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, [2]UCLA Department of Statistics

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 multi-dimensional 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 shall review recent advances in learning representations from a statistical perspective. In particular, we will review the following two themes. (1) Unsupervised learning of vector representations. (2) Learning of both vector and matrix representations.

Key words: unsupervised learning, generative representations, relative representations, predictive representations, vector representations, 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 shall 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, it's essence can be traced back to familiar examples in statistics.

1.1.1 Factor analysis — generative representation

One prototypical example of learning representations in statistics is factor analysis (Rubin and 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 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* of x, which is also called a *code*. The mapping from h to x is called a *decoder*, while the mapping from x to h is called an *encoder*. 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 and Field, 1997), non-negative matrix factorization (Lee and Seung, 2001), recommender system (Koren et al., 2009), restricted Boltzmann machine (Hinton, 2012), etc., by modifying the prior distribution or prior assumption on h. If we generalize the linear mapping from h to x to a non-linear mapping parametrized by a deep network (LeCun et al., 1998;

Krizhevsky et al., 2012), then the resulting model becomes what is commonly called *generator network* (Goodfellow et al., 2014; Kingma and Welling, 2014).

Factor analysis is an example of what we may call "*generative representation*", where the hidden vector h generates the observed vector x.

1.1.2 Multi-dimensional scaling — relative representation

The other prototypical example of learning representation in statistics is multi-dimensional scaling (Kruskal, 1964). Let $(x_i, i = 1, ..., n)$ be a set of D-dimensional observations. We want to represent them by a corresponding set of d-dimensional hidden vectors $(h_i, i = 1, ..., n)$, 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 factor analysis, there is no explicit mapping (encoding or decoding) between h and x.

Various modifications of multi-dimensional scaling focus on preserving local adjacency or neighborhood relations between (x_i), such as spectral embedding (Bengio et al., 2004), tSNE (t stochastic neighbor embedding) (Maaten and Hinton, 2008), local linear embedding (Roweis and Saul, 2000), etc.

Multi-dimensional scaling is an example of what we may call "*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 non-linear regression model from the training examples $\{(x_i, y_i)\}$, where x_i is D-dimensional continuous predictor vector, and y_i is 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 iid noises.

Assuming $(x_i, y_i) \sim p(x, y)$, and assuming $\mathrm{E}(x) = 0$, 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 non-linear link function f. We may call $h = Wx$ as encoding, and $y = f(h, \varepsilon)$ as decoding.

Sliced inverse regression is an example of what we may call "*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 multi-dimensional 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 paper, we shall focus on unsupervised learning.

Another learning problem that lies in between supervised and unsupervised learning is reinforcement learning (Sutton and 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.

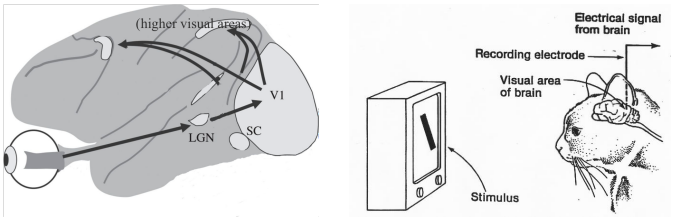
1.3 Plan for the remainder of the paper

Section 2 reviews vector representations based on linear models. We first review 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 review continuous vector representations of discrete data, in predictive and relative representation schemes. Section 3 reviews the learning of both vector and matrix representations in a relative representation scheme. Section 4 reviews the learning of non-linear vector representation based on the generator model, which generalizes linear mapping in the factor analysis model to non-linear mapping parameterized by deep neural network. 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 shall review learning vector representations of data using models that generalize the factor analysis model.

2.1 Sparse vector representation



(a) (b)

Figure 1: (a) Primary visual cortex or V1 is the first step in representing retina image data. (b) Cells in V1 respond to bars of different locations, orientations and sizes.

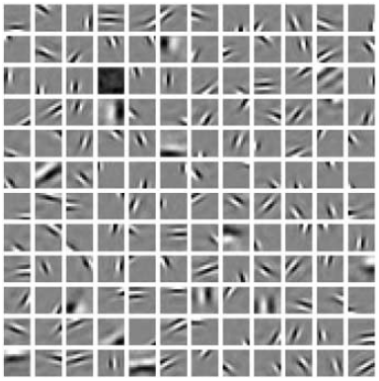


Figure 2: Olshausen-Field sparse coding model. The plot displays the 144 learned basis vectors, each displayed as an image patch (orders of the patches carry no meaning). These basis vectors represent local image structures such as edges and bars. The training data were obtained by extracting 12×12 image patches at random from ten 512×512 images of natural scenes (trees, rocks, mountains etc.).

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 and Wiesel, 1959). Figure 1 (a) illustrates the V1 area. Hubel and Wiesel 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. See Figure 1 (b). Some V1 cells are called simple cells, which behave like linear wavelets. A mathematical model of a simple cell is Gabor wavelet, which is sine or cosine plane wave multiplied by an elongate Gaussian function.

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

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

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

Unlike factor analysis, in the sparse coding model, the dimensionality of h, i.e., d, is assumed to be larger than the dimensionality of x, which is D, 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 non-zero or significantly different from zero. For different x, the non-zero elements of h can be different. Thus unlike principal component analysis, sparse coding leads to adaptive dimension reduction. $W = (W_k, k = 1, ..., d)$ 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, $\{x_i, i = 1, ..., n\}$, where each x_i is a training example of image patch. Each x_i is represented by an $h_i = (h_{ik}, k = 1, ..., d)$, 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

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

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 the Lasso, where (W_k) serve as variables or regressors. Compared to the 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 and 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 non-negative matrix factorization (Lee and Seung, 2001), which assumes $h_k \geq 0$, and restricted Boltzmann machine (Hinton, 2012), which assumes 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 visualization of x if h is 2-dimensional ($d = 2$). The input x is usually continuous.

The input x can also be discrete, such as 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 j-th word in the dictionary, then x is a D-dimensional vector so that the j-th element of x is 1 and all the other elements are 0's. We represent x by 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 j-th word is represented by the j-th 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 i-th column of W, i.e., the vector representation of word i in the encoding pass, and $\tilde{W}_j$ is the j-th 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 system (Koren et al., 2009). Let X_{ij} be the rating of user i on item j, 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, and $\tilde{W}_j$ 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 i-th column is W_i, and $\tilde{W}$ is the $d \times D$ matrix whose j-th column is $\tilde{W}_j$.

For discrete x such as a word, the vector representation h is continuous, dense, and distributed, where each component of h captures partial semantic meaning of x. Such dense vector representations have revolutionized the 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).

The vector representation has also been applied to encode the nodes in graphs (Hamilton et al., 2017), which can be conveniently used for subsequent analysis (Kipf and 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 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 multi-dimensional scaling. The matrices represent the relations between the vectors, and can be part of a relative representation. An early example is (Paccanaro and Hinton, 2001).

In terms of 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

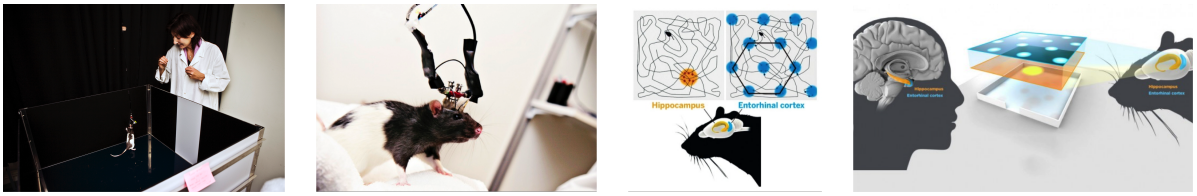
like "nouns", while the matrix representations are like "verbs" that transform the "nouns".

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. Purely based on the movements or self-motion, you know the 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



(a) (b) (c) (d)

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.

Figure 3 (a) depicts Dr. May-Britt Moser, who together with Dr. Edvard Moser, won the 2014 Nobel Prize for Physiology or Medicine, for their discovery of the grid cells (Hafting et al., 2005) in 2005. Their thesis advisor, Dr. John O'keefe, shared the prize for his discovery of the place cells (O'Keefe, 1979). Both the 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. See 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 cover the whole square region. The discovery of grid cells was much more surprising and unexpected. The grid cells exist in the Entorphinal cortex. Each grid cell fires at multiple locations, and these locations form a regular hexagon grid. See Figure 3 (c). The grid cells have been identified across many mammalian species, including human. See Figure 3 (d).

3.1.2 A simple addition problem

There are two problems in navigation. One is 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, then you can 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 $(\Delta x_t, t = 1, ..., T)$, we want to calculate the positions over time with $x_t = x_{t-1} + \Delta x_t$ for $t = 1, ..., T$. The path planning problem is

such that given the starting position x and the target position y, we want to plan a sequence of displacements $(\Delta x_t, t = 1, ..., T)$, 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 this 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, for any 2D position $x \in D$, we represent it by a d-dimensional vector $h(x)$. Suppose at a position x, the self-motion or displacement is Δx, so that the agent moves to $x + \Delta x$ after one step. We assume the following motion model:

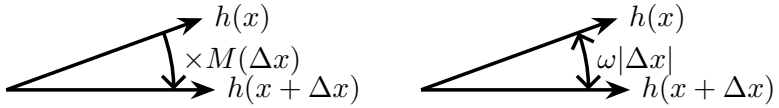
$$h(x + \Delta x) = M(\Delta x)h(x), \tag{3}$$

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 will 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 a rotation in the d-dimensional sphere. We can illustrate the motion model by the following diagram:

$$\begin{array}{ccc} x_t & \xrightarrow{+\Delta x} & x_{t+1} \\ \downarrow & \downarrow & \downarrow \\ h(x_t) & \xrightarrow{M(\Delta x)\times} & h(x_{t+1}) \end{array} \tag{4}$$

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. See Figure 4 (a) for an illustration. See also (Paccanaro and Hinton, 2001) for an earlier treatment of the addition problem.



(a) Vector-matrix multiplication (b) Magnified local isometry

Figure 4: Grid cells form a high-dimensional vector representation of 2D self-position. Two sub-models: (a) Local motion is modeled by vector-matrix multiplication. (b) Angle between two nearby vectors magnifies the Euclidean distance.

3.1.4 Error correction

In data visualization such as t-SNE (Maaten and Hinton, 2008), we represent high dimensional data by 2D points. In grid cells, we do the opposite, where we represent 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 sub-manifold of $(h(x), x \in [0,1]^2)$, we can eliminate most of the noises.

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:

$$\langle h(x), h(x+\Delta x)\rangle = 1 - \alpha|\Delta x|^2, \tag{5}$$

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 $h(x+\Delta x)$, 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 4 (b) for an illustration.

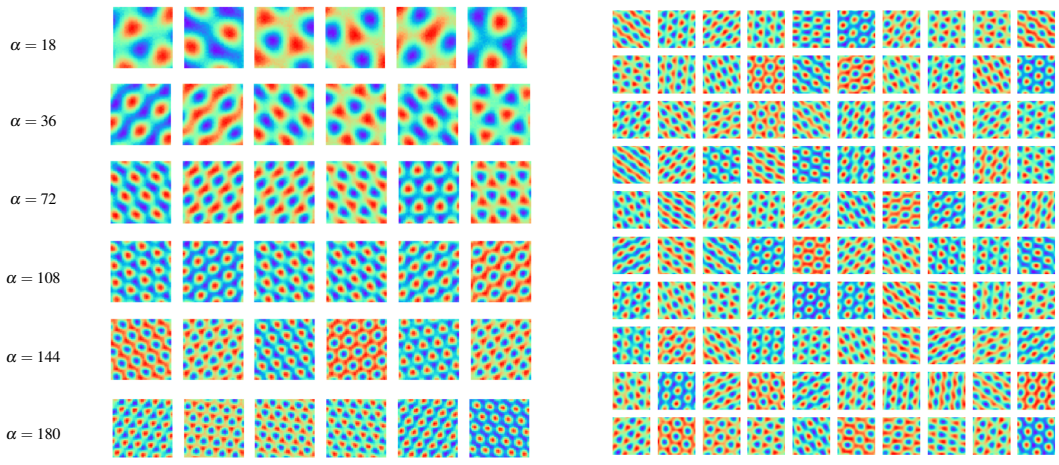
Gao et al. (2018b) showed that even if they randomly shut down (i.e., set to zero) 70% of the neurons in each step, their learned system can still perform path integral accurately. Such dropout error may occur due to internal noises, asynchrony of neuron activities, as well as aging and diseases like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

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:

$$\mathrm{E}_{x,\Delta x}[\|h(x+\Delta x) - M(\Delta x)h(x)\|^2] + \lambda \mathrm{E}_{x,\Delta x}[(\langle h(x), h(x+\Delta x)\rangle - (1 - \alpha|\Delta x|^2))^2]. \tag{6}$$

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.



(a) Learned $h(x)$ with 6 units. (b) Learned $h(x)$ with 100 units

Figure 5: 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 $\alpha = 72$.

The experiments of Gao et al. (2018b) show that as long as the dimension of $h(x)$, $d \geq 6$, 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 5 (a), each row displays the learned $h(x)$ for a given value of α, where $d = 6$. Figure 5 (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. Thus one may 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.

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 vector and represent the change of the state caused by motion or action by matrix. For example, recent work of (Gao et al., 2018c) proposes a model of V1 simple cells that is different from the sparse coding model (Olshausen and Field, 1997) and the independent component analysis model (Bell and Sejnowski, 1997) reviewed in Section 2.1. (Gao et al., 2018c) proposes 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., 2018c) represents 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 represent 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 be adopted by the 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 non-linear 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 parametrized by a deep network. We shall also review 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 non-linear 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 non-linear link functions.

Specifically, consider a non-linear 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 $l = 1, ..., L$, 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 non-linear link function or the so-called rectification function that is applied coordinate-wise. $f(x)$ is a recursive composition of GLM (generalized linear model) structures.

Modern deep networks usually use $r_l(s) = \max(0, s)$, the so-called rectified linear unit (ReLU). For such non-linear link function, $f(x)$ is a multivariate linear spline where the linear pieces are recursively partitioned. This is similar to but more general than the recursive partitions in CART (classification and regression trees) (Breiman, 2017) and MARS (multivariate adaptive regression splines) (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 and Schmidhuber, 1997), which are commonly applied to sequence data such as speech and nature language.

The 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 shall review in the next subsection, where h_l at multiple layers can be considered generative representations.

4.1.1 Non-linear generalization of logistic regression

For the deep network reviewed in the previous subsection, let $\alpha = (W_l, b_l, l = 1, ..., L)$ collect all the weight and bias parameters, and let $f_\alpha(x)$ be the resulting non-linear transformation.

We can generalize logistic regression model to

$$P(y = 1|x) = D(x) = \frac{1}{1 + \exp(-f_\alpha(x))}. \tag{7}$$

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

4.1.2 Non-linear generalization of exponential family model

We can also generalize exponential family model to

$$\pi_\alpha(x) = \frac{1}{Z(\alpha)} \exp(f_\alpha(x))\rho(x), \tag{8}$$

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

The connection between the two models are as follows. Suppose $\rho(x)$ is the distribution of negative examples, i.e., $P(x|y = 0) = \rho(x)$, and $\pi_\alpha(x)$ is the distribution of positive examples, i.e., $P(x|y = 1) = \pi_\alpha(x)$. Suppose there are equal numbers of positive and negative examples, then according to the Bayes rule, $P(y = 1|x)$ is given by (7).

We shall make use of the above two models later on as the complementary models to the generator model we shall review next.

4.2 Non-linear generalization of factor analysis and maximum likelihood learning

While sparse coding and independent component analysis etc. 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.,

$$h \sim \mathrm{N}(0, I_d),\ x = g_\theta(h) + \varepsilon, \tag{9}$$

where g is parametrized 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 $l = L-1, ..., 0$, 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. $\varepsilon \sim \mathrm{N}(0, \sigma^2 I_D)$ 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 $h_0 = x \to h_1 \to ... \to h_L$, g in this subsection is a top-down network in the sense that it defines $h_L = h \to h_{L-1} \to ... \to h_0 = x$.

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 $p_\theta(x) = \int p_\theta(h,x)dh$. 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 $x_i, i = 1, ..., n$. For large n, the maximum likelihood estimation of θ is to minimize the Kullback-Leibler 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

$$-\frac{\partial}{\partial \theta}\mathrm{KL}(q_{\text{data}}(x)\|p_\theta(x)) = \mathrm{E}_{q_{\text{data}}(x)p_\theta(h|x)}\left[\frac{\partial}{\partial \theta}\log p_\theta(h,x)\right]. \tag{10}$$

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

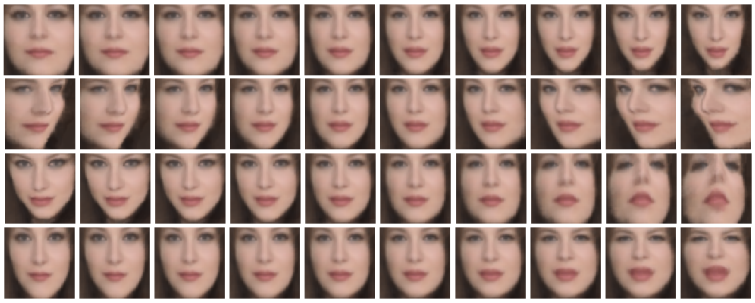


Figure 6: Each dimension of the geometric latent vector h_1 encodes geometric information such as shape and viewing angle. In the fist row, the shape of the face changes from fat to thin from left to right. In the second row, the pose of the face varies from left to right. In the third row, from left to right, the vertical tilt of the face varies from downward to upward. In the fourth row, the face width changes from stretched to cramped. The deformable generator model is trained on the 10,000 face images from CelebA dataset. The training images are cropped to 64×64 pixels. These faces have different colors, illuminations, identities, viewing angles, shapes, and expressions.

(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 CelebA dataset (Liu et al., 2015). Figure 6 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 7 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.

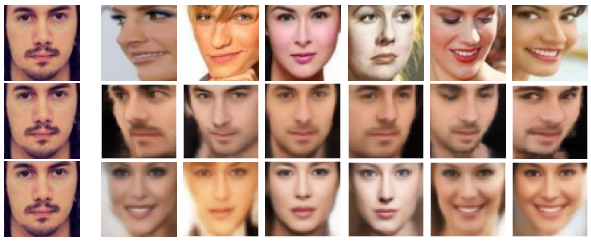


Figure 7: Transferring and recombining geometric and appearance vectors. The first row shows 7 faces from CelebA. The second row shows the generated faces by transferring and recombining the 2th-7th faces' geometric vectors h_1 with the first face's appearance vector h_2 in the first row. The third row shows the generated faces by transferring and recombining the 2th-7th 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, which are cropped to 64×64 pixels. The faces in the training data have a wide and diverse variety of colors, illuminations, identities, viewing angles, shapes, and expressions.

Figure 8: Generating dynamic textures. The dynamic generator model is learned from one single training video exhibiting burning flame, with the size 64 $\times$ 64 pixels $\times$ 60 frames, by maximum likelihood. A "longer length" dynamic texture can be generated from a relatively "short" training sequence by just drawing "longer" iid samples from 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 in length, which are generated by the learned model.

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

$$h_t = f_\alpha(h_{t-1}, z_t), \tag{11}$$

$$x_t = g_\beta(h_t) + \varepsilon_t, \tag{12}$$

where $t = 1, ..., T$. (11) is the transition model, and (12) is the emission model. h_t is the d-dimensional hidden state vector. $z_t \sim \mathrm{N}(0, I)$ is the noise vector of a certain dimensionality. The Gaussian noise vectors $(z_t, t = 1, ..., T)$ are independent of each other. The sequence of $(h_t, t = 1, ..., T)$ follows a non-linear auto-regressive model, where the noise vector z_t encodes the randomness in the transition from h_{t-1} to h_t in the

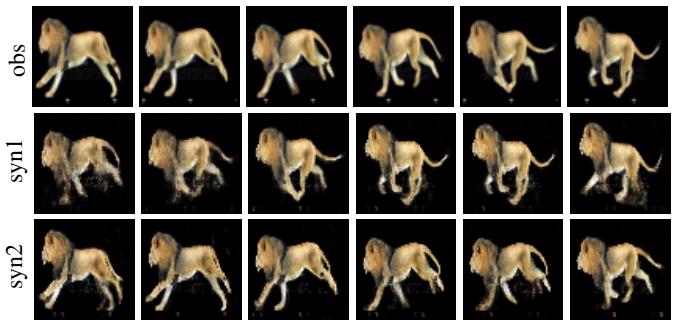


Figure 9: Generated action patterns. The dynamic generator model is trained on an animal action dataset including 20 videos of 10 animals performing running and walking. Each observed video is scaled to 64 $\times$ 64 pixels $\times$ 30 frames. The first row displays 6 frames of the observed sequence, and the second and third rows show the corresponding frames of two synthesized sequences generated by the learned model.

d-dimensional state space. f_α is a feedforward 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. $\varepsilon_t \sim \mathrm{N}(0, \sigma^2 I_D)$ is the residual error. The model is a state-space model or hidden Markov model. (Xie et al., 2019a) learned the dynmaic generator model by maximum likelihood. Figures 8 and 9 show examples of learning the model from video data. Once the model is learned, we can synthesize dynamic textures from the learned model by firstly randomly initializing the initial hidden state h_0, and then following (11) and (12) 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, 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.

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

5 Learning 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. (1) The learning algorithm requires MCMC sampling of the posterior distribution $p_\theta(h|x)$ as an inner loop, which can be expensive. (2) 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 variational auto-encoder (VAE) (Kingma and Welling, 2014; Rezende et al., 2014; Mnih and 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 which specifies the distribution of x explicitly up to a normalizing constant.

While the generator model is parametrized by a top-down network as show in the left diagram of (13), the complementary model is parameterized by a separate bottom-up network as shown in the right diagram of (13).

$$\begin{array}{cc} \text{Top-down mapping} & \text{Bottom-up mapping} \\ \text{hidden vector } h & \text{inference } q_\phi(h|x) \text{ or energy } f_\alpha(x) \\ \Downarrow & \Uparrow \\ \text{signal } x \approx g_\theta(h) & \text{signal } x \\ \text{(a) Generator model} & \text{(b) Complementary model} \end{array} \tag{13}$$

5.2 Variational auto-encoder: joint learning with inference model

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

$$q_\phi(h|x) \sim \mathrm{N}(\mu_\phi(x), \mathrm{diag}(v_\phi(x))), \tag{14}$$

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 reparametrization trick (Kingma and 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 MLE objective:

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

We define the conditional Kullback-Leibler 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

$$\min_\theta \min_\phi \mathrm{KL}(q_{\mathrm{data}}(x)q_\phi(h|x)\|p_\theta(h,x)), \tag{16}$$

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 $\min_\theta \min_\phi \mathrm{KL}(Q\|P)$.

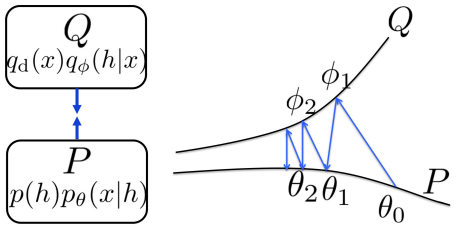


Figure 10: Variational auto-encoder as joint minimization by alternating projection.

We may interpret VAE as alternating projection between Q and P. See Figure 10 for illustration. The wake-sleep algorithm (Hinton et al., 1995) is similar to 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.

5.2.1 MLE algorithm from VAE perspective

Recall the MLE is to minimize $\mathrm{KL}(q_{\rm data}\|p_\theta)$. Suppose θ_t is the current estimate in the MLE algorithm. We can write

$$\mathrm{KL}(q_{\rm data}(x)p_{\theta_t}(h|x)\|p(h)p_\theta(x|h)) = \mathrm{KL}(q_{\rm data}(x)\|p_\theta(x)) + \mathrm{KL}(p_{\theta_t}(h|x)\|p_\theta(h|x)), \tag{17}$$

where we replace $q_\phi(h|x)$ in 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, $\mathrm{KL}(q_{\rm data}(x)p_{\theta_t}(h|x)\|p(h)p_\theta(x|h))$ majorizes $\mathrm{KL}(q_{\rm data}(x)\|p_\theta(x))$ as functions of θ, and both functions touch at θ. Thus minimizing the left-hand side will decrease $\mathrm{KL}(q_{\rm data}\|p_\theta)$, 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 (10).

5.2.2 Comparison with traditional variational inference

In VAE, the model $q_\phi(h|x)$ and the parameter ϕ is shared by all the training examples x, so that $\mu_\phi(x)$ and $v_\phi(x)$ in 14 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 VAE are to approximate the iterative minimization algorithm by direct mappings.

5.3 Generative adversarial net (GAN): joint learning with discriminator

The generator model learned by MLE or VAE usually cannot generate very realistic images. Both MLE and VAE target $\mathrm{KL}(q_{\rm data}\|p_\theta)$, though VAE only minimizes an upper bound of $\mathrm{KL}(q_{\rm data}\|p_\theta)$. 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 p if p cannot fit the modes of p closely. As a result, p_θ learned by MLE or 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 generative adversarial networks (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 parametrized by a bottom-up network $f_\alpha(x)$, so that $D(x) = 1/(1+\exp(-f_\alpha(x))$, i.e., logistic regression. See Subsection 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

$$V(D,G) = \mathrm{E}_{q_{\mathrm{data}}}[\log D(X)] + \mathrm{E}_{h\sim p(h)}[\log(1-D(G(h))], \tag{18}$$

where $\mathrm{E}_{q_{\mathrm{data}}}$ 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 $\mathrm{E}_{h\sim p(h)}[\log D(G(h))]$ 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 the Bayes rule $D(x) = q_{\mathrm{data}}(x)/(q_{\mathrm{data}}(x) + p_\theta(x))$ (assuming equal numbers of real and faked examples). Then θ minimizes the Jensen-Shannon divergence

$$\mathrm{JS}(q_{\mathrm{data}}\|p_\theta) = \mathrm{KL}(p_\theta\|p_{\mathrm{mix}}) + \mathrm{KL}(q_{\mathrm{data}}\|p_{\mathrm{mix}}), \tag{19}$$

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

In JS-divergence, the model p_θ also appears on the left side of KL-divergence. This encourages p_θ to fit some major modes of q_{data}, while ignoring others. As a result, the GAN learning suffers from 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 p_θ learned by MLE or 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, 2018c; 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 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

$$\pi_\alpha(x) = \frac{1}{Z(\alpha)} \exp\left[f_\alpha(x)\right], \tag{20}$$

where $f_\alpha(x)$ is parametrized by a bottom-up deep network with parameters α, and $Z(\alpha)$ is the normalizing constant. It is the non-linear generalization of the exponential family model, see subsection 4.1.2. 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 diagram (b) in (13).

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.

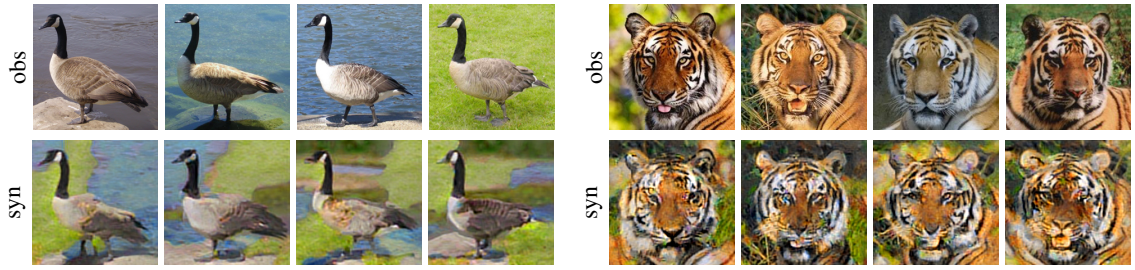
5.4.2 Maximum likelihood

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

$$-\frac{\partial}{\partial \alpha}\text{KL}(q_{\text{data}}(x)\|\pi_\alpha(x)) = \text{E}_{q_{\text{data}}}\left[\frac{\partial}{\partial \alpha} f_\alpha(x)\right] - \text{E}_{\pi_\alpha}\left[\frac{\partial}{\partial \alpha} f_\alpha(x)\right]. \tag{21}$$

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; Xie et al., 2016) learned the energy-based model using such a learning method. See Figure 11 for illustration.



(a) goose (b) tiger

Figure 11: Learning energy-based model by maximum likelihood. (a) goose (b) tiger. For each category, the first row displays 4 of the training images, and the second row displays 4 of the images generated by the learning algorithm. $f_\alpha(x)$ is parametrized by a 4-layer bottom-up deep network, where the first layer has 100 7×7 filters with sub-sampling size of 2, the second layer has 64 5×5 filters with subsampling size of 1, the third layer has 20 3×3 filters with sub-sampling size of 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. The number of Langevin iterations between every two consecutive updates of parameters is 10. The training images are of size 224×224 pixels.

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 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 the Bayes rule, see Subsection 4.1.2. See also (Dai et al., 2014; Wu et al., 2019). The model can be learned discriminatively by fitting a logistic regression model, see (Tu, 2007; Lazarow et al., 2017; Jin et al., 2017; Lee et al., 2018).

5.4.3 Adversarial contrastive divergence (ACD): 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 as 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, i.e., p_θ is an approximate direct sampler of π_α.

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

$$\min_\alpha \max_\theta [\mathrm{KL}(q_{\mathrm{data}} \| \pi_\alpha) - \mathrm{KL}(p_\theta \| \pi_\alpha)], \tag{22}$$

or equivalently

$$\max_\alpha \min_\theta [\mathrm{KL}(p_\theta \| \pi_\alpha) - \mathrm{KL}(q_{\mathrm{data}} \| \pi_\alpha)]. \tag{23}$$

The gradient for updating α becomes

$$\frac{\partial}{\partial \alpha} [\mathrm{E}_{q_{\mathrm{data}}}(f_\alpha(x)) - \mathrm{E}_{p_\theta}(f_\alpha(x))], \tag{24}$$

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

Because of the negative sign in front of the second KL-divergence in (22), we need $\max_\theta$ in (22) or $\min_\theta$ in (23), so that the learning becomes adversarial. See Figure 12 for illustration. Inspired by (Hinton, 2002), (Han et al., 2019) called (22) the adversarial contrastive divergence (ACD). It underlies (Kim and Bengio, 2016; Dai et al., 2017).

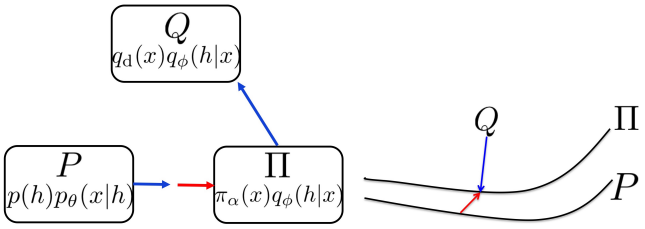


Figure 12: Adversarial contrastive divergence where the energy-based model favors real data against generator.

The adversarial form (22) or (23) 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 get away from p_θ. The red arrow in Figure 12 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, 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.

The above chasing game is different from 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 GAN, the discriminator eventually becomes a confused one 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 (24) bears similarity 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

$$\min_\theta \min_\phi \mathrm{KL}(p(h)p_\theta(x|h) \| \pi_\alpha(x) q_\phi(h|x)). \tag{25}$$

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

$$\mathrm{KL}(P\|\Pi) = \mathrm{KL}(p_\theta(x)\|\pi_\alpha(x)) + \mathrm{KL}(p_\theta(h|x)\|q_\phi(h|x)). \tag{26}$$

Thus, we can modify (23) 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 VAE (32), it is in the form of KL + KL, whereas in ACD (22), it is in the form of KL - KL. In both (32) and (22), the first KL is about maximum likelihood. The KL+KL form of 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 (22), the ACD objective function becomes an adversarial one or a minimax game.

Also note that in 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 MLE algorithm from ACD perspective

Recall 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

$$\mathrm{KL}(q_{\mathrm{data}}\|\pi_\alpha) - \mathrm{KL}(\pi_{\alpha_t}\|\pi_\alpha), \tag{27}$$

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. This leads to identity (21). For K-step MCMC in (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 studies the learned K-step MCMC as a model in itself.

5.5 Divergence triangle: VAE + ACD, joint learning of three models

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

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, q_ϕ by the following divergence triangle loss functional $\mathscr{D}$

$$\begin{aligned} &\max_\alpha \min_\theta \min_\phi \mathscr{D}(\alpha,\theta,\phi), \\ &\mathscr{D} = \mathrm{KL}(Q\|P) + \mathrm{KL}(P\|\Pi) - \mathrm{KL}(Q\|\Pi). \end{aligned} \tag{28}$$

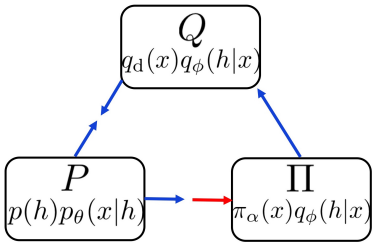


Figure 13: Divergence triangle is based on the Kullback-Leibler divergences between three joint distributions of (h,x). The blue arrow indicates the "running toward" behavior and the red arrow indicates the "running away" behavior.

See Figure 13 for illustration. The divergence triangle is based on the three KL-divergences between the three joint distributions on (h,x). It has a symmetric and anti-symmetric form, where the anti-symmetry is due to the negative sign in front of the last KL-divergence and the maximization over α. Comparing to the VAE and ACD objective functions in the previous subsections, $\mathrm{KL}(Q\|P)$ is the VAE part. $\mathrm{KL}(P\|\Pi) - \mathrm{KL}(Q\|\Pi)$ is the ACD part.

The divergence triangle leads to the following dynamics between the three models: (1) Q and P seek to get close to each other. (2) P seeks to get close to Π. (3) π seeks to get close to $q_{\rm data}$, but it seeks to get away from P, as indicated by the red arrow. Note that $\mathrm{KL}(Q\|\Pi) = \mathrm{KL}(q_{\rm data}\|\pi_\alpha)$, because $q_\phi(h|x)$ is canceled out. The effect of (2) and (3) is that π gets close to $q_{\rm data}$, while inducing P to get close to $q_{\rm data}$ as well, or in other words, P chases π_α toward $q_{\rm data}$.

(Han et al., 2019) also employed a layer-wise training scheme of (Karras et al., 2017) to learn models by divergence triangle from the CelebA-HQ dataset (Liu et al., 2015) including 200K celebrity face images with resolutions of up to 1024×1024 pixels. The learning algorithm converges stably, without extra tricks, to obtain realistic results as shown in Figure 14.

The top row of Figure 14 displays a few 1024×1024 images generated by the learned generator model with 512-dimensional latent vector. The bottom row of Figure 14 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 non-linear mapping leads to a highly non-linear interpolation in the image space. The interpolation experiment shows that the algorithm can learn a smooth generator model that traces the manifold of the data distribution.

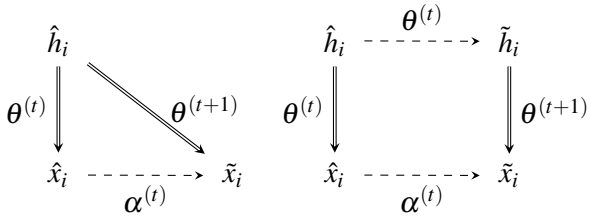
5.6 Cooperative learning via MCMC teaching

In adversarial contrastive divergence (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. (Xie et al., 2018a,b) 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 the 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 $i = 1, ..., \tilde{n}$. Starting from $\{\hat{x}_i, i = 1, ..., \tilde{n}\}$, we run MCMC such as Langevin dynamics for a finite number of steps toward π_α to get $\{\tilde{x}_i, i = 1, ..., \tilde{n}\}$, 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 (21).

Figure 14: Learning generator model by divergence triangle from the CelebA-HQ dataset that includes 200K high resolution celebrity face images. Top: Generated face images with 1024×1024 resolution sampled from the learned generator model with 512-dimensional latent vector. Bottom: Linear interpolation of the vector representations. The images at the two ends are generated from latent vectors randomly sampled from 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.

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 $\{\tilde{x}_i, i = 1, ..., \tilde{n}\}$ 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 $\{(\hat{h}_i, \tilde{x}_i); i = 1, ..., \tilde{n}\}$, which is a supervised learning problem, or more specifically, a non-linear 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 diagram in (29) illustrates the basic idea.



(29)

In the two diagrams in (29), the double-line arrows indicate generation and reconstruction by the generator model, while the dashed-line arrows indicate Langevin dynamics for MCMC sampling and inference in the two models. The diagram on the right in (29) illustrates 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 $\{(\tilde{h}_i, \tilde{x}_i), i = 1, ..., \tilde{n}\}$.

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 $(M_\alpha p_\theta)(x) = \int M_\alpha(x', x) p_\theta(x') dx'$ be the marginal distribution by running M_α initialized from p_θ. Then similar to the traditional contrastive divergence, the learning gradient of the evaluator model α at iteration t is the gradient of $\mathrm{KL}(q_{\text{data}} \,\|\, \pi_\alpha) - \mathrm{KL}(M_{\alpha^{(t)}} p_{\theta^{(t)}} \,\|\, \pi_\alpha)$ 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 diagram of (29) is the gradient of $\mathrm{KL}(M_{\alpha^{(t)}} p_{\theta^{(t)}} \,\|\, p_\theta)$ 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 $\text{KL}(M_{\alpha^{(t)}} p_{\theta^{(t)}} \,\|\, p_\theta)$ than minimizing $\text{KL}(q_{\text{data}} \,\|\, p_\theta)$ because the latent variables are essentially known in the former, so that 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 15.

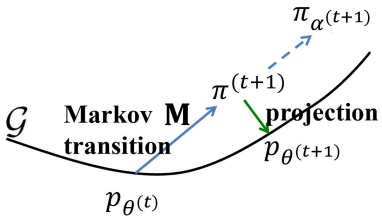


Figure 15: The MCMC teaching of the generator alternates between Markov transition and projection. The family of the generator models $\mathcal{G}$ is illustrated by the black curve. Each distribution is illustrated by a point.

Figure 16 displays two examples of image synthesis by cooperative learning algorithm on datasets LSUN bedrooms (Yu et al., 2015) and CelebA human faces (Liu et al., 2015).

(a) bedroom (b) face

Figure 16: 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 LSUN dataset with 3,033K training images. (b) Generating human face images (128×128 pixels). The synthesized images are generated by the cooperative learning algorithm that learns from celebA dataset with 200K training images. For each category, the top panel shows some examples of the training images, and the bottom panel shows some examples of the synthesized images generated by the learned models.

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.

6.1 Conditional generator, conditional VAE and GAN

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 dataset of the pairs $\{(x_i, c_i), i = 1, ..., n\}$ 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:

$$h \sim \text{N}(0, I_d),\ x = g_\theta(h, c) + \varepsilon, \tag{30}$$

where $g_\theta(h, c)$ is a top-down 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 parametrize 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, i.e., 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 Kullback-Leibler divergence $\text{KL}(q_{\text{data}}(x|c) \| p_\theta(x|c))$ over θ. The gradient of the conditional log-likelihood is computed by

$$-\frac{\partial}{\partial \theta} \text{KL}(q_{\text{data}}(x|c) \| p_\theta(x|c)) = \text{E}_{q_{\text{data}}(x,c) p_\theta(h|x,c)} \left[\frac{\partial}{\partial \theta} \log p_\theta(h, x|c) \right], \tag{31}$$

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 variational auto-encoder (CVAE) (Sohn et al., 2015) trains 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

$$\text{KL}(q_{\text{data}}(x|c) q_\phi(h|x, c) \| p_\theta(h, x|c)) = \text{KL}(q_{\text{data}}(x|c) \| p_\theta(x|c)) + \text{KL}(q_\phi(h|x, c) \| p_\theta(h|x, c)). \tag{32}$$

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 GANs (Mirza and Osindero, 2014), whose objective function of a two-player minimax game is

$$V(D, G) = \text{E}_{q_{\text{data}}}[\log D(x|c)] + \text{E}_{h \sim p(h)}[\log(1 - D(G(h|c))]. \tag{33}$$

The conditional generator models have had 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., 2017) 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., 2019b) 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

$$\pi_\alpha(x|c) = \frac{1}{Z(c, \alpha)} \exp[f_\alpha(x, c)], \tag{34}$$

Figure 17: 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 MNIST handwritten digit images along with their class labels. The image size is 64×64 pixels.

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 convolutional network (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 field (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 Subsection 5.6.

Figure 17 shows some examples of learning the conditional distribution of an image given a class label. The two models are jointly learned on 30,000 MNIST (LeCun et al., 1998) handwritten digit images 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 18 shows some examples of pattern completion on the CMP Facades dataset (Tyleček and 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.

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 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 and 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 to 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.

Figure 18: Pattern completion by conditional learning. Each row displays one example, where the first image displays 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 recovered result by the initializer (i.e., conditional generator model), the fourth image shows the recovered result by the solver (i.e., the MCMC sampler of the conditional energy-based model, initialized from the result of the initializer), and the last image shows the recovered result by the conditional GAN as a comparison.

7 Conclusion

This paper 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 multi-dimensional scaling.

A generative representation is a latent variable model. In this paper, 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 non-linear transformation. Such a model can be and should be extended to multiple layers of hidden vectors, or a hierarchical or graphical model (Lee et al., 2009; Salakhutdinov and Hinton, 2009). While statisticians tend to learn such models by maximum likelihood or Bayesian method, with the help of MCMC, people in deep learning prefer to learn such models by variational approximations or adversarial training. It is our hope that this paper explains the latter methods and connect 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.

Between vector representations and matrix representations, the latter are much less studied than the former, but the brain appears to need both for representing the sensory data, where 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 notion of the outside world are the vector and matrix representations that the brain invents to explain the sensory data. In other words, only data are real, and the outside world as we see it is more imaginary than real.

Acknowledgments

The work is supported by DARPA XAI project N66001-17-2-4029; ARO project W911NF1810296; and 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.

References

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning*, pages 1–8.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223.

Bell, A. J. and Sejnowski, T. J. (1997). The independent components of natural scenes are edge filters. *Vision Research*, 37(23):3327–3338.

Bengio, Y., Delalleau, O., Roux, N. L., Paiement, J.-F., Vincent, P., and Ouimet, M. (2004). Learning eigenfunctions links spectral embedding and kernel pca. *Neural Computation*, 16(10):2197–2219.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877.

Breiman, L. (2017). *Classification and regression trees*. Routledge.

Dai, J., Lu, Y., and Wu, Y. N. (2014). Generative modeling of convolutional neural networks. In *International Conference on Learning Representations*.

Dai, Z., Almahairi, A., Bachman, P., Hovy, E., and Courville, A. (2017). Calibrating energy-based generative adversarial networks. In *International Conference on Learning Representations*.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (methodological)*, pages 1–38.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *The Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 4171–4186.

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

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2017). Density estimation using real nvp. In *International Conference on Learning Representations*.

Dornhoff, L. L. (1972). *Group Representation Theory: Modular Representation Theory*.

Friedman, J. H. (1991). Multivariate adaptive regression splines. *The Annals of Statistics*, pages 1–67.

Gao, R., Lu, Y., Zhou, J., Zhu, S.-C., and Wu, Y. N. (2018a). Learning generative convnets via multi-grid modeling and sampling. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 9155–9164.

Gao, R., Xie, J., Zhu, S.-C., and Wu, Y. N. (2018b). Learning grid cells as vector representation of self-position coupled with matrix representation of self-motion. In *International Conference on Learning Representations*.

Gao, R., Xie, J., Zhu, S.-C., and Wu, Y. N. (2018c). Learning vector representation of content and matrix representation of change: Towards a representational model of v1. *arXiv preprint arXiv:1902.03871*.

Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. (2018). Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680.

Grathwohl, W., Chen, R. T., Betterncourt, J., Sutskever, I., and Duvenaud, D. (2019). Ffjord: Free-form continuous dynamics for scalable reversible generative models. In *International Conference on Learning Representations*.

Hafting, T., Fyhn, M., Molden, S., Moser, M.-B., and Moser, E. I. (2005). Microstructure of a spatial map in the entorhinal cortex. *Nature*, 436(7052):801.

Hamilton, W. L., Ying, R., and Leskovec, J. (2017). Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, 40:52–74.

Han, T., Lu, Y., Zhu, S.-C., and Wu, Y. N. (2017). Alternating back-propagation for generator network. In *The AAAI Conference on Artificial Intelligence*, volume 3, pages 1976–1984.

Han, T., Nijkamp, E., Fang, X., Hill, M., Zhu, S.-C., and Wu, Y. N. (2019). Divergence triangle for joint training of generator model, energy-based model, and inference model. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800.

Hinton, G. E. (2012). A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer.

Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. (1995). The "wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Hubel, D. H. and Wiesel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3):574–591.

Hyvärinen, A., Karhunen, J., and Oja, E. (2004). *Independent component analysis*, volume 46.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1125–1134.

Jin, L., Lazarow, J., and Tu, Z. (2017). Introspective classification with convolutional nets. In *Advances in Neural Information Processing Systems*, pages 823–833.

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

Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2017). Progressive growing of gans for improved quality, stability, and variation. *International Conference on Learning Representations*.

Kim, T. and Bengio, Y. (2016). Deep directed generative models with energy-based probability estimation. *ICLR Workshop*.

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

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. *International Conference for Learning Representations*.

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, (8):30–37.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105.

Kruskal, J. B. (1964). Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27.

Lafferty, J., McCallum, A., and Pereira, F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning*, pages 282–289.

Lazarow, J., Jin, L., and Tu, Z. (2017). Introspective neural networks for generative modeling. In *IEEE International Conference on Computer Vision*, pages 2774–2783.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lee, D. D. and Seung, H. S. (2001). Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems*, pages 556–562.

Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *International Conference on Machine Learning*, pages 609–616.

Lee, K., Xu, W., Fan, F., and Tu, Z. (2018). Wasserstein introspective neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3702–3711.

Li, K.-C. (1991). Sliced inverse regression for dimension reduction. *Journal of the American Statistical Association*, 86(414):316–327.

Liu, Z., Luo, P., Wang, X., and Tang, X. (2015). Deep learning face attributes in the wild. In *International Conference on Computer Vision*.

Lu, Y., Zhu, S.-C., and Wu, Y. N. (2016). Learning FRAME models using CNN filters. In *The AAAI Conference on Artificial Intelligence*.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *ICLR workshop*.

Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Mnih, A. and Gregor, K. (2014). Neural variational inference and learning in belief networks. In *International Conference on Machine Learning*, pages 1791–1799.

Neal, R. M. (2011). Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2.

Ngiam, J., Chen, Z., Koh, P. W., and Ng, A. Y. (2011). Learning deep energy models. In *International Conference on Machine Learning*, pages 1105–1112.

Nijkamp, E., Zhu, S.-C., and Wu, Y. N. (2019). On learning non-convergent short-run mcmc toward energy-based model. *arXiv preprint arXiv:1904.09770*.

O'Keefe, J. (1979). A review of the hippocampal place cells. *Progress in Neurobiology*, 13(4):419–439.

Olshausen, B. A. and Field, D. J. (1997). Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision Research*, 37(23):3311–3325.

Paccanaro, A. and Hinton, G. E. (2001). Learning distributed representations of concepts using linear relational embedding. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):232–244.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543.

Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. In *International Conference on Learning Representations*.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*.

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., and Lee, H. (2016). Generative adversarial text to image synthesis. *International Conference on Machine Learning*.

Rezende, D. J. and Mohamed, S. (2015). Variational inference with normalizing flows. *International Conference on Machine Learning*.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *International Conference on Machine Learning*.

Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326.

Rubin, D. B. and Thayer, D. T. (1982). Em algorithms for ml factor analysis. *Psychometrika*, 47(1):69–76.

Salakhutdinov, R. and Hinton, G. E. (2009). Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*.

Sohn, K., Lee, H., and Yan, X. (2015). Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, pages 3483–3491.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to reinforcement learning*, volume 135. MIT press Cambridge.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288.

Tu, Z. (2007). Learning generative models via discriminative approaches. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

Tyleček, R. and Šára, R. (2013). Spatial pattern templates for recognition of objects with regular structure. In *German Conference on Pattern Recognition*, pages 364–374.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Wang, T.-C., Liu, M.-Y., Zhu, J.-Y., Liu, G., Tao, A., Kautz, J., and Catanzaro, B. (2018). Video-to-video synthesis. *Advances in Neural Information Processing Systems*.

Wu, Y. N., Gao, R., Han, T., and Zhu, S.-C. (2019). A tale of three probabilistic families: Discriminative, descriptive and generative models. *Quarterly of Applied Mathematics*, 77(2):423–465.

Xie, J., Gao, R., Zheng, Z., Zhu, S.-C., and Wu, Y. N. (2019a). Learning dynamic generator model by alternating back-propagation through time. In *The AAAI Conference on Artificial Intelligence*.

Xie, J., Lu, Y., Gao, R., and Wu, Y. N. (2018a). Cooperative learning of energy-based model and latent variable model via mcmc teaching. In *The AAAI Conference on Artificial Intelligence*.

Xie, J., Lu, Y., Gao, R., Zhu, S.-C., and Wu, Y. N. (2018b). Cooperative training of descriptor and generator networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Xie, J., Lu, Y., Zhu, S.-C., and Wu, Y. N. (2016). A theory of generative convnet. In *International Conference on Machine Learning*, pages 2635–2644.

Xie, J., Zheng, Z., Fang, X., Zhu, S.-C., and Wu, Y. N. (2019b). Multimodal conditional learning with fast thinking policy-like model and slow thinking planner-like model. *arXiv preprint arXiv:1902.02812*.

Xie, J., Zheng, Z., Gao, R., Wang, W., Zhu, S.-C., and Wu, Y. N. (2018c). Learning descriptor networks for 3d shape synthesis and analysis. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 8629–8638.

Xie, J., Zhu, S.-C., and Nian Wu, Y. (2017). Synthesizing dynamic patterns by spatial-temporal generative convnet. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 7093–7101.

Xing, X., Han, T., Gao, R., Zhu, S.-C., and Wu, Y. N. (2019). Unsupervised disentangling of appearance and geometry by deformable generator network. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 10354–10363.

Yu, F., Seff, A., Zhang, Y., Song, S., Funkhouser, T., and Xiao, J. (2015). Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*.

Zee, A. (2016). *Group theory in a nutshell for physicists*. Princeton University Press.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *The AAAI Conference on Artificial Intelligence*.