_{\tau(t)}|s_{\tau(t-1)}; \theta)\right), \tag{1}$$

where $\Pi(n)$ is the set of all possible permutations over $[n]$. Unfortunately, direct optimization of (1) is computationally intractable. For each fixed θ and $\mathcal{P}$, the number of possible permutations, i.e., $|\Pi(n)|$, is $n!$. As we will see shortly, in what follows we will first introduce GMN and then present an efficient greedy algorithm in Sec. 3.3 to approximate this expensive function evaluation.

3.1 Parametrized Transition Operator

In practice, when the state space is huge, often we cannot afford to maintain the tabular transition matrix directly, which takes up to $O(d^2)$ space, where d is the number of states in the chain. For

example, if the state refers to the binary image $I \in \{0,1\}^p$, the size of the state space is $d = 2^p$. Hence, before optimizing (1), we should first find a family of functions to parametrize the transition operator $\mathcal{T}(\cdot|\cdot)$.

Being universal function approximators (Hornik et al., 1989), neural networks could be used to approximate the discrete structures which led to the recent success of deep reinforcement learning (Mnih et al., 2013). In our case, we utilize neural networks to approximate the discrete and tabular transition matrix. The advantages are two-fold: first, it significantly reduces the space complexity by amortizing the space required by each separate state into a unified model. Since all the states share the same model as the transition operator, there is no need to store the transition vector for each separate state explicitly. Second, neural networks allow better generalization of the transition probabilities across states. In most real-world applications, states, represented as feature vectors, are not independent from each other. As a result, the differentiable approximation to a discrete structure has the additional smoothness properties, which allows the transition operator to have a good estimate even for unseen states.

In more detail, let θ be the parameters of the neural network to approximate the transitional operator. We define

$$f_\theta(s, s') = \mathcal{T}(s'|s;\theta) : \mathbb{R}^p \times \mathbb{R}^p \to [0,1]$$

to be the transition function that takes two states s and s' as inputs and returns the corresponding transition probability. Note that one can consider each discrete transitional operator as a lookup table. For example, we use s and s' to locate the corresponding row and column of the table and read out its probability. From this perspective, the neural network works as a soft lookup table that outputs the transition probability given two states (features).

3.2 Greedy Approximation of the Optimal Order

As mentioned above, a direct evaluation of the objective function in (1) is computationally intractable at each given $\mathcal{P}$ and θ. Nevertheless, we can still develop a coordinate ascent style training algorithm to optimize (1) efficiently. The key insight comes from the following observation: for each fixed θ, there exists a point mass distribution over $\Pi(n)$ that achieves the maximum value for eq. (1). In other words, the maximum value of eq. (1) is given by the distribution that puts all the probability mass on τ^*, where τ^* is defined as

$$\tau^* = \operatorname*{arg\,max}_{\tau \in \Pi(n)} \sum_{t=2}^{n} \log \mathcal{T}(s_{\tau(t)}|s_{\tau(t-1)};\theta). \tag{2}$$

To see this, for any $\mathcal{P}$ over $\Pi(n)$, we have:

$$\begin{aligned}\log\left(\sum_\tau \mathcal{P}(\tau)\prod_{t=2}^{n}\mathcal{T}(s_{\tau(t)}|s_{\tau(t-1)};\theta)\right) &\le \log\left(\sum_\tau \mathcal{P}(\tau)\prod_{t=2}^{n}\mathcal{T}(s_{\tau^*(t)}|s_{\tau^*(t-1)};\theta)\right)\\ &= \log\left(\sum_\tau \mathbb{I}(\tau=\tau^*)\prod_{t=2}^{n}\mathcal{T}(s_{\tau^*(t)}|s_{\tau^*(t-1)};\theta)\right),\end{aligned}$$

where $\mathbb{I}(\tau = \tau^*)$ is the indicator function that takes value 1 iff $\tau = \tau^*$ otherwise 0. Realize that $\mathbb{I}(\tau = \tau^*)$ also defines a valid distribution over $\Pi(n)$, which proves our claim. In other words, given each θ, the optimization problem over $\mathcal{P}$ now reduces to finding the optimal permutation τ^* that gives the maximum likelihood on generating the data. However, without further assumption on the structure of the transitional operator, this is still a hard problem, which takes time $O(n!)$. Here we propose a greedy algorithm to approximate the optimal order, which takes time $O(n^2 \log n)$. We list the pseudocode in Alg. 1.

At a high level, Alg. 1 finds the approximate order by first enumerating all the possible states appearing in the first time step. For each of the following steps, it finds the next state by maximizing the transition probability at the current step, i.e., a local search to find the next state. The final approximate order is then defined to be the maximum of all these n orders. A naive implementation of this algorithm has time complexity $O(n^3)$, but we can reduce it to $O(n^2 \log n)$ by first pre-computing $\mathcal{T}(s_i|s_j;\theta)$, $\forall i, j \in [n]$ and then sort them so that the maximum finding operation in line 5 can be done in constant time.

Algorithm 1 Greedy Approximate Order

Input: Input data $\{s_i\}_{i=1}^n$ and transitional operator $\mathcal{T}(s_i|s_j;\theta)$

$v^* \leftarrow -\infty$
for $i = 1$ to n **do**
  $\tau_i(1) \leftarrow i$
  for $j = 2$ to n **do**
    $\tau_i(j) \leftarrow \max_{k \notin \{\tau_i(1),\ldots,\tau_i(j-1)\}} \mathcal{T}(s_{\tau_i(j-1)}|s_k;\theta)$
  end for
  $v_i \leftarrow \sum_{t=2}^n \log \mathcal{T}(s_{\tau_i(j-1)}|s_{\tau_i(t)};\theta)$
  if $v_i > v^*$ **then**
    $\hat{\tau} \leftarrow \tau_i$
    $v^* \leftarrow v_i$
  end if
end for
return $\hat{\tau}$

Algorithm 2 Optimization with Batch-Wise Permutation Training

Input: $\{s_i\}_{i=1}^n$, b_o, b, t, γ

Initialize $\theta^{(0)}$, $\{x_i^{(0)}\}_{i=1}^b$
for $k = 1$ to ∞ **do**
  if $k \equiv 1 (\mathrm{mod}\ t)$ **then**
    Sample $\{x_i^{(k)}\}_{i=1}^{b-b_o} \sim \{s_i\}_{i=1}^n$
    $\{x_i^{(k)}\}_{i=1}^b = \{\{x_i^{(k)}\}_{i=1}^{b-b_o}, \{x_i^{(k-1)}\}_{i=1}^{b_o}\}$
  end if
  Compute $\hat{\tau}^{(k)}$ using the Greedy Approximate Order (Alg. 1)
  Compute $\nabla_\theta^{(k-1)} \log \mathcal{P}(\{x_i\}_{i=1}^b;\theta^{(k-1)}) = \partial_\theta \sum_{t=2}^b \log \mathcal{T}(x_{\hat{\tau}^{(k)}(t)}|x_{\hat{\tau}^{(k-1)}(t-1)};\theta^{(k-1)})$
  $\theta^{(k)} = \theta^{(k-1)} + \gamma \nabla_\theta^{(k-1)} \log \mathcal{P}(\{x_i\}_{i=1}^b;\theta^{(k-1)})$
end for

Given the approximate order $\hat{\tau}$, we then proceed to optimize the model parameter θ by gradient based optimization. By now it should be clear that the whole algorithm is an instance of the famous coordinate descent algorithm, where we alternatively optimize over the order τ and the model parameter θ. Under this framework, since both optimizations over θ and τ will increase the objective function, the algorithm is guaranteed to converge.

3.3 Batch-Wise Permutation Training

The $O(n^2 \log n)$ computation to find the approximate order in Alg. 1 can be expensive when the size of the data is large. In this section we provide batch-wise permutation training to get around of this issue. The idea is to partition the original training set into batches with size b, and perform greedy approximate order on each batch. Assume $b \ll n$ is a constant, then now the effective time complexity becomes: $O(b^2 \log b) \cdot n/b = O(nb \log n)$, which is linear in n.

One issue on using batch-wise permutation training compared with using the whole dataset is that, because training data are partitioned into chunks, the learned transitional operator is not guaranteed to have nonzero transition probabilities between different chunks of data. In other words, it may happen that the learned transitional operator does not induce an ergodic Markov chain because of the isolated states. To ensure an ergodic Markov chain, we propose a simple strategy to enforce some samples are overlapped between the consecutive batches. We show the pseudocode in Alg. 2 for training our GMNs. In Alg. 2, b means the batch size, γ is the learning rate and $b_0 < b$ is the number of overlap states between consecutive batches.

3.4 INTRODUCING STOCHASTIC LATENT VARIABLES VIA VARIATIONAL BAYES INFERENCE

In this section we give a detailed description on how to implement the transitional operator where the state can be both discrete or continuous. As a first step, to prevent our GMNs from simply memorizing all the training data and their transitions, we introduce stochastic latent variables $z \in \mathbb{R}^{\mathbf{z}}$ via Variational Bayes Inference (Wainwright et al., 2008). The evidence lower bound (ELBO) of the log likelihood for the transition operator (i.e., $\log \mathcal{T}(x'|x;\theta)$) becomes:

$$\log \mathcal{T}(x'|x;\theta) \approx \mathbf{E}_{z\sim\mathcal{Q}(z|x;\phi)}\Big[\log \mathcal{P}(x'|x,z;\psi)\Big] - \mathcal{KL}\Big(\mathcal{Q}(z|x;\phi) \,||\, \mathcal{P}(z)\Big), \tag{3}$$

where $\mathcal{T}(x'|x;\theta)$ has been replaced by a distribution $\mathcal{P}(x'|x,z;\psi)$ parametrized by ψ, which allows us to make the dependence of x on z. Moreover, $\mathcal{KL}$ is the KL-divergence, $\mathcal{Q}(z|x;\phi)$ is an encoder function parametrized by ϕ that encodes latent code z given current state x, and $\mathcal{P}(z)$ is a fixed prior which we take its form as Gaussian distribution $\mathcal{N}(0,\mathbf{I})$. Typically, we use reparametrized trick to draw $\mathcal{Q}(z|x;\phi)$ from Gaussian $\mathcal{N}\big(\mu_{\mathcal{Q},\phi}(x), \sigma^2_{\mathcal{Q},\phi}(x)\mathbf{I}\big)$ where $\mu_{\mathcal{Q},\phi}(x)$ and $\sigma_{\mathcal{Q},\phi}(x)$ are learnable function.

Next, we consider two types of distribution family for $\mathcal{P}(x'|x,z;\theta)$: Bernoulli and Gaussian.

If $x \in \{0,1\}^p$ (i.e., a binary image), we define $\log \mathcal{P}(x'|x,z;\psi)$ as:

$$\log \mathcal{P}(x'|x,z;\psi) = x' \odot \log \Big(g_\psi(x,z)\Big) + (1-x') \odot \log \Big(1 - g_\psi(x,z)\Big), \tag{4}$$

when $\odot$ is element-wise multiplication and $g_\psi(x,z) : \Big[\{0,1\}^p, \mathbb{R}^{\mathbf{z}}\Big] \to [0,1]^p$.

If $x \in \mathbb{R}^p$ (i.e., a real-valued feature vector), we choose $\mathcal{P}(x'|x,z;\psi)$ to be fixed variance factored Gaussian $\mathcal{N}\Big(\mu_{\mathcal{P},\psi}(x,z), \sigma^2_{\mathcal{P}}\mathbf{I}\Big)$, where $\mu_{\mathcal{P},\psi}(x,z) : \mathbb{R}^{p+\mathbf{z}} \to \mathbb{R}^p$ and $\sigma_{\mathcal{P}}$ is a fixed variance. $\log \mathcal{P}(x'|x,z;\theta)$ can thus be defined as

$$\log \mathcal{P}(x'|x,z;\psi) = -\frac{1}{2\sigma^2_{\mathcal{P}}}||x' - \mu_{\mathcal{P},\psi}(x,z)||^2_2 + const., \tag{5}$$

where $const.$ is not related to the optimization of ψ.

For simplicity, we specify $\theta = \{\psi \cup \phi\}$. Therefore, the model parameters update for θ in (3) refers to the updates for ψ and ϕ.

4 EXPERIMENTS

4.1 DISCOVERING ORDERS IN DATASETS

We perform experiments on ordering data in three datasets: MNIST (LeCun et al., 1990), Horse (Borenstein & Ullman, 2002), and MSR_SenseCam (Jojic et al., 2010). We also provide another experiments on Moving MNIST (Srivastava et al., 2015) in Supplementary. Among these datasets, MNIST, Horse, and MSR_SenseCam do not have explicit orders. On the other hand, Moving MNIST can be seen as a collection of short video clips and thus it has an explicit order for each sequence of frames. Due to space limit, we only show partial ordering results. Please see Supplementary for the full version.

<**MNIST**> MNIST (LeCun et al., 1990) is a well-studied dataset that contains 60,000 training examples. Each example is a digit image with size 28x28. We rescale the pixel values to $[0,1]$.

<**Horse**> Horse dataset (Borenstein & Ullman, 2002) consists of 328 horse images collected from the Internet. Each horse is centered in a 30x40 image. For the preprocessing, the object-background segmentation is applied and the binary pixel value is set to 1 and 0 for object and background, respectively. Examples are show in Supplementary.

<**MSR_SenseCam**> MSR_SenseCam (Jojic et al., 2010) is a dataset consisting of images taken by SenseCam wearable camera. It contains 45 classes and there are approximately 150 images per class. Each image has size 480x640. We resize each image into 224x224 and extract the feature from VGG-19 network (Simonyan & Zisserman, 2014). In this dataset, we consider only *office* category which has 362 images.

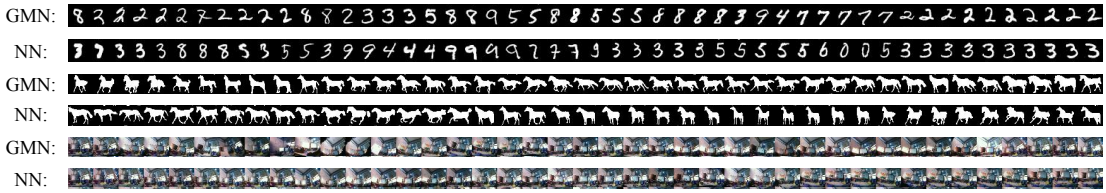

Figure 1: For MNIST, Horse, and MSR_SenseCam datasets: the implicit order observed from GMN and the oder implied from Nearest Neighbor sorting.

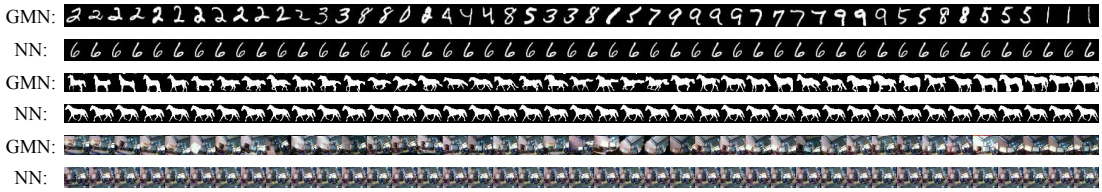

Figure 2: For MNIST, Horse, and MSR_SenseCam datasets: data generation from the learned transition operator in GMN and Nearest Neighbor search.

4.1.1 Implicit Orders in Datasets

We apply Alg. 2 to train our Generative Markov Networks. When the training converges, we plot the images following permutation $\hat{\tau}$ in Alg. 1. Note that $\hat{\tau}$ can be seen as the implicit order found by GMNs. For comparison, we also plot the images following nearest neighbor sorting using Euclidean distances. The parameters $\{b_{overlap}, b, t\}$ in Alg. 2 are $\{50, 500, 600\}$, $\{328, 328, 1\}$, and $\{362, 362, 1\}$ for MNIST, Horse, and MSR_SenseCam. Network architectures for parameterizing $\mathcal{T}(\cdot|\cdot;\theta)$ are specified in Supplementary.

The results are shown in Fig. 1. We first observe that data following the order suggested by our proposed GMN have visually high autocorrelation. This result implies that our proposed GMN is able to discover nice implicit orders for the dataset. Comparing to the strong ordering baseline Nearest Neighbor sorting, one could hardly tell which one is better. Nevertheless, GMN is a *distance-metric-free* model which requires no predefined distance metric. Moreover, the implicit order suggested by GMN considers a generative model viewpoint: the order is the optimal permutation under the Markov chain data generation assumption (see Sec. 3.2).

4.1.2 Transition Operator as a Generative Model

Next, we examine the data generation using the learned transition operator. Conditioned on a given sample x, instead of sampling $x' \sim \mathcal{T}(x'|x;\theta)$ directly, we sample $x' = \arg\max_{x^\dagger \in \{\{x_i\}_{i=1}^b \setminus x\}} \mathcal{T}(x^\dagger|x;\theta)$. We make this modification based on the reason that our model aims at discovering datasets' orders, while other *iterative sampling* models (Bengio et al., 2013; 2014; Sohl-Dickstein et al., 2015; Bordes et al., 2017; Song et al., 2017) intended to denoise generated samples. Similar to Sec. 4.1.1, we exploit nearest neighbor search using Euclidean distance for comparison. More precisely, $x'_{NN} = \arg\max_{x^\dagger \in \{\{x_i\}_{i=1}^b \setminus x\}} d(x^\dagger, x)$ with $d(\cdot,\cdot)$ denoting Euclidean distance.

Fig. 2 illustrate the sampling of GMN and Nearest Neighbor search. We can see that Nearest Neighbor search is not able to perform efficient sampling since it would stick between two similar images. On the other hand, our proposed GMN is able to perform consecutive sampling. This huge difference manifests the distinction between the discriminative (sampling by a fixed distance metric) and the generative model (sampling through the transition operator in a Markov chain) model.

4.2 One-Shot Recognition

Now, we perform one-shot recognition task on the miniImageNet dataset (Vinyals et al., 2016; Ravi & Larochelle, 2017), which is a benchmark dataset designed for the evaluation of few-shot learning (Vinyals et al., 2016; Ravi & Larochelle, 2017). Being a subset of ImageNet (Russakovsky et al.,

Table 1: 5-way 1-shot recognition task for miniImageNet. The results are reported averagely in $10,000$ episodes with 95% confidence intervals.

Model	Basic/ Advanced Model	Discriminative/ Generative	Parametric/ Nonparametric	Accuracy
Meta-Learner LSTM (Ravi & Larochelle, 2017)	Basic	Discriminative	Parametric	43.44±0.77
Model-Agnostic Meta-Learning (Finn et al., 2017)	Basic	Discriminative	Parametric	48.70±1.84
Meta Networks (Munkhdalai & Yu, 2017)	Advanced	Discriminative	Parametric	49.21±0.96
Meta-SGD (Li et al., 2017)	Basic	Discriminative	Parametric	50.47±1.87
Temporal Convolutions Meta-Learning (Mishra et al., 2017)	Advanced	Discriminative	Parametric	55.71±0.99
Nearest Neighbor with Cosine Distance	Basic	Discriminative	Nonparametric	41.08±0.70
Matching Networks FCE (Vinyals et al., 2016)	Basic	Discriminative	Nonparametric	43.56±0.84
Siamese (Koch et al., 2015)	Basic	Discriminative	Nonparametric	48.42±0.79
mAP-Direct Loss Minimization (Triantafillou et al., 2017)	Basic	Discriminative	Nonparametric	41.64±0.78
mAP-Structural Support Vector Machine (Triantafillou et al., 2017)	Basic	Discriminative	Nonparametric	47.89±0.78
Prototypical Networks (Snell et al., 2017)	Basic	Discriminative	Nonparametric	49.42±0.78
Attentive Recurrent Comparators (Shyam et al., 2017)	Not Specified	Discriminative	Nonparametric	49.1
Skip-Residual Pairwise Networks (Mehrotra & Dukkipati, 2017)	Advanced	Discriminative	Nonparametric	55.2
Generative Markov Networks without fine-tuning (ours)	Basic	Generative	Nonparametric	45.36±0.94
Generative Markov Networks with fine-tuning (ours)	Basic	Generative	Nonparametric	48.87±1.10

2015), it contains 100 classes and each class has 600 images. Each image is down sampled to size 84x84. As suggested in (Ravi & Larochelle, 2017), the dataset is divided into three parts: 64 classes for training, 16 classes for validation, and 20 classes for testing. Identical to (Ravi & Larochelle, 2017), we consider the $5-$way $1-$shot problem. That is, from testing classes, we sample 5 classes with each class containing 1 labeled example. The labeled examples refer to support instances. Then, we randomly sample 500 unlabeled query examples in these 5 classes for evaluation. We repeat this procedure for $10,000$ times and report the average with 95% confidence intervals in Tbl. 1.

4.2.1 Training Details

Instead of viewing one-shot recognition as a discriminative task, we hold it as a generative one. To achieve this goal, we train our Generative Markov Networks on training classes and then apply it to testing classes. More precisely, for each training episode, we sample 1 class from the training classes and let $\{s_i\}_{i=1}^n$ to be all the data from this class. Then, we apply Alg. 2 with $\{b_{overlap}, b, t\}$ = $\{20, 100, 10\}$. We consider $3,000$ training episodes.

On the other hand, for each testing episode, we apply GMNs to generate a chain from each support instance:

$$\tilde{x}_1^c \sim \mathcal{T}(\cdot|x_0^c;\theta)\,,\ \tilde{x}_2^c \sim \mathcal{T}(\cdot|\tilde{x}_1^c;\theta)\,,\ \cdots\,,\tilde{x}_t^c \sim \mathcal{T}(\cdot|\tilde{x}_{k-1}^c;\theta),$$

where x_0^c is the support instance belonging to class c and $\tilde{x}^c$ is the generated samples from the Markov chain.

Next, we fit each query example into each chain by computing the average approximating log-likelihood. Namely, the probability for generating the query sample x_q in the chain of class c is

$$\mathcal{P}(x_q|c) := \frac{1}{k+1}\ \sum_{i=0}^{k}\ \log\ \mathcal{T}(x_q|x_i;\theta). \tag{6}$$

In a generative viewpoint, the predicted class $\hat{c}$ for x_q is determined by

$$\hat{c} = \arg\max_{c}\ \mathcal{P}(x_q|c). \tag{7}$$

For fair comparisons, we use the same architecture specified in (Ravi & Larochelle, 2017) to extract 1600-dimensional features. We pretrain the architecture using standard softmax regression on image-label pairs in training and validation classes. The architecture consists of 4 blocks. Each block comprises a CNN layer with 64 3x3 convolutional filters, Batch Normalization (Ioffe & Szegedy, 2015) layer, ReLU activation, and 2x2 Max-Pooling layer. Then, we train our Generative Markov Networks based on these $1,600$ dimensional features. Network architecture for parameterizing $\mathcal{T}(\cdot|\cdot;\theta)$ is specified in Supplementary.

4.2.2 Results

For a comprehensive analysis, we also provide the variant of our GMN with fine-tuning. In other words, we fine tune GMN by applying Alg. 2 with $\{b_{overlap}, b, t\}$ = $\{20, 100, 10\}$ on support and

query instances. Note that in eq. (6), k is chosen to be 1 and 5 for the non-fine-tuned and fine-tuned version, respectively. We compare our proposed method and the related approaches in Tbl. 1, in which *Basic* model refers the architecture in (Ravi & Larochelle, 2017) and *Advanced* models refer to more complicated designs. Generally, it is not fair to compare methods using different models; therefore, we only discuss the methods using *Basic* model in the following.

First, we observe that the performance for GMN is comparable to other works. For example, the best result is reported by Meta-SGD (Munkhdalai & Yu, 2017) with 50.47 ± 1.87. Although GMN suffers from slight performance drop, it requires much less computational budget. The reason is that the meta-learning (parametric) approaches (Ravi & Larochelle, 2017; Finn et al., 2017; Munkhdalai & Yu, 2017; Li et al., 2017; Mishra et al., 2017) rely on huge networks to manage complicated intersections between meta and base leaner, while parameters for GMN exist only in θ which is a relatively tiny network. On the other hand, the best performance reported in the distance-metric learning (nonparametric) approach is Prototypical Networks (Snell et al., 2017) with 49.42 ± 0.78. Sacrificing from little performance deterioration, our proposed GMN enjoys more flexibility without the need of defining any distance metrics as in (Vinyals et al., 2016; Koch et al., 2015; Triantafillou et al., 2017; Snell et al., 2017; Shyam et al., 2017; Mehrotra & Dukkipati, 2017). More importantly, except for our proposed GMN, all the works belong to discriminative models, which means they are optimized based on carefully chosen objectives for one-shot learning purpose.

Next, our proposed GMN enjoys a significant improvement ($45.36 \pm 0.94 \rightarrow 48.87 \pm 1.10$) from fine-tuning over support and query instances. This result verifies that our proposed GMN is able to mimic the Markov chain data generation process since the query instances can be better fitted in the chains generated from the support instances.

5 Conclusion

In this paper, we argue that data i.i.d. assumption is not always the case in most of the datasets. Often, data instances are exhibiting some implicit orders which may benefit our understanding and analysis of the dataset. To observe the implicit orders, we propose a novel Generative Markov Network which considers a Markov chain data generation scheme. Specifically, we simultaneously learn the transition operator as a generative model in the Markov chain as well as find the optimal orders of the data under all possible permutations. In lots of experiments, we show that our model is able to observe implicit orders from unordered datasets, and also perform well on the one-shot recognition task.

REFERENCES

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pp. 3981–3989, 2016.

Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. In *Advances in Neural Information Processing Systems*, pp. 899–907, 2013.

Yoshua Bengio, Eric Laufer, Guillaume Alain, and Jason Yosinski. Deep generative stochastic networks trainable by backprop. In *International Conference on Machine Learning*, pp. 226–234, 2014.

Florian Bordes, Sina Honari, and Pascal Vincent. Learning to generate samples from noise through infusion training. *arXiv preprint arXiv:1703.06975*, 2017.

Eran Borenstein and Shimon Ullman. Class-specific, top-down segmentation. In *European conference on computer vision*, pp. 109–122. Springer, 2002.

Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 2786–2793. IEEE, 2017.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 513–520, 2011.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Nebojsa Jojic, Alessandro Perina, and Vittorio Murino. Structural epitome: a way to summarize ones visual experience. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta (eds.), *Advances in Neural Information Processing Systems 23*, pp. 1027–1035. Curran Associates, Inc., 2010. URL `http://papers.nips.cc/paper/4092-structural-epitome-a-way-to-summarize-ones-visual-experience.pdf`.

Łukasz Kaiser, Ofir Nachum, Aurko Roy, and Samy Bengio. Learning to remember rare events. *arXiv preprint arXiv:1703.03129*, 2017.

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

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

Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pp. 396–404, 1990.

Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

Akshay Mehrotra and Ambedkar Dukkipati. Generative adversarial residual pairwise networks for one shot learning. *arXiv preprint arXiv:1703.08033*, 2017.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. Meta-learning with temporal convolutions. *arXiv preprint arXiv:1707.03141*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Tsendsuren Munkhdalai and Hong Yu. Meta networks. *arXiv preprint arXiv:1703.00837*, 2017.

Masatoshi Nei and Sudhir Kumar. *Molecular evolution and phylogenetics*. Oxford university press, 2000.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. *ICLR*, 2017.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Pranav Shyam, Shubham Gupta, and Ambedkar Dukkipati. Attentive recurrent comparators. *arXiv preprint arXiv:1703.00767*, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Jake Snell, Kevin Swersky, and Richard S Zemel. Prototypical networks for few-shot learning. *arXiv preprint arXiv:1703.05175*, 2017.

Jascha Sohl-Dickstein, Eric A Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *arXiv preprint arXiv:1503.03585*, 2015.

Jiaming Song, Shengjia Zhao, and Stefano Ermon. A-nice-mc: Adversarial training for mcmc. *arXiv preprint arXiv:1706.07561*, 2017.

Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using LSTMs. In *ICML*, 2015.

Eleni Triantafillou, Richard Zemel, and Raquel Urtasun. Few-shot learning through an information retrieval lens. *arXiv preprint arXiv:1707.02610*, 2017.

Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pp. 3630–3638, 2016.

Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305, 2008.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pp. 2048–2057, 2015.

SUPPLEMENTARY FOR DISCOVERING ORDER IN UNORDERED DATASETS: GENERATIVE MARKOV NETWORKS

Anonymous authors
Paper under double-blind review

1 EXAMPLE IMAGES FOR HORSE DATASET

Figure 1: RGB images of Horse Dataset.

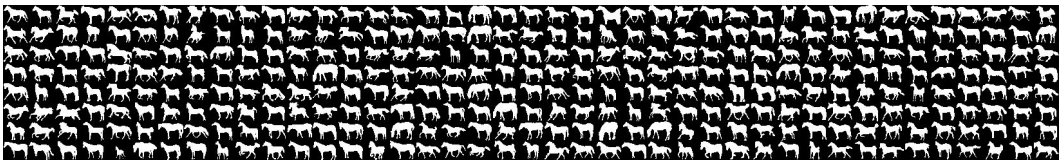

Figure 2: Example images of Horse Dataset.

Horse dataset Borenstein & Ullman (2002) consists of images collected from Internet. Fig. 1 illustrates these RGB images. Object-background segmentations are applied on these images and the horses are centered in 30x40 images. The processed images are shown in Fig. 2.

2 FULL ORDERING RESULTS FOR MNIST, HORSE, AND MSR_SENSECAM

Fig. 3, 4, and 5 show the results of the implicit order observed from GMN the order implied from Nearest Neighbor sorting. On the other hand, Fig. 6, 7, and 8 illustrate the sampling of GMN and Nearest Neighbor search.

(a)

(b)

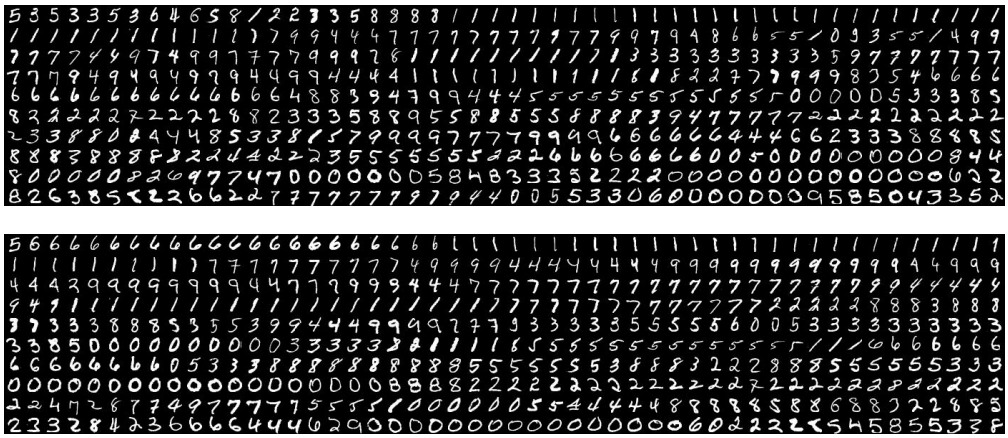

Figure 3: For MNIST dataset: (a) implicit order observed from GMNs (b) order implied from nearest neighbor sorting using Euclidean distance.

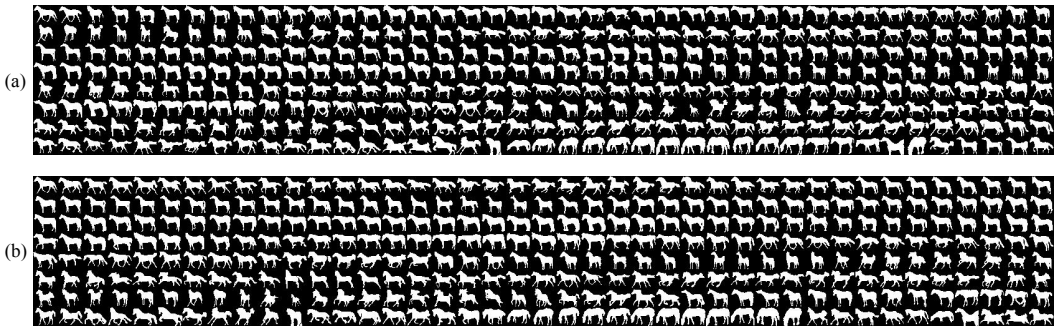

Figure 4: For Horse dataset: (a) implicit order observed from GMNs (b) order implied from nearest neighbor sorting using Euclidean distance.

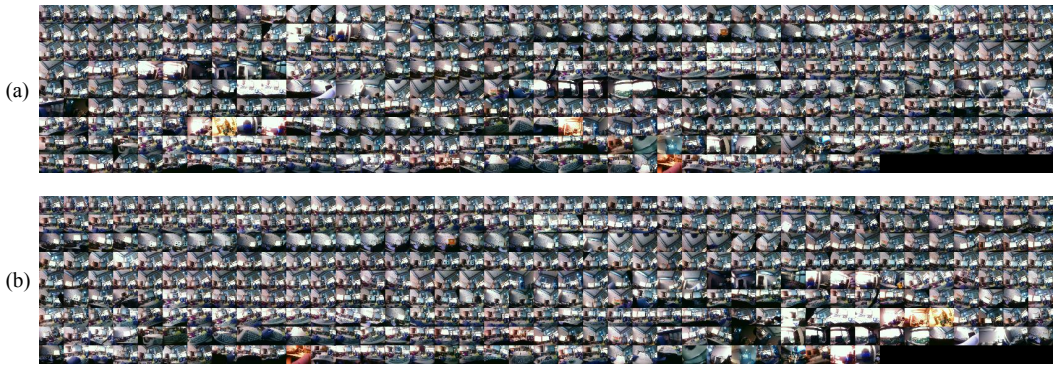

Figure 5: For *office* category in SenseCam dataset: (a) implicit order observed from GMNs (b) order implied from nearest neighbor sorting using Euclidean distance.

3 MOVING MNIST

<**Moving MNIST**> Moving MNIST (Srivastava et al., 2015) contains $10,000$ sequences each of length 20 showing 2 digits moving in a 64x64 frame. We rescale the pixel values to $[0, 1]$. For each training episode, we apply Alg. 1 to train GMN on one randomly chosen sequence with parameters $\{b_{overlap}, b, t\}$ set as $\{0, 20, 10\}$. We consider $6,000$ training episodes. For evaluation, we randomly sample a disjoint sequence from training sequences and observe the optimal permutation (implicit order) from Alg. 2.

Fig. 9 illustrates the results for the implicit order observed from Generative Markov Networks, the order inferred from Nearest Neighbor sorting using Euclidean distance, and the suggested explicit order. We find that both the orders observed from GMN and NN sorting manifest smooth motions for two digits in the frame. It is worth noting that our proposed GMN enjoys the freedom of not defining any distance metric.

The sampling results for Moving MNIST dataset are shown in Fig. 10. We consider two approaches: the proposed Generative Markov Networks and Nearest Neighbor search. We find that, by learning the transition operator in a Markov chain as a generative model, GMN performs much better sampling results than Nearest Neighbor search which is a discriminative model.

4 NETWORK ARCHITECTURES FOR TRANSITION OPERATOR

We elaborate the design of the transition operator in Fig. 11. In our design, U can be seen as a gating mechanism between input X_t and the learned update $\tilde{X}$. More precisely, the output can be written as

$$X_{t+1} = U \odot \tilde{X} + (\mathbf{1} - U) \odot X_t, \tag{1}$$

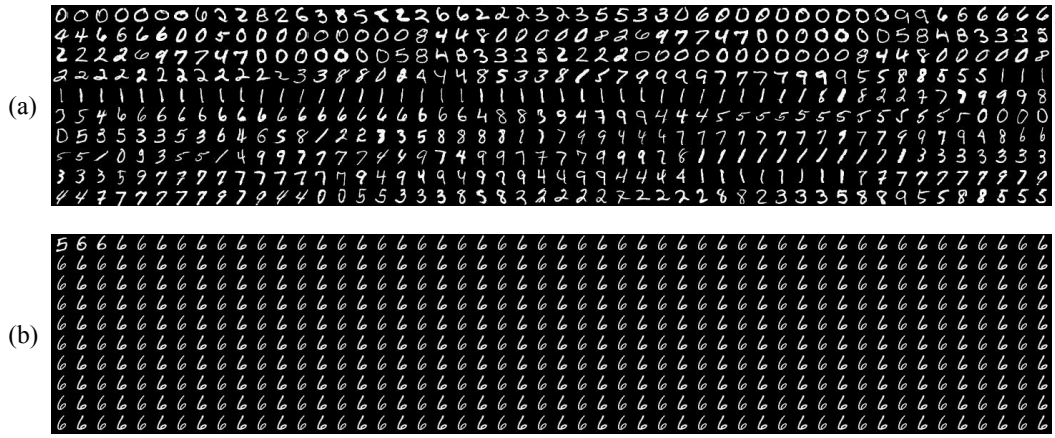

Figure 6: For MNIST dataset, data generation from (a) learned transition operator in GMNs (b) nearest neighbor search using Euclidean distance.

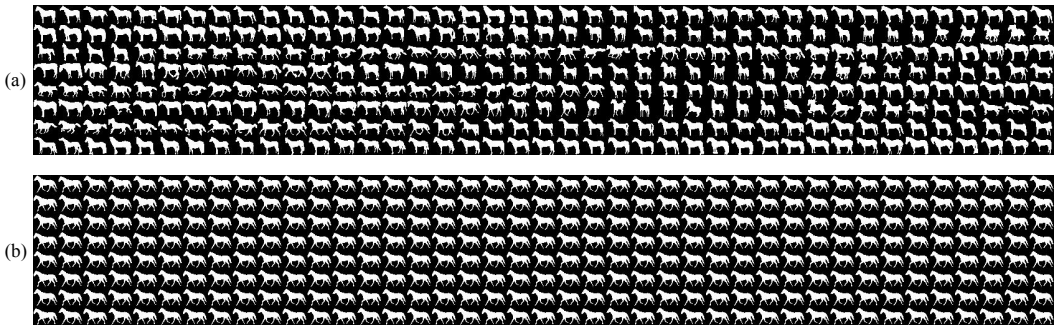

Figure 7: For Horse dataset, data generation from (a) learned transition operator in GMNs (b) nearest neighbor search using Euclidean distance.

where $\odot$ denotes element-wise product. We specify each function f in Tbl. 1, 2, 3, 4, and 5. Note that we omit the bias term for simplicity. We use ADAM (Kingma & Ba, 2014) with learning rate 0.001 and 0.2 dropout rate to train our $\mathcal{T}(\cdot|\cdot;\theta)$.

Table 1: Details of functions for MNIST experiments.

function	details
f1	784x512 FC layer with ReLU
f21	512x128 FC layer
f22	512x128 FC layer
f3	912x512 FC layer with ReLU
f41	512x784 FC layer with sigmoid
f42	512x784 FC layer with sigmoid

REFERENCES

Eran Borenstein and Shimon Ullman. Class-specific, top-down segmentation. In *European conference on computer vision*, pp. 109–122. Springer, 2002.

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

Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using LSTMs. In *ICML*, 2015.

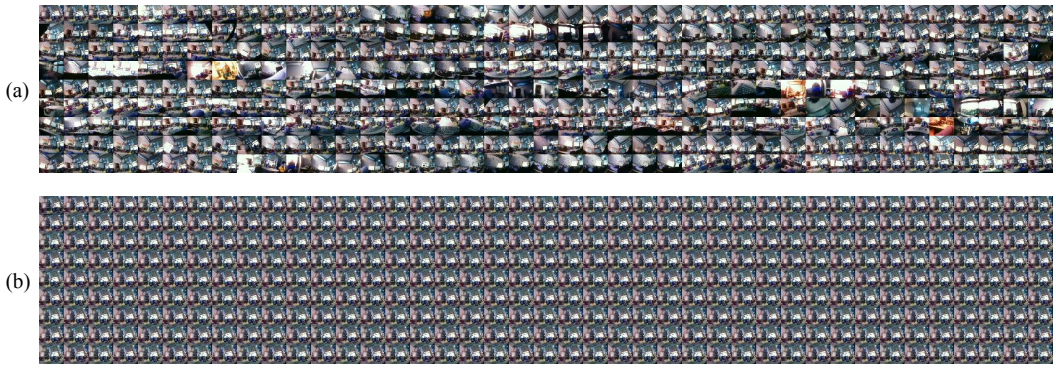

Figure 8: For *office* category in SenseCam dataset, data generation from (a) learned transition operator in GMNs (b) nearest neighbor search using Euclidean distance.

Figure 9: For Moving MNIST dataset: (a) implicit order observed from GMNs (b) order implied from nearest neighbor sorting using Euclidean distance (c) suggested explicit order.

Figure 10: For Moving MNIST dataset, data generation from (a) learned transition operator in GMNs (b) nearest neighbor search using Euclidean distance.

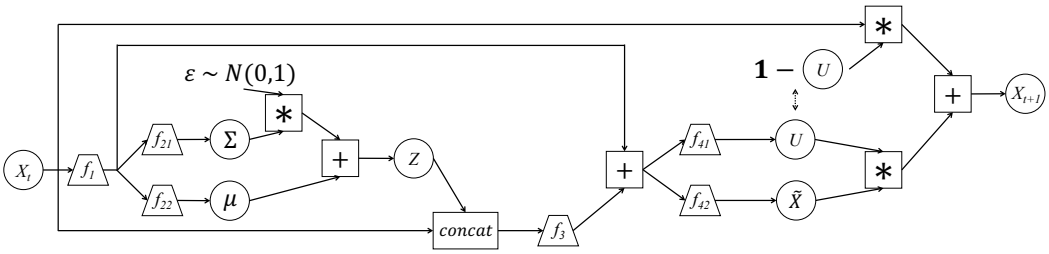

Figure 11: Network design for $\mathcal{T}(\cdot|\cdot;\theta)$.

Table 2: Details of functions for Horse experiments.

function	details
f1	1200x512 FC layer with ReLU
f21	512x128 FC layer
f22	512x128 FC layer
f3	912x512 FC layer with ReLU
f41	512x1200 FC layer with sigmoid
f42	512x1200 FC layer with sigmoid

Table 3: Details of functions for MSR_SenseCam experiments.

function	details
f1	4096x1024 FC layer with ReLU
f21	1024x256 FC layer
f22	1024x256 FC layer
f3	4352x1024 FC layer with ReLU
f41	1024x4096 FC layer with sigmoid
f42	1024x4096 FC layer

Table 4: Details of functions for Moving MNIST experiments.

function	details
f1	4096x1024 FC layer with ReLU
f21	1024x256 FC layer
f22	1024x256 FC layer
f3	4352x1024 FC layer with ReLU
f41	1024x4096 FC layer with sigmoid
f42	1024x4096 FC layer with sigmoid

Table 5: Details of functions for miniImageNet experiments.

function	details
f1	1600x1024 FC layer with ReLU // 1024x512 FC layer with ReLU // 512x256 FC layer with ReLU
f21	256x64 FC layer
f22	256x64 FC layer
f3	1664x256 FC layer with ReLU
f41	256x512 FC layer with ReLU // 512x1024 FC layer with ReLU // 1024x1600 FC layer with sigmoid
f42	256x512 FC layer with ReLU // 512x1024 FC layer with ReLU // 1024x1600 FC layer