ON LEARNING COMPOSABLE AND DECOMPOSABLE GENERATIVE MODELS USING PRIOR INFORMATION

Yeu-Chern Harn

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2019

ABSTRACT

Yeu-Chern Harn: On Learning Composable and Decomposable Generative Models Using Prior Information
(Under the direction of Vladimir Jojic)

Within the field of machining learning, supervised learning has gained much success recently, and the research focus moves towards unsupervised learning. A generative model is a powerful way of unsupervised learning that models data distribution. Deep generative models like generative adversarial networks (GANs), can generate high-quality samples for various applications. However, these generative models are not easy to understand. While it is easy to generate samples from these models, the breadth of the samples that can be generated is difficult to ascertain. Further, most existing models are trained from scratch and do not take advantage of the compositional nature of the data.

To address these deficiencies, I propose a composition and decomposition framework for generative models. This framework includes three types of components: part generators, composition operation, and decomposition operation. In the framework, a generative model could have multiple part generators that generate different parts of a sample independently. What a part generator should generate is explicitly defined by users. This explicit ”division of responsibility” provides more modularity to the whole system. Similar to software design, this modular modeling makes each module (part generators) more reusable and allows users to build increasingly complex generative models from simpler ones. The composition operation composes the parts from the part generators into a whole sample, whereas the decomposition operation is an inversed operation of composition.

On the other hand, given the composed data, components of the framework are not necessarily identifiable. Inspired by other signal decomposition methods, we incorporate prior information to the model to solve this problem. We show that we can identify all of the components by incorporating prior information about one or more of the components. Furthermore, we show both theoretically and experimentally how much prior information is needed to identify the components of the model.

Concerning the applications of this framework, we apply the framework to sparse dictionary learning (SDL) and offer our dictionary learning method, MOLDL. With MOLDL, we can easily include prior information about part generators; thus, we learn a generative model that results in a better signal decomposition operation. The experiments show our method decomposes ion mass signals more accurately than other signal decomposition methods. Further, we apply the framework to generative adversarial networks (GANs). Our composition/decomposition GAN learns the foreground part and background part generators that are responsible for different parts of the data. The resulting generators are easier to control and understand. Also, we show both theoretically and experimentally how much prior information is needed to identify different components of the framework. Precisely, we show that we can learn a reasonable part generator given only the composed data and composition operation. Moreover, we show the composable generators has better performance than their non-composable generative counterparts. Lastly, we propose two use cases that show transfer learning is feasible under this framework.

ACKNOWLEDGMENTS

First of all, I want to thank my advisor, Vladimir Jojic, for always providing helpful advice and support. He is willing to share his excellent research idea with me. When I encountered problems in my research projects, he worked with me patiently to solve the problems. Moreover, he gave comments on my academic writings and presentations so I can improve my writing and presenting skills. When I lacked the motivation to graduate, he is the person who continually reminds me to move on. I also want to thank my committee members, Leonard Mcmillan, Marc Niethammer, Nebojsa Jojic, Ron Alterovitz, for supporting my research and giving me feedback on my works. Without them, the quality of my thesis could be much lower.

Secondly, I want to thank my senior labmate: Tianxiang Gao. When I was a new graduate student, Tianxang shared his experience with machine learning and bioinformatics so I could catch up in these two areas quickly. Moreover, I want to thank all my project collaborators: Elizabeth Shank, Matthew Powers, Zhenghao Chen. By working with them, I learn how other great people approach challenging problems. I had excellent experience in collaboration with them.

Thirdly, I want to thank all UNC-CS faculty members that are helpful for my graduation. Faculties and staffs in UNC-CS are all very lovely and very helpful. Specifically, I want to thank my former and current student service manager, Jodie and Denise. They are enthusiastic about helping me with various documents for graduation. I also want to thank Leonard Mcmillan for giving me good advice and help proceed with my graduation procedure.

I think getting out of a PhD is a unique journey for everyone. For me, this journey was not smooth. I have encountered many failures in my research and difficulties in my life. Fortunately, I have my families and friends, who support me without any conditions, I am really grateful for their support. On the other hand, these failures and difficulties help me understand myself more, and make me live a religious life. I am glad for these changes.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Generative Model

Given an observable variable X and a target variable Y, a generative model is a statistical model of probability $p(X, Y)$ (Ng and Jordan, 2002). Informally speaking, a generative model assigns a probability for a sample. One benefit of this modeling is it allows including prior knowledge and explicit claims about how data is generated. Therefore, our framework that has assumptions about data generation and including prior knowledge naturally builds on the generative model.

1.2 Matrix-assisted Laser Desorption/Ionization Imaging Mass Spectrometry (MALDI-IMS)

In this dissertation, I applied my framework on common data types (2d images), but also on a specific type of data generated by Matrix-assisted Laser Desorption/Ionization Imaging Mass Spectrometry (MALDI-IMS) (McDonnell and Heeren, 2007). MALDI-IMS is a technique used in mass spectrometry, that is an analytical technique to measure the masses within a sample. From these masses, we can infer the possible molecules within the sample. Precisely, with MALDI-IMS we do not measure the masses directly; instead, we measure the masses of ion fragments ablated from the molecules. Each molecule may contribute to multiple ion fragments' measurements, and different molecules may produce ions of the same mass and charge, all contributing to the same measurement. Therefore, it is not trivial to uncover the molecules within a sample from MALDI-IMS data directly. Signal decompositions methods have been applied to uncover the source signals (signal of each molecule) from the mixed one.

Imaging Mass Spectrometry (IMS) is another technique used in mass spectrometry to visualize the spatial distribution of molecules. Instead of measuring samples in bulk, IMS measures multiple spatially related samples and thus formes a spatial distribution of measurements. Figure 1.1 is an example where MALDI-IMS is applied to a specimen containing two bacterial colonies. It is useful to know not only which ions were generated but also where and in what amount. For example, the figure shows that only the right bacteria colony generates the ion in column two.

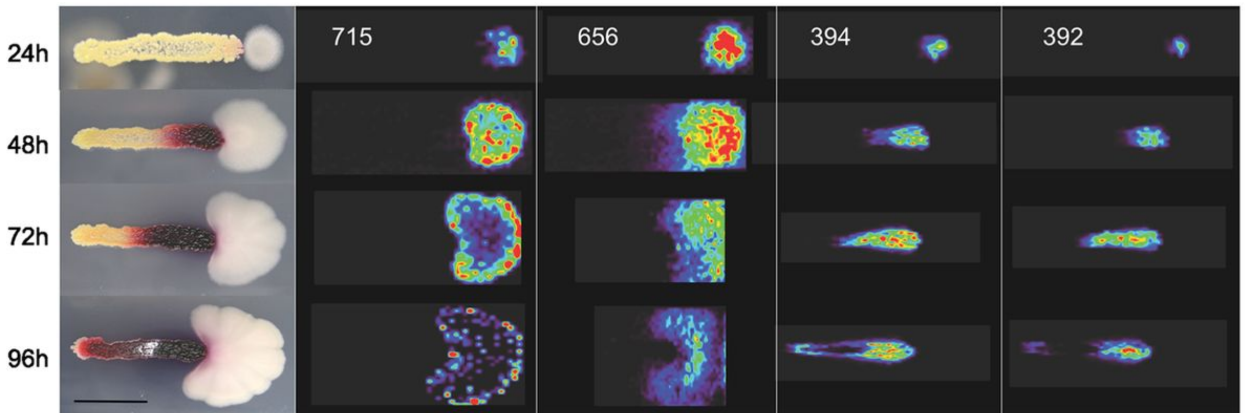


Figure 1.1: Interaction between two baceria colonies (Bacillus subtilis PY79 and Streptomyces coelicolor A3) over time (Yang et al., 2012). Column one are the photos of the specimen. Column two, three, and four represent different ion fragments and their distributions in this sample. The color gradient is from least-intense ion (purple) to most-intense ions (red).

1.3 Signal Decomposition

One approach to analyzing MALDI-IMS data is signal decomposition. Signal decomposition aims to separate a signal into multiple source signals. The decomposition provides a better understanding and a simpler model of input signals. Figure 1.2 is an example where two source images are decomposed (the images in the third row), given only the superpositioned-mixed images (the images in the second row) and no other information about the source images. Signal decomposition methods have been widely applied in many problems, including identification of molecules from MALDI-IMS data (Harn et al., 2015), denoising brain images (Congedo et al., 2008), decomposition of gene expressions into un-correlated groups (Ma and Dai, 2011), speech denoising (Schmidt et al., 2007), and object recognition (Koniusz et al., 2017).

Signal Decomposition is an ill-posed problem because there are multiple ways to decompose a mixed signal. Therefore, signal decomposition methods seek to narrow the set of possible ways of decompositions by imposing different constraints on the data or source signals. For example, Principal Component Analysis (PCA) (Bentler and Weeks, 1980) seeks linearly uncorrelated source signals, while independent component analysis (ICA) (Comon, 1994) seeks source signals that are maximally independent in a probabilistic sense. Non-negative matrix factorization (NNMF) (Sra and Dhillon, 2006) assumes the mixed and source signals are non-negative. Sparse Dictionary learning (SDL) (Engan et al., 1999) assumes a mixed signal is a sparse linear combination of the source signals (most of the signals' contributions to the mixed signal are zero), such that they can be decomposed into a few source signals. While methods with strict constraints provide easily

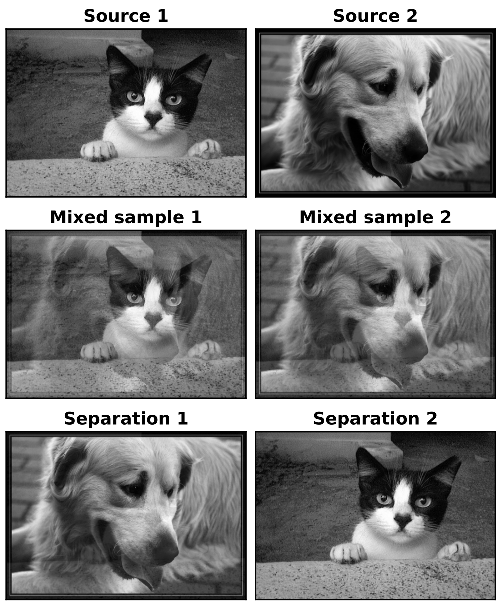


Figure 1.2: An image example of signal decomposition. We use FastICA (Hyvärinen and Oja, 2000) implemented by scikit-learn (Pedregosa et al., 2011) to separate the mixed signals. Note the separated signals only approximate the source signals.

interpretable models for data, they are not applicable in many real use cases. For example, ICA assumes independent source signals, to which we can easily assign meanings; however, it is unsuitable to apply ICA in a foreground detection task where we want to decompose the foreground object from an image because foreground and background are not independent.

1.4 Generative Adversarial Networks (GANs)

One of the main contributions of this work is the application of the framework for learning interpretable generative adversarial networks (GANs) (Goodfellow et al., 2014). GANs are a class of machine learning algorithms, implemented by a system of two neural networks contesting with each other. One network generates candidates (generators), and the other evaluates them (discriminators). While the discriminators' training objective is to correctly classify if a sample is coming from real data or the generators, the generators' training objective is to increase the error rate of the discriminators. In this way, the generators try to "fool" the discriminators by synthesizing samples that look as if they are coming from the true data distribution. The generator and discriminator are iteratively updated until convergence. After enough training, the generator can transform a set of latent codes into a realistic sample.

GANs have many advantages, including sampling at low cost, implicitly modeling of data, and last but not the least, GANs have many successful applications including realistic face generation (Karras et al., 2017), style transfer (Zhu et al., 2017), super-resolution imaging(Ledig et al., 2017), text to image synthesis (Reed et al., 2016), text generation (Fedus et al., 2018). Due to these advantages, GANs have been gaining much attention recently. Despite great interest in GANs, trained generators are typically monolithic and difficult to interpret. One approach to understanding generators is to investigate the effects of latent codes one-by-one. This method is not scalable to a large number of codes. Other approaches to obtaining interpretable generators are to assign meaning to code from the start (Odena et al., 2016; Donahue et al., 2017), or learn disentangled codes from data (Chen et al., 2016), but there are not many ways to do this. All in all, the generators, though they are powerful, are used in the manner of a black box.

1.5 Motivation from Signal Decomposition Methods

Signal decomposition methods have their constraints (or in a milder form: regularizations) to narrow down the possible ways of decompositions. A decomposition method is suitable for a type of data if its constraints reflect the nature of data. As mentioned before, some decomposition methods with strict constraints provide easily interpretable models but usually do not reflect the data, like PCA and ICA. Decomposition methods, such as NNMF or SDL, do not have the constraints as strict as PCA or ICA. Instead; NNMF assumes input data, and source signals are non-negative, and SDL assumes input data has a sparse representation. Non-negativity and sparsity are constraints that reflect the nature of many important data types. For example, audio spectrograms are inherently non-negative, and natural images naturally have sparse representations for fixed bases (e.g., Fourier) (Wright et al., 2010). Therefore, these constraints are useful in general. Motivated by this, we propose to incorporate prior knowledge about data, from which we can construct flexible constraints for our models. Since the constraints come from data, they inherently reflect the nature of data.

1.6 Thesis Statement

Thesis: *A generative model of a sample that is a mixture of source signals can be identified by using prior information about one or more of the following components: part generators, composition operation of* *parts into a sample, and an operator which decomposes a sample into parts. Dictionaries following the prior can be learned using this formalism. Modern deep models such as GANs can also be trained and analyzed using this formalism.*

This dissertation proposes a framework for generative models. This framework aims at modeling samples that are mixtures of a set of source signals. This framework includes three components: part generators, composition operation, and decomposition operation. A model could have multiple part generators that generate different parts of a sample. What a part generator generates is explicitly defined by users. The composition operation composes the parts from the part generators into a complete sample, whereas the decomposition operation is the inverse of composition. As for the signal decomposition problem, the uniqueness of part generators may not be guaranteed. Hence, correctly identifying the part generators, composition operation, and decomposition operation is a challenge. We show that we can identify the three components by using prior information about one or more of the components. We apply this framework to a traditional signal decomposition method, dictionary learning, as well as a modern deep model, GAN. The applications show the framework helps identify the components. We also derive sufficient conditions such that the generative models under this framework are identifiable.

1.7 Limitations of Previous Signal Decomposition Methods

Our framework includes both composition and decomposition operations. In contrast, most signal decomposition methods only assume a specific composition operation and discover decomposition function as the inference of latent variable, or Maximum a posteriori (MAP) estimation of parameters. For example, in dictionary learning, a signal decomposition method, a mixture sample is modeled as a linear combination of basic elements taken as the columns of the dictionary matrix $\boldsymbol{D}$. If also combing mixture samples in the column, we have a mixture samples matrix $\boldsymbol{Y}$, and the model for dictionary learning is:

$$\boldsymbol{Y} = \boldsymbol{D}\boldsymbol{X} \tag{1.1}$$

In this case, $\boldsymbol{D}$ is given as a predefined composition operation, and $\boldsymbol{X}$ is the decomposition solution. To get the decomposition, we time the pseudo-inverse of $\boldsymbol{D}$ on both sides of equation 1.1, and having the following:

$$\boldsymbol{X} = \boldsymbol{D}^{+}\boldsymbol{Y} \tag{1.2}$$

On the other hand, probabilistic modelings of signals define the composition operation and give a way to compute the probability distribution $p(Y|X, D)$. We can have the following to compute the decomposition distribution

$$p(X|Y, D) = \frac{p(Y|X, D)}{\sum_D p(Y|X, D)} \tag{1.3}$$

As shown in equation 1.3, in a probabilistic model, one has to compute probability over all possible $\boldsymbol{X}$ explicitly; thus, invoking the composition operation many times which is very expensive. In contrast, In my framework, decomposition is a given operation, and we can get the decomposition solution by this operation in a single pass, instead of invoking the composition operation multiple passes to infer decomposition in a probabilistic model.

Another benefit for explicitly defining the two operations is that it provides a possibility for modeling the two operators with different functions. The need for explicitly modeling the two operators happens in real cases. Consider the foreground object and background in an image for example; the composition operation is more straightforward than the decomposition operation. While the task of the composition operation is correctly putting the foreground on top of the background, the task of decomposition operation includes 1) distinguishing between the foreground and the background, and 2) inferring the complete background and foreground from its input. Under our framework, we can define a simpler model for the composition operation and a more complex model for the decomposition operation. A minor benefit for having the two operations defined separately is that we can execute each individually. In the case where we are only interested in decomposition (such as foreground detection), we do not need to execute the composition operation.

1.8 Limitations of Previous GANs

GANs have been successfully applied to many different tasks. The generators of GANs can produce realistic samples, but the generators remain opaque. Since our framework explicitly defines multiple part generators that generate specific parts of a sample and how they compose to form a complete sample, we can start to understand the building blocks of complex generators. Take images as an example; we train a part generator generating foreground objects and another part generator generating backgrounds. Therefore, we know what the foreground/background generator is responsible for, and we can analyze/use each generator independently. This division of responsibility also adds more extensibility to the original GANs. As the concept of modular programming is to separate the functionality of a program into independent modules, we can start to build toward ”modularity programming of GANs” by separating a complex generator into multiple simpler, independent generators. Each part generator should have an independent functionality so that it can be reused in other generative models.

1.9 Comparisons to Related Frameworks for GANs

Recently the idea of composition/decomposition of GANs has been explored. LR-GAN (Yang et al., 2017) learns a composite generator that invokes each part generator recursively and learns a composition operation to stitch generated outputs together. The recursive formalism of the part generators makes the later generators depends on the earlier generators. Compared to our framework, LR-GAN learns the non-independent part generators and composition operation, but not the decomposition operation. Compositional GAN (C-GAN) (Azadi et al., 2018) learns composition and decomposition operations that have the same purposed as ours. However, C-GAN does not learn part generators, so it is not a generative model. ST-GAN (Lin et al., 2018) learns to find the correct geometric warping parameters for a foreground object, given a background and a foreground, so that the foreground and background are composed reasonably. Therefore, ST-GAN learns a composition operation but not a decomposition operation and part generators. Table 1.1 summarizes the features of each framework. Compared to others, our framework is the most comprehensive because it provides a generative model capable of learning all three components. A benefit coming from this comprehensive framework is that the composition and decomposition operations together provides a regularization for part generators. In experiments, we show we improve the quality of generators by this regularization.

Method	Learn parts	Learn composition	Learn decomposition	Generative model
LR-GAN(Yang et al., 2017)	Background	True	False	True
C-GAN (Azadi et al., 2018)	False	True	True	False
ST-GAN (Lin et al., 2018)	False	True	False	False

Table 1.1: Various GAN frameworks can learn some, but not all, components of our framework. These components may exist implicitly in each of the models, but their extraction is non-trivial.

1.10 Contributions and Organization

The above thesis is supported by the following contributions made in this dissertation:

- Chapter 2 first presents a signal decomposition algorithm and neural network algorithms that our framework builds on. These include sparse dictionary learning, convolutional neural networks, and generative adversarial networks. Then it presents the evaluation metrics used in the dissertation. The evaluation of common signal decomposition algorithms is simple due to explicitly defined scoring function, whereas the evaluation of the algorithms of GANs is more complicated because GANs do not have such a scoring function. Consequently, we applied a GAN scoring function commonly used in the research community in this dissertation.

- Chapter 3 presents an application of our framework. Also, we include the prior information about the part generators in this algorithm so that we can correctly decompose mixed signals (decomposition operation) and recover the source signals (part generators). The experiments on synthetic datasets show this prior information is critical for recovering the correct source signals. We apply the proposed algorithm to several MALDI-IMS datasets. In these applications, we are interested in the source signals in the samples. Therefore, we evaluate the quality of the source signals resulting from the algorithm.

- Chapter 4 presents another application of our framework. For purposes of exposition, we defined a family of learning tasks in a progressively harder manner. The easier tasks have more prior information about the three components (composition operation, decomposition operation, and part generator), and the harder tasks have less prior information. We perform the experiments of these tasks on different datasets. We compare our generative model with other compositional/decompositional GANs approaches, and we evaluate the decomposition/composition operations, as well as the generative model both qualitatively and quantitatively. Last but not least, we also derive sufficient conditions such that these generative models are identifiable.

- Chapter 5 concludes this dissertation and discusses the limitations of the proposed methods as well as the future works.

CHAPTER 2: BACKGROUND

2.1 Sparse Dictionary Learning (SDL)

In the following section, I will introduce a sparse dictionary learning method, that is used in this thesis. This introduction includes the objective, the mathematical formulation, and the training algorithm of this method.

2.1.1 Introduction

Dictionary learning is a signal decomposition algorithm that aims at finding the source signals and representing the input data (mixed signals) as a linear combination of these source signals. These source signals are called dictionary elements, and together they form a dictionary. Using this dictionary, Sparse dictionary learning (SDL) further aims to find a sparse representation of the input data. This sparse representation has several advantages. First, it is easier to interpret, because most entries are zero in the representation. Second, it fits the nature of many data types, including natural images (Mairal et al., 2009) and audio (Zubair et al., 2013). Lastly, it allows learning a more flexible dictionary (e.g., dictionary elements are not required to be orthogonal) from a few high-dimensional samples (Aharon et al., 2006).

2.1.2 Mathematical Notations

Here I listed the mathematical notations that will be used later.

- Bold upper case letter represents a matrix, for example, $\boldsymbol{X}$.
- Bold bold lower case letter represents a vector, for example, $\boldsymbol{x}$.
- Normal letter represents a scalar, for example, i.
- $\|\boldsymbol{x}\|_1$, $\|\boldsymbol{x}\|_2$ represents L1-norm and L2-norm on vector $\boldsymbol{x}_1$. $\|\boldsymbol{X}\|_F$ represents Frobenius norm of matrix $\boldsymbol{X}$.

2.1.3 Mathematical Formulation

Given the input dataset as a matrix $\boldsymbol{X} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_K], \boldsymbol{x}_i \in \mathbb{R}^d$, we want to find a dictionary $\boldsymbol{D} \in \mathbb{R}^{d\times n}$: $\boldsymbol{D} = [\boldsymbol{d}_1, ..., \boldsymbol{d}_n]$, and a representation $\boldsymbol{W} = [\boldsymbol{w}_1, ..., \boldsymbol{w}_K], \boldsymbol{w}_i \in \mathbb{R}^n$ such that 1) the difference between $\|\boldsymbol{X} - \boldsymbol{DW}\|_F^2$ is minimized, and 2) the representation $\boldsymbol{W}$ is sparse. This can be formulated as the following optimization problem:

$$\underset{\boldsymbol{D}\in C, w_i \in \mathbb{R}^n}{\arg\min} \sum_{i=1}^{K} \|\boldsymbol{x}_i - \boldsymbol{D}\boldsymbol{w}_i\|_2^2 + \lambda \|\boldsymbol{w}_i\|_1, \text{ where } C \equiv \{\boldsymbol{D} \in \mathbb{R}^{d\times n} : \|\boldsymbol{d}_i\| \leq 1, \forall i = 1, ..., n\}, \lambda > 0 \quad (2.1)$$

In equation 2.1, the first term minimizes the difference. Since the second term is an L1-norm of the representation, it promotes the sparsity of the representation. In most cases, for large underdetermined systems of linear equation (that is the setting of SDL), the minimal L1-norm representation is also the sparsest one (Donoho, 2006). The hyperparameter λ controls the degree of this sparsity. λ is selected by an additional development dataset outside the training dataset. A constraint (C) on $\boldsymbol{D}$ is required so that its elements ($[\boldsymbol{d}_1, ..., \boldsymbol{d}_n]$) would not become arbitrarily large while $\boldsymbol{w}_i$ would become arbitrarily small.

2.1.4 Training of SDL

The optimization problem that SDL aims to solve is a bi-convex problem, that is the problem becomes convex with respect to each of the variables $\boldsymbol{D}$ and $\boldsymbol{W}$ when the other one is fixed. One training algorithm for a bi-convex problem is to train $\boldsymbol{D}$ and $\boldsymbol{W}$ alternately. This type of algorithm is called Alternate Convex Search (ACL). In theory, ACL gives a convergent sequence of objective function and a convergent sequence of feasible solutions if its solution sets are compact (Gorski et al., 2007). In SDL, $\boldsymbol{D}$ and $\boldsymbol{W}$ are both bounded so ACL can converge to a local minimum. ACL is not guaranteed to reach the global minimum. However, ACL usually reaches good enough solutions in practice (Mairal et al., 2009).

Since updating either $\boldsymbol{D}$ and $\boldsymbol{W}$ alone becomes a convex optimization problem, one could apply convex optimization algorithms to obtain an update. In theory, an update which approximates optimizer of the convex problem can be used. For example, in the work (Mairal et al., 2009), Least-angle regression (LARS) to find a sparse solution of $\boldsymbol{W}$ and use coordinate descent to find $\boldsymbol{D}$. In the work (Lee et al., 2007), Quadratic Programming is used to update a subset of $\boldsymbol{W}$ and update $\boldsymbol{D}$ according to the analytical solution of the Lagrangian dual form of the problem.

2.2 Generative Adversarial Networks (GANs)

Since GANs are built on deep neural networks, we will first introduce the networks and their building blocks.

2.2.1 Deep Neural Networks and Layers

A deep neural network consists of an input layer, an output layer, and several hidden layers between the input layer and the output layer. Data and application determine the number of hidden layers. There are several widely-used types of layers for deep neural networks. The three types of layers most relevant to this work are a fully connected layer, convolutional layer, and transposed convolutional layer. Note that all three layers perform linear operations. In order to have a deep neural network approximates non-linear operation, an element-wise non-linear operation is usually applied after a linear layer. In this thesis, the non-linear operation refers to a rectified linear unit (ReLU), or a leaky ReLU. ReLU is the following function:

$$f(x) = \max(0, x)$$

And leaky ReLU is the following function:

$$f(x, a) = \begin{cases} x & \text{if } x > 0, \\ ax & \text{otherwise,} \end{cases}$$

where a is usually a small value, e.g. 0.3.

A fully connected layer consists of neurons that connect all units in one layer to all units in the next layer. In a fully connect layer, the formula to compute the output for a neuron k, given the N inputs from $(x_1, ..., x_N)$ previous layer, is:

$$o_k = \sum_{i=1}^{N} w_i x_i + b_k, \tag{2.2}$$

where w_i are the weights connected to the input x_i from the previous neuron and b_k is the offset of neuron k. Both w_i and b_k are model parameters that need to be learned. On the other hand, the number of neurons in a fully connected layer is a hyperparameter of this layer. We will use the notation **FC(k)** to represent a fully connected layer with k neurons.

A convolutional layer consists of neurons that perform convolution operation over the inputs from the previous layer. In a convolutional layer, the formula to compute the output of the neuron k at a location i is:

$$o_{i,k} = \sum_{m=0}^{F} x_{i+m} \times w_{m,k} + b_k, \tag{2.3}$$

where $w_{f,k}$ and b_k are the weight and offset for the model. Same as fully-connected layers, the number of neurons is also a hyperparameter of this layer. There are three significant differences between a fully connected layer and a convolutional layer. First, a neuron in a convolutional layer only connects to F^2(filter size) inputs at a time. This restriction greatly decreases a convolutional layer's parameter number, but still enables the neuron connects to the inputs near the target location i. Second, the output from a neuron in a convolutional layer is not a scalar. In the above formula, the output dimension for neuron k is the same as its input. Also, an hyper-parameter S(stride) controls the output dimension. For example, a formula to compute the output of neuron k with stride equals to two is:

$$o_{i,k} = \sum_{m=0}^{F} x_{2i+m} \times w_{m,k} + b_k \tag{2.4}$$

With stride equals to two the output dimension is halved by two. Third, convolutional layers usually are applied to the input of 2D images, which is of two dimensions: (height, width). The two-dimensional version of the above formula is:

$$o_{i,j,k} = \sum_{m=0}^{F} \sum_{n=0}^{F} x_{2i+m,2j+n} \times w_{m,n,k} + b_k, \tag{2.5}$$

where $x_{i,j}$ is a pixel in a 2D image, and every neuron k has a weight matrix of size $F \times F$. To sum up, A convolutional layer applies convolution operation over its inputs. The hyperparameter F controls the number of connections to its inputs. The hyperparameter S controls the size of the output. If $S = 1$ the output has the same size as the input. If $S = 2$ the output size is halved by two. We will use the notation **CONV(k, f, s)** to represent a convolutional layer with k neurons, f filter size, and s stride.

When the stride is greater than one results in downsampling. This operation is useful when we want to down-sampling the inputs. When we want to up-sample the inputs, we use transposed convolutional layer. Up-sampling is needed if a network needs to go from a small dimensional input to a large dimensional output (e.g., generators in GANs). Informally speaking, we compute transposed-convolution by a reversing the direction of equation 2.5. Consider the convolution operation as a matrix multiplication operation. If we transpose the weight matrix and multiply it by the output of the convolution operation, we will get a matrix having the same dimension as the input of the convolution operation. In the case of stride equals to two, the output of that transposed convolutional layer has size double by two. A transposed convolutional layer has same hyperparameters as a convolutional layer. We will use the notation **TCONV(k, f, s)** to represent a transposed convolutional layer with k neurons, f filter size, and s stride.

2.2.2 The Network Architectures in GANs

In generative adversarial networks, there are one generator and one discriminator network. They are trained against each other. While the discriminator's goal is to correctly classify if a sample comes from real data or the generators, the generator's goal is to increase the error rate of discriminators. Both the discriminator and the generator in GANs are deep neural networks. DCGAN (Radford et al., 2015) is a widely-used network architecture for GANs. The discriminator in DCGAN (DCGAN-disc) is a series of convolutional layers with stride of two. Since the stride equals to two, the output dimension is halved. For a 32×32 image, the output dimension becomes 4×4 after it goes through three such convolutional layers. The neurons' number doubles across each layer. In DCGAN-disc, when the input dimension becomes 4×4, a fully-connected layer is on top of it. And the output of this fully-connected layer is a scalar indicating whether the input data is realistic, a DCGAN-disc for 32×32 images with one color channel has the architecture of three convolutional layers and one fully connected layer: **Input data-CONV(k, 5, 2)-CONV(2k, 5, 2)-CONV(4k, 5, 2)-FC(1)**.

The generator in DCGAN (DCGAN-gen) aims at generating large dimensional data from small dimensional codes. The Transposed convolutional layer is used to achieve this up-sampling operation. The transposed convolutional layer in DCGAN also has its stride equals to two. This doubles the dimension everytime an input goes through the layer. DCGAN-gen follows the design principle of reversing the operations in DCGAN-disc. The input codes first go through a fully-connected layer, with neuron numbers equals to $4 \times 4k$. Later the output of this fully connected layer is reshaped to 4×4 "images" with $4k$ channels, and goes through a series of transposed convolutional layers. While the dimension is doubled, the channel size is halved across each layer. The output of the final transposed convolutional layers should have the same dimension and channel size of the real data. For example, a DCGAN-gen for 32×32 image with one color channel has the architecture of one fully connected layer and three transposed convolutional layers: **Input codes-FC(4 $\times$ 4 $\times$ 4k)-TCONV(2k, 5, 2)-TCONV(k, 5, 2)-TCONV(1, 5, 2)**

For DCGAN network architecture, the layer number of a network is fixed once the input dimension is fixed. If the input dimension is 32×32, the network has three (transposed)convolutional layers. If the input dimension is 64×64, the network has four such layers. On the other hand, The neuron number k is a tunable hyperparameter in DCGAN.

2.2.3 GAN Mathmatical Formulation

A generative adversarial network is a framework that estimates generative models via an adversarial process. In this process we simultaneously train two models: a generator model G that captures the data distribution, and a discriminator model D that estimates the likeliness that a sample came from the training data rather than G. Since in this adversarial process the two networks are trained against each other, training GANs is not a minimization problem but a two-player minimax game. In the original formulation of GANs (Goodfellow et al., 2014), D and G play the following minimax game :

$$\min_{G} \max_{D} V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \tag{2.6}$$

$\mathbf{x} \sim p_{data}(\mathbf{x})$ represents a real sample sampling from the training set. And $\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})$ represents the generated data distribution. $\mathbf{z}$ is a vector of codes, each drawn independently from a simple distribution, such as normal distribution or uniform distribution. The discriminator D outputs a probability that a sample came from the training data rather than G.

There are many variants of GANs that use different cost functions, and there is no best GAN variant for every data and application (Lucic et al., 2018). In this work, I applied a variant called Wasserstein GAN with gradient penalty (WGAN-GP). Therefore, I will introduce WGAN-GP in details.

2.2.4 WGAN-GP

Wasserstein GAN (Arjovsky et al., 2017) is a variant of GAN trained by solving following minimax problem:

$$\min_{G} \max_{\|D\|_L \leq 1} V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[D(G(\mathbf{z}))] \tag{2.7}$$

This equation is similar to equation 2.6: D, G aim to decrease and increase the objective, respectively. However, there are differences. There is no $\log$ on the output of D. And D no longer outputs a probability but a real value. Finally, D must be a 1-Lipschitz function. A function that has gradient smaller or equal to one is by definition 1-Lipschitz. In other words, the rate of changes of this function is bounded.

By the Kantorovich-Rubinstein duality (Villani, 2008), the Wasserstein distance between real data distribution and generated data distribution is the maximum over all the 1-Lipschitz functions $D : \mathbb{R}^m \to \mathbb{R}$ ($\mathbf{x} \in \mathbb{R}^m$)

$$W(p_{data}, p_{\mathbf{z}}) = \max_{\|D\| \leq 1} \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})}[D(G(\mathbf{z}))] \tag{2.8}$$

And the above equation is the objective function that D maximizes in equation 2.7; thus, the discriminator in Wasserstein GAN maximize the Wasserstein distance between the true and generated data distribution. And G, training adversarially against D, minimize the Wasserstein distance.

In practice, there are several ways of making D a 1-Lipschitz function. In this thesis, I applied a method called Wasserstein GAN with gradient penalty (WGAN-GP), (Gulrajani et al., 2017). The method includes an additional penalty term on D to regularize it to be a 1-Lipschitz function.

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})}[D(G(\mathbf{z}))] + \lambda \mathbb{E}_{\hat{\mathbf{x}} \sim p_{\hat{\mathbf{x}}}(\hat{\mathbf{x}})}[(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1)^2] \tag{2.9}$$

The last term in the above equation is the penalty term on the gradients of D. λ controls the importance of this penalty term. Usually, λ can be fixed to 5, and it is robust enough for different data and applications. On the other hand, the penalty term is not put on every input domain ($\mathbf{x}$) of D. Instead, it is put only on a subset of D's input domain. The member of the subset is computed from the following process:

$$\begin{aligned}
&\mathbf{x} \sim p_{data}(\mathbf{x}) \\
&\tilde{\mathbf{x}} \sim G(\mathbf{z}), \mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z}) \\
&\epsilon \sim U[0, 1] \\
&\hat{\mathbf{x}} \sim \epsilon \mathbf{x} + (1 - \epsilon)\tilde{\mathbf{x}}
\end{aligned}$$

$U[0, 1]$ stands for Uniform distribution between zero and one. $\hat{\mathbf{x}}$ is an interpolation between the real sample and the generated sample. Although this penalty is not applied on every $\mathbf{x}$, the work (Gulrajani et al., 2017) shows this is sufficient to find the optimal solution for D.

In the original paper of the WGAN (Arjovsky et al., 2017), it shows there are several good properties of using the Wasserstein distance as a metric between two probability distrubtions. In practice, I also find WGAN-GP is more robust to different hyperparameters than the original GAN. An empirical comparison of this is shown in (Gulrajani et al., 2017). Also, the Wasserstein distance is a good indicator of the GAN training status. The generating quality has a high positive correlation with the distance. This does not happen in the original GAN formulation, where the objective function is bouncing around high and low values even in the late training iterations. Moreover, the WGAN computation does not use use logarithm and is a more straightforward computation than the original GAN.

2.2.5 Training of GANs

As training other deep neural networks, we train GANs with backpropagation algorithm. Backpropagation refers to an algorithm which uses chain rule to efficiently compute gradient of network's output with respect to its parameters. In principle, if we can compute these gradients, we can apply any gradient descent algorithms for optimization. In this work, we apply Adam (Kingma and Ba, 2014) to optimize our model because it is has been widely used to train GANs. Adam is a type of stochastic gradient descent algorithm which only use a random batch of samples to compute the gradients. The sample size of a batch is a hyperparameter for training GANs. Also, there are two hyperparameters β_1, β_2 that are the forgetting factors for gradients and second moments gradients, respectively (Kingma and Ba, 2014). These hyperparameters are either tuned with a validation dataset or set to default values suggested by the author of Adam.

2.3 Other Deep Neural Networks

2.3.1 Classifier Network

There are three additional network architectures applied in this work. The first is a classifier network that classifies its input samples (Krizhevsky et al., 2012). If we view the samples from training data as one class and the samples from the generator as another class, the discriminator in original GAN formulation equation 2.6 is a classifier network because it classifies the samples as coming from the training data or the generator. The classifier network outputs probability that input sample belongs to different classes. Therefore, the final layer of a classifier network is a sigmoid or a softmax layer to squish the outputs between zero and one. In terms of the network architectures, the classifier network does not have many restrictions. The hyperparameters, including the filter size, neuron numbers, and stride are all determined by application and data. We will specify the hyperparameters of each classifier networks in the experiment section.

2.3.2 Encoder-Decoder Network and U-Net

Concerning 2d image, The composition operation and decomposition operation both takes images as input and generate images as output. The two operations work similarly to the image translation in that they translate one image to another with possibly different number of channels. Encoder-decoder network architecture has been applied to solve this problem (Isola et al., 2017; Zhu et al., 2017); therefore, we also apply this network to our composition operation and decomposition operation. An encoder-decoder network is a network contains first an encoder network that extracts the high-level features from its input images, and then a decoder networks that reconstructs images from these high-level features. The encoder network contains a series of convolutional layers with stride equals to two, so it progressively down-samples its inputs and extracts complex features at this down-sampling level. The decoder network contains a series of transposed convolutional layers with stride equals to two, so the features are progressively up-sampled back to images. The encoder and decoder should have a similar number of parameters. There are no strict restrictions for this encoder-decoder network. For simplicity, we use encoder-decoder networks that have architectures similar to that of DCGAN. For example, for 32×32 images, our encoder-decoder network has the following architecture: **Input data-CONV(k, 5, 2)-CONV(2k, 5, 2)-CONV(4k, 5, 2)-TCONV(2k, 5, 2)-TCONV(k, 5, 2)-TCONV(1, 5, 2)**

Since the decoder networks' reconstruction relies purely on high-level features, they required encoders to extract informative high-level features. Also, for the composition or decomposition operation, there is much low-level information shared between the input and output. For example, a good foreground/background composition operation should directly copy pixels from the background if the foreground does not occlude them. Based on these observations, I chose enhanced network architecture called U-Net. U-Net has additional connections that pass the information between the encoder and decoder. Specifically, U-Net adds connections between each layer i and layer n-i, where n is the total number of layers. Each additional connection concatenates all inputs at layer i with those at layer n-i. Figure 2.1 shows examples of the encoder-decoder network and the U-Net.

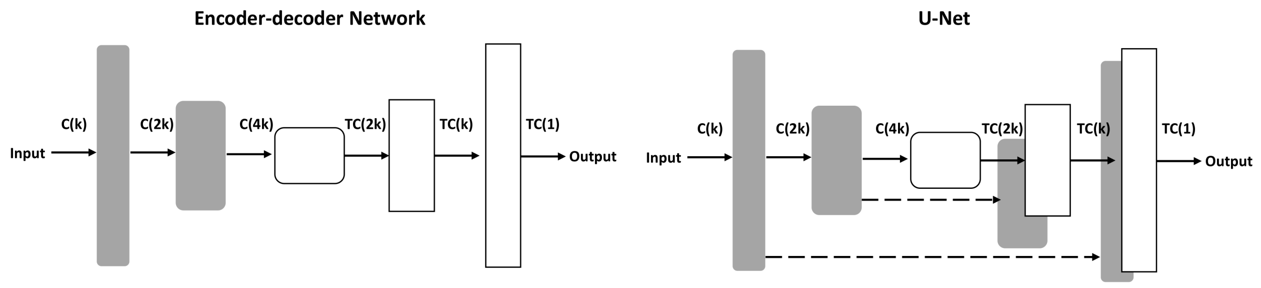


Figure 2.1: Illustrations of the encoder-decoder network and U-net. Each arrow represents an operation (either **CONV** or **TCONV**), while each dotted arrow represents an additional connection that copies the featrues from layer i to layer n-i and concatenates them together. **C(k)** stands for **CONV(k, 5, 2)** and **TC(k)** for **TCONV(k, 5, 2)**. Figure credit: (Isola et al., 2017)

2.4 Evaluations of Models

Since we have an objective function for sparse dictionary learning, we can use that function to evaluate data-fit of the model. Precisely, we optimize the function given different dictionaries on a validation dataset. For example, if we learn two dictionaries $\boldsymbol{D}_1, \boldsymbol{D}_2$ that come from different hyperparameter λ_1, λ_2, we optimize equation 2.1 with different dictionaries ($\boldsymbol{D}_1$ or $\boldsymbol{D}_2$) and compare their optimization results. Note that when evaluating the λ is fixed to zero, so we have a comparison of dictionaries with other conditions fixed. On the other hand, if we have the ground-truth dictionary of data, we can compare the learned dictionary and the ground-truth dictionary directly. If the dictionary's dimension is small, we can show all entries of the dictionaries in a figure; otherwise, we can define a similarity index between two dictionaries so we can have a quantitative comparison. In this work, we make cosine similarity between dictionary elements as the similarity index. The details of this evaluation are shown in chapter three.

Evaluating generative adversarial networks is more complicated because the model does not minimize an objective function, but plays a minimax game. Precisely, we want to evaluate the generators learned in GANs. A suitable generator should 1) generate realistic samples and 2) should generate a variety of samples as in the training dataset. A straight-forward evaluation is to make the target generator generate a fixed number of samples, and check the samples one-by-one. In this case, we can look at these generated images samples and have a general idea if our generator was doing a good job or not. However, as other qualitative evaluations, this evaluation is subjective. Also, it is not scalable to a large number of generated samples. There have been many different quantitative evaluation metrics proposed (Lucic et al., 2018; Xu et al., 2018). The first widely-used evaluation metric is the Inception Score. Since its computation is only based on the generated data, this metric cannot detect model collapse, a problem that a generator only generates data of low diversity compared to the real data. To cover this case, people proposed the evaluation metrics aims at computing the distance (or dissimilarity) between the two data distributions: the real data distribution $\mathbb{P}_r$ and the generated data distribution $\mathbb{P}_g$. These metrics are sampling-based metrics, meaning that they rely on random draw from $\mathbb{P}_r$ and $\mathbb{P}_g$. Among all these distance computing metrics, we selected and applied the Fréchet Inception Distance (FID) in our work because 1) it performs well compared to other metrics, and 2) Other GANs also use FID so that we can compare our results with others with fewer efforts. FID computes the follows:

$$FID(\mathbb{P}_r, \mathbb{P}_g) = \|\mu_r - \mu_g\| + Tr(C_r + C_g - 2(C_r C_g)^{1/2}), \tag{2.10}$$

where μ, C are the mean vector and the covariance matrice. FID models the two distributions as multivariate Gaussian distributions. While the 2D images are certainly not Gaussian distributed, the high-level features of the images could be close to Gaussian (Heusel et al., 2017). In this thesis, the high-level features are the outputs from the second to the last layer of a well-trained classifier network. The second to the last layer is the highest layer before fully connected layers for classification, so it still retain much information. Since other related works use Inception-v3 network (Szegedy et al., 2016) as the classifier network to compute FID, we also use Inception-v3 network.

CHAPTER 3: Learning Part Generators with Sparse Dictionary Learning (SDL)

In this chapter, I show the way of applying my formalism in sparse dictionary learning and propose a novel algorithm MOLDL. The proposed algorithm MOLDL is applied to MALDI-IMS data. It is shown that MOLDL has better performance than other state-of-the-art signal decomposition methods on this type of data.

3.1 MALDI-IMS Data and a Preprocessing method

In this section, I introduce the MALDI-IMS data in details and a preprocessing method for this data. As stated in the introduction section, MALDI-IMS is an analytical technique to measure the masses within a sample plate. This technique is based on a technique called matrix-assisted laser desorption/ionization time of flight (MALDI-TOF). MALDI-TOF is an analytical technique that measures the masses within a sample. MALDI-TOF analysis is a four-step process. In the first step, the input sample is mixed with the matrix. In the next step, a pulsed laser irradiates the sample mixture, triggering desorption of the mixture. In the third step, the analyte molecules in the mixture are ionized by being protonated or deprotonated. Finally, these ionized molecules are accelerated by an electric field of known strength, and their mass-to-charge ratios are measured (Karas and Krüger, 2003; Wolff and Stephens, 1953). For example, consider a molecule with a weight M is in an input sample. It is ionized as $[M + H]^+$ after the third step, and it provides a large signal of 440 m/z in the fourth step. Then we can compute the weight of $M + H$ by $\frac{M+H}{1} = 440$, and $M = 438.99$ (as $H = 1.01$).

There are two main challenges of recovering molecules' weights from the MALDI-TOF measurements. First, an analyte molecule could generate more than one ionized form, which is called an adduct. For example, a positively charged molecule could generate adducts like $[M + H]^+$, $[M + Na]^+$, $[M + K]^+$. Therefore, multiple molecules contribute to an m/z signal, and you can not recover a molecule from one signal. In this work, we applied our version of dictionary learning to solve this problem.

Second, a low-resolution mass spectrometry needs an algorithm to detect the ”real” m/z location. In figure 3.1, a molecule of 500 m/z not only has signal at that m/z but also has signals of m/zs around it. Thus, one ionization adduct generates multiple signals, but it contains only one peak at 500 m/z. This problem is alleviated when the spectrometry is of high-resolution (e.g. figure 3.2), but our instrument, MALDI-TOF, is not. To cope with this problem, we applied the Continuous Wavelet Transform (CWT)-based peak detection algorithm (Du et al., 2006). The main idea of this algorithm is not only considering the signal-to-noise ratio, but also the typical shape of a peak. The authors of this algorithm also showed it has the best peak-detection performance for MALDI data.

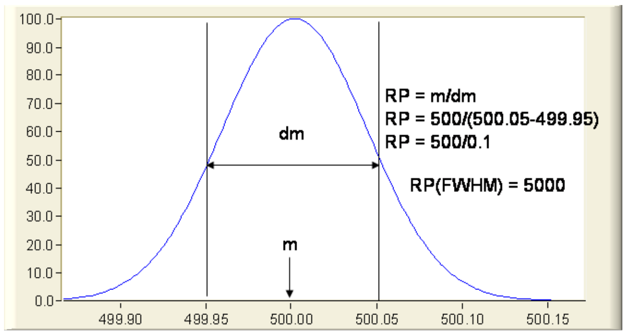


Figure 3.1: An example of a peak from low-resolution mass spectrometry. RP stands for resolution power. The figure is adopted from https://fiehnlab.ucdavis.edu/projects/seven-golden-rules/mass-resolution

Since there are still many false positive signals after the peak detection algorithm, a further reduction algorithm is needed. It is given to us that our m/z accuracy is one parts-per-million (ppm). With this information in mind, we develop a simple yet effective binning algorithm to further combining the peaks that have m/z values within the range of 1 ppm as one peak. This algorithm first sums up signals for each m/z across different samples. Since an m/z with a higher signals sum is more likely to be the center of a peak, the algorithm then bins each m/z in the descending order of signals sum we just computed. This heuristic makes binning wrong peaks less likely to happen. For an m/z m, its bin includes all the peaks in the range of [m - 0.5 ppm, m + 0.5 ppm], and the algorithm takes the maximum value in the bin to represent the value in this bin. After this step, we finish preprocessing our data. The code for the preprocessing algorithm can be found in `https://github.com/frizfealer/IMS_project/tree/master/Bruker_files_conversion`.

MALDI-IMS is an analytical technique based on MALDI-TOF. In this MALDI-IMS, a sample plate is divided into multiple spots, and each spot is analyzed with a MALDI-TOF. Therefore we have multiple

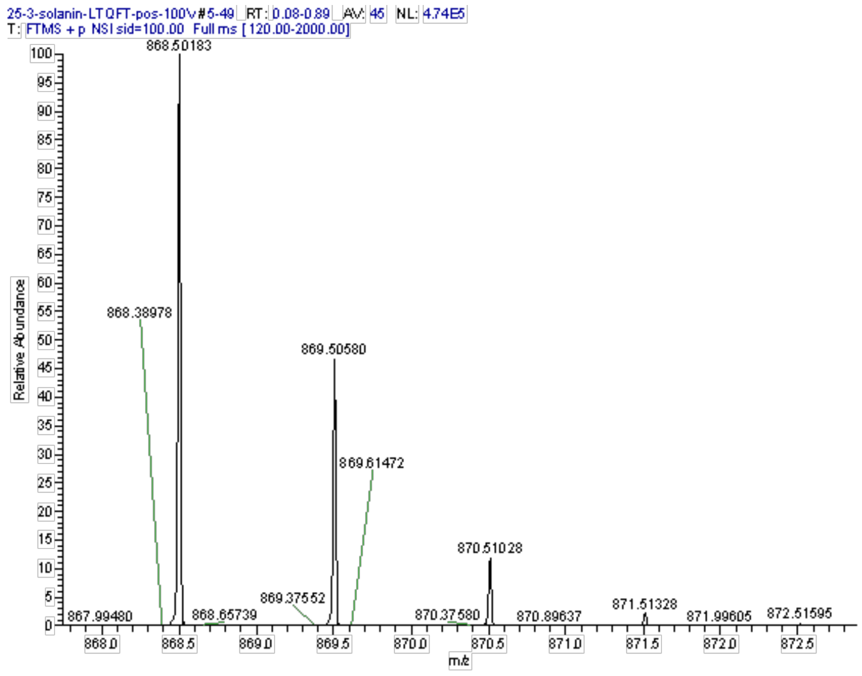


Figure 3.2: An example of peaks from high-resolution mass spectrometry. The figure is adopted from https://fiehnlab.ucdavis.edu/projects/seven-golden-rules/mass-resolution

samples of MALDI-TOF with a spatial relation. We will use $\boldsymbol{y}_i$ as a set of m/z signals of MALDI-TOF for a sample of index i. The number of m/z signals in each sample is determined by the preprocessing algorithm mentioned above. Consider a two-dimensional sample plate; we use two indices to represent a sample. For a sample at location i, j, we denote it as $\boldsymbol{y}^{i,j}$. Therefore, data for a sample plate for MALDI-IMS is a three-dimensional cube, with one dimension for m/z values, and the other two dimension for the width and the height of the plate.

3.2 Applying Our Framework to SDL

In the SDL formulation, given an input dataset as a matrix $\boldsymbol{Y} = [\boldsymbol{y}_1, ..., \boldsymbol{y}_k], \boldsymbol{y}_i \in \mathbb{R}^d$, we want to find a dictionary $\boldsymbol{D} \in \mathbb{R}^{d \times n} : \boldsymbol{D} = [\boldsymbol{d}_1, ..., \boldsymbol{d}_n]$, and a representation $\boldsymbol{W} = [\boldsymbol{w}_1, ..., \boldsymbol{w}_k]$, so the input matrix is best represented. We write our dictionary learning as a generative model as follows:

$$\boldsymbol{y}_i | \boldsymbol{D}, \boldsymbol{w}_i \sim \text{Poisson}(\boldsymbol{D}\boldsymbol{w}_i), i \in [1, ..., k]$$

$$\boldsymbol{w}_i \sim \text{Laplace}(\lambda), i \in [1, ..., k]$$

This formulation is similar to the original SDL model, except the data is modeled as Poisson distributed rather than Gaussian. We assume Poisson here because our data (MALDI-IMS) is non-negative and discrete. The L1-penalty on $\boldsymbol{w}_i$ in equation 2.1 corresponds to a Laplace prior on $\boldsymbol{w}_i$ here.

We then apply our framework to this generative model. **We regard each dictionary element $\boldsymbol{d}_j, j \in [1, ..., n]$ as a part generator** because it should generate only a part of the data measurements. There are n such part generators. Given these part generators and the generative model, **the composition and the decomposition operation are defined as a linear combination specified with w_i for each sample i.** The measurements of a sample i ($\boldsymbol{y}_i$) are formed by a linear combination of the measurements of the part generators.

3.3 Including Prior information

Learning $\boldsymbol{D}$ and $\boldsymbol{W}$ is undetermined if we do not provide the prior information to narrow down possible solutions. In this application, we can leverage two kinds of prior information on the part generators. The first prior information is that the measurements of the part generators have non-zero patterns. **A part generator models a set of measurements generated by a specific molecule.** Theoretically, a molecule only generates a few measurements in MALDI-IMS. And a molecule with a fixed experiment condition generates a fixed set of measurements. Table 3.1 shows a set of possible measurements that a molecule could generate. From the table, we can construct a sparse dictionary of these non-zero measurements (figure 3.3). Note that the non-zero entries in the dictionary are still treated as learnable parameters. If the data show no support for a non-zero entry of a part generator, the corresponding entry in that part generator will be zero.

Table 3.1: The possible ion products a molecule with molecular weight M could generate, under different experimental settings: positive mode and negative mode. Each row represents a type of signal coming from an ion product, named by "Ionization types." The field "m/z" is the measurement of the corresponded signal.

(a) Postive Mode

Ionization types	*m/z*
M+H	M+1.01
M+NH_4	M+18.04
M+Na	M+22.99
M+K	M+39.10
M+2Na-H	M+44.97
M+2K-H	M+77.19

(b) Negative Mode

Ionization types	*m/z*
M-H_2O-H	M-19.02
M-H	M-1.01
M-Na-2H	M+20.98
M+Cl	M+34.97
M-K-2H	M+37.08

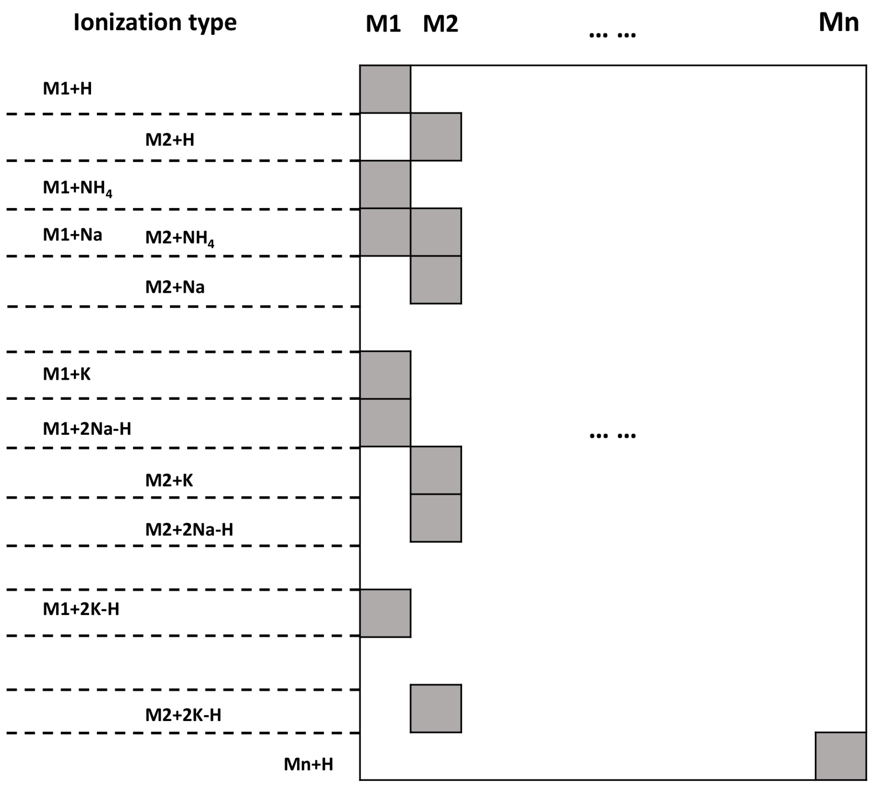


Figure 3.3: A dictionary with non-zero patterns is constructed from the table 3.1. Each column is a part generator modeling the measurements of each molecule. Each of the M1, M2, and Mn stands for a molecule's weight. The shaded entries are the non-zero entries. Given the experiment setting (here is the positive mode) and molecular weights, we can construct the pattern for each column. Note that it is possible that two molecules share the same measurement. In this example, M1+Na and M2+NH_4 have the same m/z measurement. Signal decomposition is useful for this case.

The second prior information is that the signals generated from molecules should be sparse. Theoretically, a molecule only generates a few signals in MALDI-IMS. Thus, a part generator should also have a few measurements in the data. We write out this prior as a Laplace prior on $\boldsymbol{d}_j$

$$\boldsymbol{d}_j \sim \text{Laplace}(\phi), j \in [1, ..., n]$$

Moreover, we have another helpful prior information on $\boldsymbol{W}$ that affects the composition and the decomposition operation. MALDI-IMS measures multiple samples distributed in a 2d plate. If we arrange these samples' measurements according to their distribution in the plate, we have a measurement tensor $\boldsymbol{Y}$ of size $d \times w \times h$, where d is the number of measurements and w and h are the width and height of the plate. Similarly, we have an abundance tensor $\boldsymbol{W}$ of size $n \times w \times h$, where n is the number of molecules in the dictionary and w and h are also the width and the height (figure 3.4). **We use $\boldsymbol{w}^{i,j}$ to model the abundance of the molecules in the sample at height i and width j.** In this arrangement, the nearby samples should contain similar abundances of molecules. One way to introduce this prior information to the generative model is to put Gaussian priors on the abundance differences between neighboring locations.

$$\boldsymbol{w}^{i+1,j} - \boldsymbol{w}^{i,j} \sim \text{Gaussian}(0, \theta)$$
$$\boldsymbol{w}^{i,j+1} - \boldsymbol{w}^{i,j} \sim \text{Gaussian}(0, \theta)$$

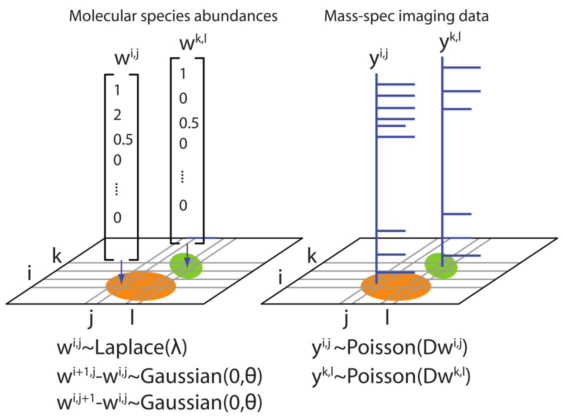

Molecular species abundances
Mass-spec imaging data
w^{i,j}
w^{k,l}
1
2
0.5
0
0
1
0
0.5
0
0
y^{i,j}
y^{k,l}
k
i
j
l
w^{i,j}~Laplace(λ)
w^{i+1,j}-w^{i,j}~Gaussian(0,θ)
w^{i,j+1}-w^{i,j}~Gaussian(0,θ)
y^{i,j}~Poisson(Dw^{i,j})
y^{k,l}~Poisson(Dw^{k,l})

Figure 3.4: The illustrations of the tensor $\boldsymbol{W}$ and $\boldsymbol{Y}$. We arrange the samples in a 2d plate, and each sample in the plate contains d measurements (Mass-spec imaging data). On the other hand, $\boldsymbol{W}$ models the abundances of the molecules in the dictionary $\boldsymbol{D}$. Each sample has a corresponded abundance vector. Gaussian priors on the abundance differences between neighboring samples are included in our model.

3.4 Training Algorithm of the Generative Model (MOLDL)

Given the generative model and the prior information, we write out the objective function as follows.

$$\begin{aligned} \underset{\boldsymbol{D},\boldsymbol{w}^{i,j},\lambda,\phi,\theta;\boldsymbol{y}}{\arg\min} \quad & \sum_{i,j}(\boldsymbol{y}^{i,j}) \cdot \log\left(\boldsymbol{D}\boldsymbol{w}^{i,j}\right) - \sum_{i,j}\{\boldsymbol{D}\boldsymbol{w}^{i,j}\} \\ & - \lambda \sum_{i,j} \|\boldsymbol{w}^{i,j}\|_1 - \phi \sum_{k=1}^{n} \|\boldsymbol{d}_k\|_1 \\ & - \theta \sum_{i,j} \|\boldsymbol{w}^{i,j} - \boldsymbol{w}^{i+1,j}\|_2^2 - \theta \sum_{i,j} \|\boldsymbol{w}^{i,j} - \boldsymbol{w}^{i,j+1}\|_2^2 \end{aligned} \tag{3.1}$$

In this equation, the first line corresponds to the log-likelihood function of Poisson regression. The second line corresponds to sparsity priors on $\boldsymbol{D}, \boldsymbol{W}$. The third line corresponds to Gaussian priors on the differences. We called the proposed learning problem molecular dictionary learning, abbreviated as MOLDL.

We also propose a learning algorithm to optimize this function (Harn et al., 2015). Similar to SDL, MOLDL iteratively updates dictionary $\boldsymbol{D}$ and abundance $\boldsymbol{W}$ until the objective function converges or the changes in $\boldsymbol{W}$ and $\boldsymbol{D}$ is smaller than a specific value.

Minimizing $\boldsymbol{D}$ with $\boldsymbol{W}$ fixed is a non-linear optimization with two constraints. The first constraint is that the elements in D are positive. And the second constraint is that the L2-norm of each dictionary element is less than or equal to one ($\|\boldsymbol{d}\|_2 \leq 1$). We applied the interior-point method to solve this optimization problem (Wächter and Biegler, 2006). On the other hand, minimizing $\boldsymbol{W}$ with $\boldsymbol{D}$ fixed is a complex optimization problem due to its objective function. We applied the alternating direction method of multipliers (ADMM) algorithm (Boyd et al., 2011). The main idea of the ADMM is solving the problem through a divide-and-conquer approach. We divide the objective function into several sub-functions and update each sub-functions individually and alternately. We first write out the objective function as

$$\begin{aligned} \underset{\boldsymbol{z}_0^{i,j}, \boldsymbol{z}_1^{i,j}, \boldsymbol{z}_2^{i,j}}{\arg\min} \quad & \sum_{i,j} (\boldsymbol{y}^{i,j}) \cdot \log(\boldsymbol{z}_0^{i,j}) - \sum_{i,j} \{\boldsymbol{z}_0^{i,j}\} \\ & - \lambda \sum_{i,j} \|\boldsymbol{z}_1^{i,j}\|_1 - \theta \sum_{i,j} \|\boldsymbol{z}_2^{i,j}\|_2^2 \end{aligned} \tag{3.2}$$

And it is subject to

$$\begin{aligned} \boldsymbol{z}_0^{i,j} &= \boldsymbol{D}\boldsymbol{w}^{i,j}, \\ \boldsymbol{z}_1^{i,j} &= \boldsymbol{w}^{i,j}, \\ \boldsymbol{z}_2^{i,j} &= [\boldsymbol{w}^{i,j} - \boldsymbol{w}^{i+1,j}; \boldsymbol{w}^{i,j} - \boldsymbol{w}^{i,j+1}], \end{aligned}$$

where the ";" symbol in the definition of $\boldsymbol{z}_2$ is the row-combined operator. Then we write out this constraint optimization as the following:

$$\begin{aligned}
\operatorname*{arg\,min}_{\boldsymbol{z}_0,\boldsymbol{z}_1,\boldsymbol{z}_2,\boldsymbol{u}_0,\boldsymbol{u}1,\boldsymbol{u}2^{i,j},\boldsymbol{w}} \quad & \sum_{i,j}(\boldsymbol{y}^{i,j}) \cdot \log(\boldsymbol{z}_0^{i,j}) - \sum_{i,j}\{\boldsymbol{z}_0^{i,j}\} \\
& - \lambda \sum_{i,j} \|\boldsymbol{z}_1^{i,j}\|_1 - \theta \sum_{i,j} \|\boldsymbol{z}_2^{i,j}\|_2^2 \\
& - \sum_{i,j}(\boldsymbol{u}_0^{i,j}) \cdot (\boldsymbol{z}_0^{i,j} - \boldsymbol{D}\boldsymbol{w}^{i,j}) - \sum_{i,j}(\boldsymbol{u}_1^{i,j}) \cdot (\boldsymbol{z}_1^{i,j} - \boldsymbol{w}^{i,j}) \\
& - \sum_{i,j}(\boldsymbol{u}_2^{i,j}) \cdot (\boldsymbol{z}_2^{i,j} - [\boldsymbol{w}^{i,j} - \boldsymbol{w}^{i+1,j}; \boldsymbol{w}^{i,j} - \boldsymbol{w}^{i,j+1}]) \\
& - \sum_{i,j} \frac{\rho}{2} \|\boldsymbol{z}_0^{i,j} - \boldsymbol{D}\boldsymbol{w}^{i,j}\|_2^2 - \sum_{i,j} \frac{\rho}{2} \|\boldsymbol{z}_1^{i,j} - \boldsymbol{w}^{i,j}\| \\
& - \sum_{i,j} \frac{\rho}{2} \|\boldsymbol{z}_2^{i,j} - [\boldsymbol{w}^{i,j} - \boldsymbol{w}^{i+1,j}; \boldsymbol{w}^{i,j} - \boldsymbol{w}^{i,j+1}]\|
\end{aligned} \tag{3.3}$$

In this form, we can perform the ADMM algorithm, that is updating each parameter alternately. Updating $\boldsymbol{w}$ is a linear optimization with a simple constraint $\boldsymbol{w} > 0$. Updating $\boldsymbol{z}_0$ is a non-linear optimization problem without constraints, that can be solved by the trust-region algorithm (Byrd et al., 1987). Updating $\boldsymbol{z}_1$ and updating $\boldsymbol{z}_2$ are both linear optimization problems with an L1-penalty and L2-penalty, respectively. Both of them has a closed-form formulation for their updates. After updating all primal parameters, we update the dual parameters $\boldsymbol{u}_0, \boldsymbol{u}_1, \boldsymbol{u}_2$ with the followings:

$$\begin{aligned}
\boldsymbol{u}_0 &\leftarrow \boldsymbol{u}_0 + \rho \sum_{i,j}(\boldsymbol{z}_0^{i,j} - \boldsymbol{D}\boldsymbol{w}^{i,j}) \\
\boldsymbol{u}_1 &\leftarrow \boldsymbol{u}_1 + \rho \sum_{i,j}(\boldsymbol{z}_1^{i,j} - \boldsymbol{w}^{i,j}) \\
\boldsymbol{u}_2 &\leftarrow \boldsymbol{u}_2 + \rho \sum_{i,j}(\boldsymbol{z}_2^{i,j} - [\boldsymbol{w}^{i,j} - \boldsymbol{w}^{i+1,j}; \boldsymbol{w}^{i,j} - \boldsymbol{w}^{i,j+1}])
\end{aligned}$$

We also implement the adaptive ρ algorithm (Byrd et al., 1987) to have a faster convergence speed.

3.5 Biconvexity of MOLDL

Biconvexity not only holds in SDL, but also in MOLDL. If MOLDL is bi-convex, it means that updating $\boldsymbol{D}$ with $\boldsymbol{W}$ fixed is a convex optimization, and updating $\boldsymbol{W}$ with $\boldsymbol{D}$ fixed is also a convex optimization.

Theorem 3.1. *The objective in equation 3.1 is bi-convex in abundances, $\boldsymbol{W}$, and dictionary, $\boldsymbol{D}$.*

Proof. Sketch: To show that the objective function is bi-convex in $\boldsymbol{W}$ and $\boldsymbol{D}$, we need to show that the function is convex in $\boldsymbol{W}$ for a fixed $\boldsymbol{D}$ and vice versa. For a fixed $\boldsymbol{D}$ the objective is a sum of a convex function with an affinely transformed argument $\sum_{i,j} \log\left(\boldsymbol{D}\boldsymbol{w}^{i,j}\right)$, a linear functions of $\boldsymbol{w}$, a convex function $\|\boldsymbol{w}^{i,j}\|_1$ and another convex function with affinely transformed arguments $\|\boldsymbol{w}^{i,j} - \boldsymbol{w}^{i,j+1}\|_2^2$. As a sum of convex functions, the objective function is convex for a fixed $\boldsymbol{D}$. For a fixed $\boldsymbol{W}$, the objective function has three terms that are influenced by D. The first term is a convex function with an affinely transformed argument linear function of $\boldsymbol{D}$, $\sum_{i,j} \log \boldsymbol{D}\boldsymbol{w}^{i,j}$. The second term is a linear function of D, $-(\sum_{i,j}\{\boldsymbol{D}\boldsymbol{w}^{i,j}\})$. The third term is a convex function $\|\boldsymbol{d}_k\|_1$. As a sum of convex function, the objective is convex for a fixed $\boldsymbol{W}$. Hence, the objective is bi-convex in abundances and dictionary. $\square$

3.6 Experimental Results

3.6.1 Synthetic Data Results

We present three synthetic experiments with different purposes. We evaluate the experimental results by comparing the learned dictionary and the ground-truth dictionary. If the size of the dictionary is small, we can compare the learned dictionary to the ground truth entry-by-entry. Otherwise; we use cosine similarity to compute agreement between the pairs of dictionary elements. We compute cosine similarity for the two dictionary elements $\boldsymbol{d}_1, \boldsymbol{d}_2$ by $cos(\boldsymbol{d}_1, \boldsymbol{d}_2) = \frac{\boldsymbol{d}_1 \cdot \boldsymbol{d}_2}{\|\boldsymbol{d}_1\|\|\boldsymbol{d}_2\|}$. There are two details for using cosine similarity. First, since a dictionary contains many dictionary elements and cosine similarity is defined for a pair of elements, we cannot directly apply it to measure the similarity of the two dictionaries. In this work, we compute cosine similarity for each pair of dictionary elements, and we show a distribution of these cosine similarities. Second, two similar dictionaries might have a very different order of their dictionary elements. If we directly compute the cosine similarity of the dictionary elements on the given orders of the two dictionaries, we might get very low cosine similarities even for the two very similar dictionaries. To overcome this issue, we apply the

Hungarian algorithm (Kuhn, 1955) to find the best matching of the dictionary elements. Hungarian algorithm ensures we get the best cosine similarities we could have for two given dictionaries.

The first synthetic experiment shows that the prior of non-zero patterns is useful for learning more accurate dictionaries. In the experiment, we created a ground truth dictionary that has the non-zero pattern $[1, 1, 0; 0, 1, 1; 1, 0, 0]$. and made the ground truth dictionary $\boldsymbol{D}$ to be $[0.707, 0.707, 0; 0, 0.707, 0.707; 1, 0, 0]$. This dictionary has three dictionary elements, with the first two dictionary elements have two non-zero values, and the last dictioanry element has one non-zero value. The values of $\boldsymbol{W}$ were generated by taking absolute values of the sample from a normal distribution $(\mu = 0, \sigma = 1)$. The sample size (width times height) is 20×20, and $\boldsymbol{W}$ is a tensor of size $3 \times 20 \times 20$. The result in figure 3.5 shows with the non-zero pattern (MOLDL w p), the dictionary is similar to the ground truth. Without the non-zero pattern (MOLDL w/o p), a false positive appears in entry 6, and two false negative appear in entry 3 and 5.

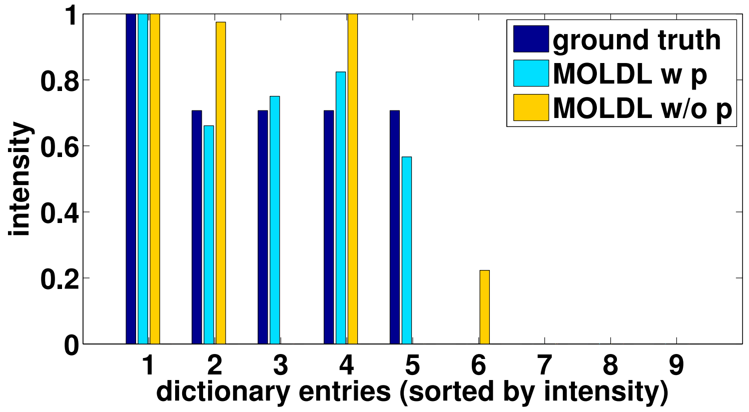


Figure 3.5: The entry-by-entry comparison of the ground truth dictionary with MOLDL with the non-zero pattern (MOLDL w p), and without the pattern (MOLDL w/o p). The entries are sorted in the descending order of the intensity in the ground truth dictionary. The ground-truth value for each entry is $1, 0.707, 0.707, 0.707, 0.707, 0, 0, 0, 0$.

The second synthetic experiment shows MOLDL performs better than NNMF and pLSA when data consists of measurements shared by several dictionary elements. In the experiment, we made a ground truth dictionary as a three-by-three matrix: $[0.89, 0.45, 0; 0, 0.89, 0.45; 0.89, 0, 0.45]$. Thus each of the dictionary element shares measurements with the other two. The ground truth abundances were generated as follows: a location on the 2D plane for a particular dictionary element was chosen and smoothing of its abundance was performed. The smoothing was implemented using a mean filtering kernel of size 3×3 iterated 10 times. This smoothing algorithm is to simulate the real case where nearby samples should contain similar abundances. The $\boldsymbol{W}$ is a tensor of size $3 \times 20 \times 20$. Figure 3.6 shows MOLDL decomposed dictionary elements correctly. Both NNMF and pLSA were unable to learn the correct dictionary elements in this dataset.

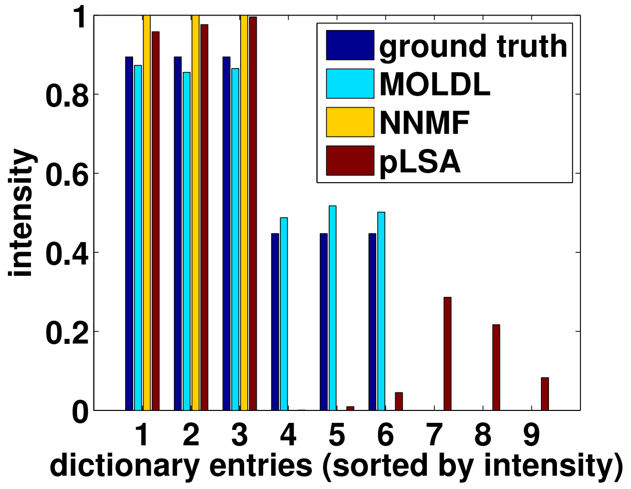


Figure 3.6: The entry-by-entry comparison of the ground truth dictionary with the dictionary from MOLDL, NNMF, and pLSA. The entries are sorted in the descending order of the intensity in the ground truth dictionary.

In the third synthetic experiment, we compare by the distributions of cosine similarity. The way we compute the distributions is as follows. We apply the Hungarian algorithm to find the best matching of the two dictionaries. We define the best matching as the matching having the largest overall cosine similarities. For the best matching, it contains n pairs of cosine similarities, where n is the number of dictionary elements in the compared dictionary. Therefore, we have a distribution of cosine similarities for these pairs, and we call these ”true dictionary elements.” On the other hand, any pair is regarded as a mismatching if it is not included in the best matching. Their cosine similarities are also computed; so we obtain this distribution. We call this group of pairs ”false dictionary elements.” In figure 3.7, we compared the learned dictionary from each method (MODL, NNMF, and pLSA) to the ground truth dictionary, and we also show in figure 3.7A the comparison of the ground truth dictionary to itself as a reference. In figure 3.7A, all true dictionary elements have a cosine similarity of 1, and most mismatched dictionary elements have cosine similarities closed to 0. For sparse-NNMF and sparse-pLSA, $\sim 20\%$ of the matched dictionary elements' cosine similarities are around 1, and the whole distribution lies in a broad range of $[0.2, 1]$. However, there are $\sim 10\%$ of the cosine similarities around 0.4 for both algorithms and $\sim 10\%$ of cosine similarities around 0.2 in sparse-NNMF (figure 3.7B, C). In contrast, in MOLDL, there are $\sim 95\%$ of the cosine similarities larger than 0.5 and $\sim 80\%$ of the similarities larger than 0.8 (figure 3.7D).

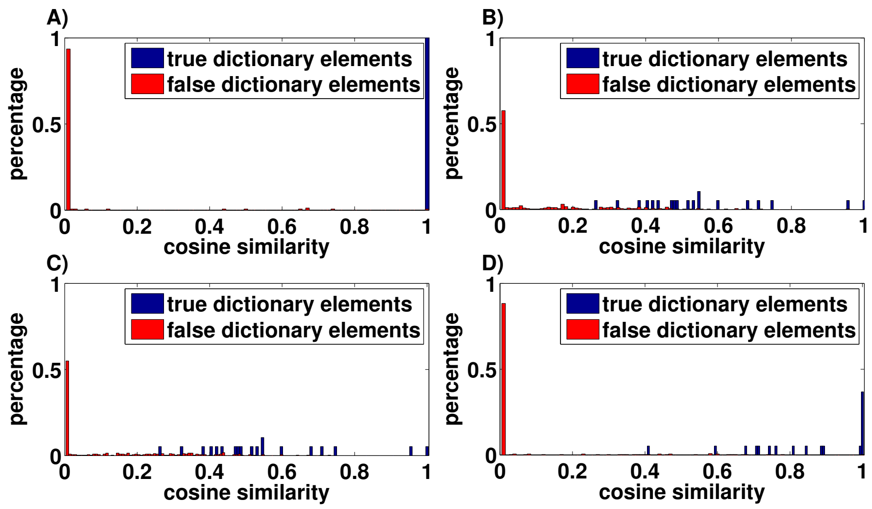


Figure 3.7: The distribution of the cosine similarities of the true dictionary elements and false dictionary elements for each pair of methods comparison. (A) ground truth dictionary compared to itself. (B) The dictionary from sparse-pLSA compared to the ground truth dictionary. (C) The dictionary from sparse-NNMF compared to the ground truth dictionary. (D) The dictionary from MOLDL compared to the ground truth dictionary.

3.6.2 Real Data Results

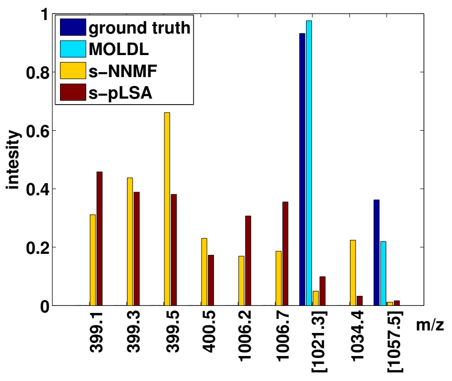
Our real data samples consist of two bacteria strains (*Bacillus cereus* and *Bacillus subtilits*). We are interested in finding out what molecules are and how they distributed in the samples. The sample preparation includes preparing bacteria strains and media on which the strains grow. *Bacillus cereus* ATCC14579 and *Bacillus subtilis* NCIB 3610 were resuspended into Luria Broth from growth on agar plates, and resuspended to an OD_{600} of 0.5 One μl of these cell suspensions were then spotted onto 10 ml agar plates (0.1 X Luria Broth, Lennox: 1 g tryptone, 5 g yeast extract, 5 g NaCl and 15 g Bacto-agar per L). Four bacterial spots (two of each bacterial species) were put onto the agar plates in a line, with the two spots of the same species next to each other at a 1 cm distance, and the spots of the different bacterial species 0.5 cm away from each other. Colonies were grown at 30°C for 12 or 40 hr before being harvested for MALDI-TOF imaging.

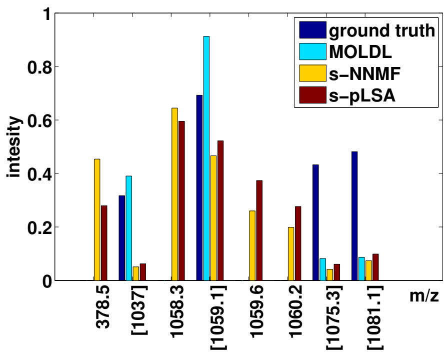
The next step is MALDI-IMS sample preparation. Agar-grown microbial samples were prepared for MALDI as described in (Yang et al., 2012). Briefly, rectangular regions of agar containing the bacterial co-cultures and the distal control colonies were excised from the agar plate, placed onto a MALDI-TOF ground steel target plate (Bruker part no. 224990) and covered with Universal MALDI matrix (Sigma, Fluka 50149) using a 53 μm stainless steel sieve (Hogentogler & Co, part 1312). After matrix application, the sample was dried overnight at 37°C. Excess matrix was physically removed to clean the plate, and a peptide calibration standard was spotted onto it (Bruker part no. 206195, Pepmix4).

Here is our MALDI-IMS experiment protocol. After mass calibration using the Pepmix standard, samples were imaged using a MALDI-TOF mass spectrometer (Microflex LRF, Bruker) with a Microscout ion source (Nitrogen UV laser, $\lambda = 337$ nm) in both linear positive and linear negative mode. FlexControl and FlexImaging software (Bruker) was used for image acquisition with 80 shots averaged from each pixel of 400 to 800 μm across an m/z of 0 to 5000Da.

After these experiments, we use the method mentioned before to preprocess the raw data. On the other hand, to evaluate MOLDL on the dataset, we need a ground truth dictionary to compare. From literature, we found surfactin-C13, surfactin-C14, surfactin-C15 are the molecules that should appear in our dataset. To have the ground truth dictionary elements for these molecules, we conducted MALDI-IMS experiments with purified surfactin-C13, C14, and C15. We then compare the dictionary elements learned from MOLDL with the ground truth elements. Since Surfactin-C14 and surfactin-C15 have the most complex signals and measurements, we provide the results of these molecules. The results of other molecules can be found in the supplementary of citepharn2015deconvolving. Figure 3.8a and b show the entry-by-entry comparisons of the dictionary elements from ground truth and different methods. The m/z with a bracket stands for the measurement that appears in the ground truth dictionary. Concerning these true positive measurements, the dictionary element of MOLDL has the measurements that are the most similar to the true positive measurements. Also, the dictionary element of MOLDL has the least false positive measurements. Both sparse-NNMF and sparse-pLSA have a lot of false positives, and they affect the true positives because of the shared measurements between many molecules. Including the prior information of the dictionary and abundance reduces these false positive, and result in the success of MOLDL.

Since we do not have the ground truth dictionary for each molecule in the samples, we cannot show by experiments that the overall dictionary learned by MOLDL is better. However, the shared measurements are widespread in the MALDI-IMS data. Our method that excludes the infeasible solutions and removes false-positives can excel at this type of data. Moreover, if we know more prior information about the dictionary in data, we can further decrease the number of non-zero entries and thus excludes more infeasible solution. One possible future direction of this work is investigating the learned abundances. The abundances represent the distribution of molecules in the dictionary. For the molecule of interest, we should check if the location of its high abundance corresponds to that of the bacteria colony that could secrete this molecule.





(a) The surfactin-C14 in the negative mode data (b) The surfactin-C15 in the positive mode data

Figure 3.8: The surfactin-C14 and surfaction-C15 dictionary element from the ground truth (purified surfactin), MOLDL, s-pLSA (sparse pLSA), and s-NNMF (sparse NNMF) in the negative and positve mode data. The negative and positive mode are experimental settings that are useful to capture certain types of molecules. The bars from left to right for each m/z are the intensities of that m/z in each dictionary elements. The m/z values in the brackets are the true positive m/z in the ground truth dictionary element (purified surfactin).

CHAPTER 4: Composition and Decomposition of GANs

4.1 Motivation

Generative Adversarial Networks (GANs) have proven to be a successful framework for training generative models that can produce realistic samples across a variety of domains, including natural images and languages. However, existing GANs approaches mostly attempt to model a data distribution directly and fail to exploit the compositional nature inherent in many data distributions. Many data distributions consist of different components that are mixed through some composition process. For example, natural scenes often include various objects, composed via some combinations of translating, scaling, rotation, occlusion, etc. Our framework applying to GANs should enable it to model this compositional nature of data distributions.

4.2 Compositional/Decompositional Framework of GANs

The framework is consists of three components: part generators, a composition operation, and a decomposition operation. Here we define each component under the context of GANs.

1. **Part generators** $g_i(\boldsymbol{z}_i)$ A part generator is a standard generator in GANs. It maps some noise vector $\boldsymbol{z}$ sampled from a simple distribution, e.g., standard normal distribution, to a component sample. We assume there are m part generators, from g_1 to g_m. Let $o_i := g_i(\boldsymbol{z}_i)$ be the output for part generator i.

2. **Composition operation** $(c : (\mathbb{R}^n)^m \to \mathbb{R}^n)$ A function which composes m inputs of dimension n, that come from the part generators to a single output (composed sample).

3. **Decomposition operation** $(d : \mathbb{R}^n \to (\mathbb{R}^n)^m)$ A function which decomposes one input of dimension n, that is a composed sample, to m outputs (components). We denote the i-th output of the decomposition function by $d(\cdot)_i$. Note that the decomposition operation is the inversed operation of the composition operation.

Without loss of generality, we will assume that the composed sample has the same dimensions as each of its components.

This compositional framework provides additional modularity to the original GANs because part generators in the framework are separated and allowed to be swapped or be reused by other compositional models. Also, this framework will enable us to explicitly incorporating the prior information about the compositional structure of the data, so that we assign a high-level meaning for each part generator. Lastly, given enough prior information, we can learn the part generator from composed data directly. We perform experiments with different amount of prior information and derive sufficient conditions under which our compositional models are identifiable.

Below we present an applications this framework.

4.2.1 Application: Image with Foreground Object(s) on a Background

In this setting, we assume that each image consists of one or more foreground object over a background. In this case, m$\leq$ 2, one part generator is responsible for generating the background, and other part generators are accountable for their foreground objects.

An example is shown in figure 4.1. Here the foreground object is a single MNIST digit, and the composition operation takes a uniform background and overlays the digit over the background. The decomposition function takes a composed image and outputs the foreground digit and the uniform background.

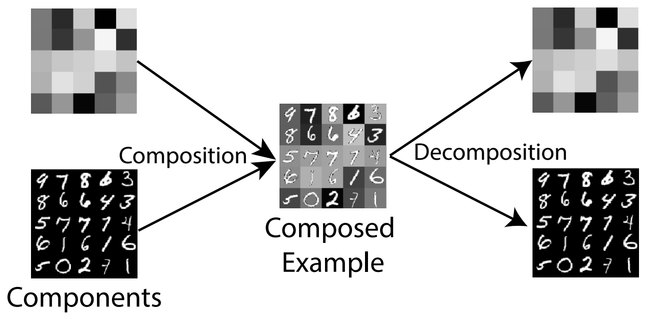


Figure 4.1: An example of composition and decomposition for the application on images

4.3 Loss Functions

As mentioned in chapter 2, we applied Wasserstein GAN with gradient penalty (WGAN-GP) in this work; thus, all the adversarial losses discussed below is the type of WGAN-GP adversarial loss. We choose WGAN-GP adversarial loss due to its simplicity in its formulation All theoretical results also hold for other adversarial losses.

Here We define the data terms used in the loss function. Let $\mathbf{x}_1$ be a part sample. There are m such samples. Let $\mathbf{y}$ be a composite sample be obtained by composition of all parts $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m$. We denote vector L^1 norm by $\|\mathbf{a}\|_1$ ($\|\mathbf{a}\|_1 = \sum_i |a_i|$). Finally, we use capital D to denote discriminators involved in different losses.

Part Generator Adversarial Loss ($l_{\mathbf{g_i}}$) Given the part samples, we can train the part generator ($\mathbf{g}_i$) to match the distribution of the part samples using this loss:

$$l_{\mathbf{g_i}} \equiv \mathbb{E}_{\mathbf{x}_i \sim p_{\text{data}}(\mathbf{x}_i)}[D_i(\mathbf{x}_i)] - \mathbb{E}_{\mathbf{z}_i \sim p_{\mathbf{z}_i}}[D_i(\mathbf{g_i}(\mathbf{z}_i))].$$

Composition Adversarial Loss ($l_{\mathbf{c}}$) Given the part generators and composite samples, we can train a composition operation ($\mathbf{c}$) such that the distribution of the generated composite samples matches the distribution of the real composite samples using this loss:

$$l_{\mathbf{c}} \equiv \mathbb{E}_{\mathbf{y} \sim p_{\text{data}}(\mathbf{y})}[D_c(\mathbf{y})] - \mathbb{E}_{\mathbf{z}_1 \sim p_{\mathbf{z}_1}, ..., \mathbf{z}_m \sim p_{\mathbf{z}_m}}[D_c(\mathbf{c}(\mathbf{o}_1, ..., \mathbf{o}_m))]$$

Decomposition Adversarial Loss ($l_{\mathbf{d}}$) Given the part generators and composite samples, we can train a decomposition operation ($\mathbf{d}$) such that the distribution of the decompositions of the composite samples matches the distribution of the generated part samples using this loss:

$$l_{\mathbf{d}} \equiv \mathbb{E}_{\mathbf{z}_1 \sim p_{\mathbf{z}_1}, ..., \mathbf{z}_m \sim p_{\mathbf{z}_m}}[D_f(\mathbf{o}_1, ..., \mathbf{o}_m)] - \mathbb{E}_{\mathbf{y} \sim p_{\text{data}}(\mathbf{y})}[D_f(\mathbf{d}(\mathbf{y}))].$$

Composition/Decomposition Cycle Losses ($l_{\mathbf{c-cyc}}, l_{\mathbf{d-cyc}}$) Additionally, we adopt the cyclic consistency loss (Zhu et al. (2017)) to encourage composition and decomposition functions to be inverses of each other.

$$\begin{aligned} l_{\mathbf{c-cyc}} &\equiv \mathbb{E}_{\mathbf{z}_1 \sim p_{\mathbf{z}_1}, ..., \mathbf{z}_m \sim p_{\mathbf{z}_m}}\left[\sum_i \|\mathbf{d}(\mathbf{c}(\mathbf{o}_1, ..., \mathbf{o}_m))_i - \mathbf{o}_i\|_1\right] \\ l_{\mathbf{d-cyc}} &\equiv \mathbb{E}_{\mathbf{y} \sim p_{\text{data}}(\mathbf{y})}\left[\|\mathbf{c}(\mathbf{d}(\mathbf{y})) - \mathbf{y}\|_1\right] \end{aligned}$$

Table 4.1 summarizes all the losses. Training of discriminators (D_i, D_c, D_f), composition operation ($\mathbf{c}$), decomposition operation ($\mathbf{d}$), and part generators $\mathbf{g_i}, i = [1, ..., m]$ is achieved by maximization of their

Table 4.1: Table for all losses

Loss name	**Detail**
$l_{\mathbf{g_i}}$	$\mathbb{E}_{\mathbf{x}_i \sim p_{\text{data}}(\mathbf{x}_i)}[D_i(\mathbf{x}_i)] - \mathbb{E}_{\mathbf{z}_i \sim p_{\mathbf{z}}}[D_i(\mathbf{g_i}(\mathbf{z}_i))]$
$l_{\mathbf{c}}$	$\mathbb{E}_{\mathbf{y} \sim p_{\text{data}}(\mathbf{y})}[D_c(\mathbf{y})] - \mathbb{E}_{\mathbf{z}_1 \sim p_{\mathbf{z}_1}, \ldots, \mathbf{z}_m \sim p_{\mathbf{z}_m}}[D_c(\mathbf{c}(\mathbf{o}_1, ..., \mathbf{o}_m))]$
$l_{\mathbf{d}}$	$\mathbb{E}_{\mathbf{z}_1 \sim p_{\mathbf{z}_1}, \ldots, \mathbf{z}_m \sim p_{\mathbf{z}_m}}[D_f(\mathbf{o}_1, ..., \mathbf{o}_m)] - \mathbb{E}_{\mathbf{y} \sim p_{\text{data}}(\mathbf{y})}[D_f(\mathbf{d}(\mathbf{y}))]$
$l_{\mathbf{c-cyc}}$	$\mathbb{E}_{\mathbf{z}_1 \sim p_{\mathbf{z}_1}, \ldots, \mathbf{z}_m \sim p_{\mathbf{z}_m}}[\sum_i \lVert \mathbf{d}(\mathbf{c}(\mathbf{o}_1, ..., \mathbf{o}_m))_i - \mathbf{o}_i \rVert_1]$
$l_{\mathbf{d-cyc}}$	$\mathbb{E}_{\mathbf{y} \sim p_{\text{data}}(\mathbf{y})}[\lVert \mathbf{c}(\mathbf{d}(\mathbf{y})) - \mathbf{y} \rVert_1]$

respective losses. We use the following composite loss in all experiments:

$$l_{\mathbf{c}} + l_{\mathbf{d}} + \alpha(l_{\mathbf{c-cyc}} + l_{\mathbf{d-cyc}}),$$

where $\alpha \geq 0$ controls the importance of consistency. While the loss function is the same for the tasks, the parameters to be optimized are different. In each task, only the parameters of the trained networks are optimized.

4.4 Network architecture and Training Algorithm

As mentioned in chapter 2, we follow the network architecture of DCGAN (Radford et al., 2015). The discriminators for all loss functions follow the architecture of DCGAN (DCGAN discriminator, DCGAN-disc), and the part generators also follow the architecture of DCGAN (DCGAN generator, DCGAN-gen). For the composition and decomposition, we applied the encoder-decoder network and U-Net. The two networks have similar architectures, and we found the two networks do not make a significant difference in our experiments. Besides the network architecture, we use different neuron numbers for different experiments because more complex dataset needs more neurons to model. We will show these numbers under the experiment section. Concerning training GANs, we applied the algorithm in (Heusel et al., 2017), that update the generator and the discriminator alternatively with one iteration but different learning rate. We use the version of gradient penalty (Petzka et al., 2017) and set $\lambda = 5$ according to its default. Concerning optimization algorithms, we use Adam algorithm (Kingma and Ba, 2014) because it has been widely applied when training GANs, and we use the default hyperparameters suggested by the authors.

4.5 Different Tasks with Different Prior Information Given

Under this composition/decomposition framework, we focus on a set of tasks that give different amount of prior information of data and train the components that do not have any prior information.

Task 1: given part generators $\mathbf{g_i}, i \in \{1, \ldots, m\}$ and $\mathbf{c}$, train $\mathbf{d}$.

Task 2: given part generators $\mathbf{g_i}, i \in \{1, \ldots, m\}$, train $\mathbf{d}$ and $\mathbf{c}$.

Task 3: given part generators $\mathbf{g_i}, i \in \{1, \ldots, m-1\}$ and $\mathbf{c}$, train $\mathbf{g_m}$ and $\mathbf{d}$.

Task 4: given $\mathbf{c}$, train all $\mathbf{g_i}, i \in \{1, \ldots, m\}$ and $\mathbf{d}$

The more prior information is given, the easier the tasks. The tasks listed above increase in difficulty. In task 1, all components in the framework except the decomposition operation are provided. In task 2 and task 3, most of the components in the framework are provided, and we need to learn two components together. In task 4, only composition operation is provided, and we need to learn the rest of the components in the framework.

4.6 Experiment Results

4.6.1 Datasets

1. **MNIST-MB** MNIST digits (LeCun et al., 1998) are superimposed on a monochromic one-color-channel background (value ranged from 0-200) (figure 4.2). The image size is 28 x 28.

2. **MNIST-BB** MNIST digit are rotated and scaled to fit a box of size 32 x 32 placed on a monochrome background of size 64 x 64. The box is positioned in one of the four possible locations (top-right, top-left, bottom-right, bottom-left), with rotation between $(-\pi/6, \pi/6)$ (figure 4.3).

3. **fashion-MNIST-MB** fashion-MNIST is a dataset that focues on fashion items and has similar style of MNIST (Xiao et al., 2017). The images are also superimposed on a monochromic one-color-channel background (value ranged from 0-200) (figure 4.4). The image size is 28 x 28.

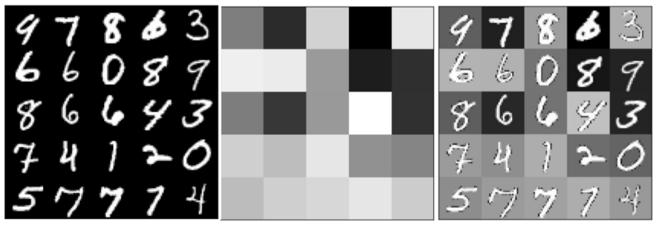


Figure 4.2: Examples of MNIST-MB dataset. 5x5 grid on the left shows examples of MNIST digits (first part), middle grid shows examples of monochromatic backgrounds (second part), grid on the right shows examples of composite images.

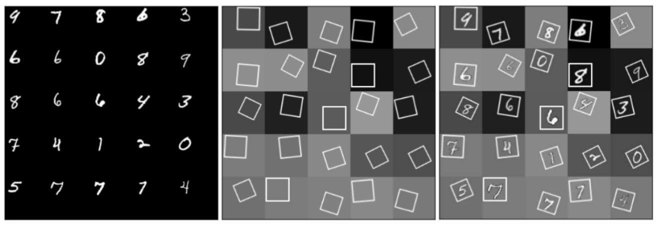


Figure 4.3: Examples of MNIST-BB dataset. 5x5 grid on the left shows examples of MNIST digits (first part), middle grid shows examples of monochromatic backgrounds with shifted, rotated, and scaled boxes (second part), grid on the right shows examples of composite images with digits transformed to fit into appropriate box.

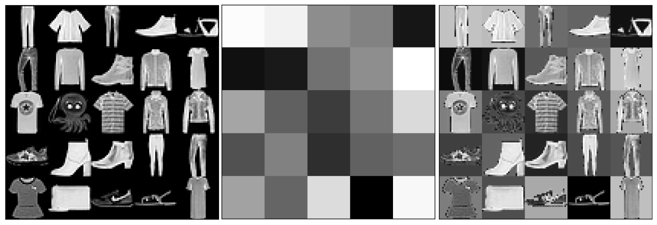


Figure 4.4: Examples of fashion-MNIST-MB dataset. 5x5 grid on the left shows examples of fasion-MNIST item (first part), middle grid shows examples of monochromatic backgrounds (second part), grid on the right shows examples of composite images.

4.6.2 Results

4.6.3 Experiments on MNIST-MB

Throughout this section we assume that composition operation is known and given by

$$c(\mathbf{o}_1, \mathbf{o}_2)_i = \begin{cases} o_{1,i} & \text{if } o_{2,i} = 0 \\ o_{2,i} & \text{otherwise.} \end{cases}$$

The composition operation is learned from the data. In tasks where one or more generators are given, the generators have been independently trained using corresponding adversarial loss $l_{\mathbf{g_i}}$. The network architectures used in this dataset are as follows. The composition and decomposition networks are both encoder-decoder networks with $k = 32$. The discriminators and part generators are DCGAN-disc and DCGAN-gen network with $k = 32$.

Task 1: given $\mathbf{g_i}, i \in \{1, 2\}$ and c, train $\mathbf{d}$. This is the simplest task in the framework. The decomposition network learns to decompose the digits and backgrounds correctly (figure 4.5) given c and pre-trained generative models for both digits and background components.

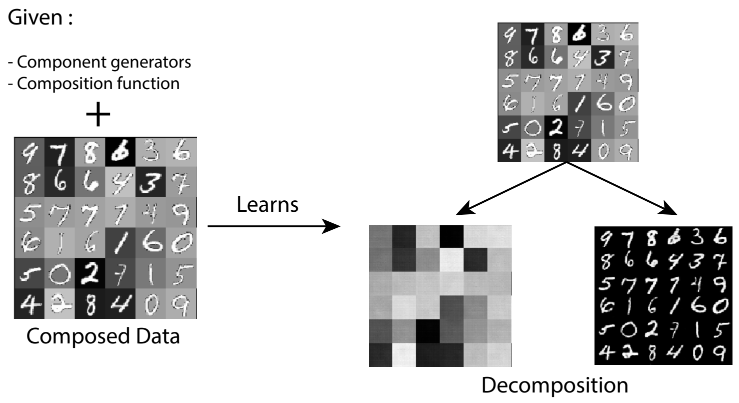


Figure 4.5: Given component generators and composite data, decomposition can be learned.

Task 2: given $\mathbf{g_1}$ and $\mathbf{g_2}$ train $\mathbf{d}$ and $\mathbf{c}$. Here we learn composition and decomposition jointly (figure 4.6). We find that the model learns to decompose digits accurately; interestingly however, we note that backgrounds from decomposition network are inverted in intensity ($\mathbf{d}(b) = 255 - b$) and that the model learns to undo this inversion in the composition function ($\mathbf{c}(\mathbf{d}(b)) = b$) so that cyclic consistency ($\mathbf{d}(\mathbf{c}(\mathbf{o}_1, \mathbf{o}_2)) \approx [\mathbf{o}_1, \mathbf{o}_2]$ and $\mathbf{c}(\mathbf{d}(\mathbf{y})) \approx \mathbf{y}$ are satisfied. We note that this is an interesting case where symmetries in the background distributions results in the model learning the background distribution only up to a phase flip.

Task 3: given $\mathbf{g_1}$ and $\mathbf{c}$, train $\mathbf{g_2}$ and $\mathbf{d}$. Here we train decomposition network and background generator with digit generator and composition network given to us (figure 4.7). We see that decomposition network learns to generate nearly uniform backgrounds, and the decomposition network learns to draw the occluded backgrounds.

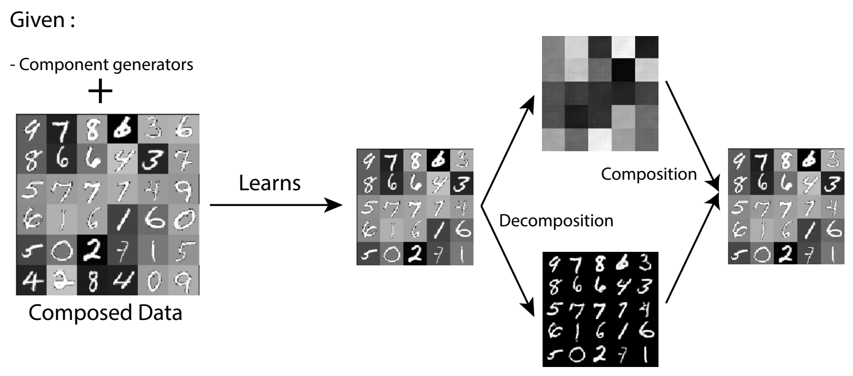


Figure 4.6: Training composition and decomposition jointly can lead to “incorrect” decompositions that still satisfy cyclic consistency. Results from the composition and decomposition network. We note that decomposition network produces inverted background (compare decomposed backgrounds to original), and composition network inverts input backgrounds during composition (see backgrounds in re-composed image). Consequently decomposition and composition perform inverse operations, but do not correspond to the way the data was generated.

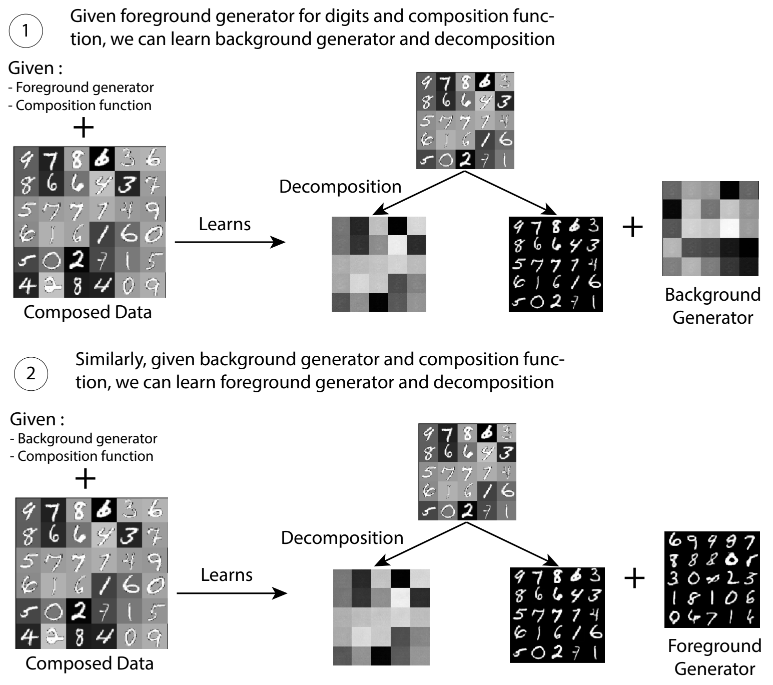

1
Given foreground generator for digits and composition function, we can learn background generator and decomposition
Given :
- Foreground generator
- Composition function
+
Composed Data
Learns
Decomposition
+
Background Generator
2
Similarly, given background generator and composition function, we can learn foreground generator and decomposition
Given :
- Background generator
- Composition function
+
Composed Data
Learns
Decomposition
+
Foreground Generator

Figure 4.7: Given one part distribution, decomposition function and the other part distribution can be learned.

FID evaluation In Table 4.2, we illustrate the performance of the generators trained using the setting of Task 3, compared to baseline monolithic models which are not amenable to decomposition. Concerning the foreground data, training the foreground generators with the composed data has a slightly worse FID (~ 9.74) than training with the foreground data directly ~ 6.62. However, concerning the composed data,

Methods	Foreground		Foreground+background	
	Digits	Fashion	Digits	Fashion
WGAN-GP	6.622 ± 0.116	20.425 ± 0.130	25.871 ± 0.182	21.914 ± 0.261
By decomposition	9.741 ± 0.144	21.865 ± 0.228	13.536 ± 0.130	21.527 ± 0.071

Table 4.2: We show Frechet inception distance (FID)(Heusel et al., 2017) for generators trained using different datasets and methods. The ”Foreground” column and ”Foreground+background” column reflect the performance of trained generators on generating corresponding images. WGAN-GP is trained on the foreground and the composed images directly. Generators evaluated in the ”By decomposition” row are obtained as described in Task 3 - on composed images, given background generator and composition operator. The information processing inequality (Cover and Thomas, 2012) guarantees that the resulting generator cannot beat the WGAN-GP on clean foreground data. However, the composed images are better modeled using the learned foreground generator and known composition and background generator.

training the composed generators has better FID(~ 13.54) than the original generator without composition structure. The FIDs shown here are comparable to the related work (Lucic et al., 2018); so it suggested our generative models have reasonable performance.

Task 4: given $\mathbf{c}$, train $\mathbf{g_1}, \mathbf{g_2}$ and $\mathbf{d}$. Given just composition, learn all part generators and decomposition. We show that for a simple composition function, there are many ways to assign responsibilities to different part generators. Some are trivial, for example, the whole composite image is generated by a single part generator (figure 4.8).

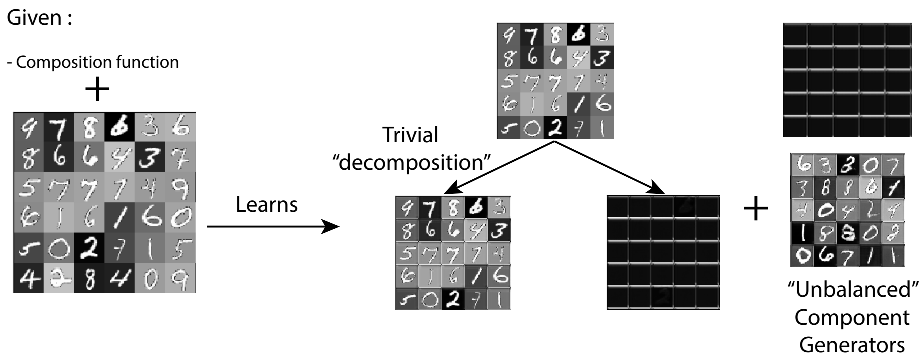


Figure 4.8: Knowing the composition function is not sufficient to learn components and decomposition. Instead, the model tends to learn a “trivial” decomposition whereby one of the component generators tries to generate the entire composed example.

4.6.4 Chain Learning - Experiments on MNIST-BB

In task 3 above, we demonstrated on MNIST-MB dataset that we can learn to model the background component and the decomposition function from composed data assuming we are given a model for the foreground component and a composition network. The success of this task suggests the natural follow-up question: if we have a new dataset consisting of a previously unseen class of foreground objects on the same distribution of backgrounds, can we then use this background model we've learned to learn a new foreground model?

We call this concept **"chain learning"**, since training proceeds sequentially and relies on the model trained in the previous stage. To make this concrete, consider this proof-of-concept chain (using the MNIST-BB dataset):

0. Train a model for the digit "1" (or obtain a pre-trained model).
1. Using the model for digit "1" from step 1 (and a composition network), learn the decomposition network and background generator from composed examples of "1"s.
2. Using the background model from step 2, learn a model for digit "2" from composed examples of "2"s.

MNIST-BB has a more complex composition operation than MNIST-MB. In this composition, the foreground digits go through a set of linear transformation: (small-scale) scaling, translating and rotation. These transformations are simple as they can be represented by matrix multiplication. But the parameters of these transformations need to be inferred from the background, that is a non-trivial operation. We learn all these operations from the data via larger networks. The composition and decomposition network are U-Net with $k = 64$. The discriminators and part generators are DCGAN-disc and DCGAN-gen network with $k = 64$. Since the dimensionality of MNIST-BB is 64×64, according to the constructing rule of DCGAN, both DCGAN-disc and DCGAN-gen have four layers. The encoder network and decoder network both have four layers, too.

As shown in figure 4.9, we can learn both the background generator (in step 1) and the foreground generator for "2" (in step 2) correctly. More generally, the ability to learn a part data model from composed data (given models for all other components) allows one to learn new part data models directly from composed data incrementally.

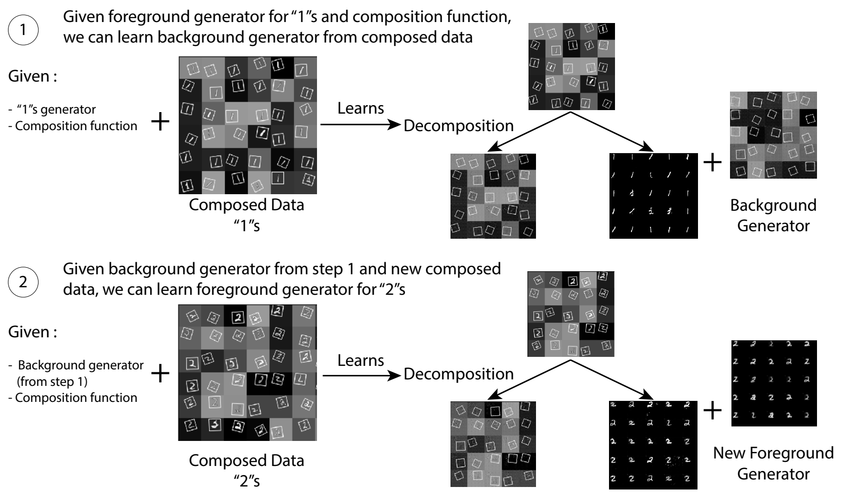

1
Given foreground generator for "1"s and composition function, we can learn background generator from composed data
Given :
- "1"s generator
- Composition function
Composed Data "1"s
Learns
Decomposition
Background Generator
2
Given background generator from step 1 and new composed data, we can learn foreground generator for "2"s
Given :
- Background generator (from step 1)
- Composition function
Composed Data "2"s
Learns
Decomposition
New Foreground Generator

Figure 4.9: Some results of chain learning on MNIST-BB. First we learn a background generator given foreground generator for "1" and composition network, and later we learn the foreground generator for digit "2" given background generator and composition network.

4.6.5 Transfer Learning - Experiments on Fashion-MNIST-MB

In this section, we train our model on a more complex dataset, fashion-MNIST-MB, where the foreground objects are various fashion items. We perform three tasks on this dataset. The first one is the learning task 3 mentioned before(learning 1 component given the other component and composition). The second one is an example of cross-domain chain learning (learning the background on MNIST-MB and using that to learn a foreground model for T-shirts from Fashion-MNIST). The third one is a transfer-learning that we do learning task 3 on the composition operation learned from MNIST-MB dataset.

The composition operation here is the same as MNIST-MB. We learn this operation from this dataset (fashion-MNIST-MB) in the first two tasks. However, we use the composition operation trained from MNIST-MB dataset in the last task. The composition and decomposition operation are both modeled as an encoder-decoder network with $k = 32$. The discriminators and part generators are DCGAN-disc and DCGAN-gen with $k = 32$.

For the first task, as before, given one component and the composition operation, we can learn the other component. The FIDs for the foreground and composed data of fashion-MNIST are shown in Table 4.2.

For the second task, it is an example of reusing part generators. We show that a background generator learned from MNIST-MB dataset can be used to learn a foreground model on the fashion-MNIST-MB dataset.

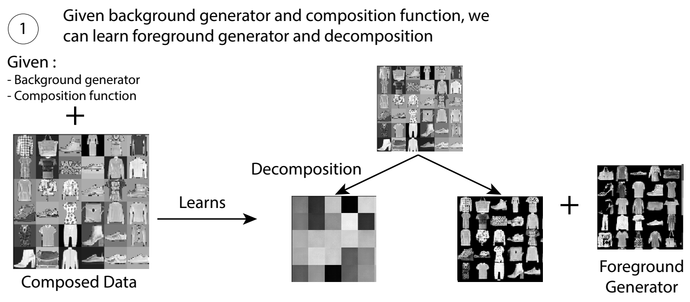


Figure 4.10: For fasion-MNIST-MB dataset, given one part data distribution and composition function, decomposition function and the other part generators can be learned.

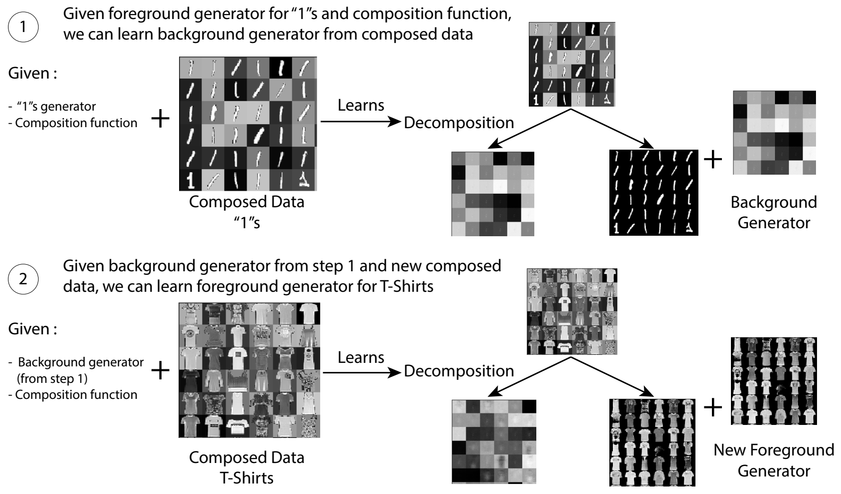

1
Given foreground generator for "1"s and composition function, we can learn background generator from composed data
Given :
- "1"s generator
- Composition function
+
Composed Data "1"s
Learns
Decomposition
+
Background Generator
2
Given background generator from step 1 and new composed data, we can learn foreground generator for T-Shirts
Given :
- Background generator (from step 1)
- Composition function
+
Composed Data T-Shirts
Learns
Decomposition
+
New Foreground Generator

Figure 4.11: Some results of chain learning on MNIST to Fashion-MNIST. First we learn a background generator given foreground generator for digit "1" and composition network, and later we learn the foreground generator for T-shirts given the background generator.

In the third task, we show that can also reuse the composition operation. We can learn a foreground generator and a decomposition operation from the composed data (fashin-MNIST-MB), given the composition operation learned from MNIST-MB dataset.

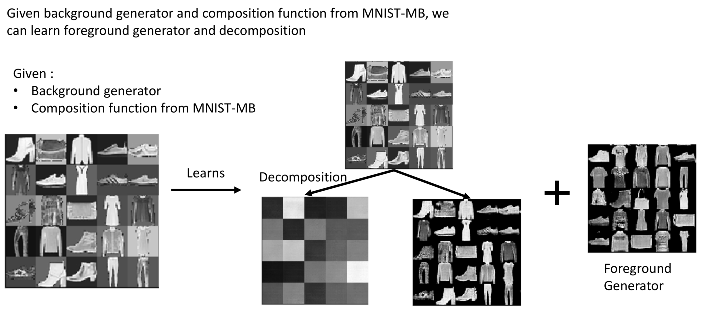

Given background generator and composition function from MNIST-MB, we can learn foreground generator and decomposition
Given :
• Background generator
• Composition function from MNIST-MB
Learns
Decomposition
+
Foreground Generator

Figure 4.12: Given one part data distribution and composition function from MNIST-MB, decomposition function and the other part generator can be learned for fashion-MNIST-MB dataset.

4.7 Identifiability Results

In the experimental section, we highlighted tasks which suffer from identifiability problems. Here we state sufficient conditions for identifiability of different parts of our framework. For simplicity, we consider the output of a generator network as a random variable and do away with explicit reference to generators. Specifically, we use random variables X and Y to refer to component random variables, and Z to a composite random variable. Let range$(\cdot)$ denote range of a random variable. We define indicator function, $\mathbb{1}[a]$ is 1 if a is true and 0 otherwise.

Definition 4.1. A **resolving matrix**, R, for a composition function c and random variable X, is a matrix of size $|\text{range}(Z)| \times |\text{range}(Y)|$ with entries $R_{z,y} = \sum_{x \in \text{range}(X)} p(X = x)\mathbb{1}[z = c(x, y)]$ (see figure 4.13).

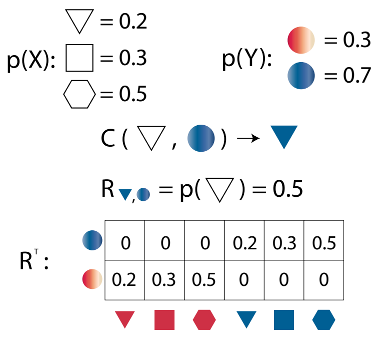


Figure 4.13: A simple example illustrating a bijective composition and corresponding resolving matrix.

Definition 4.2. A composition function c is bijective if it is surjective and there exists a decomposition function d such that

1. $d(c(x,y)) = x, y; \forall x \in \text{range}(X), y \in \text{range}(Y)$

2. $c(d(z)) = z; \forall z \in \text{range}(Z)$

equivalently, c is bijective when $c(x,y) = c(x',y')$ iff $x = x'$ and $y = y'$. We refer to decomposition function d as **inverse** of c.

In the following results, we use assumptions:

Assumption 1. X, Y are finite discrete random variables.

Assumption 2. For variables X and Y, and composition function c, let random variable Z be distributed according to

$$p(Z = z) = \sum_x \sum_y p(Y = y) p(X = x) \mathbb{1}[z = c(x,y)]. \tag{4.1}$$

Theorem 4.1. *Let Assumptions 1 and 2 hold. Further, assume that resolving matrix of X and c has full column-rank. If optimum of*

$$\inf_{p(Y')} \sup_{\|D\|_L \leq 1} \mathbb{E}_Z [D(Z)] - \mathbb{E}_{X,Y'} \left[D\left(c\left(X, Y'\right)\right)\right] \tag{4.2}$$

is achieved for some random variable Y', then Y and Y' have the same distribution.

Theorem 4.2. *Let Assumptions 1 and 2 hold. Further, assume that c is bijective. If optimum of*

$$\inf_d \mathbb{E}_{X,Y} \left[\|d(c(X,Y))_x - X\|_1\right] + \mathbb{E}_Z \left[\|d(c(X,Y))_y - Y\|_1\right] + \mathbb{E}_Z \left[\|c(d(Z)) - Z\|_1\right] \tag{4.3}$$

is 0 and it is achieved for some d' then d' is equal to inverse of c.

4.8 Identifiability proofs

We prove several results on identifiability of part generators, composition and decomposition functions as defined in the main text. These results take form of assuming that all but one object of interest are given, and the missing object is obtained by optimizing losses specified in the main text.

Let X, Y and Z denote finite three discrete random variables. Let range$(\cdot)$ denote range of a random variable. We refer to $c : \text{range}(X) \times \text{range}(Y) \to \text{range}(Z)$ as composition function, and $d : \text{range}(Z) \to$

range$(X) \times$ range(Y) as decomposition function. We define indicator function, $\mathbb{1}[a]$ is 1 if a is true and 0 otherwise.

Lemma 4.1. *The resolving matrix of any bijective composition c has full column rank.*

Proof. Let $R_{\cdot,y}$ denote a column of R. Let $d(z)_x$ denote the x part of $d(z)$, and $d(z)_y$ analogously.

Assume that:

$$\sum_y \alpha_y R_{\cdot,y} = 0 \tag{4.4}$$

or equivalently, $\forall z \in$ range(Z):

$$\begin{aligned} \sum_y \alpha_y \sum_x P(X=x)\mathbb{1}[z=c(x,y)] = 0 \\ \sum_y \alpha_y \sum_x P(X=x)\mathbb{1}[x=d(z)_x]\mathbb{1}[y=d(z)_y] = 0 \\ \alpha_{d(z)_y} P(X=d(z)_x) = 0 \end{aligned} \tag{4.5}$$

using the the definition of R in the first equality, making the substitution $\mathbb{1}[z=c(x,y)] = \mathbb{1}[x = d(z)_x]\mathbb{1}[y=d(z)_y]$ implied by the bijectivitiy of c in the second equality and rearranging / simplifying terms for the third.

Since $P(X=x) > 0$ for all $x \in$ range(X), $\alpha_y = 0$ for all $y \in \{y \mid y = d(z)_y\}$. By the surjectivity of c, $\alpha_y = 0$ for all $y \in$ range(Y), and R has full column rank. □

Theorem 4.3. *Let Assumptions 1 and 2 hold. Further, assume that resolving matrix of X and c has full column-rank. If optimum of*

$$\inf_{p(Y')} \sup_{\|D\|_L \leq 1} \mathbb{E}_Z\left[D\left(Z\right)\right] - \mathbb{E}_{X,Y'}\left[D\left(c\left(X,Y'\right)\right)\right] \tag{4.6}$$

is achieved for some random variable Y', then Y and Y' have the same distribution.

Proof. Let Z' be distributed according to

$$p(Z'=z) = \sum_x \sum_y p(Y'=y)p(X=x)\mathbb{1}[z=c(x,y)]. \tag{4.7}$$

The objective in equation 4.6 can be rewritten as

$$\underbrace{\inf_{p(Y')} \sup_{\|D\|_L \leq 1} \overbrace{\mathbb{E}_Z \left[D\left(Z\right)\right] - \mathbb{E}_{Z'}\left[D\left(Z'\right)\right]}^{W(Z,Z')}}_{C(Z')} \tag{4.8}$$

where dependence of Z' on Y' is implicit.

Following (Arjovsky et al., 2017), we note that $W(Z, Z') \to 0$ implies that $p(Z) \xrightarrow{\mathcal{D}} p(Z')$, hence the infimum in equation 4.8 is achieved for Z distributed as Z'. Finally, we observe that Z' and Z are identically distributed if Y' and Y are. Hence, distribution of Y if optimal for equation 4.6.

Next we show that there is a unique of distribution of Y' for which Z' and Z are identically distributed, by generalizing a proof by (Bora et al., 2018) For a random variable X we adopt notation p_x denote a vector of probabilities $p_{x,i} = p(X = i)$. In this notation, equation 4.1 can be rewritten as

$$p_z = R p_y. \tag{4.9}$$

Since R is of rank $|\text{range}(Y)|$ then $R^T R$ is of size $|\text{range}(Y)| \times |\text{range}(Y)|$ and non-singular. Consequently, $(R^T R)^{-1} R^T p_z$ is a unique solution of equation 4.9. Hence, optimum of equation 4.6 is achieved only Y' which are identically distributed as Y. ☐

Corollary 1. Let Assumptions 1 and 2 hold. Further, assume that c is a bijective. If, an optimum of equation 4.6 is achieved is achieved for some random variable Y', then Y and Y' have the same distribution.

Proof. Using Lemma 4.1 and Theorem 4.3. ☐

Theorem 4.4. *Let Assumptions 1 and 2 hold. Further, assume that c is bijective. If optimum of*

$$\inf_d \mathbb{E}_{X,Y}\left[\|d\left(c\left(X,Y\right)\right)_x - X\|_1\right] + \mathbb{E}_Z\left[\|d\left(c\left(X,Y\right)\right)_y - Y\|_1\right] + \mathbb{E}_Z\left[\|c(d(Z)) - Z\|_1\right] \tag{4.10}$$

is 0 and it is achieved for some d' then d' is equal to inverse of c.

Proof. We note that for a given distribution, expectation of a non-negative function – such as norm – can only be zero if the function is zero on the whole support of the distribution.

Assume that optimum of 0 is achieved but d' is not equal to inverse of c, denoted as d^*. Hence, there exists a z' such $(x', y') = d'(z') \neq d^*(z') = (x^*, y^*)$. By optimality of d', $c(d'(z')) = z'$ or the objective would be positive. Hence, $c(x', y') = z'$. By Definition 4.2, $c(d^*(z')) = z'$, hence $c(x^*, y^*) = z'$. However, $d'(c(x^*, y^*)) \neq (x^*, y^*)$ and expectation in equation 4.10 over X or Y would be positive. Consequently, the objective would be positive, violating assumption of optimum of 0. Hence, inverse of c is the only function which achieves optimum 0 in equation 4.10.

$\square$

CHAPTER 5: SUMMARY, DISCUSSION AND FUTURE WORK

5.1 Summary

5.1 Summary In this thesis, I presented a framework for generative models. The framework takes advantage of the compositional nature of many data types. The framework defines three components: part generators that generate these parts of data, a composition operation that composes different parts into a complete sample, and a decomposition operation that is a reversed operation of the composition operation. Moreover, the part generators can be identified given prior information on one or more of the components in the framework. I derive sufficient conditions for this model identifiability. Along with the theoretical proof, I conduct experiments with different amount of prior information, on different datasets. The results show my framework can take advantage of the prior information, learn part generators from composed data, and learn reasonable decomposition operations without labeled data.

I apply my framework on the traditional signal decomposition models, sparse dictionary learning (SDL), as well as a more recently proposed deep generative models, generative adversarial networks (GANs). The first application results in a novel dictionary learning method, MOLDL. MOLDL is a biconvex problem as SDL; so MOLDL shares similar optimization algorithm and properties of SDL. With MOLDL, we can easily include prior information of part generators and thus learn a better generative model. I compare my method with other signal decomposition methods: non-negative matrix factorization (NNMF) and probabilistic latent semantic analysis (PLSA), on both synthetic datasets and real datasets. The comparisons show MOLDL can discover more accurate dictionary. The prior information of the dictionary excludes the false-positive solutions. For MALDI-IMS data, we show that MOLDL has better performance than other state-of-the-art methods.

The second application to GANs results in a composition and decomposition framework of GANs. With adversarial training, we can learn the data distribution of part generators and composable data. In addition to the adversarial training losses, cycle losses are also included in the model such that the composition operation is encouraged to be reversed to the decomposition operation. Based on the framework, I further define a series of learning tasks that gives a different amount of prior information. I experiment with these learning tasks on three image datasets, showing the feasibility of each of them. Since a part generator model might not be identifiable with the composed data and insufficient prior information, I also derive sufficient conditions for identifiability. Moreover, I show experimentally how much prior information is needed to identify the model. Compared to other GANs models, I show the foreground generators trained from the composed data have similar performance to the generators trained with foreground data, given one additional prior information, composition operation. I also show our composable generative models are better than their non-composable generative counterparts. Last but not least, I show transfer learning is feasible under the framework. In the transfer learning scenario, we can learn the composition operation and the background part generators from one dataset and use it on other datasets.

The contributions of this thesis can be summarized as follows. First, I propose a composition and decomposition framework for generative models that allows us to incorporate prior information about data easily. The framework has been applied to two learning methods: sparse dictionary learning (SDL) and generative adversarial networks(GANs). By applying my framework on SDL, we can learn a generative model that enable us to decompose the mass signals more accurate than existing methods. For GANs, my composition/decomposition GAN framework learns several part generators that are responsible for different parts of the data. This ”division of responsibility” makes the part generators easy to control and understand. Also, I show both theoretically and experimentally how much prior information is needed to identify different components of the framework. Moreover, I show the composable generators have better generation quality (by FID score) than the non-composable generators. Lastly, I propose two use cases that show transfer learning is feasible under this framework.

5.2 Discussion

An essential ingredient for the success of deep neural networks is network architectures. DCGAN discriminator and DCGAN generator are robust architectures for GANs. However, there are no such robust architectures for composition or decomposition operations. Related works use the architectures like encoder-decoder, U-Net, or a network combing relative appearance flow network (RAFN) and spatial transformer network (STN). In my experiments, I find that encoder-decoder can learn the translation and rotation, while STN fails because of the large translation in the data making the loss hard to propagate back. Still, STN might be useful when the translation is small. The encoder-decoder network alone might be sufficient to learn the common composition function for natural images, but we might further boost the performance by combing both encoder-decoder and transformer networks. On the other hand, U-Net, or other skip-connections architectures might be useful for learning the part generators with the composed data. Since the losses for the part generators are passing from the discriminator to the composition network and down to the part generators, the losses go through many layers. It has been shown that skip connections are necessary for deep neural networks (He et al., 2016). In my experiments, I find U-Net has slightly better performance than encoder-decoder architectures. All in all, I think that exploration of network architectures including the transformation-related networks and skip-connection should enable us to find more robust architectures for our framework.

A possible extension to my framework is to make the composition/decomposition operation also generators that capture conditional distributions. There are pros and cons to this extension. The advantage is the composition generator now generates several possible composed images for a fixed set of inputs. This powerful composition is useful when there are several reasonable compositions for given inputs. For example, for a system with two foreground objects A and B, it is possible that object A occludes object B or the other way around. Both cases are reasonable, and a composition generator can capture both variations. Broadly speaking, when part generators are fixed, and there are other variations in the dataset, the ability to captures these variations might be very useful. On the other hand, this non-deterministic behavior of composition/decomposition generators breaks the cycle consistency. The cycle consistency is essential for correct mappings between two domains (in our case these are parts of samples and composed samples). With this extension, the cycle consistency in my framework needs to be modified.

We need enough prior information for the proposed framework. This prior information includes the composition operation and some part generators to identify the remaining components in the framework. In practice, this prior information might be hard to get for a dataset. Transfer learning might be helpful in this case. In my experiments, I show a simple composition operation, as well as simple background generators, can be learned in one dataset, and later be used in another. Since it is easier to transfer simple models, my framework that encourages ”division of responsibility” might be helpful for transfer learning. Further experiments of transfer learning are needed to prove this point. In addition to providing the prior information like background and foreground data directly, other helpful prior information includes class labels, attributes, text descriptions of images. This information can be embedded and included in my framework.

This information should be useful for identifying part generators. For example, given all forty attributes and five landmarks of human faces, e.g., CelebA dataset ((Liu et al., 2015)), we might be able to learn the face generators that generate all kinds of faces in the dataset.

5.3 Future Work

Since most data has composable/decomposable nature, taking advantage of this nature in the modeling results in a more accurate model or a model with fewer parameters. Also, an intuitive way to have basic building blocks is to learn part generative models for the simple form of the data. After we have these part models, we can build a more complex model based on these part models. Similar to software engineering, the simple part models are easier to understand and maintain. This thesis proposes a composable/decomposable framework that makes uses of this property. At the same time, the thesis shows the ability and limitation of this framework. In this section, I list a few directions that may be interesting to explore. Some directions result from the limitation of the current method; others are not from the limitation but are the works that could be done to make the thesis more convincing.

Comparison of composed generators and non-composed generators. A comprehensive comparison for composed generators from our framework and noncomposed generators from original GANs on various datasets provides more evidence that composed generators have better performance than the non-composed generators. At the same time, we can conduct experiments with the different number of model parameters to see the correlation between the parameter number, the composable models' performance, and the decomposable models' performance.

Extending the composition/decomposition network to conditional generators. The composition/decomposition networks do not learn distributions but learn to output the mode of data. Learning the distribution is useful for both operations. (Yang et al., 2018) proposed a method to learn conditional generators. This method can be adapted to learn a composition/decomposition generator conditioned on its inputs. At the same time, the cycle consistency term in my framework needs to be modified to address the non-deterministic nature. A possible modification is using a conditional discriminator (Mirza and Osindero, 2014) to discriminate whether it has a good consistency or not.

Conducting experiments on more than one foreground. Though our framework allows more than one foreground part generators, our experiments are limited to just one foreground. Since in many use cases, there are more than one foreground objects in an image, it is useful to see if the framework works on multiple foregrounds. The case of multiple foregrounds might have occlusions for the foregrounds. And it might be impossible to learn the foreground that is always occluded. This means we might derive additional sufficient conditions for model identifiability.

Conducting transfer learning experiments on more complex datasets. Complex datasets include the one with complex foreground objects or complex composition operation, e.g., fashion-MNIST object composed with a bounding box background, or shapenet dataset having 3D object models (Chang et al., 2015), or complex composition operation involving changes both foregrounds/backgrounds non-linearly. With these experiments, we can assure that transfer learning works in more general cases.

Taking labels, attributes, and text description as prior information and incorporating into our framework. Since labels, attributes and text descriptions are very common prior information among image datasets, having a correct way to incorporate these is very useful. There are different methods of incorporating them into GANs model (Odena et al., 2016; Reed et al., 2016; Miyato and Koyama, 2018; Mirza and Osindero, 2014), and finding out a proper way to use the information for our framework is essential.

BIBLIOGRAPHY

Aharon, M., Elad, M., Bruckstein, A., et al. (2006). K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on signal processing*, 54(11):4311.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

Azadi, S., Pathak, D., Ebrahimi, S., and Darrell, T. (2018). Compositional gan: Learning conditional image composition. *arXiv preprint arXiv:1807.07560*.

Bentler, P. M. and Weeks, D. G. (1980). Linear structural equations with latent variables. *Psychometrika*, 45(3):289–308.

Bora, A., Price, E., and Dimakis, A. G. (2018). Ambientgan: Generative models from lossy measurements. In *International Conference on Learning Representations (ICLR)*.

Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., et al. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122.

Byrd, R. H., Schnabel, R. B., and Shultz, G. A. (1987). A trust region algorithm for nonlinearly constrained optimization. *SIAM Journal on Numerical Analysis*, 24(5):1152–1170.

Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., et al. (2015). Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*.

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016). Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180.

Comon, P. (1994). Independent component analysis, a new concept? *Signal processing*, 36(3):287–314.

Congedo, M., Gouy-Pailler, C., and Jutten, C. (2008). On the blind source separation of human electroencephalogram by approximate joint diagonalization of second order statistics. *Clinical Neurophysiology*, 119(12):2677–2686.

Cover, T. M. and Thomas, J. A. (2012). *Elements of information theory*. John Wiley & Sons.

Donahue, C., Lipton, Z. C., Balsubramani, A., and McAuley, J. (2017). Semantically decomposing the latent spaces of generative adversarial networks. *arXiv preprint arXiv:1705.07904*.

Donoho, D. L. (2006). For most large underdetermined systems of linear equations the minimal l1-norm solution is also the sparsest solution. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 59(6):797–829.

Du, P., Kibbe, W. A., and Lin, S. M. (2006). Improved peak detection in mass spectrum by incorporating continuous wavelet transform-based pattern matching. *Bioinformatics*, 22(17):2059–2065.

Engan, K., Aase, S. O., and Husoy, J. H. (1999). Method of optimal directions for frame design. In *Acoustics, Speech, and Signal Processing, 1999. Proceedings., 1999 IEEE International Conference on*, volume 5, pages 2443–2446. IEEE.

Fedus, W., Goodfellow, I., and Dai, A. M. (2018). Maskgan: Better text generation via filling in the _. *arXiv preprint arXiv:1801.07736*.

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.

Gorski, J., Pfeuffer, F., and Klamroth, K. (2007). Biconvex sets and optimization with biconvex functions: a survey and extensions. *Mathematical methods of operations research*, 66(3):373–407.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777.

Harn, Y.-C., Powers, M., Shank, E. A., and Jojic, V. (2015). Deconvolving molecular signatures of interactions between microbial colonies. *Bioinformatics*, 31(12):i142–i150.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637.

Hyvärinen, A. and Oja, E. (2000). Independent component analysis: algorithms and applications. *Neural networks*, 13(4-5):411–430.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. *arXiv preprint*.

Karas, M. and Krüger, R. (2003). Ion formation in maldi: the cluster ionization mechanism. *Chemical reviews*, 103(2):427–440.

Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2017). Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Koniusz, P., Yan, F., Gosselin, P.-H., and Mikolajczyk, K. (2017). Higher-order occurrence pooling for bags-of-words: Visual concept detection. *IEEE transactions on pattern analysis and machine intelligence*, 39(2):313–326.

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.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

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.

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A. P., Tejani, A., Totz, J., Wang, Z., et al. (2017). Photo-realistic single image super-resolution using a generative adversarial network. In *CVPR*, volume 2, page 4.

Lee, H., Battle, A., Raina, R., and Ng, A. Y. (2007). Efficient sparse coding algorithms. In *Advances in neural information processing systems*, pages 801–808.

Lin, C.-H., Yumer, E., Wang, O., Shechtman, E., and Lucey, S. (2018). St-gan: Spatial transformer generative adversarial networks for image compositing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9455–9464.

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

Lucic, M., Kurach, K., Michalski, M., Gelly, S., and Bousquet, O. (2018). Are gans created equal? a large-scale study. In *Advances in neural information processing systems*, pages 697–706.

Ma, S. and Dai, Y. (2011). Principal component analysis based methods in bioinformatics studies. *Briefings in bioinformatics*, 12(6):714–722.

Mairal, J., Bach, F., Ponce, J., and Sapiro, G. (2009). Online dictionary learning for sparse coding. In *Proceedings of the 26th annual international conference on machine learning*, pages 689–696. ACM.

McDonnell, L. A. and Heeren, R. M. (2007). Imaging mass spectrometry. *Mass spectrometry reviews*, 26(4):606–643.

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

Miyato, T. and Koyama, M. (2018). cgans with projection discriminator. *arXiv preprint arXiv:1802.05637*.

Ng, A. Y. and Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems*, pages 841–848.

Odena, A., Olah, C., and Shlens, J. (2016). Conditional image synthesis with auxiliary classifier gans. *arXiv preprint arXiv:1610.09585*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Petzka, H., Fischer, A., and Lukovnicov, D. (2017). On the regularization of wasserstein gans. *arXiv preprint arXiv:1709.08894*.

Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., and Lee, H. (2016). Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*.

Schmidt, M. N., Larsen, J., and Hsiao, F.-T. (2007). Wind noise reduction using non-negative sparse coding. In *Machine Learning for Signal Processing, 2007 IEEE Workshop on*, pages 431–436. IEEE.

Sra, S. and Dhillon, I. S. (2006). Generalized nonnegative matrix approximations with bregman divergences. In *Advances in neural information processing systems*, pages 283–290.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

Villani, C. (2008). *Optimal transport: old and new*, volume 338. Springer Science & Business Media.

Wächter, A. and Biegler, L. T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57.

Wolff, M. and Stephens, W. (1953). A pulsed mass spectrometer with time dispersion. *Review of Scientific Instruments*, 24(8):616–617.

Wright, J., Ma, Y., Mairal, J., Sapiro, G., Huang, T. S., and Yan, S. (2010). Sparse representation for computer vision and pattern recognition. *Proceedings of the IEEE*, 98(6):1031–1044.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

Xu, Q., Huang, G., Yuan, Y., Guo, C., Sun, Y., Wu, F., and Weinberger, K. Q. (2018). An empirical study on evaluation metrics of generative adversarial networks. *CoRR*, abs/1806.07755.

Yang, D., Hong, S., Jang, Y., Zhao, T., and Lee, H. (2018). Diversity-sensitive conditional generative adversarial networks.

Yang, J., Kannan, A., Batra, D., and Parikh, D. (2017). Lr-gan: Layered recursive generative adversarial networks for image generation. *arXiv preprint arXiv:1703.01560*.

Yang, J. Y., Phelan, V. V., Simkovsky, R., Watrous, J. D., Trial, R. M., Fleming, T. C., Wenter, R., Moore, B. S., Golden, S. S., Pogliano, K., et al. (2012). Primer on agar-based microbial imaging mass spectrometry. *Journal of bacteriology*, 194(22):6023–6028.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint*.

Zubair, S., Yan, F., and Wang, W. (2013). Dictionary learning based sparse coefficients for audio classification with max and average pooling. *Digital Signal Processing*, 23(3):960–970.