ANNUAL REVIEWS

Annual Review of Statistics and Its Application

Representation Learning: A Statistical Perspective

Jianwen Xie,[1] Ruiqi Gao,[2] Erik Nijkamp,[2] Song-Chun Zhu,[2] and Ying Nian Wu[2]

[1]Hikvision Research Institute, Santa Clara, California 95054, USA

[2]Department of Statistics, University of California, Los Angeles, California 90095, USA; email: ywu@stat.ucla.edu

Annu. Rev. Stat. Appl. 2020. 7:11.1–11.33

The *Annual Review of Statistics and Its Application* is online at statistics.annualreviews.org

https://doi.org/10.1146/annurev-statistics-031219-041131

Keywords

unsupervised learning, generative representations, relative representations, predictive representations, vector representations, matrix representations

Abstract

Learning representations of data is an important problem in statistics and machine learning. While the origin of learning representations can be traced back to factor analysis and multidimensional scaling in statistics, it has become a central theme in deep learning with important applications in computer vision and computational neuroscience. In this article, we review recent advances in learning representations from a statistical perspective. In particular, we review the following two themes: (*a*) unsupervised learning of vector representations and (*b*) learning of both vector and matrix representations.

1. INTRODUCTION

Statistics is about understanding data. If the input data are complex, it is desirable to find representations for the data so that they become easier to understand and process. In this article, we review learning representations of data with various models, including models with linear structures and models that are based on deep neural networks.

1.1. Prototypes of Learning Representations in Statistics

Although representation learning is a central theme in deep learning, its essence can be traced back to familiar examples in statistics.

1.1.1. Factor analysis—generative representation. One prototypical example of learning representation in statistics is factor analysis (Rubin & Thayer 1982). Here, multivariate observations (e.g., test scores on different subjects) are explained by latent factors (e.g., verbal and analytical intelligence). Let h be a d-dimensional hidden vector that consists of d latent factors. Let x be the observed D-dimensional vector. Usually, $d < D$. Then, the model is of the form $x = Wh + \varepsilon$, where W is the $D \times d$ loading matrix that transforms h to x. It is assumed that $h \sim \mathrm{N}(0, I_d)$, where I_d denotes the d-dimensional identity matrix, and $\varepsilon \sim \mathrm{N}(0, \sigma^2 I_D)$, which is independent of h. This model can be learned by maximum likelihood via the expectation-maximization (EM) algorithm (Dempster et al. 1977), where the E step is based on the posterior distribution of h given x.

h is said to be a vector representation, also called a code, of x. The mapping from h to x is called a decoder, while the mapping from x to h is called an encoder, and both can be formally written as conditional distributions. While the decoder $p(x|h)$ and the prior $p(h)$ define a top-down generative model, the encoder $p(h|x)$ defines an inference model.

Factor analysis is related to principal component analysis, where W is obtained by the first d eigenvectors of the covariance matrix $\mathrm{Cov}(x)$. The factor analysis model can be generalized to independent component analysis (Hyvärinen et al. 2004), sparse coding (Olshausen & Field 1997), nonnegative matrix factorization (Lee & Seung 2001), recommender systems (Koren et al. 2009), restricted Boltzmann machines (Hinton 2012), and so on, by modifying the prior distribution or prior assumption on h. If we generalize the linear mapping from h to x to a nonlinear mapping parameterized by a deep network (LeCun et al. 1998, Krizhevsky et al. 2012), then the resulting model is commonly called generator network (Goodfellow et al. 2014, Kingma & Welling 2014). Factor analysis is an example of generative representation, where the hidden vector h generates the observed vector x.

1.1.2. Multidimensional scaling—relative representation. The other prototypical example of learning representation in statistics is multidimensional scaling (Kruskal 1964). Let $(x_i, i = 1, \ldots, n)$ be a set of D-dimensional observations. We want to represent them by a corresponding set of d-dimensional hidden vectors $(h_i, i = 1, \ldots, n)$, so that (h_i) preserve the relations such as distances between (x_i). For instance, we may find (h_i) by minimizing $\sum_{i \neq j}(\|h_i - h_j\| - \|x_i - x_j\|)^2$, which enforces global isometry.

Again, h is said to be a vector representation of x, which is also called an embedding of x. Unlike in factor analysis, there is no explicit mapping (encoding or decoding) between h and x.

Various modifications of multidimensional scaling focus on preserving local adjacency or neighborhood relations between x_i, such as spectral embedding (Bengio et al. 2004), t-distributed stochastic neighbor embedding (t-SNE) (van der Maaten & Hinton 2008), and local linear

embedding (Roweis & Saul 2000). Multidimensional scaling is an example of relative representation, where the hidden vectors $\{h_i\}$ are to preserve the relations between the observed vectors $\{x_i\}$.

1.1.3. Sliced inverse regression—predictive representation. The third prototypical example of learning representation in statistics is sliced inverse regression (Li 1991). It learns a nonlinear regression model from the training examples $\{(x_i, y_i)\}$, where x_i is a D-dimensional continuous predictor vector, and y_i is a one-dimensional continuous outcome. The sliced inverse regression model assumes a d-dimensional hidden vector $h_i = Wx_i$, where W is $d \times D$, so that $y_i = f(h_i, \varepsilon_i)$ where ε_i are independent and identically distributed (i.i.d.) noise.

Assume $(x_i, y_i) \sim p(x, y)$, $\mathrm{E}(x) = 0$, and $\mathrm{Cov}(x) = I_D$ under $p(x, y)$ (which can be achieved by standardizing x). Then, under mild conditions, W can be obtained by the top d eigenvectors of $\mathrm{Cov}[\mathrm{E}(x|y)]$, where $\mathrm{E}(x|y)$ can be obtained by dividing the range of y into slices, and $\mathrm{E}(x|y)$ is the inverse regression. W can be obtained without knowledge of the nonlinear link function f. We may refer to $h = Wx$ as encoding and $y = f(h, \varepsilon)$ as decoding.

Sliced inverse regression is an example of predictive representation, where the hidden vector h_i contains all the information of x_i for predicting y_i, i.e., h_i is a sufficient summary of x_i as far as predicting y_i is concerned.

1.2. Unsupervised, Supervised, and Reinforcement Learning

Sliced inverse regression is a supervised learning problem where, for each input x_i, an output y_i is given as supervision. Factor analysis and multidimensional scaling are unsupervised learning problems where only x_i are observed without y_i. Learning representations is of fundamental importance for both supervised and unsupervised learning. In this article, we shall focus on unsupervised learning.

Another learning problem that lies in between supervised and unsupervised learning is reinforcement learning (Sutton & Barto 1998), where the input x is the state and the output y is the action. In training, the optimal y is not directly given, but a reward for an action is provided. For this problem, learning a good representation of state x is important for learning value and policy functions that are defined on the state.

1.3. Plan for the Remainder of the Article

Section 2 presents vector representations based on linear models. We first describe a generalization of the factor analysis model in which the hidden vector is assumed to be sparse (or have independent components) in the generative representation scheme. We then explain continuous vector representations of discrete data in predictive and relative representation schemes. Section 3 presents the learning of both vector and matrix representations in a relative representation scheme. Section 4 is about the learning of nonlinear vector representation based on the generator model, which generalizes linear mapping in the factor analysis model to nonlinear mapping parameterized by deep neural networks. Section 5 reviews the joint learning of generator model and various complementary models. Section 6 reviews the learning of the conditional generator model.

2. LEARNING VECTOR REPRESENTATIONS

In this section, we review learning vector representations of data using models that generalize the factor analysis model.

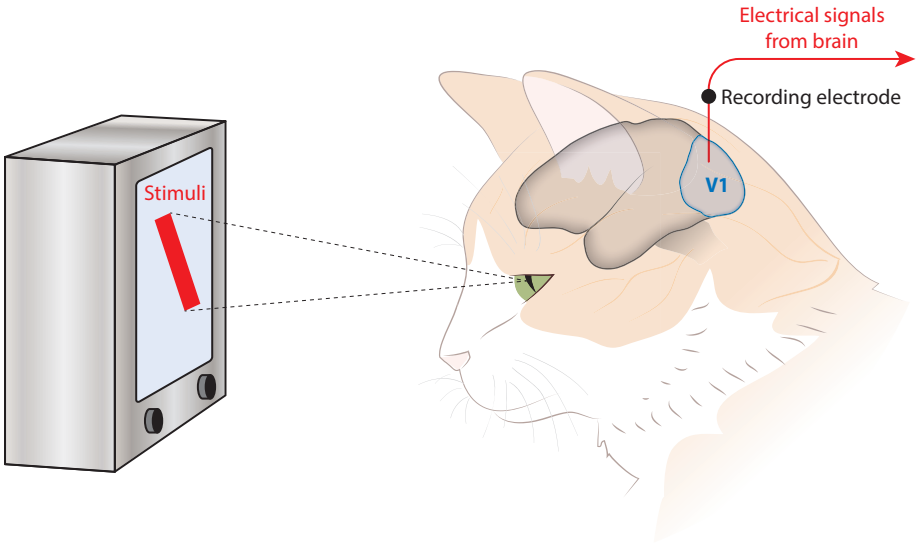

Figure 1

Visual area of brain. Primary visual cortex, or V1, is the first step in representing retina image data. Cells in V1 respond to bars of different locations, orientations, and sizes.

2.1. Sparse Vector Representation

David Hubel and Torsten Wiesel earned the Nobel Prize for Physiology or Medicine in 1981 for their discovery of simple and complex cells in the primary visual cortex, or V1 (Hubel & Wiesel 1959). They discovered that cells in V1 of the cat brain responded to bars of different locations, orientations, and sizes, and each cell responded to the bar at a particular location, orientation, and scale (**Figure 1** gives an illustration). Some V1 cells are called simple cells, which behave like linear wavelets. A mathematical model of a simple cell is the Gabor wavelet, which is a sine or cosine plane wave multiplied by an elongate Gaussian function.

Olshausen & Field (1997) proposed a sparse coding model for the V1 simple cells by generalizing the factor analysis model. Recall that in factor analysis,

$$x = Wh + \varepsilon = \sum_{k=1}^{d} W_k h_k + \varepsilon, \tag{1.}$$

where W_k is the kth column of W and is of the same dimensionality as x, and h_k is the kth element of h. The above model expresses x as a linear superposition of the basis vectors W_k, with coefficients h_k.

Unlike in factor analysis, in the sparse coding model, the dimensionality d of h is assumed to be larger than the dimensionality D of x (i.e., $d > D$). However, h is assumed to be a sparse vector, i.e., for each x, only a small number d_0 ($d_0 < D < d$) of h_k are nonzero or significantly different from zero. For different x, the nonzero elements of h can be different. Thus, unlike principal component analysis, sparse coding leads to adaptive dimension reduction. $W = (W_k, k = 1, \ldots, d)$ is sometimes called a dictionary, from which a small number of words are chosen to describe x. h is called a sparse code of x.

The training data are in the form of image patches sampled from natural images, $\{x_i, i = 1, \ldots, n\}$, where each x_i is a training example image patch. Each x_i is represented by an

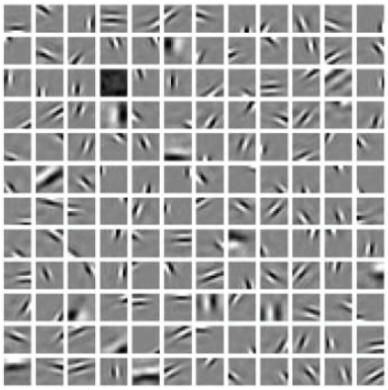

Figure 2

Olshausen-Field sparse coding model. The plot displays the 144 learned basis vectors, each displayed as an image patch (ordering of the patches carries no meaning). These basis vectors represent local image structures such as edges and bars. The training data were obtained by extracting 12 × 12 image patches at random from ten 512 × 512 images of natural scenes (trees, rocks, mountains, etc.).

$h_i = (h_{ik}, k = 1, \ldots, d)$, but all the examples share the same W, where each W_k has the same dimensionality as x_i, so that $x_i = Wh_i + \varepsilon_i = \sum_{k=1}^{d} W_k h_{ik} + \varepsilon_i$. The learning of W can be accomplished by minimizing the following objective function:

$$L(W, \{h_i\}) = \frac{1}{n} \sum_{i=1}^{n} \left[\|x_i - Wh_i\|^2 + \sum_{k=1}^{d} \rho(h_{ik}) \right], \qquad 2.$$

where $\rho(h_{ik})$ is a sparsity-inducing term, e.g., $\rho(r_{ik}) = |r_{ik}|$, which leads to the lasso estimator (Tibshirani 1996) of h_i. The minimization can be accomplished by alternating gradient descent over W and $\{h_i\}$. **Figure 2** displays the learned (W_k), where each W_k is displayed as an image patch of the same size as x_i. The basis vectors (W_k) represent local image structures such as bars and edges.

Given W, the inference of h_i from each x_i can be accomplished by lasso, where the (W_k) serve as variables or regressors. Compared with lasso, the sparse coding has an added layer of depth in that W (i.e., the regressors) is to be learned from the training data. The sparse coding model has had a profound impact on computational neuroscience and applied harmonic analysis, in addition to machine learning.

A related model is independent component analysis (Bell & Sejnowski 1997, Hyvärinen et al. 2004), which assumes that $D = d$, $\varepsilon = 0$, and h_k are independent. It assumes an invertible transformation $x = Wh$, and $h = W^{-1}x$, so that the distribution of x can be obtained in closed form from the prior distribution of h: $p(x) = p_0(W^{-1}x)|W|^{-1}$, where $p_0(h)$ is the prior distribution of h, and $|W|$ is the absolute value of the determinant of W.

Other related models include nonnegative matrix factorization (Lee & Seung 2001), which assumes $h_k \geq 0$, and restricted Boltzmann machines (Hinton 2012), which assume a binary h and a joint distribution $p(x, h) \propto \exp(-x^\top Wh)$ (where we omit bias terms for simplicity), which is an energy-based model on (x, h) with pairwise potentials defined on (x, h). For this model, both the decoder $p(x|h)$ and the encoder $p(h|x)$ are in closed form. But the prior distribution $p(h)$ is not in closed form.

2.2. Continuous Vector Representation of Discrete or Symbolic Input

The vector representation h of the original input x can be considered a dimension reduction of x, or a visualization of x if h is 2D ($d = 2$). The input x is usually continuous.

The input x can also be discrete, like a word in the dictionary. In that case, x can be expressed as a one-hot vector. Let D be the number of words in the dictionary. If x is the jth word in the dictionary, then x is a D-dimensional vector so that the jth element of x is 1 and all the other elements are zeroes. We represent x with a d-dimensional continuous hidden vector h. We can write $h = Wx$, where W is a $d \times D$ dimensional encoding matrix, so that the jth word is represented by the jth column of the encoding matrix W. h is called a semantic embedding, or word2vec (Mikolov et al. 2013, Pennington et al. 2014). In Mikolov et al. (2013), h is learned to predict nearby words, i.e., it is a predictive representation. Specifically, for a particular word y, again expressed as a one-hot vector, in the context of word x in a random sentence, we predict this word y based on the decoded vector $\tilde{W}^\top h$, where $\tilde{W}$ is the $d \times D$ decoding matrix of the same dimensionality as the encoding matrix W, so that $p(y) \propto \exp(y^\top \tilde{W}^\top h)$. More specifically, let Q_{ij} be the probability that word j is within the context of word i, then $Q_{ij} = \exp(\langle W_i, \tilde{W}_j \rangle) / \sum_j \exp(\langle W_i, \tilde{W}_j \rangle)$, the so-called soft-max classifier, where W_i is the ith column of W, i.e., the vector representation of word i in the encoding pass, and $\tilde{W}_j$ is the jth column of $\tilde{W}$, i.e., the vector representation of word j in the decoding pass. In Pennington et al. (2014), $h = Wx$ is learned as a relative representation so that for two words i and j, $\log Q_{ij} = \langle W_i, \tilde{W}_j \rangle + b_i + \tilde{b}_j$, where b_i and $\tilde{b}_j$ are bias terms.

The above form is similar to matrix factorization in recommender systems (Koren et al. 2009). Let X_{ij} be the rating of user i on item j, and then the model is $X_{ij} = \langle W_i, \tilde{W}_j \rangle + b_i + \tilde{b}_j$, where W_i is the vector representation of user i, $\tilde{W}_j$ is the vector representation of item j, and b_i and $\tilde{b}_j$ are the bias terms. The elements of the d-dimensional vector W_i can be interpreted as the desires of user i in various aspects, and the elements of the d-dimensional $\tilde{W}_j$ can be interpreted as the desirabilities of item j in the corresponding aspects. In terms of matrix, let X be the $n \times D$ matrix of ratings where n is the number of users and D is the number of items. Then $X = W^\top \tilde{W}$, where W is the $d \times n$ matrix whose ith column is W_i, and $\tilde{W}$ is the $d \times D$ matrix whose jth column is $\tilde{W}_j$.

For a discrete x such as a word, the vector representation h is continuous, dense, and distributed, where each component of h captures a partial semantic meaning of x. Such dense vector representations have revolutionized natural language processing in recent years, and they are at the foundation of recent natural language models (Vaswani et al. 2017, Radford et al. 2018, Devlin et al. 2019).

Vector representations have also been applied to encode the nodes in graphs (Hamilton et al. 2017), which can be conveniently used for subsequent analysis (Kipf & Welling 2016).

In Gómez-Bombarelli et al. (2018), each molecular compound, which is a graph structure, is represented by a continuous vector, which can be used to learn to predict the chemical activity of the compound. One can also optimize the activity by maximizing over the continuous vector using a gradient-based method, and the optimized vector can then generate the corresponding compound. Such continuous representation is much more convenient to operate on than the original discrete input.

3. LEARNING BOTH VECTOR AND MATRIX REPRESENTATIONS

This section reviews recent work on learning models based on vector and matrix representations. The representations are of a relative nature, similar to multidimensional scaling. The matrices represent the relations between the vectors and can be part of a relative representation. An early example is that of Paccanaro & Hinton (2001).

In computational neuroscience, the vector representations can be interpreted as neuron activities, and the matrix representations can be stored in the synaptic connections. The vector representations are like nouns, while the matrix representations are like verbs that transform the nouns.

Matrix representations of groups underlie much of modern mathematics (Dornhoff 1972) and hold the key to modern physics (Zee 2016).

3.1. Learning Grid Cells

You may imagine moving in your living room at night in the dark. Based purely on the movements or self-motion, you know your current position by summing up the displacements. The grid cells in our brain accomplish this computation, albeit in a very sophisticated manner.

3.1.1. Hexagon patterns. **Figure 3*a*** depicts Dr. May-Britt Moser who, together with Dr. Edvard Moser, won the 2014 Nobel Prize for Physiology or Medicine for the discovery of grid cells (Hafting et al. 2005). Their thesis advisor, Dr. John O'Keefe, shared the prize for his discovery of place cells (O'Keefe 1979). Both place and grid cells are used for navigation. The discoveries of these cells were made by recording the activities of the neurons of a rat when it moves within a square region (**Figure 3*b***). Some neurons in the hippocampus area are place cells. Each place cell fires when the rat moves to a particular location, and different place cells fire at different locations; the whole collection of place cells covers the whole square region. The discovery of grid cells, found in the entorhinal cortex, was much more surprising and unexpected. Each grid cell fires at multiple locations, and these locations form a regular hexagonal grid (**Figure 3*c***). The grid cells have been identified across many mammalian species, including human (**Figure 3*d***).

3.1.2. A simple addition problem. There are two problems in navigation. One is the path integral. Imagine you walk in your living room at night. If you know the position of your starting point, then by summing over your displacements over time, you can calculate where you are at any time. The other problem is path planning. Suppose you want to go to a target position such as the light switch, which is a position that you know, so you plan a sequence of displacements that will lead you from the starting point to the target.

More specifically, consider an agent (e.g., a rat or a human) navigating within a domain $D = [0, 1] \times [0, 1]$. We can discretize D into an $N \times N$ lattice. Let $x = (x^{(1)}, x^{(2)}) \in D$ be the self-position of the agent. Let $\Delta x = (\Delta x^{(1)}, \Delta x^{(2)})$ be the displacement or self-motion of the agent at a certain time. The path integral problem is such that, given the starting point x_0 and the sequence of self-displacements $(\Delta x_t, t = 1, \ldots, T)$, we want to calculate the positions over time with $x_t = x_{t-1} + \Delta x_t$ for $t = 1, \ldots, T$. The path planning problem is, given the starting position x and the target position y, to plan a sequence of displacements $(\Delta x_t, t = 1, \ldots, T)$ such that $x_0 = x$ and $x_T = y$.

Both problems appear to be quite simple, especially path integral, which is merely an addition problem—but the brain uses a system of grid cells to solve this problem. What is the purpose of this system, and how does the system work? Why the hexagon patterns?

3.1.3. A representational scheme. Recently, Gao et al. (2018b) proposed an explanation of grid cells as a representational system. The basic idea is that the grid cells form a d-dimensional vector representation of the 2D position. Specifically, we represent any 2D position $x \in D$ by a d-dimensional vector $h(x)$. Suppose at a position x, the self-motion or displacement is Δx, so that

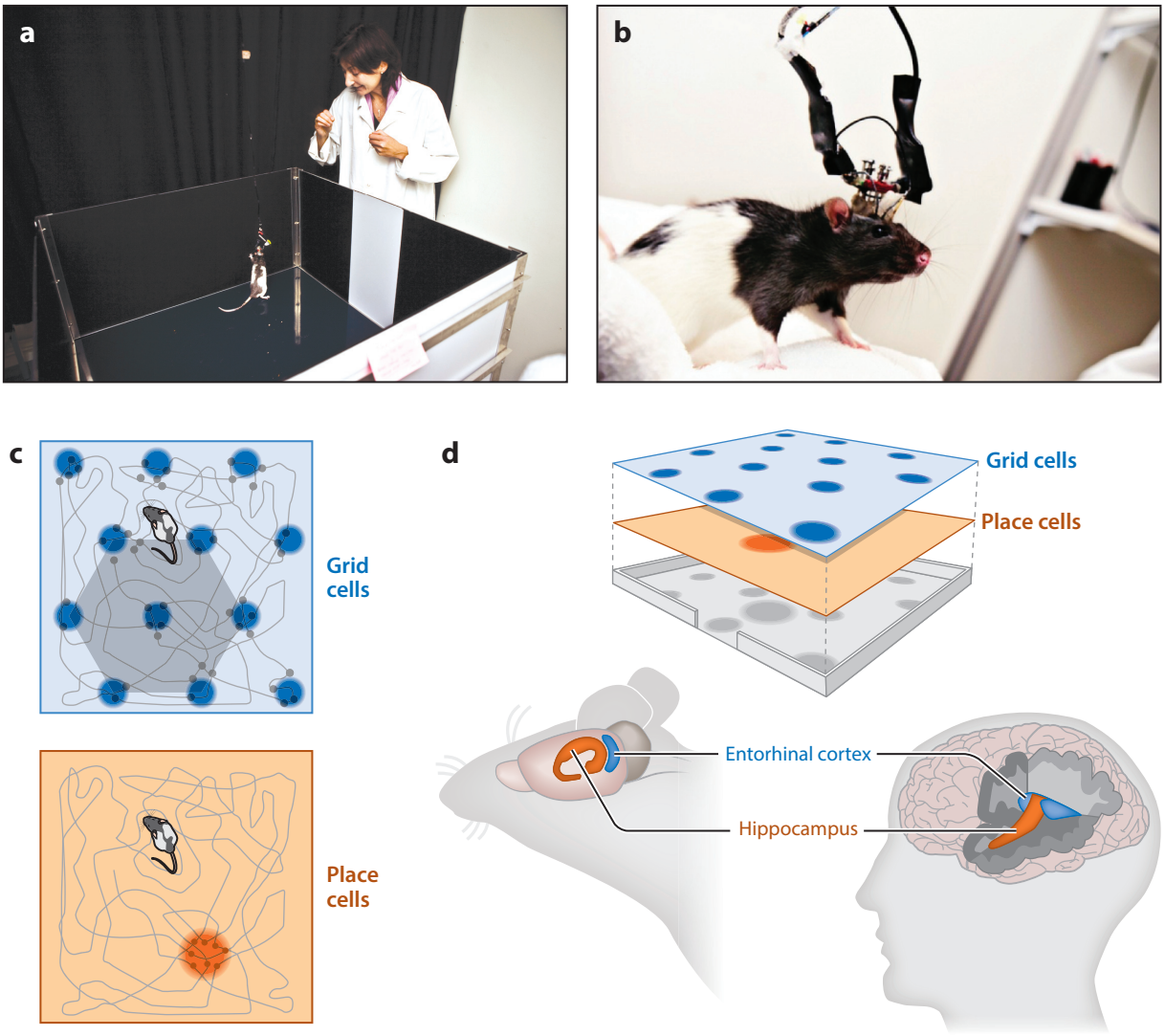

Figure 3

Place cells and grid cells. (*a*) The rat is moving within a square region. (*b*) The activity of a neuron is recorded. (*c*) When the rat moves around (the curve is the trajectory), each place cell fires at a particular location, but each grid cell fires at multiple locations that form a hexagon grid. (*d*) The place cells and grid cells exist in the brains of both rat and human.

the agent moves to $x + \Delta x$ after one step. We assume the following motion model:

$$h(x + \Delta x) = M(\Delta x)h(x), \qquad 3.$$

where $M(\Delta x)$ is a $d \times d$ matrix that depends on Δx. While $h(x)$ is the vector representation of the self-position x, $M(\Delta x)$ is the matrix representation of the self-motion Δx. As we show below, $\|h(x)\| = 1$ for all x, thus $M(\Delta x)$ is a rotation matrix, and the self-motion in 2D is represented by

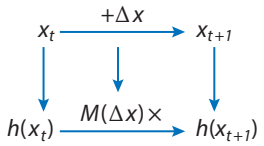

Figure 4

An illustration of the representational model for grid cells. Learning grid cells as vector representation of self-position coupled with matrix representation of self-motion. In this model, the 2D self-position x of the agent is represented by a high-dimensional vector $h(x)$, and the 2D self-motion Δx of the agent is represented by a matrix that transforms the vector $M(\Delta x)$. The vector of the next position, x_{t+1}, is obtained by multiplying the matrix of the motion $M(\Delta x)$ of the current position x_t, i.e., $h(x_{t+1}) = M(\Delta x)h(x_t)$.

a rotation in the d-dimensional sphere. We can illustrate the motion model using the diagram in **Figure 4**. Both $h(x)$ and $M(\Delta x)$ are to be learned.

Gao et al. (2018b) proposed that the brain uses the above representational scheme to carry out the simple addition calculation (illustrated in **Figure 5*a***; see also Paccanaro & Hinton 2001 for an earlier treatment of the addition problem).

3.1.4. Error correction. In data visualization such as t-SNE (van der Maaten & Hinton 2008), we represent high-dimensional data by 2D points. In grid cells, we do the opposite, representing 2D coordinates by high-dimensional vectors. Why does the brain bother with a high-dimensional representation of 2D coordinates? The answer lies in error correction. The neurons are intrinsically noisy. For a noisy observation of $h(x)$, by projecting it onto the submanifold of $(h(x), x \in [0, 1]^2)$, we can eliminate most of the noise.

In order to reduce the noise, we can use a high-dimensional h to record multiple noisy copes of x; then, a simple averaging will reduce the variance of noise. Apparently, the brain goes much further than that. It represents 2D x by a high-dimensional h, so that the angle between $h(x)$ and $h(x + \Delta x)$ is $\omega|\Delta x|$ for $\omega \gg 1$. This makes the system even more robust to noise, because $h(x)$ and $h(x + \Delta x)$ are very far apart when $\omega \gg 1$.

More specifically, we assume a magnified local isometry model:

$$\langle h(x), h(x + \Delta x)\rangle = 1 - \alpha|\Delta x|^2, \qquad 4.$$

which is a second-order Taylor expansion of a function of $|\Delta x|$ whose maximum is 1 at $|\Delta x| = 0$. For $\Delta x = 0$, we have $\|h(x)\|^2 = 1$ for all x. Let $\Delta\theta$ be the angle between $h(x)$ and $h(x + \Delta x)$, and then, $\langle h(x), h(x + \Delta x)\rangle = \cos(\Delta\theta) \approx 1 - \Delta\theta^2/2$ for small $\Delta\theta$. Thus, $\Delta\theta$ is proportional to $|\Delta x|$, i.e., $\Delta\theta = \omega|\Delta x|$, where $\omega = \sqrt{2\alpha} \gg 1$ (see **Figure 5*b***).

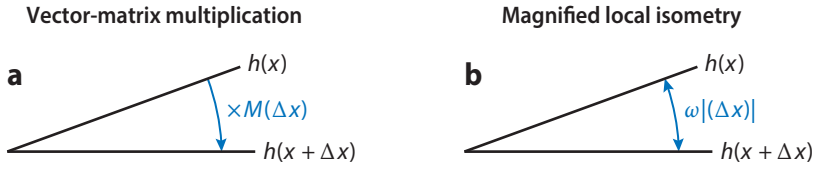

Figure 5

Grid cells form a high-dimensional vector representation of 2D self-position. Two submodels: (*a*) Local motion is modeled by vector-matrix multiplication. (*b*) Angle between two nearby vectors magnifies the Euclidean distance. x is a 2D position, Δx is a 2D self-motion, $h(x)$ is a high-dimensional vector representation of x, and $M(\Delta x)$ is a matrix representation of Δx. ω is a magnifying factor.

Gao et al. (2018b) showed that even if they randomly shut down (i.e., set to zero) 70% of the neurons in each step, their learned system can still perform a path integral accurately. Such dropout error could occur due to internal noises and asynchrony of neuron activities, as well as aging and diseases like Alzheimer's.

3.1.5. Emergence of hexagon patterns. For a fixed α, we can learn $(h(x), \forall x)$ and $M(\Delta x)$ by minimizing the least-squares loss:

$$\mathrm{E}_{x,\Delta x}[\|h(x+\Delta x) - M(\Delta x)h(x)\|^2] + \lambda \mathrm{E}_{x,\Delta x}[(\langle h(x), h(x+\Delta x)\rangle - (1-\alpha|\Delta x|^2))^2]. \quad 5.$$

The above loss function can be minimized by stochastic gradient descent, where for stochastic approximation of the expectations, we randomly sample $(x, \Delta x)$ uniformly where $x \in [0,1]^2$ and Δx is within a limited range.

The experiments of Gao et al. (2018b) show that as long as the dimension of $h(x)$ is $d \geq 6$, then the learning algorithm always learns the hexagon grid pattern for each element of $h(x)$. Even if $d = 100$, the algorithm still learns regular hexagon patterns. In **Figure 6*a***, each row displays the learned $h(x)$ for a given value of α, where $d = 6$. **Figure 6*b*** shows the learned $h(x)$ for $\alpha = 72$, where $d = 100$. If $d < 6$, the algorithm tends to learn square grid patterns.

Thus, if we move from x to $x + \Delta x$, the corresponding h will be rotated by a matrix $M(\Delta x)$, and h will rotate at a much faster speed $\omega|\Delta x|$. As a result, it will quickly rotate back to itself, causing the periodic grid patterns, which in turn causes the global ambiguities in position, because the same h may correspond to multiple positions. One could say that the grid patterns are almost an unwanted consequence of error correction. To resolve the ambiguities, Gao et al. (2018b) combined multiple blocks of grid cells to determine the position uniquely, and for each block, the magnifying parameter α can be learned automatically.

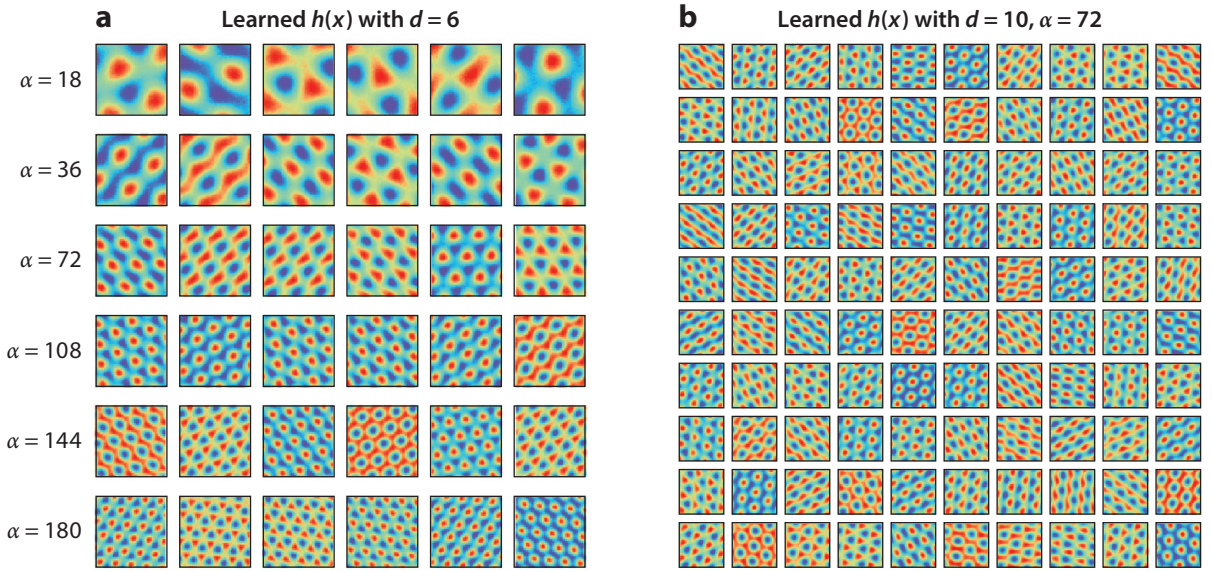

Figure 6

Learned grid cells. (*a*) Each row shows a component or a unit of $h(x)$ with a certain metric parameter α, where the number of units d is 6. (*b*) Learned units where the number of units d is 100 and $\alpha = 72$.

3.2. Vector Representation of State and Matrix Representation of Motion or Action

We may generalize the model in the previous subsection into a more general model for dynamic systems, where we represent the state by a vector and the change of the state caused by motion or action by a matrix. For example, Gao et al. (2019) recently proposed a model of V1 simple cells that is different from the sparse coding model (Olshausen & Field 1997) and the independent component analysis model (Bell & Sejnowski 1997) reviewed in Section 2.1. Gao et al. (2019) propose that a direct purpose of V1 cells is to perceive the displacements of pixels over time, where the displacements of pixels are caused by the relative motion between the agent (a rat or a human) and the surrounding 3D environment. Specifically, Gao et al. (2019) represent the local image contents by vectors and the local displacements of pixels by matrices, so that when the pixels undergo displacements, the vectors are rotated by the matrices representing the displacements. After learning this representational system, the agent will be able to sense the displacements of pixels based on the rotations of the vectors.

More generally, for a video sequence, we can represent the image frames by vectors and the motions or actions of the agent or the objects in the image by matrices. This will enable the agent to perceive the objects and their motions and actions while the agent is moving or taking actions.

In terms of neuroscience, the vectors correspond to the activities of neurons, and the matrices correspond to the synaptic connections. Interestingly, such a representational scheme appears to occur in nature. In quantum theory, the states are represented by vectors in a Hilbert space, and the changes of the states are represented by matrices or operators (Zee 2016). Similar to the creation and annihilation operators in quantum field theory, the matrix representations in vision may also account for discrete events such as the appearance and disappearance of objects. Perhaps the brain speaks the same mathematical language as nature.

4. LEARNING NONLINEAR VECTOR REPRESENTATION BY GENERATOR NETWORK

This section reviews the generator network that is a generalization of factor analysis, where the mapping from the latent factors to the signal is parameterized by a deep network. We also discuss the maximum likelihood learning algorithm that learns various generator models.

4.1. Deep Neural Networks

The models reviewed so far are based on linear structures. They can be generalized to nonlinear transformations, such as deep neural networks (LeCun et al. 1998, Krizhevsky et al. 2012), which are compositions of multiple layers of linear transformations and coordinate-wise nonlinear link functions.

Specifically, consider a nonlinear transformation $f(x)$ that can be decomposed recursively as $s_l = W_l h_{l-1} + b_l$, and $h_l = r_l(s_l)$, for $l = 1, \ldots, L$, with $f(x) = h_L$ and $h_0 = x$. W_l is a weight matrix at layer l, and b_l is the bias vector at layer l. Both s_l and h_l are vectors of the same dimensionality, and r_l is a one-dimensional nonlinear link function, the rectification function, that is applied coordinate-wise. $f(x)$ is a recursive composition of generalized linear model (GLM) structures.

Modern deep networks usually use $r_l(s) = \max(0, s)$, the rectified linear unit, or ReLU. For such nonlinear link functions, $f(x)$ is a multivariate linear spline where the linear pieces are recursively partitioned. This is similar to but more general than the recursive partitions in classification and regression trees (CART) (Breiman 2017) and multivariate adaptive regression splines (MARS) (Friedman 1991).

In computational neuroscience, each element or unit in h_l can be interpreted as a neuron or a cell, whose value can be related to the firing rate. Sometimes h_l is colloquially called a thought vector.

There are two special classes of neural networks. One consists of convolutional neural networks (LeCun et al. 1998, Krizhevsky et al. 2012), which are commonly applied to images, where the same linear transformations are applied around each pixel locally. The other class consists of recurrent neural networks (Hochreiter & Schmidhuber 1997), which are commonly applied to sequence data such as speech and natural language.

Neural networks are commonly used in supervised learning and reinforcement learning, where h_l at multiple layers can be considered predictive representations. They are also useful for unsupervised learning of generative models, as we review in the next subsection, where h_l at multiple layers can be considered generative representations.

4.1.1. Nonlinear generalization of logistic regression. For the deep network reviewed in the previous subsection, let $\alpha = (W_l, b_l, l = 1, \ldots, L)$ collect all the weight and bias parameters, and let $f_\alpha(x)$ be the resulting nonlinear transformation.

We can generalize the logistic regression model to

$$\mathrm{P}(y = 1|x) = D(x) = \frac{1}{1 + \exp(-f_\alpha(x))}. \quad 6.$$

The model is also called a discriminator network, and h_l at different layers can be considered predictive representations of x.

4.1.2. Nonlinear generalization of the exponential family model. We can also generalize the exponential family model to

$$\pi_\alpha(x) = \frac{1}{Z(\alpha)} \exp(f_\alpha(x))\rho(x), \quad 7.$$

where $\rho(x)$ is a reference measure such as the uniform distribution and $Z(\alpha)$ is the normalizing constant. This model is also called the energy-based model or Gibbs distribution.

The connection between the two models are as follows. Suppose $\rho(x)$ is the distribution of negative examples, i.e., $\mathrm{P}(x|y = 0) = \rho(x)$, and $\pi_\alpha(x)$ is the distribution of positive examples, i.e., $\mathrm{P}(x|y = 1) = \pi_\alpha(x)$. Suppose there are equal numbers of positive and negative examples; then, according to Bayes' law, $\mathrm{P}(y = 1|x)$ is given by Equation 6.

Later in the article, we make use of the above two models as the complementary models to the generator model, which we review next.

4.2. Nonlinear Generalization of Factor Analysis and Maximum Likelihood Learning

While sparse coding and independent component analysis generalize the prior assumption on the hidden vector h in factor analysis, the generator model generalizes the mapping from the hidden vector h to the input x, i.e.,

$$h \sim \mathrm{N}(0, I_d), \; x = g_\theta(h) + \varepsilon, \quad 8.$$

where g is parameterized by a deep network, similar to f in the previous subsection, i.e., $s_l = W_l h_{l+1} + b_l$, and $h_l = r_l(s_l)$, for $l = L-1, \ldots, 0$, with $h_L = h$, and $x = h_0$. W_l is a weight matrix at layer l, and b_l is the bias vector at layer l. θ collects all the weight and bias parameters at all the layers, and $\varepsilon \sim \mathrm{N}(0, \sigma^2 I_D)$ is the residual noise image that is independent of h.

While f in the previous subsection is a bottom-up network in the sense that it defines $h_0 = x \to h_1 \to \ldots \to h_L$, g in this subsection is a top-down network in the sense that it defines $h_L = h \to h_{L-1} \to \ldots \to h_0 = x$.

As in factor analysis, the model can be learned by maximum likelihood. We can write the prior distribution as $h \sim p(h)$, where $p(h)$ is the density of $\mathrm{N}(0, I_d)$. The conditional distribution of x given h is $p_\theta(x|h)$, which is the density of $\mathrm{N}(g_\theta(h), \sigma^2 I_D)$. The joint distribution, or the complete-data model, is $p_\theta(h, x) = p(h)p_\theta(x|h)$. The marginal distribution, or the observed-data model, is $p_\theta(x) = \int p_\theta(h, x)\mathrm{d}h$. The posterior distribution of h given x is $p_\theta(h|x) = p_\theta(h, x)/p_\theta(x)$. Unlike in factor analysis, the marginal $p_\theta(x)$ and the conditional $p_\theta(h|x)$ are not in closed form.

Let q_{data} be the distribution that generates the observed examples $x_i, i = 1, \ldots, n$. For large n, the maximum likelihood estimation of θ is to minimize the Kullback-Leibler (KL) divergence $\mathrm{KL}(q_{\text{data}} \| p_\theta)$ over θ, where the KL divergence is defined as $\mathrm{KL}(q|p) = \mathrm{E}_q[\log(q(x)/p(x))]$. In practice, the expectation with respect to q_{data} is approximated by the average over the observed examples. The gradient of the log-likelihood can be computed based on

$$-\frac{\partial}{\partial \theta}\mathrm{KL}(q_{\text{data}}(x)\|p_\theta(x)) = \mathrm{E}_{q_{\text{data}}(x)p_\theta(h|x)}\left[\frac{\partial}{\partial \theta}\log p_\theta(h, x)\right]. \quad 9.$$

The expectation with respect to the posterior distribution $p_\theta(h|x)$ can be approximated via Markov chain Monte Carlo (MCMC) sampling of $p_\theta(h|x)$, such as Langevin dynamics or Hamiltonian Monte Carlo (HMC) (Neal 2011). It can be efficiently implemented by gradient computation via back-propagation.

Han et al. (2017) learned the generator model by maximum likelihood. More recently, Xing et al. (2019) generalized the model to a deformable generator model with two hidden vectors (h_1, h_2), where h_1 is the geometric hidden vector that generates the displacements of the pixels, or the displacement field, and h_2 is the appearance hidden vector that generates the appearance image before deformation. The observed image is assumed to be generated by deforming or warping the appearance image by the displacement field. Such a model can be learned by maximum likelihood, and the learned model disentangles variations in shape and appearance.

Xing et al. (2019) trained the deformable generator on the 10,000 face images from the CelebA data set (Liu et al. 2015). **Figure 7** illustrates the change of the image if we vary the components of h_1, while keeping h_2 fixed at a certain value; different dimensions of h_1 capture different aspects of shape change. **Figure 8** displays an example of transferring and recombining the vectors. For two images, we can exchange their geometric vectors, so that each image changes its shape but retains its appearance.

Xie et al. (2019a) generalized the generator model to a dynamic generator model for a video sequence $(x_t, t = 1, \ldots, T)$, where x_t is an image frame at time t, by assuming a model of the form

$$h_t = f_\alpha(h_{t-1}, z_t), \quad 10.$$

$$x_t = g_\beta(h_t) + \varepsilon_t, \quad 11.$$

where $t = 1, \ldots, T$. Equation 10 is the transition model, and Equation 11 is the emission model. h_t is the d-dimensional hidden state vector, and $z_t \sim \mathrm{N}(0, I)$ is the noise vector of a certain

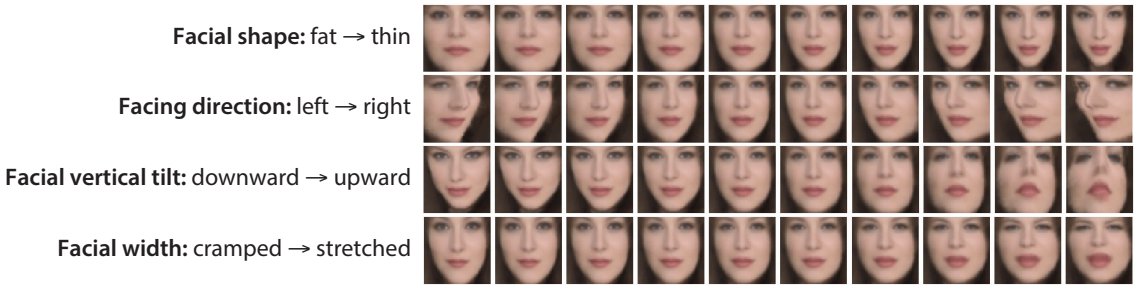

Figure 7

Each dimension of the geometric latent vector h_1 encodes geometric information such as shape and viewing angle. In each row, a characteristic of the face changes from left to right. In the first row, the shape of the face changes from wide to thin, and in the second row, the pose of the face changes from facing left to right. In the third row, the vertical tilt of the face varies from downward to upward, and in the fourth row, the face width changes from cramped to stretched. The deformable generator model is trained on the 10,000 face images randomly selected from the CelebA data set (Liu et al. 2015). The training images are cropped to 64 × 64 pixels, and the faces have different colors, illuminations, identities, viewing angles, shapes, and expressions.

dimensionality. The Gaussian noise vectors $(z_t, t = 1, \ldots, T)$ are independent of each other. The sequence of $(h_t, t = 1, \ldots, T)$ follows a nonlinear auto-regressive model, where the noise vector z_t encodes the randomness in the transition from h_{t-1} to h_t in the d-dimensional state space. f_α is a feed-forward neural network or multi-layer perceptron, where α denotes the weight and bias parameters of the network. x_t is the D-dimensional image, which is generated by the d-dimensional hidden state vector h_t. g_β is a top-down network, where β denotes the weight and bias parameters of this network, and $\varepsilon_t \sim \mathrm{N}(0, \sigma^2 I_D)$ is the residual error. The model is a state-space model or hidden Markov model. Xie et al. (2019a) learned the dynamic generator model by maximum likelihood. **Figures 9** and **10** show examples of learning the model from video data. Once the model is learned, we can synthesize dynamic textures from the learned model by first randomly initializing

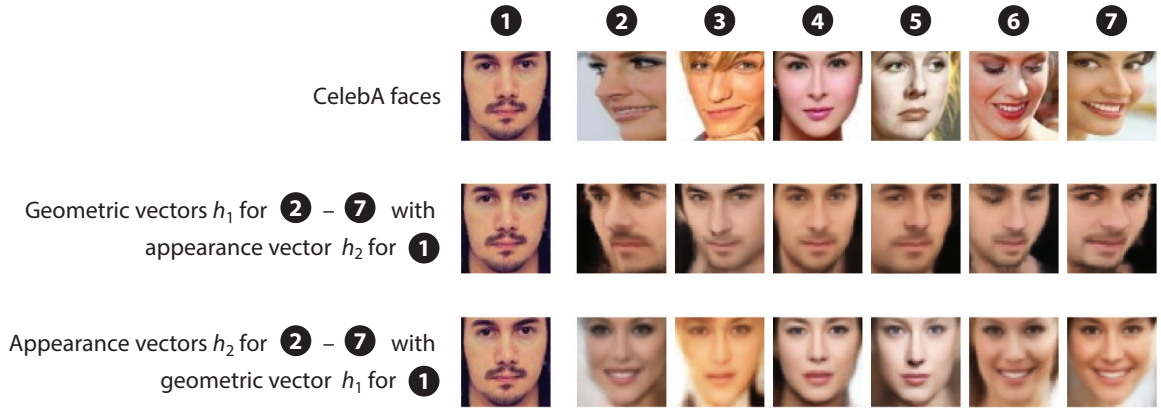

Figure 8

Transferring and recombining geometric and appearance vectors. The first row shows seven faces from the CelebA data set. The second row shows the faces generated by transferring and recombining the second through seventh faces' geometric vectors h_1 with the first face's appearance vector h_2 in the first row. The third row shows the faces generated by transferring and recombining the second through seventh faces' appearance vectors h_2 with the first face's geometric vector h_1 in the first row. The deformable generator model is trained on the 10,000 face images from the CelebA data set, which are cropped to 64 × 64 pixels, and the faces in the training data have a wide and diverse variety of colors, illuminations, identities, viewing angles, shapes, and expressions.

Figure 9

Generating dynamic textures. The dynamic generator model is learned by maximum likelihood from a single training video exhibiting a burning flame, sized 64 pixels × 64 pixels × 60 frames. A longer-length dynamic texture can be generated from a relatively short training sequence by just drawing longer independent and identically distributed samples from a Gaussian distribution. The first row displays 6 frames of the 60-frame observed sequence, and the second and third rows show 6 frames of two synthesized sequences of 120 frames, which are generated by the learned model.

the initial hidden state h_0, and then following Equations 10 and 11 to generate a sequence of images with a sequence of innovation vectors z_t sampled from Gaussian noise distribution.

4.3. Flow-Based Models

A flow-based model is of the form $x = g_\theta(h)$, but h is of the same dimensionality as x, and g_θ is a composition of a sequence of simple invertible transformations, so that the probability density of x can be obtained in closed form, $p_\theta(x) = p_0(g_\theta^{-1}(x))|\partial g_\theta(x)/\partial x|^{-1}$, where p_0 is the density of h,

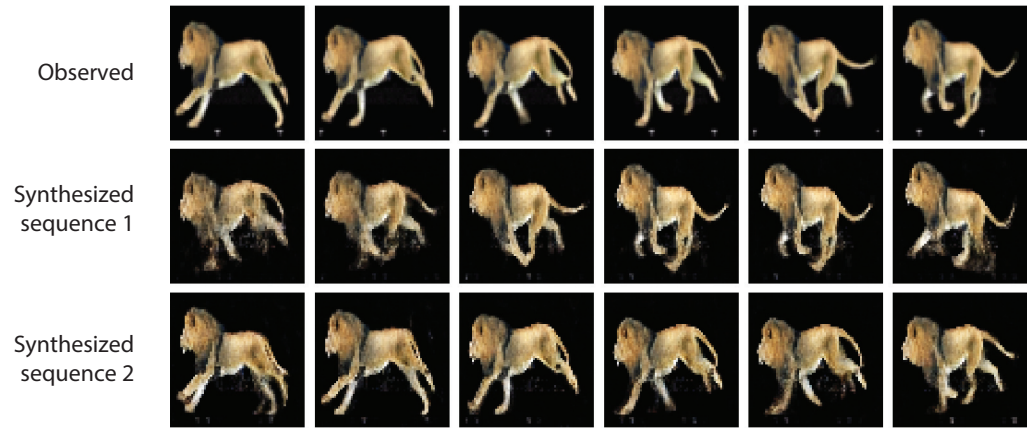

Figure 10

Generating action patterns. The dynamic generator model is trained on an animal action data set, including 20 videos of 10 animals performing running and walking. Each observed video is scaled to 64 pixels × 64 pixels × 30 frames. The first row displays 6 frames of the observed sequence, and the second and third rows show the corresponding frames of two synthesized sequences generated by the learned model.

and $|\partial g_\theta(x)/\partial x|$ is the absolute value of the determinant of the Jacobian of g_θ. Such a model can be considered a special generator model with invertible mapping between the hidden vector and the signal.

Flow-based models (Dinh et al. 2014, Rezende & Mohamed 2015, Dinh et al. 2017, Kingma & Dhariwal 2018, Grathwohl et al. 2019) can be traced back to independent component analysis reviewed in Section 2.1 by, for example, Dinh et al. (2014). They also arise from efforts to strengthen the inference model in variational auto-encoders (VAEs) (e.g., Rezende & Mohamed 2015), which are reviewed in the next section. The advantage of such models is that the normalized probability density of x can be obtained in closed form, so maximum likelihood learning is simple. A disadvantage is that the mapping g_θ may be of a rather contrived form in order to ensure that the mapping is invertible and the Jacobian can be efficiently computed.

5. LEARNING THE GENERATOR MODEL JOINTLY WITH COMPLEMENTARY MODELS

In modern deep learning literature, the generator model is usually learned jointly with a complementary model, and the learning is not based on maximum likelihood. Such learning methods are unconventional in statistics, but they can be quite powerful and can be interesting to statisticians.

5.1. Issues with Maximum Likelihood

The maximum likelihood learning of the generator network in the previous section has two issues. First, the learning algorithm requires MCMC sampling of the posterior distribution $p_\theta(h|x)$ as an inner loop, which can be expensive. Second, the maximum likelihood estimator, which minimizes $\mathrm{KL}(p_{\mathrm{data}}\|p_\theta)$ over θ, seeks to cover all the local modes of p_{data}, and as a result, the learned p_θ tends to be smoother than p_{data}, and images generated by the learned p_θ tends to be less sharp than the observed images.

To address the first issue, the VAE (Kingma & Welling 2014, Rezende et al. 2014, Mnih & Gregor 2014) learns an inference model to approximate the posterior distribution. To address the second issue, the generator model can be learned jointly with a discriminator, as in generative adversarial networks (GAN) (Goodfellow et al. 2014, Radford et al. 2015), or an energy-based model that specifies the distribution of x explicitly up to a normalizing constant.

While the generator model is parameterized by a top-down network as show in the left panel of **Figure 11**, the complementary model is parameterized by a separate bottom-up network as shown in the right panel of **Figure 11**.

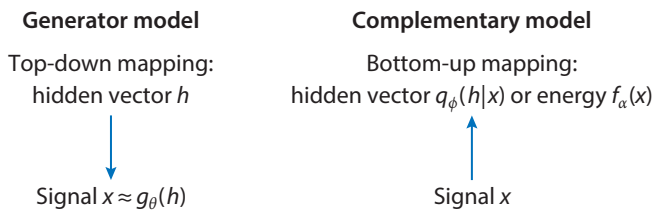

Figure 11

A generator model coupled with a complementary model. The generator model is parameterized by a top-down network, g_θ, which maps the hidden vector h to signal x, while the complementary model is parameterized by a bottom-up network q_ϕ that maps signal x to hidden vector h or f_α that maps signal x to energy.

5.2. Variational Auto-Encoder: Joint Learning with Inference Model

In order to avoid MCMC sampling from the posterior $p_\theta(h|x)$, the VAE (Kingma & Welling 2014, Rezende et al. 2014, Mnih & Gregor 2014) approximates $p_\theta(h|x)$ by a tractable $q_\phi(h|x)$, such as

$$q_\phi(h|x) \sim \mathrm{N}(\mu_\phi(x), \mathrm{diag}(v_\phi(x))), \tag{12.}$$

where both μ_ϕ and v_ϕ are bottom-up networks that map x to d-dimensional vectors, with ϕ collecting all the weight and bias parameters of the bottom-up networks. For $h \sim q_\phi(h|x)$, we can write $h = \mu_\phi(x) + \mathrm{diag}(v_\phi(x))^{1/2} z$, where $z \sim \mathrm{N}(0, I_d)$. Thus, expectation with respect to $h \sim q_\phi(h|x)$ can be written as expectation with respect to z. This reparameterization trick (Kingma & Welling 2014) helps reduce the variance in Monte Carlo integration. We may consider $q_\phi(h|x)$ as an approximation to the iterative MCMC sampling of $p_\theta(h|x)$. In other words, $q_\phi(h|x)$ is the learned inferential computation that approximately samples from $p_\theta(h|x)$.

The VAE objective is a modification of the maximum likelihood estimation (MLE) objective:

$$\mathrm{KL}(q_{\mathrm{data}}(x)q_\phi(h|x)\|p_\theta(h,x)) = \mathrm{KL}(q_{\mathrm{data}}(x)\|p_\theta(x)) + \mathrm{KL}(q_\phi(h|x)\|p_\theta(h|x)). \tag{13.}$$

We define the conditional KL divergence as $\mathrm{KL}(q(x|y)\|p(x|y)) = \mathrm{E}_{q(x,y)}[\log(q(x|y)/p(x|y))]$, where the expectation is with respect the joint distribution $q(x, y)$. We estimate θ and ϕ jointly by

$$\min_\theta \min_\phi \mathrm{KL}(q_{\mathrm{data}}(x)q_\phi(h|x)\|p_\theta(h,x)), \tag{14.}$$

which can be accomplished by gradient descent.

Define $Q(h, x) = q_{\mathrm{data}}(x)q_\phi(h|x)$. Define $P(h, x) = p(h)p_\theta(x|h)$. Q is the distribution of the complete data (h, x), where $q_\phi(h|x)$ can be interpreted as an imputer that imputes the missing data h. P is the distribution of the complete-data model. The VAE is $\min_\theta \min_\phi \mathrm{KL}(Q\|P)$.

We may interpret the VAE as an alternating projection between Q and P (**Figure 12** provides an illustration). The wake-sleep algorithm (Hinton et al. 1995) is similar to the VAE, except that it updates ϕ by $\min_\phi \mathrm{KL}(P\|Q)$, where the order is flipped. Xing et al. (2019) implemented the VAE learning of the deformable generator model, and the results are similar to maximum likelihood learning.

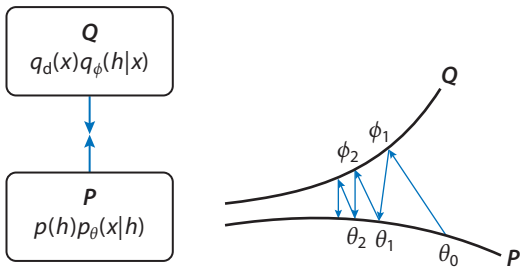

Figure 12

Variational auto-encoder as joint minimization by alternating projection. $P = p(h)p_\theta(x|h)$ is the distribution of the complete-data model, where $p(h)$ is the prior distribution of hidden vector h and $p_\theta(x|h)$ is the conditional distribution of x given h. $Q = q_d(x)q_\phi(h|x)$ is the distribution of the complete data (h, x), where $q_d(x)$ is the data distribution and $q_\phi(h|x)$ is the learned inferential computation that approximately samples from the posterior distribution $p_\theta(x|h)$. (*Left*) Interaction between the models. (*Right*) Alternating projection. The two models run toward each other.

5.2.1. Maximum likelihood estimation algorithm from the variational auto-encoder perspective. Recall that the MLE objective is to minimize $\mathrm{KL}(q_{\mathrm{data}}\|p_\theta)$. Suppose θ_t is the current estimate in the MLE algorithm. We can write

$$\mathrm{KL}(q_{\mathrm{data}}(x)p_{\theta_t}(h|x)\|p(h)p_\theta(x|h)) = \mathrm{KL}(q_{\mathrm{data}}(x)\|p_\theta(x)) + \mathrm{KL}(p_{\theta_t}(h|x)\|p_\theta(h|x)), \quad 15.$$

where we replace $q_\phi(h|x)$ in the VAE by $p_{\theta_t}(h|x)$.

The above identity underlies the EM algorithm (Dempster et al. 1977), where we find θ_{t+1} by maximizing the left-hand side over θ. Because $\mathrm{KL}(p_{\theta_t}(h|x)\|p_\theta(h|x))$, as a function of θ, is minimized at $\theta = \theta_t$, with minimum value 0, $\mathrm{KL}(q_{\mathrm{data}}(x)p_{\theta_t}(h|x)\|p(h)p_\theta(x|h))$ majorizes $\mathrm{KL}(q_{\mathrm{data}}(x)\|p_\theta(x))$ as functions of θ, and both functions touch at θ. Thus minimizing the left-hand side will decrease $\mathrm{KL}(q_{\mathrm{data}}\|p_\theta)$, which leads to the monotonicity of the EM algorithm. Moreover, the derivative of $\mathrm{KL}(p_{\theta_t}(h|x)\|p_\theta(h|x))$, as a function of θ, is zero at θ_t. Thus, the gradient of the KL divergence on the left-hand side at θ_t agrees with the gradient of the first KL divergence on the right-hand side at θ_t. This leads to the identity in Equation 9.

5.2.2. Comparison with traditional variational inference. In the VAE, the model $q_\phi(h|x)$ and the parameter ϕ are shared by all the training examples x, so that $\mu_\phi(x)$ and $v_\phi(x)$ in Equation 12 can be computed directly for each x given ϕ. This is different from traditional variational inference (Jordan et al. 1999, Blei et al. 2017), where, for each x, a model $q_{\mu,v}(h)$ is learned by minimizing $\mathrm{KL}(q_{\mu,v}(h)\|p_\theta(h|x))$ with x fixed, so that (μ, v) is computed by an iterative algorithm for each x, which is an inner loop of the learning algorithm. This is similar to maximum likelihood learning, except that in maximum likelihood learning, the inner loop is an iterative algorithm that samples $p_\theta(h|x)$ instead of minimizing over (μ, v). The learned networks $\mu_\phi(x)$ and $v_\phi(d)$ in the VAE are to approximate the iterative minimization algorithm by direct mappings.

5.3. Generative Adversarial Net: Joint Learning with Discriminator

The generator model learned by MLE or the VAE usually cannot generate very realistic images. Both MLE and the VAE target $\mathrm{KL}(q_{\mathrm{data}}\|p_\theta)$, though the VAE only minimizes an upper bound of $\mathrm{KL}(q_{\mathrm{data}}\|p_\theta)$. Consider minimizing $\mathrm{KL}(q\|p)$ over p within a certain model class. If q is multi-modal, then p is obliged to fit all the major modes of q because $\mathrm{KL}(q\|p)$ is an expectation with respect to q. Thus, p tends to interpolate the major modes of q if p cannot fit the modes of q closely. As a result, p_θ learned by MLE or the VAE tends to generate images that are not as sharp as the observed images.

The behavior of minimizing $\mathrm{KL}(q\|p)$ over p is different from minimizing $\mathrm{KL}(q\|p)$ over q. If p is multi-modal, q tends to capture some major modes of p while ignoring the other modes of p, because $\mathrm{KL}(q\|p)$ is an expectation with respect to q. In other words, $\min_q \mathrm{KL}(q\|p)$ encourages mode chasing, whereas $\min_p \mathrm{KL}(q\|p)$ encourages mode covering.

Sharp synthesis can be achieved by GAN (Goodfellow et al. 2014, Radford et al. 2015), which pairs a generator model G with a discriminator model D. For an image x, $D(x)$ is the probability that x is an observed (real) image instead of a generated (faked) image. It can be parameterized by a bottom-up network $f_\alpha(x)$, so that $D(x) = 1/(1 + \exp(-f_\alpha(x))$, i.e., logistic regression (see Section 4.1.1). We can train the pair of (G, D) by an adversarial, zero-sum game. Specifically, let $G(h) = g_\theta(h)$ be a generator. Let

$$V(D, G) = \mathrm{E}_{q_{\mathrm{data}}}[\log D(X)] + \mathrm{E}_{h\sim p(h)}[\log(1 - D(G(h))], \quad 16.$$

where $E_{q_{\text{data}}}$ can be approximated by averaging over the observed examples, and E_h can be approximated by Monte Carlo average over the faked examples generated by the generator model. We learn D and G by $\min_G \max_D V(D, G)$. $V(D, G)$ is the log-likelihood for D, i.e., the log-probability of the real and faked examples. However, $V(D, G)$ is not a very convincing objective for G. In practice, the training of G is usually modified into maximizing $E_{h\sim p(h)}[\log D(G(h))]$ to avoid the vanishing gradient problem.

For a given θ, let p_θ be the distribution of $g_\theta(h)$ with $h \sim p(h)$. Assuming a perfect discriminator. Then, according to Bayes' theorem, $D(x) = q_{\text{data}}(x)/(q_{\text{data}}(x) + p_\theta(x))$ (assuming equal numbers of real and faked examples). Then, θ minimizes the Jensen-Shannon (JS) divergence

$$\text{JS}(q_{\text{data}} \| p_\theta) = \text{KL}(p_\theta \| p_{\text{mix}}) + \text{KL}(q_{\text{data}} \| p_{\text{mix}}), \tag{17.}$$

where $p_{\text{mix}} = (q_{\text{data}} + p_\theta)/2$.

In JS divergence, the model p_θ also appears on the left-hand side of KL divergence. This encourages p_θ to fit some major modes of q_{data} while ignoring others. As a result, GAN learning suffers from the mode collapsing problem, i.e., the learned p_θ may miss some modes of q_{data}. However, the p_θ learned by GAN tends to generate sharper images than the p_θ learned by MLE or the VAE.

5.4. Energy-Based Model

Similar to GAN, we can pair the generator model with an energy-based model (Ngiam et al. 2011; Dai et al. 2014; Lu et al. 2016; Xie et al. 2016, 2017, 2018b; Gao et al. 2018a) instead of a discriminator model. Similar to the discriminator model, the energy-based model is also defined by a bottom-up network. Also similar to the discriminator model, which seeks to tell apart the images generated by the generator model and the real images, the energy-based model plays the role of an evaluator, evaluating the images generated by the generator model against the real images. We may intuitively consider the generator model as an actor or a student and the energy-based model as a critic or a teacher.

5.4.1. Generalizing the exponential family model. The energy function in the energy-based model, $-f_\alpha(x)$, defines the energy of x, and a low energy x is assigned a high probability. Specifically, we have the following probability model:

$$\pi_\alpha(x) = \frac{1}{Z(\alpha)} \exp\left[f_\alpha(x)\right], \tag{18.}$$

where $f_\alpha(x)$ is parameterized by a bottom-up deep network with parameters α, and $Z(\alpha)$ is the normalizing constant. It is the nonlinear generalization of the exponential family model (see Section 4.1.2), and it is also a Gibbs distribution and a random field model. Here, we drop the reference measure $\rho(x)$, or we assume it is uniform measure. In contrast to the discriminator model $D(x)$, we may intuitively call π_α the evaluator model, where f_α assigns the value to x, and π_α evaluates x by a normalized probability distribution (see the right panel of **Figure 11**).

In terms of learning representations, the generator model represents the observed x by a vector h, and the energy-based model learns multiple layers of features in the network $f_\alpha(x)$. The energy-based model learned by maximum likelihood tends to have stronger synthesis ability than the generator model learned by maximum likelihood because the former directly approximates q_{data} by f_α, while the latter approximates q_{data} by p_θ, which is obtained by integrating out h.

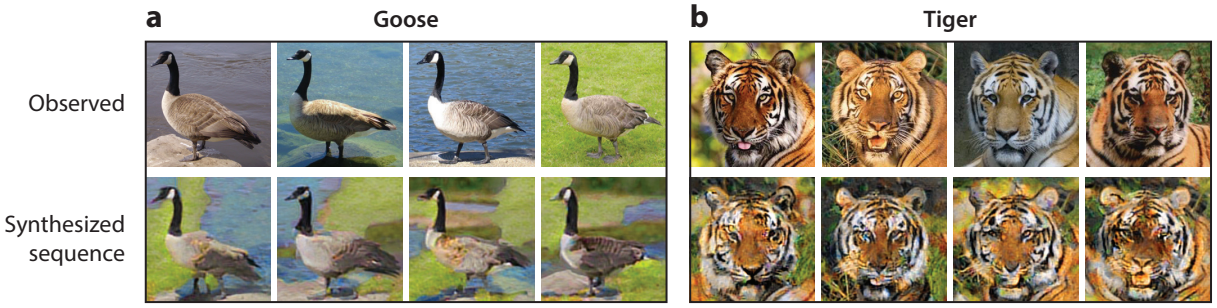

Figure 13

Learning the energy-based model by maximum likelihood: (*a*) goose, (*b*) tiger. For each category, the first row displays four of the training images, and the second row displays four of the images generated by the learning algorithm. $f_\alpha(x)$ is parameterized by a four-layer bottom-up deep network, where the first layer has 100 7×7 filters with subsampling size 2, the second layer has 64 5×5 filters with subsampling size 1, the third layer has 20 3×3 filters with subsampling size 1, and the fourth layer is a fully connected layer with a single filter that covers the whole image. The number of parallel chains for Langevin sampling is 16, and the number of Langevin iterations between every two consecutive updates of parameters is 10. The training images are 224×224 pixels.

5.4.2. Maximum likelihood. To learn the energy-based model π_α, the maximum likelihood estimator minimizes $\mathrm{KL}(q_{\mathrm{data}} \| \pi_\alpha)$ over α. We can update α by a gradient descent:

$$-\frac{\partial}{\partial \alpha}\mathrm{KL}(q_{\mathrm{data}}(x)\|\pi_\alpha(x)) = \mathrm{E}_{q_{\mathrm{data}}}\left[\frac{\partial}{\partial \alpha} f_\alpha(x)\right] - \mathrm{E}_{\pi_\alpha}\left[\frac{\partial}{\partial \alpha} f_\alpha(x)\right]. \quad 19.$$

The above identity follows from the fact that the derivative of the cumulant or log partition function $\log Z(\alpha)$ is the expectation of the derivative of $f_\alpha(x)$.

To implement the above update, we need to compute the expectation with respect to the current model π_α. It can be approximated by MCMC, such as Langevin dynamics or HMC, that samples from π_α. Again, it can be efficiently implemented by gradient computation via back-propagation. Lu et al. (2016) and Xie et al. (2016) learned the energy-based model using such a learning method (see **Figure 13** for an illustration).

More recently, Nijkamp et al. (2019) studied a very simple implementation of the learning algorithm where, within each learning iteration, we run K-step MCMC starting from a uniform noise distribution. After convergence, the K-step MCMC is capable of generating realistic images.

The energy-based model is related to the discriminator model via Bayes' law (see Section 4.1.2 and also Dai et al. 2014, Wu et al. 2019). The model can be learned discriminatively by fitting a logistic regression model (see Tu 2007, Jin et al. 2017, Lazarow et al. 2017, Lee et al. 2018).

5.4.3. Adversarial contrastive divergence: joint learning of generator and energy-based model. To avoid MCMC sampling of π_α, we may approximate it by a generator model p_θ, which can generate synthesized examples directly, i.e., sampling h from $p(h)$, and transforming h to x by $x = g_\theta(h)$. We may consider p_θ an approximation to the iterative MCMC sampling of π_α. In other words, p_θ is the learned computation that approximately samples from π_α—it is an approximate direct sampler of π_α.

We can learn both π_α and p_θ using the following objective function (Kim & Bengio 2016, Dai et al. 2017):

$$\min_\alpha \max_\theta [\mathrm{KL}(q_{\mathrm{data}}\|\pi_\alpha) - \mathrm{KL}(p_\theta\|\pi_\alpha)], \quad 20.$$

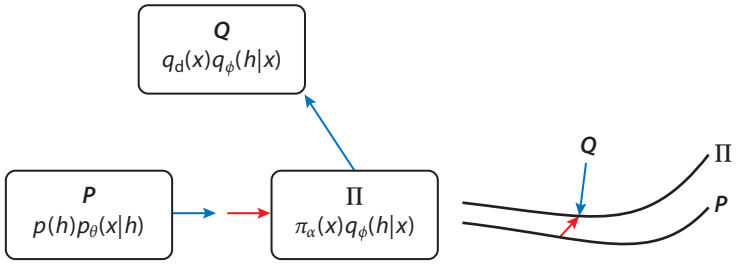

Figure 14

Adversarial contrastive divergence where the energy-based model favors real data against the generator. (*Left*) Interaction between the models. Red arrow indicates a chasing game, where the red arrow pointing to Π indicates that Π seeks to move away from P. The blue arrow pointing from P to Π indicates that P seeks to move close to Π. (*Right*) Contrastive divergence.

or equivalently,

$$\max_{\alpha} \min_{\theta} [\mathrm{KL}(p_\theta \| \pi_\alpha) - \mathrm{KL}(q_{\mathrm{data}} \| \pi_\alpha)]. \tag{21.}$$

The gradient for updating α becomes

$$\frac{\partial}{\partial \alpha} [\mathrm{E}_{q_{\mathrm{data}}}(f_\alpha(x)) - \mathrm{E}_{p_\theta}(f_\alpha(x))], \tag{22.}$$

where the intractable $\log Z(\alpha)$ term is canceled.

Because of the negative sign in front of the second KL divergence in Equation 20, we need $\max_\theta$ in Equation 20 or $\min_\theta$ in Equation 21, so that the learning becomes adversarial (illustrated in **Figure 14**). Inspired by the work of Hinton (2002), Han et al. (2019) called Equation 20 the adversarial contrastive divergence (ACD). It underlies the work of Kim & Bengio (2016) and Dai et al. (2017).

The adversarial form (Equation 20 or 21) defines a chasing game with the following dynamics: The generator p_θ chases the energy-based model π_α in $\min_\theta \mathrm{KL}(p_\theta \| \pi_\alpha)$, while the energy-based model π_α seeks to get closer to q_{data} and away from p_θ. The red arrow in **Figure 14** illustrates this chasing game. The result is that π_α lures p_θ toward q_{data}. In the idealized case, p_θ always catches up with π_α, and then π_α will converge to the maximum likelihood estimate $\min_\alpha \mathrm{KL}(q_{\mathrm{data}} \| \pi_\alpha)$, and p_θ converges to π_α. This chasing game is different from the VAE $\min_\theta \min_\phi \mathrm{KL}(Q \| P)$, which defines a cooperative game where q_ϕ and p_θ run toward each other.

Even though the above chasing game is adversarial, both models are running toward the data distribution. While the generator model runs after the energy-based model, the energy-based model runs toward the data distribution. As a consequence, the energy-based model guides or leads the generator model toward the data distribution. It is different from GAN (Goodfellow et al. 2014), in which the discriminator eventually becomes confused because the generated data become similar to the real data. In the above chasing game, the energy-based model becomes close to the data distribution.

The updating of α by Equation 22 is similar to Wasserstein GAN (WGAN) (Arjovsky et al. 2017), but unlike WGAN, f_α defines a probability distribution π_α, and the learning of θ is based on $\min_\theta \mathrm{KL}(p_\theta \| \pi_\alpha)$, which is a variational approximation to π_α. This variational approximation only requires knowing $f_\alpha(x)$, without knowing $Z(\alpha)$. However, unlike $q_\phi(h|x)$, $p_\theta(x)$ is still intractable; in particular, its entropy does not have a closed form. Thus, we can again use variational

approximation, by changing the problem $\min_\theta \mathrm{KL}(p_\theta \| \pi_\alpha)$ to

$$\min_\theta \min_\phi \mathrm{KL}(p(h)p_\theta(x|h) \| \pi_\alpha(x)q_\phi(h|x)). \quad 23.$$

Define $\Pi(h, x) = \pi_\alpha(x)q_\phi(h|x)$, and then the problem is $\min_\theta \min_\phi \mathrm{KL}(P\|\Pi)$, which is analytically tractable and underlies the work of Dai et al. (2017). In fact,

$$\mathrm{KL}(P\|\Pi) = \mathrm{KL}(p_\theta(x)\|\pi_\alpha(x)) + \mathrm{KL}(p_\theta(h|x)\|q_\phi(h|x)). \quad 24.$$

Thus, we can modify Equation 21 into $\max_\alpha \min_\theta \min_\phi [\mathrm{KL}(P\|\Pi) - \mathrm{KL}(Q\|\Pi)]$, because $\mathrm{KL}(Q\|\Pi) = \mathrm{KL}(q_{\mathrm{data}}\|\pi_\alpha)$.

Note that in the VAE (Equation 13), the objective function is in the form of KL + KL, whereas in ACD (Equation 20), it is in the form of KL - KL. In both Equations 20 and 13, the first KL is about maximum likelihood. The KL+KL form of the VAE makes the computation tractable by changing the marginal distribution of x to the joint distribution of (h, x). The KL-KL form of ACD makes the computation tractable by cancelling the intractable $\log Z(\alpha)$ term. Because of the negative sign in Equation 20, the ACD objective function becomes an adversarial one or a minimax game.

Also note that in the VAE, p_θ appears on the right-hand side of KL, whereas in ACD, p_θ appears on the left-hand side of KL. Thus, in ACD, p_θ may exhibits mode chasing behavior, i.e., fitting the major modes of π_α, while ignoring other modes.

5.4.4. Maximum likelihood estimation algorithm from the adversarial contrastive divergence perspective. Recall that the maximum likelihood is to minimize $\mathrm{KL}(q_{\mathrm{data}}\|\pi_\alpha)$. Suppose α_t is the current estimate of the MLE algorithm. We can consider the contrastive divergence

$$\mathrm{KL}(q_{\mathrm{data}}\|\pi_\alpha) - \mathrm{KL}(\pi_{\alpha_t}\|\pi_\alpha), \quad 25.$$

where we replace p_θ in ACD by π_{α_t}. Again, $\mathrm{KL}(\pi_{\alpha_t}\|\pi_\alpha)$ as a function of α is minimized at α_t, where the gradient is zero. Thus the gradient of the above contrastive divergence at α_t agrees with the gradient of the first KL divergence $\mathrm{KL}(q_{\mathrm{data}}\|\pi_\alpha)$ for MLE, which leads to the identity in Equation 19. For the K-step MCMC of Nijkamp et al. (2019), we can replace π_{α_t} above by the marginal distribution obtained by K-step MCMC toward π_{α_t}, initialized at the uniform distribution. Nijkamp et al. (2019) also studied the learned K-step MCMC as a model in itself.

5.5. Divergence Triangle: Variational Auto-Encoder Plus Adversarial Contrastive Divergence, Joint Learning of Three Models

We can combine the VAE and ACD into a divergence triangle, which involves the following three joint distributions on (h, x) defined above:

1. Q distribution: $Q(h, x) = q_{\mathrm{data}}(x)q_\phi(h|x)$
2. P distribution: $P(h, x) = p(h)p_\theta(x|h)$
3. Π distribution: $\Pi(h, x) = \pi_\alpha(x)q_\phi(h|x)$

Han et al. (2019) proposed to learn the three models, p_θ, π_α, and q_ϕ, by the following divergence triangle loss functional $\mathcal{D}$ (see **Figure 15** for an illustration):

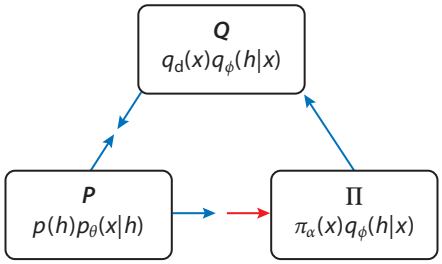

Figure 15

Divergence triangle based on the Kullback-Leibler divergences between three joint distributions, Q, P, and Π, of (h, x). The blue arrow indicates the "running toward" behavior and the red arrow indicates the "running away" behavior.

$$\max_{\alpha} \min_{\theta} \min_{\phi} \mathcal{D}(\alpha, \theta, \phi),$$

$$\mathcal{D} = \mathrm{KL}(Q\|P) + \mathrm{KL}(P\|\Pi) - \mathrm{KL}(Q\|\Pi). \tag{26.}$$

The divergence triangle is based on the three KL divergences between the three joint distributions on (h, x). It has a symmetric and antisymmetric form, where the antisymmetry is due to the negative sign in front of the last KL divergence and the maximization over α. Compared to the VAE and ACD objective functions in the previous subsections, $\mathrm{KL}(Q\|P)$ is the VAE part, and $\mathrm{KL}(P\|\Pi) - \mathrm{KL}(Q\|\Pi)$ is the ACD part.

The divergence triangle leads to the following dynamics between the three models: (*a*) Q and P seek to get close to each other. (*b*) P seeks to get close to Π. (*c*) π seeks to get close to q_{data}, but it seeks to get away from P, as indicated by the red arrow. Note that $\mathrm{KL}(Q\|\Pi) = \mathrm{KL}(q_{\text{data}}\|\pi_\alpha)$, because $q_\phi(h|x)$ is canceled out. The effect of (*b*) and (*c*) is that π gets close to q_{data} while inducing P to get close to q_{data} as well, or in other words, P chases π_α toward q_{data}.

Han et al. (2019) also employed the layer-wise training scheme of Karras et al. (2017) to learn models by divergence triangle from the CelebA-HQ data set (Karras et al. 2017), including 30,000 celebrity face images with resolutions of up to 1,024 × 1,024 pixels. The learning algorithm converges stably, without extra tricks, to obtain realistic results, as shown in **Figure 16**.

Figure 16*a* displays a few 1,024 × 1,024 images generated by the learned generator model with a 512-dimensional latent vector. **Figure 16*b*** shows an example of interpolation. The two images at the two ends are generated by two different latent vectors. The images in between are generated by the vectors that are linear interpolations of the two vectors at the two ends. Even though the interpolation is linear in the latent vector space, the nonlinear mapping leads to a highly nonlinear interpolation in the image space. We first do linear interpolation between the latent vectors at the two ends, i.e., $(1 - \alpha) \times h_0 + \alpha \times h_1$, where h_0 and h_1 are two latent vectors at two ends, respectively, and α is in the closed unit interval [0, 1]. The images in between are generated by mapping those interpolated vectors to image space via the learned generator. The interpolation experiment shows that the algorithm can learn a smooth generator model that traces the manifold of the data distribution.

5.6. Cooperative Learning via Markov Chain Monte Carlo Teaching

In ACD, the generator model p_θ is used to approximate the energy-based model π_α, and we treat the examples generated by p_θ as if they are generated from π_α for the sake of updating α. The gap between p_θ and π_α can cause bias in learning. In the work of Xie et al. (2018a, 2019b), we

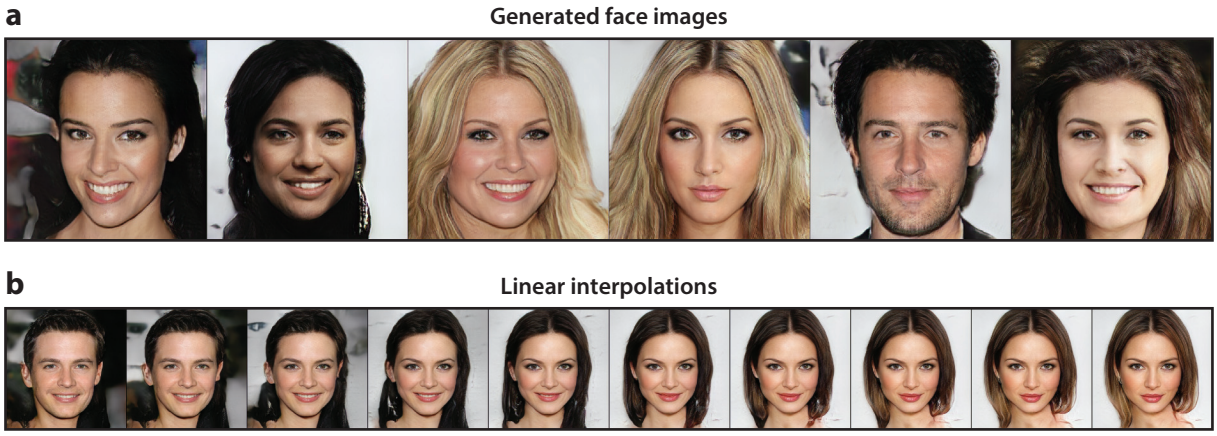

Figure 16

Learning generator model by divergence triangle from the CelebA-HQ data (Karras et al. 2017) set that includes 30,000 high-resolution celebrity face images. (*a*) Generated face images with 1,024 × 1,024 resolution sampled from the learned generator model with a 512-dimensional latent vector. (*b*) Linear interpolation of the vector representations. The images at the two ends are generated from latent vectors randomly sampled from a Gaussian distribution. Each image in the middle is obtained by first interpolating the two vectors of the two end images, and then generating the image using the generator.

proposed to bring back MCMC to bridge the gap. Instead of running MCMC from scratch, we run a finite-step MCMC toward π_α, initialized from the examples generated by p_θ. We then use the examples produced by the finite-step MCMC as the synthesized examples from π_α for updating α. Meanwhile, we update p_θ based on how the finite-step MCMC revises the initial examples generated by p_θ; in other words, the energy-based model (as a teacher) π_α distills the MCMC into the generator (as a student) p_θ. We call this scheme cooperative learning.

Specifically, we first generate $\hat{h}_i \sim \mathrm{N}(0, I_d)$, and then generate $\hat{x}_i = g_\theta(\hat{h}_i) + \varepsilon_i$, for $i = 1, \ldots, \tilde{n}$. Starting from $\{\hat{x}_i, i = 1, \ldots, \tilde{n}\}$, we run MCMC such as Langevin dynamics for a finite number of steps toward π_α to get $\{\tilde{x}_i, i = 1, \ldots, \tilde{n}\}$, which are revised versions of $\{\hat{x}_i\}$. $\{\tilde{x}_i\}$ are used as the synthesized examples from the energy-based model. We can then update α according to Equation 19.

The energy-based model can teach the generator via MCMC. The key is that in the generated examples, the latent h is known. In order to update θ of the generator model, we treat $\{\tilde{x}_i, i = 1, \ldots, \tilde{n}\}$ as the training data for the generator. Since these $\{\tilde{x}_i\}$ are obtained by the Langevin dynamics initialized from $\{\hat{x}_i\}$, which are generated by the generator model with known latent factors $\{\hat{h}_i\}$, we can update θ by learning from the complete data $\{(\hat{h}_i, \tilde{x}_i); i = 1, \ldots, \tilde{n}\}$, which is a supervised learning problem, or more specifically, a nonlinear regression of $\tilde{x}_i$ on $\hat{h}_i$. At $\theta^{(t)}$, the latent factors $\hat{h}_i$ generates and thus reconstructs the initial example $\hat{x}_i$. After updating θ, we want $\hat{h}_i$ to reconstruct the revised example $\tilde{x}_i$. That is, we revise θ to absorb the MCMC transition from $\hat{x}_i$ to $\tilde{x}_i$. The left panel of **Figure 17** illustrates the basic idea. The right panels shows a more rigorous method, where we initialize the MCMC for inferring $\{\tilde{h}_i\}$ from the known $\{\hat{h}_i\}$ and then update θ based on $\{(\tilde{h}_i, \tilde{x}_i), i = 1, \ldots, \tilde{n}\}$.

The theoretical understanding of the cooperative learning scheme is given below:

1. Modified contrastive divergence for the energy-based model: In the traditional contrastive divergence (Hinton 2002), $\hat{x}_i$ is taken to be the observed x_i. In cooperative learning, $\hat{x}_i$ is generated by $p_{\theta^{(t)}}$. Let M_α be the Markov transition kernel of finite steps of Langevin dynamics that samples π_α. Let $(M_\alpha p_\theta)(x) = \int M_\alpha(x', x) p_\theta(x') \mathrm{d}x'$ be the marginal distribution by running M_α initialized from p_θ. Then, similar to the traditional contrastive

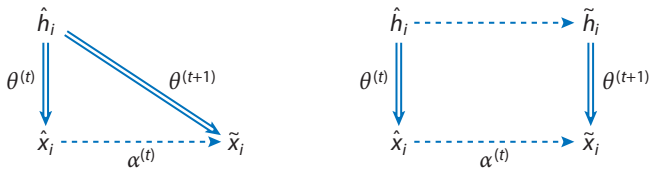

Figure 17

Diagrams of cooperative learning via MCMC teaching. The double-line arrows represent generation and reconstruction by the generator model, while the dashed-line arrows represent MCMC sampling and inference in the energy-based model and the generator model. θ is the parameters of the generator model, and α is the parameters of the energy-based model. $\hat{h}$ and $\hat{x}$ are a hidden vector and signal generated by the generator model, respectively. $\tilde{x}$ is the MCMC sample of the energy-based model, which is achieved by Langevin dynamic initialized from $\hat{x}$. $\tilde{h}$ is the MCMC inference of the generator model, which is achieved by Langevin dynamic initialized from $\hat{h}$. Abbreviation: MCMC, Markov chain Monte Carlo.

divergence, the learning gradient of the evaluator model α at iteration t is the gradient of $\mathrm{KL}(q_{\mathrm{data}} \parallel \pi_\alpha) - \mathrm{KL}(M_{\alpha^{(t)}} p_{\theta^{(t)}} \parallel \pi_\alpha)$ with respect to α. In the traditional contrastive divergence, q_{data} takes the place of $p_{\theta^{(t)}}$ in the second KL divergence.

2. MCMC teaching of the generator model: The learning gradient of the generator θ in the right panel of **Figure 17** is the gradient of $\mathrm{KL}(M_{\alpha^{(t)}} p_{\theta^{(t)}} \parallel p_\theta)$ with respect to θ. Here $\pi^{(t+1)} = M_{\alpha^{(t)}} p_{\theta^{(t)}}$ takes the place of q_{data} as the data to train the generator model. It is much easier to minimize $\mathrm{KL}(M_{\alpha^{(t)}} p_{\theta^{(t)}} \parallel p_\theta)$ than to minimize $\mathrm{KL}(q_{\mathrm{data}} \parallel p_\theta)$ because the latent variables are essentially known in the former, so the learning is supervised. The MCMC teaching alternates between Markov transition from $p_{\theta^{(t)}}$ to $\pi^{(t+1)}$, and projection from $\pi^{(t+1)}$ to $p_{\theta^{(t+1)}}$, as illustrated by **Figure 18**.

Figure 19 displays two examples of image synthesis by cooperative learning algorithm on data sets, LSUN bedrooms (Yu et al. 2015) and CelebA human faces (Liu et al. 2015).

6. LEARNING CONDITIONAL GENERATOR MODEL

The models and methods in the previous section can be easily generalized to conditional versions, which can be more useful in various applications.

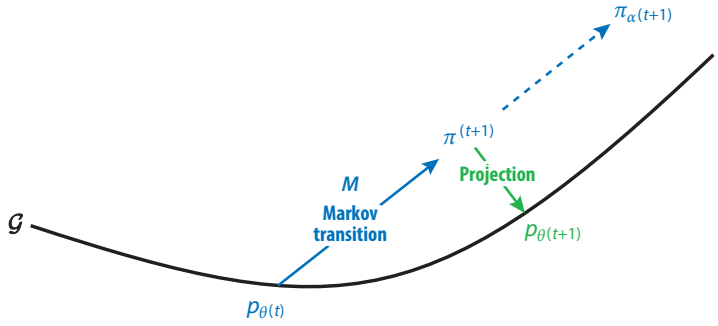

Figure 18

The MCMC teaching of the generator alternates between Markov transition and projection. The family of generator models $\mathcal{G}$ is illustrated by the black curve, and each distribution is illustrated by a point. p_θ is a generator model, and π_α is an energy-based model.

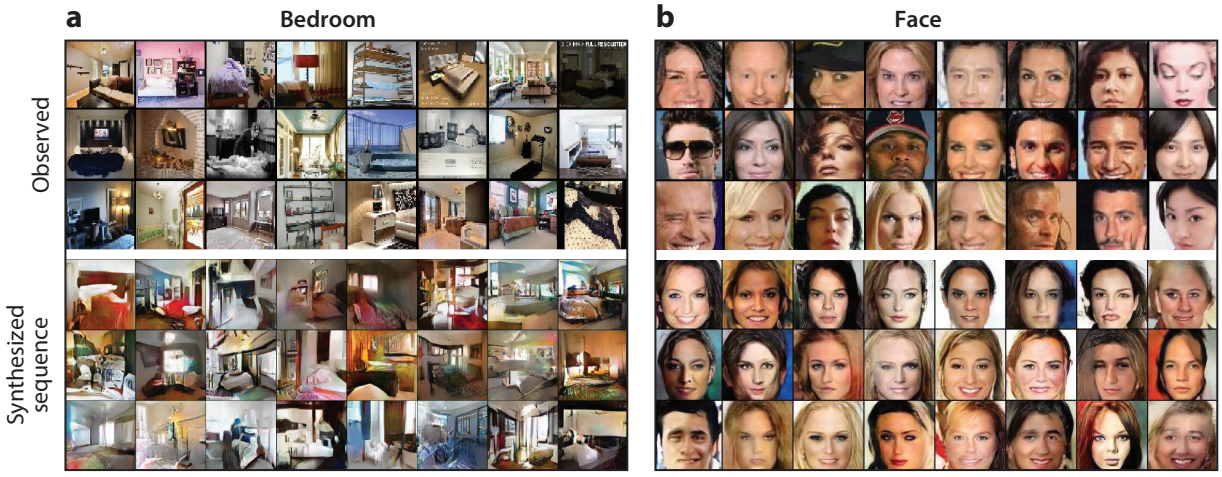

Figure 19

Image synthesis by cooperative learning. (*a*) Generating bedroom images (256 × 256 pixels). The synthesized images are generated by the cooperative learning algorithm that learns from the LSUN data set (Yu et al. 2015) with 3,033,000 training images. (*b*) Generating human face images (128 × 128 pixels). The synthesized images are generated by the cooperative learning algorithm that learns from the CelebA data set (Liu et al. 2015) with 200,000 training images. For each category, the top panel shows examples of the training images, and the bottom panel shows examples of the synthesized images generated by the learned models.

6.1. Conditional Generators, Conditional Variational Auto-Encoders, and Conditional Generative Adversarial Networks

The unconditioned generator model can be extended to a conditional model. Let x be the observed signal and c be the observed condition. For instance, x may be an image, and c may be a class label (e.g., cat or bird) or some text description (e.g., a bird is flying). The goal is to learn the conditional distribution $p_\theta(x|c)$ of the signal x given the condition c from the training data set of the pairs $\{(x_i, c_i), i = 1, \ldots, n\}$ that follow the data distribution $q_{\text{data}}(x, c)$. This is a supervised learning problem, except that x is a high-dimensional signal, and c may also be high dimensional.

The conditional generator model is of the following form:

$$h \sim \text{N}(0, I_d), \; x = g_\theta(h, c) + \varepsilon, \tag{27.}$$

where $g_\theta(h, c)$ is a top-down convolutional network (ConvNet) defined by the parameters θ. The ConvNet g maps the latent noise vector h together with the observed condition c to the signal x directly. Again, $\varepsilon \sim \text{N}(0, \sigma^2 I_D)$ is the residual noise signal that is independent of h. If c is the class label, it takes the form as a one-hot vector of label and is concatenated with h and fed into the decoder g. If the c is of high dimensionality, e.g., an image or text, we can parameterize g by an encoder-decoder structure: We first encode c into a latent vector z, and then we map the concatenation of h and z, i.e., (h, z), to x by a decoder. Given c, we can generate x from the conditional generator model by direct sampling—first, sampling h from its prior distribution, and then, mapping (h, c) into x directly.

The conditional generator model can be trained by maximum likelihood or, equivalently, minimizing the KL divergence $\text{KL}(q_{\text{data}}(x|c) \| p_\theta(x|c))$ over θ. The gradient of the conditional log-likelihood is computed by

$$-\frac{\partial}{\partial \theta} \text{KL}(q_{\text{data}}(x|c) \| p_\theta(x|c)) = \text{E}_{q_{\text{data}}(x,c) p_\theta(h|x,c)} \left[\frac{\partial}{\partial \theta} \log p_\theta(h, x|c) \right], \tag{28.}$$

where the expectation with respect to the conditional posterior distribution $p_\theta(h|x, c)$ can be approximated via MCMC sampling of $p_\theta(h|x, c)$.

Conditional VAEs (Sohn et al. 2015) train the conditional generator model by learning a tractable conditional inference model $q_\phi(h|x, c)$ to approximate the true conditional posterior distribution $p_\theta(h|x, c)$ for the sake of getting around the MCMC sampling from $p_\theta(h|x, c)$. Its objective function is given by

$$\mathrm{KL}(q_{\mathrm{data}}(x|c)q_\phi(h|x, c)\|p_\theta(h, x|c)) = \mathrm{KL}(q_{\mathrm{data}}(x|c)\|p_\theta(x|c)) + \mathrm{KL}(q_\phi(h|x, c)\|p_\theta(h|x, c)). \quad 29.$$

The adversarial learning framework can also be used to train the conditional generator model, where both the generator and discriminator are conditioned on the same condition. The resulting model is called conditional GAN (Mirza & Osindero 2014), whose objective function of a two-player minimax game is

$$V(D, G) = \mathrm{E}_{q_{\mathrm{data}}}[\log D(x|c)] + \mathrm{E}_{h\sim p(h)}[\log(1 - D(G(h|c))]. \quad 30.$$

The conditional generator models have a wide variety of application scenarios in computer vision and graphics, such as synthesizing images from text description (Reed et al. 2016), image-to-image translation (Isola et al. 2018) including synthesizing photo images from label maps or edge maps, and video-to-video translation (Wang et al. 2018) including converting an input source video, e.g., a sequence of semantic segmentation masks, to a target realistic video.

6.2. Conditional Learning via Fast Thinking Initializer and Slow Thinking Solver

Recently, Xie et al. (2019c) extended the cooperative learning scheme to the conditional learning problem by jointly learning a conditional energy–based model and a conditional generator model. The conditional energy–based model is of the following form:

$$\pi_\alpha(x|c) = \frac{1}{Z(c, \alpha)} \exp[f_\alpha(x, c)], \quad 31.$$

where x is the input signal and c is the condition. $Z(c, \alpha)$ is the normalizing constant conditioned on c. $f_\alpha(x, c)$ can be defined by a bottom-up ConvNet where α collects all the weight and bias parameters. Fixing the condition c, $f_\alpha(x, c)$ defines the value of x for the condition c, and $-f_\alpha(x, c)$ defines the conditional energy function. $\pi_\alpha(x|c)$ is also a deep generalization of conditional random fields (Lafferty et al. 2001). Both the conditional generator model and the conditional energy–based model can be learned jointly by the cooperative learning scheme in Section 5.6.

Figure 20 shows some examples of learning the conditional distribution of an image given a class label. The two models are jointly learned on 30,000 handwritten digit images from the MNIST database (LeCun et al. 1998) conditioned on their class labels, which are encoded as one-hot vectors. For each class, 10 randomly sampled images are displayed. Each column is conditioned on one label and each row is a different generated sample.

Figure 21 shows some examples of pattern completion on the CMP (Center for Machine Perception) Facade data set (Tyleček & Šára 2013) by learning a mapping from an occluded image (256×256 pixels), where a mask of the size of 128×128 pixels is centrally placed onto the original version, to the original image. In this case, c is the observed part of the signal, and x is the unobserved part of the signal.

Figure 20

Generated handwritten digits conditioned on class labels. Each column is conditioned on one class label, and each row represents a different generated handwritten digit image. The synthesized images are generated by the jointly trained initializer and solver from 30,000 handwritten digit images along with their class labels from the MNIST database. The image size is 64 × 64 pixels. Abbreviation: MNIST, Modified National Institute of Standards and Technology.

The cooperative learning of the conditional generator model and conditional energy–based model can be interpreted as follows. The conditional energy function defines the objective function or value function, i.e., it defines what solutions are desirable given the condition or the problem. The solutions can then be obtained by an iterative optimization or sampling algorithm such

Figure 21

Pattern completion by conditional learning. Each row displays one example. The first image is the testing image (256 × 256 pixels) with a hole of 128 × 128 that needs to be recovered, the second image shows the ground truth, the third image shows the result recovered by the initializer (i.e., the conditional generator model), the fourth image shows the result recovered by the solver (i.e., the Markov chain Monte Carlo sampler of the conditional energy–based model, initialized from the result of the initializer), and the last image shows the result recovered by the conditional GAN as a comparison. Abbreviation: GAN, generative adversarial networks.

as MCMC. In other words, the conditional energy–based model leads to a solver in the form of an iterative algorithm, and this iterative algorithm is a slow thinking process. In contrast, the conditional generator model defines a direct mapping from condition or problem to solutions, and it is a fast thinking process. We can use the fast thinking generator as an initializer to generate the initial solution, and then use the slow thinking solver to refine the fast thinking initialization by the iterative algorithm. The cooperative learning scheme enables us to learn both the fast thinking initializer and slow thinking solver. Unlike conditional GAN, the cooperative learning scheme has a slow thinking refining process, which can be important if the fast thinking initializer is not optimal.

In terms of inverse reinforcement learning (Abbeel & Ng 2004, Ziebart et al. 2008), the conditional energy–based model defines the reward or value function, and the iterative solver defines an optimal control or planning algorithm. The conditional generator model defines a policy. The fast thinking policy is about habitual, reflexive, or impulsive behaviors, while the slow thinking solver is about deliberation and planning. Compared with the policy, the value is usually simpler and more generalizable, because it is in general easier to specify what one wants than to specify how to produce what one wants.

7. CONCLUSIONS

This article reviews recent work on learning representations from a statistical perspective. We focus on unsupervised learning from unlabeled data. The representations can be either generative, like factor analysis, or relative, like multidimensional scaling.

A generative representation is a latent variable model. In this article, we focus on learning the model with a hidden vector at the top layer, and the hidden vector generates the signal via a linear or nonlinear transformation. Such a model can and should be extended to multiple layers of hidden vectors, or a hierarchical or graphical model (Lee et al. 2009, Salakhutdinov & Hinton 2009). While statisticians tend to learn such models by maximum likelihood or Bayesian methods, with the help of MCMC, researchers in deep learning prefer to learn such models by variational approximations or adversarial training. It is our hope that this article explains the latter methods and connects them to more traditional statistical methods.

A relative representation seeks to preserve important relations in the original observations. Such representations can be useful for exploratory data analysis or visualization. In relative representations, matrix representations can be used to represent the relations. For modeling dynamic systems, we can use vectors to represent the states and matrices to represent the changes of states caused by motions and actions.

Comparing vector representations and matrix representations, the latter are much less studied than the former, but the brain appears to need both for representing sensory data—vector representations are nouns and matrix representations are verbs. From a philosophical point of view, the brain only has access to the sensory data (including external and internal sensory data), and our notions of the outside world are the vector and matrix representations that the brain invents to explain the sensory data. In a sense, only data are real, and the outside world as we see it is more imaginary than real.

DISCLOSURE STATEMENT

The authors are not aware of any affiliations, memberships, funding, or financial holdings that might be perceived as affecting the objectivity of this review.

ACKNOWLEDGMENTS

This work is supported by DARPA XAI project N66001-17-2-4029, ARO project W911NF1810296, ONR MURI project N00014-16-1-2007, and Extreme Science and Engineering Discovery Environment (XSEDE) grant ASC170063. We thank Prof. Stu Geman, Prof. Xianfeng (David) Gu, Prof. Yali Amit, Prof. Jun Zhang, and Prof. Chao Gao for helpful discussions.

LITERATURE CITED

Abbeel P, Ng AY. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning*, ed. C Brodley, pp. 1–8. New York: ACM

Arjovsky M, Chintala S, Bottou L. 2017. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning*, ed. D Precup, YW Teh, pp. 214–23. New York: ACM

Bell AJ, Sejnowski TJ. 1997. The independent components of natural scenes are edge filters. *Vis. Res.* 37:3327–38

Bengio Y, Delalleau O, Roux NL, Paiement JF, Vincent P, Ouimet M. 2004. Learning eigenfunctions links spectral embedding and kernel PCA. *Neural Comput*. 16:2197–219

Blei DM, Kucukelbir A, McAuliffe JD. 2017. Variational inference: a review for statisticians. *J. Am. Stat. Assoc.* 112:859–77

Breiman L. 2017. *Classification and Regression Trees*. London: Routledge

Dai J, Lu Y, Wu YN. 2014. Generative modeling of convolutional neural networks. arXiv:1412.6296 [cs.CV]

Dai Z, Almahairi A, Bachman P, Hovy E, Courville A. 2017. Calibrating energy-based generative adversarial networks. arXiv:1702.01691 [cs.LG]

Dempster AP, Laird NM, Rubin DB. 1977. Maximum likelihood from incomplete data via the EM algorithm. *J. R. Stat. Soc. B* 39:1–38

Devlin J, Chang MW, Lee K, Toutanova K. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805 [cs.CL]

Dinh L, Krueger D, Bengio Y. 2014. NICE: non-linear independent components estimation. arXiv:1410.8516 [cs.LG]

Dinh L, Sohl-Dickstein J, Bengio S. 2017. Density estimation using Real NVP. arXiv:1605.08803 [cs.LG]

Dornhoff LL. 1972. *Group Representation Theory: Modular Representation Theory*. New York: Dekker

Friedman JH. 1991. Multivariate adaptive regression splines. *Ann. Stat.* 19:1–67

Gao R, Lu Y, Zhou J, Zhu SC, Wu YN. 2018a. Learning generative ConvNets via multi-grid modeling and sampling. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9155–64. Red Hook, NY: Curran

Gao R, Xie J, Zhu SC, Wu YN. 2018b. Learning grid cells as vector representation of self-position coupled with matrix representation of self-motion. arXiv:1810.05597 [stat.ML]

Gao R, Xie J, Zhu SC, Wu YN. 2019. Learning V1 simple cells with vector representations of local contents and matrix representations of local motions. arXiv:1902.03871 [cs.NE]

Gómez-Bombarelli R, Wei JN, Duvenaud D, Hernández-Lobato JM, Sánchez-Lengeling B, et al. 2018. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Sci*. 4:268–76

Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, et al. 2014. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, ed. Z Ghahramani, M Welling, C Cortes, ND Lawrence, KQ Weinberger, pp. 2672–80. San Diego, CA: NeurIPS

Grathwohl W, Chen RT, Betterncourt J, Sutskever I, Duvenaud D. 2019. Ffjord: Free-form continuous dynamics for scalable reversible generative models. arXiv:1810.01367 [cs.LG]

Hafting T, Fyhn M, Molden S, Moser MB, Moser EI. 2005. Microstructure of a spatial map in the entorhinal cortex. *Nature* 436:801–6

Hamilton WL, Ying R, Leskovec J. 2017. Representation learning on graphs: methods and applications. *IEEE Data Eng. Bull.* 40:52–74

Han T, Lu Y, Zhu SC, Wu YN. 2017. Alternating back-propagation for generator network. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ed. S Singh, S Markovitch, pp. 1976–84. Palo Alto, CA: AAAI

Han T, Nijkamp E, Fang X, Hill M, Zhu SC, Wu YN. 2019. Divergence triangle for joint training of generator model, energy-based model, and inference model. arXiv:1812.10907 [stat.ML]

Hinton GE. 2002. Training products of experts by minimizing contrastive divergence. *Neural Comput*. 14:1771–800

Hinton GE. 2012. A practical guide to training restricted Boltzmann machines. In *Neural Networks: Tricks of the Trade*, ed. G Montavon, GB Orr, KR Müller, pp. 599–619. New York: Springer

Hinton GE, Dayan P, Frey BJ, Neal RM. 1995. The "wake-sleep" algorithm for unsupervised neural networks. *Science* 268:1158–61

Hochreiter S, Schmidhuber J. 1997. Long short-term memory. *Neural Comput*. 9:1735–80

Hubel DH, Wiesel TN. 1959. Receptive fields of single neurones in the cat's striate cortex. *J. Physiol.* 148:574–91

Hyvärinen A, Karhunen J, Oja E. 2004. *Independent Component Analysis*. New York: Wiley

Isola P, Zhu JY, Zhou T, Efros AA. 2018. Image-to-image translation with conditional adversarial networks. arXiv:1611.07004 [cs.CV]

Jin L, Lazarow J, Tu Z. 2017. Introspective classification with convolutional nets. In *Advances in Neural Information Processing Systems 30*, ed. I Guyon, UV Luxburg, S Bengio, H Wallach, R Fergus, et al., pp. 823–33. San Diego, CA: NeurIPS

Jordan MI, Ghahramani Z, Jaakkola TS, Saul LK. 1999. An introduction to variational methods for graphical models. *Mach. Learn.* 37:183–233

Karras T, Aila T, Laine S, Lehtinen J. 2017. Progressive growing of GANs for improved quality, stability, and variation. arXiv:1710.10196 [cs.NE]

Kim T, Bengio Y. 2016. Deep directed generative models with energy-based probability estimation. arXiv:1606.03439 [cs.LG]

Kingma DP, Dhariwal P. 2018. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems 31*, ed. S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, R Garnett, pp. 10215–24. San Diego, CA: NeurIPS

Kingma DP, Welling M. 2014. Auto-encoding variational Bayes. arXiv:1312.6114 [stat.ML]

Kipf TN, Welling M. 2016. Semi-supervised classification with graph convolutional networks. arXiv:1609.02907 [cs.LG]

Koren Y, Bell R, Volinsky C. 2009. Matrix factorization techniques for recommender systems. *Computer* 42:30–37

Krizhevsky A, Sutskever I, Hinton GE. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, ed. F Pereira, CJC Burges, L Bottou, KQ Weinberger, pp. 1097–105. San Diego, CA: NeurIPS

Kruskal JB. 1964. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika* 29:1–27

Lafferty J, McCallum A, Pereira FC. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, ed. CE Brodley, AP Danyluk, pp. 282–89. New York: ACM

Lazarow J, Jin L, Tu Z. 2017. Introspective neural networks for generative modeling. In *IEEE International Conference on Computer Vision (ICCV)*, pp. 2774–83. New York: IEEE

LeCun Y, Bottou L, Bengio Y, Haffner P. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86:2278–324

Lee DD, Seung HS. 2001. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems 13*, ed. TK Leen, TG Dietterich, V Tresp, pp. 556–62. San Diego, CA: NeurIPS

Lee H, Grosse R, Ranganath R, Ng AY. 2009. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ed. A Danyluk, L Bottou, M Littman, pp. 609–16. New York: ACM

Lee K, Xu W, Fan F, Tu Z. 2018. Wasserstein introspective neural networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3702–11. New York: IEEE

Li KC. 1991. Sliced inverse regression for dimension reduction. *J. Am. Stat. Assoc.* 86:316–27

Liu Z, Luo P, Wang X, Tang X. 2015. Deep learning face attributes in the wild. In *International Conference on Computer Vision*, pp. 3730–38. New York: IEEE

Lu Y, Zhu SC, Wu YN. 2016. Learning FRAME models using CNN filters. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 1902–10. Palo Alto, CA: AAAI

Mikolov T, Chen K, Corrado G, Dean J. 2013. Efficient estimation of word representations in vector space. arXiv:1301.3781 [cs.CL]

Mirza M, Osindero S. 2014. Conditional generative adversarial nets. arXiv:1411.1784 [cs.LG]

Mnih A, Gregor K. 2014. Neural variational inference and learning in belief networks. In *Proceedings of the 31st International Conference on International Conference on Machine Learning*, ed. EP Xing, T Jebara, pp. 1791–99. New York: ACM

Neal RM. 2011. MCMC using Hamiltonian dynamics. In *Handbook of Markov Chain Monte Carlo*, ed. S Brooks, A Gelman, GL Jones, XL Meng, pp. 113–162. Boca Raton, FL: Chapman & Hall/CRC

Ngiam J, Chen Z, Koh PW, Ng AY. 2011. Learning deep energy models. In *Proceedings of the 28th International Conference on Machine Learning*, ed. L Getoor, T Scheffer, pp. 1105–12. New York: ACM

Nijkamp E, Zhu SC, Wu YN. 2019. On learning non-convergent short-run MCMC toward energy-based model. arXiv:1904.09770 [stat.ML]

O'Keefe J. 1979. A review of the hippocampal place cells. *Prog. Neurobiol.* 13:419–39

Olshausen BA, Field DJ. 1997. Sparse coding with an overcomplete basis set: a strategy employed by V1? *Vis. Res.* 37:3311–25

Paccanaro A, Hinton GE. 2001. Learning distributed representations of concepts using linear relational embedding. *IEEE Trans. Knowl. Data Eng.* 13:232–44

Pennington J, Socher R, Manning C. 2014. Glove: Global vectors for word representation. In *Conference on Empirical Methods in Natural Language Processing*, pp. 1532–43. Stroudsburg, PA: ACL

Radford A, Metz L, Chintala S. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv:1511.06434 [cs.LG]

Radford A, Narasimhan K, Salimans T, Sutskever I. 2018. *Improving language understanding by generative pre-training*. Tech. Rep., OpenAI, San Francisco. **https://s3-us-west-2.amazonaws.com/openai-assets/researchcovers/languageunsupervised/languageunderstandingpaper.pdf**

Reed S, Akata Z, Yan X, Logeswaran L, Schiele B, Lee H. 2016. Generative adversarial text to image synthesis. arXiv:1605.05396 [cs.NE]

Rezende DJ, Mohamed S. 2015. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning*, ed. F Bach, D Blei, pp. 1530–38. New York: ACM

Rezende DJ, Mohamed S, Wierstra D. 2014. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning*, ed. EP Xing, T Jebara, pp. 1278–86. New York: ACM

Roweis ST, Saul LK. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science* 290:2323–26

Rubin DB, Thayer DT. 1982. EM algorithms for ML factor analysis. *Psychometrika* 47:69–76

Salakhutdinov R, Hinton GE. 2009. Deep Boltzmann machines. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, ed. D van Dyk, M Welling, pp. 448–55. Brookline, MA: Microtome

Sohn K, Lee H, Yan X. 2015. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems 28*, ed. C Cortes, ND Lawrence, DD Lee, M Sugiyama, R Garnett, pp. 3483–91. San Diego, CA: NeurIPS

Sutton RS, Barto AG. 1998. *Introduction to Reinforcement Learning*. Cambridge, MA: MIT Press

Tibshirani R. 1996. Regression shrinkage and selection via the lasso. *J. R. Stat. Soc. B* 58:267–88

Tu Z. 2007. Learning generative models via discriminative approaches. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8. New York: IEEE

Tyleček R, Šára R. 2013. Spatial pattern templates for recognition of objects with regular structure. In *Pattern Recognition: 35th German Conference, GCPR 2013, Saarbrücken, Germany, September 3–6, 2013, Proceedings*, ed. J Weickert, M Hein, B Schiele, pp. 364–74. New York: Springer

van der Maaten L, Hinton G. 2008. Visualizing data using t-SNE. *J. Mach. Learn. Res.* 9:2579–605

Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, et al. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, ed. I Guyon, UV Luxburg, S Bengio, H Wallach, R Fergus, et al., pp. 5998–6008. San Diego, CA: NeurIPS

Wang TC, Liu MY, Zhu JY, Liu G, Tao A, et al. 2018. Video-to-video synthesis. In *Advances in Neural Information Processing Systems 31*, ed. S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, R Garnett, pp. 1144–56. San Diego, CA: NeurIPS

Wu YN, Gao R, Han T, Zhu SC. 2019. A tale of three probabilistic families: discriminative, descriptive and generative models. *Q. Appl. Math.* 77:423–65

Xie J, Gao R, Zheng Z, Zhu SC, Wu YN. 2019a. Learning dynamic generator model by alternating back-propagation through time. In *The Thirty-Third AAAI Conference on Artificial Intelligence*, pp. 5490–507. Palo Alto, CA: AAAI

Xie J, Lu Y, Gao R, Wu YN. 2018a. Cooperative learning of energy-based model and latent variable model via MCMC teaching. In *The Thirty-Second AAAI Conference on Artificial Intelligence*, pp. 4292–301. Palo Alto, CA: AAAI

Xie J, Lu Y, Gao R, Zhu SC, Wu YN. 2019b. Cooperative training of descriptor and generator networks. *IEEE Trans. Pattern Anal. Mach. Intel.* In press. **https://dx.doi.org/10.1109/TPAMI.2018.2879081**

Xie J, Lu Y, Zhu SC, Wu YN. 2016. A theory of generative ConvNet. In *Proceedings of the 33rd International Conference on Machine Learning*, ed. MF Balcan, KQ Weinberger, pp. 2635–44. New York: ACM

Xie J, Zheng Z, Fang X, Zhu SC, Wu YN. 2019c. Multimodal conditional learning with fast thinking policy-like model and slow thinking planner-like model. arXiv:1902.02812 [stat.ML]

Xie J, Zheng Z, Gao R, Wang W, Zhu SC, Wu YN. 2018b. Learning descriptor networks for 3D shape synthesis and analysis. arXiv:1804.00586 [cs.CV]

Xie J, Zhu SC, Wu YN. 2017. Synthesizing dynamic patterns by spatial-temporal generative ConvNet. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7093–101. New York: IEEE

Xing X, Han T, Gao R, Zhu SC, Wu YN. 2019. Unsupervised disentangling of appearance and geometry by deformable generator network. arXiv:1806.06298 [cs.LG]

Yu F, Seff A, Zhang Y, Song S, Funkhouser T, Xiao J. 2015. LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop. arXiv:1506.03365 [cs.CV]

Zee A. 2016. *Group Theory in a Nutshell for Physicists*. Princeton, NJ: Princeton Univ. Press

Ziebart BD, Maas AL, Bagnell JA, Dey AK. 2008. Maximum entropy inverse reinforcement learning. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pp. 1433–38. Palo Alto, CA: AAAI