ABSTRACT

HUANG, YUMING. Training and Network Application of Graphical Models. (Under the direction of Hamid Krim.)

While deep neural network learning has seen great advancement in recent years, there are still many problems where the sampled data have graphical structures, and the particular setting where graphical models are necessary tools. Our objectives in this dissertation are to extract noisy structural information by graphical modelling of networks, and to seek modelling small and sparse data by developing a practical training framework for a general graphical model. To this end, the dissertation proceeds along two parts.

In the first part, we investigate the widely encountered problem of detecting communities in multiplex networks, such as social networks, with an unknown arbitrary heterogeneous structure. To improve detectability, we propose a generative model that leverages the multiplicity of a single community in multiple layers, with no prior assumption on the relation of communities among different layers. Our model relies on a novel idea of incorporating a large set of generic localized community label constraints across the layers, in conjunction with the well-known Stochastic Block Model (SBM) in each layer. Accordingly, we build a probabilistic graphical model over the entire multiplex network by treating the constraints as Bayesian priors. We mathematically prove that these constraints/priors promote existence of identical communities across layers without introducing further correlation between individual communities. The constraints are further tailored to render a sparse graphical model and the numerically efficient belief propagation algorithm is subsequently employed. We further demonstrate by numerical experiments that in the presence of consistent communities between different layers, consistent communities are identified, and the detectability is improved over a single layer. We compare our model with a "correlated model" which exploits the prior knowledge of community correlation between layers. Similar detectability improvement is obtained under such a correlation,
even though our model relies on much milder assumptions than the correlated model. Our model shows a better detection performance than the correlated model over a certain correlation and signal to noise ratio (SNR) range. In the absence of community correlation, the correlation model naturally fails, while ours maintains its performance.

In the second part, we present a novel adversarial framework for training deep belief networks (DBNs), which includes replacing the generator network in the methodology of generative adversarial networks (GANs) with a DBN and developing a highly parallelizable numerical algorithm for training the resulting architecture in a stochastic manner. Unlike the existing techniques, this framework can be applied to the most general form of DBNs with no requirement for back propagation. As such, it lays a new foundation for developing DBNs on a par with GANs with various regularization units, such as pooling and normalization. Foregoing back-propagation, our framework also exhibits superior scalability as compared to other DBN and GAN learning techniques. We present a number of numerical experiments in computer vision as well as neurosciences to illustrate the main advantages of our approach.

To conclude, in this dissertation, we explore both the application and the training aspects of graphical models. Specifically, in the first part, we extend the community detection model for single layer network, to describe general multiplex networks, and we show that the community detectability is improved over single layer and a naive multiplex network model. In the second part, we develop a novel adversarial training framework for deep belief networks (DBN). We show that the adversarial framework is able to successfully train DBNs to model target dataset. The trained DBNs preserve advantages over deep neural networks, and our training algorithm shows better scalability than back-propagation.
DEDICATION

To my parents.
BIOGRAPHY

Yuming Huang received his B.S. (2013) degree in Applied Physics from University of Science and Technology Beijing and M.Sc. (2015) degree in Physics from North Carolina State University. He has been a Ph.D. student in Physics in North Carolina State University since 2013. His research interests include complex systems, especially topics related to network science, as well as developing machine learning algorithms.
ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Hamid Krim for all his help, who guided me through my scientific exploration with his broad knowledge and passionate attitude. I also thank all my committee members, Dr Huaiyu Dai, Dr. Karen Daniels, and Dr Keith Weninger, as well as the Physics Department, for supporting me in my study and research.

I would also like to express gratitude to all the colleagues in the VISSTA lab, including post-doctoral researchers Dr. Ashkan Panahi and Dr. Han Wang and other Ph.D. students: Shahin Mahdizadehaghdam, Hui Guan, Wen Tang, Sid Roheda, Kenneth Tran, Sally Ghanem, Bo Jiang, Tanmay Asthana, Reza Soleimani and Mohsen Ardakani for all the insightful discussions and inspirations.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>vii</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>viii</td>
</tr>
</tbody>
</table>

## Chapter 1 Introduction

1. **Community Detection**
   1.1.1 Community Detection in Single Layer Networks
   1.1.2 Community Detection in Multiplex Networks

2. **Deep Learning**
   2.1 Deep Neural Networks
   2.2 Generative Adversarial Networks
   2.3 Deep Belief Networks

3. **Overview**

## Chapter 2 Community Detection and Improved Detectability in Multiplex Networks

2.1 **Introduction**

2.2 **Problem and method**
   2.2.1 Problem Description
   2.2.2 Bayesian Solution by Stochastic Modeling
   2.2.3 BP algorithm for multilayer community detection

2.3 **Detectability transition of constrained multiplex networks**
   2.3.1 Homogeneous multiplex network
   2.3.2 Heterogeneous multiplex network
   2.3.3 Impact of a known number of communities $q$
   2.3.4 Practical considerations and more layers

2.4 **Conclusions**

## Chapter 3 Deep Adversarial Belief Network

3.1 **Introduction**

3.2 **Mathematical Background**
   3.2.1 Problem Formulation: Training Deep Belief Networks
   3.2.2 Proposed Method: Deep Adversarial Belief Networks

3.3 **Experiments**
   3.3.1 Generation of MNIST-Like Digits
   3.3.2 Classification of MNIST
   3.3.3 Modeling Neural Activities

3.4 **Conclusion**

## Chapter 4 Conclusion and Future Works

4.1 **Community Detection**

4.1.1 **Conclusion**

v
Table 3.1 Classification accuracies when number of training samples is 100, 1000 and 60000 respectively, averaged over 100 trials, with standard deviation in the parentheses. Compared with CNN, the use of DBN as pretraining shows better generalizability for small training set.
**LIST OF FIGURES**

| Figure 2.1 | Demonstration of potential benefit (upper row) and caveat (lower one) of aggregating multiplex layers. In both rows, left hand side shows the adjacency matrices of two multiplex layers, and right hand side shows that of the aggregated layer by adding the two adjacency matrices. The upper row shows a better community quality of the aggregated network than single layer, while the lower row shows a obscured community structure (notice in the lower row that the consistent community between two layers still gets enhanced quality). |
| Figure 2.2 | An example of a two-layer network with different community structures. |
| Figure 2.3 | An example of a two-layer network with partially consistent and partially different community structures. Each ellipse represents a community. |
| Figure 2.4 | An example of a factor graph for our model. A circle is a variable node, and a square is a factor node. There are two types of factor nodes. One is within each layer, \( f_a(t_i(l), t_j(l)) = c_{t_i(l), t_j(l)} \), representing the likelihood of node \( i \) having label \( t_i(l) \) and node \( j \) having label \( t_j(l) \). Another is between layers, \( f_{\text{check}}(t_i(l), t_i(l'), t_j(l), t_j(l')) \), representing the local constraints of community labels. |
| Figure 2.5 | The red arrow shows the interlayer message from \( i(l') \) to \( i(l) \). |
| Figure 2.6 | Detectability transition curves for various \( p_{\text{same}} \) ranging from 0 to 1. The slower the normalized agreement score \( Q \) [Dec11a] drops down, the better the detectability. Data points for \( p_{\text{same}} = 0, 0.1, 0.5, 0.9, 1 \) are connected by solid lines. The results are averaged over 100 experiments. The error bars represent standard errors. |
| Figure 2.7 | Detectability transition curves for a single layer network, a correlated double layer network and a double layer network with constraint. The synthetic network is a two-layer network, where each layer is independently generated by the same SBM model such that 200 nodes form two equal communities. Nodes have the same community labels in layer 1 and 2, while the edges are different across layers, so information between two layers can be easily combined. \( p_{\text{same}} \) is 0.9, only used as the inference parameter for the correlated model. The results are averaged over 30 experiments. Error bars represent the standard error of the experiments. |
| Figure 2.8 | An example (\( \varepsilon = 0.2 \)) of the heterogeneous network generated to test the constraint multiplex model. There are in total four distinct communities. First 100 nodes in two layers form the same community, while the rest 100 nodes form three different communities in two layers. |
Figure 2.9 Fraction of trials that result in WPP-satisfying labels for different $\epsilon$. For each of the five clusters of nodes in two layers, we determine the label by the majority of node labels in that cluster. Then we check if these five labels satisfy WPP. Refer to Fig. 2.8 for synthetic network setup.

Figure 2.10 Detectability transition curves for constrained model and correlated model on heterogeneous networks. $p_{\text{same}}$ is 0.9 for the correlated model. The results are averaged over top 20 trials where most node labels satisfy WPP. Error bars represent the standard error of the trials. Refer to Fig. 2.8 for synthetic network setup.

Figure 2.11 Probabilities of nodes being assigned certain community labels (node beliefs) at $q = 2, 3$ and 4. Various community labels are represented by different colors and markers. The lower right image shows the noisy double layer network used in this test.

Figure 2.12 Ratio of trials that satisfy WPP constraint after convergence, in the parameter space of $f_{\text{check}}$ and $N_{\text{sample}}$. We identify a band in the parameter space where the constrained model have over half the chance to reach the correct point. The numerical experiment includes 50 trials for each point, and uses the same setup as the heterogeneous two-layer experiment in Section 2.3.2, when $\epsilon = 0.2$.

Figure 3.1 Top-left: random generated samples from DBN trained using the unsupervised framework of GAN. Top-right: loss function and discriminator score of real and generated samples over a number of training iterations. Bottom-left: random generated samples from the conditional DBN, where the conditional input for each row represents digit from 0 to 9. Bottom-right: a diagram of the training framework for the conditional DBN.

Figure 3.2 Left: random samples from DBN trained using the framework of WGAN. Right: loss function over a number of training iterations, with the discriminator trained by different optimizers at the same learning rate.

Figure 3.3 Left: random samples from DBN with variance reduction. Right: loss function and discriminator score of real and generated samples over a number of training iterations.

Figure 3.4 Loss function and discriminator scores of adversarial DBN without variance reduction (top) and with Rebar (bottom), as well as respective output distributions, in 10000 iterations.

Figure 3.5 Left: real firing rate of each neuron and the firing rates given by the DBN. Right: log-likelihood of the real data on the DBN during training.
Figure 3.6  Mean correlation coefficient between \( p(h^L|h^1) \) and \( \hat{p}(h^L|h^1) \) (A), mean correlation coefficient between true and predicted neural firing probability sequences (B) and mean Kullback–Leibler divergence and Jensen-Shannon divergence between \( p(h^L_i|h^1) \) and \( \hat{p}(h^L_i|h^1) \) (C,D). Input has binary values and a dimensionality of 20, to simulate neural stimulation. Each data point is averaged over 50 trials, with the error bar being the standard error. 67

Figure 3.7  Mean correlation coefficient between \( p(h^L|h^1) \) and \( \hat{p}(h^L|h^1) \) (A), mean correlation coefficient between true and predicted neural firing probability sequences (B) and mean Kullback–Leibler divergence and Jensen-Shannon divergence between \( p(h^L_i|h^1) \) and \( \hat{p}(h^L_i|h^1) \) (C,D). Input has random values between 0 and 1 and a dimensionality of 64, to simulate visual stimuli of 8 by 8 images. Each data point is averaged over 50 trials, with the error bar being the standard error. 68
Graphical models abstract the notion of conditional dependence and conditional independence between random variables, using graph-based representations. A graphical model is a Bayesian network when the model is a directed acyclic graph. As an alternative, a model is a Markov random field when the it is a undirected graph. In this dissertation, we explore graphical models in two aspects, the application to community detection problems in networks, and the training of Deep Belief Networks to capture/emulate the dynamics of neural activity in the visual cortex.


1.1 Community Detection

1.1.1 Community Detection in Single Layer Networks

Many natural and artificial systems can be abstracted as networks, i.e. sets of nodes connected by edges [New18]. Community structure is a further abstraction of mesoscale structures in networks, which usually refer to groups of densely connected nodes. The problem of community detection is to partition nodes into such groups given the network structure. Community detection has been shown to be an effective analytical tool in wide range of areas, such as social networks [Du07][Agg11], granular materials [Hua16a][Pap16][Tao18], metabolic networks [Gui05][Pal05] and brain networks [Tel16].

There are numerous community detection methods [For10]. A common class of methods is modularity optimization [New04b] [New04a]. The basic form of the modularity function calculates the difference between the number of edges within communities and the expected number:

\[ Q = \frac{1}{2m} \sum_{ij} (A_{ij} - P_{ij}) \delta(C_i, C_j), \]

where \( m \) is the mean degree of all nodes, \( A_{ij} \) is the edge between node \( i \) and \( j \), \( P_{ij} \) is the expected connectivity depending on nodes' degrees, and \( C_i \) is the community label for node \( i \). This function represents the quality of the partition as communities, and maximizing it with greedy heuristics usually yields good results.

Another class of methods are based on statistical inference. Stochastic Block Model (SBM) [Kar11] is a common type of generative graphical model for community structure. A simple SBM describes how edges form communities based on the size of the communities and the affinity (probability of connections in a community). The probability model is given by [Dec11a],

\[ P(G, \{t_i\}|\theta) = \prod_{(i,j)\in E} p_{t_i,t_j} \prod_{(i,j)\not\in E} (1 - p_{t_i,t_j}) \prod_i n_{t_i}, \]
where \( t_i \) is the community label for node \( i \), \( E \) is the set of edges in the graph \( G \), \( p_{ab} \) is the affinity between two communities, \( n_a \) is community size, and \( \theta \) represents parameters \( \{ t_i \}, \{ n_a \}, \{ p_{ab} \} \). Solving SBM with maximum likelihood is shown to have equivalence with modularity maximization [New16a].

Both classes of methods are popular in community detection research. Modularity based methods are usually preferred for numerical efficiency. However modularity suffers from resolution limit [For10] which means small communities are difficult to detect, and hyper-parameter tuning is needed to adjust the scale of community sizes to be detected. In contrast, the stochastic block model, as a Bayesian approach, attempts to give optimal community detection results regardless of the community quality and is able to infer all parameters.

The problem of community detection is closely related to statistical physics. With the stochastic block model, the configuration of community labels of nodes in a network has a direct analogy to the spin glass systems [Méz87]. There is a phase transition of community detectability and the phase diagram of the SBM is analysed in the thermodynamic limit [Dec11a]. The method used in [Dec11a] to infer community labels, known as belief propagation, is closely related to the cavity method [Méz01] developed for spin glasses.

1.1.2 Community Detection in Multiplex Networks

Multiplex networks are composed of layers of single layer networks, where these layers share the same sets of nodes but not connections. Multiplex networks have been a more comprehensive representation for many systems, to be able to model multi-relation [Sze10] and time evolution [Tel16]. Community detection problems on multiplex networks becomes an important topic and many works addresses various situations. For example, modularity is generalized for multiplex networks with interlayer couplings [Muc10]. Some works have been focused on improving the detectability of communities for multiplex networks [Tay16; Sta16; Gha16]. Theoretical and experimental study show that aggregating network layers
that have the same community structure improves community detectability [Tay16]. A more practical method identifies which layers may have the same community structure and only aggregate them [Sta16]. For temporal networks, it is reasonable to similarly assume a correlation of community structure between adjacent layers and thus improve the detectability [Gha16].

However, one of the values of multiplex networks is to preserve the heterogeneous information across layers, which is otherwise mixed in a single layer. In other words, layer aggregation will be problematic for multiplex networks with different (heterogeneous) community structures across layers.

In this dissertation, we aim to address such a general situation where communities may exist in any layer as long as they do not overlap, and a community is always defined by the same set of nodes across layers. These assumptions are reasonable for well-decomposed systems represented by multiplex networks, where any potentially overlapped communities are separated into different layers. Our goal is to improve community detectability if a community appears in at least two layers, regardless whether the rest of the layers are consistent or not. Therefore, our model is applicable for both homogeneous and heterogeneous multiplex networks. We provide a graphical modelling of this problem by generalizing the original SBM, as based on that, we can conveniently implement our model in a rigorous way.

1.2 Deep Learning

Just as networks can be graphically modelled as SBMs, for another subject, the neural spike signals, the discrete nature of them also inspired us to represent them with a type of graphical models. In this case we opt for a rather implicit model called Deep Belief Networks (DBNs), since the underlying mechanism for neural behavior is still an open problem. In this work, we especially explored a novel training framework that enables us to train a DBN
with nice properties. Our work is related many concepts in deep learning, and we introduce each of them in this section.

Deep learning is a family of machine learning methods that uses deep and hierarchical graphical structure to learn a representation for data [Goo16]. Deep learning architectures include, but are not limited to, Deep Neural Networks (DNNs) and Deep Belief Networks (DBNs). In recent years, DBN has been less favorable than DNN, due to more difficulty in training, but it still has its value in a wide range of applications [Lee09b; Lee09a; Che15]. Recent developments in the area of DNN, especially the invention of Generative Adversarial Networks (GANs), in turn let us take the advantage and develop a novel training framework for DBNs.

1.2.1 Deep Neural Networks

A standard Deep Neural Network is typically composed of a set of feed-forward layers and defines a mapping \( y = f(x) \) between input \( x \) and output \( y \). The mapping is formed by a chain of functions defined by each layer \( f(x) = f^{(N)}(f^{(N-1)}(\ldots(f^{(1)}(x)))) \), and information flows directionally from the first layer to the \( N \)th layer (therefore the name feed-forward). The most basic layer is the linear layer:

\[
f^{(i)}(x) = g(x^T W + b),
\]

where \( W \) is the matrix of weights, \( b \) is the vector of bias parameters and \( g(\cdot) \) is called the activation function that adds non-linearity to the transformation. Common choices of the activation function include the rectifier linear unit (ReLU) [Glo11] \( g(x) = \max(0, x) \) with a lower bound of 0, and the sigmoid function \( g(x) = (1 + e^{-x})^{-1} \) bounded between 0 and 1.

The goal of a DNN is to learn the mapping between input \( x \) and output \( y^* \) in the data set. A loss/cost function \( L(y^*, f(x, \theta)) \), where \( \theta \) denotes weights and bias in all layers, can be defined to describe the difference between the real output and the predicted output
from the DNN. A DNN is trained by minimizing the loss function over the weight and bias parameters in each layer.

The minimization of the loss function is usually achieved by a gradient-based method called back-propagation [Rum86]. Specifically, it starts with calculating the gradient of the loss function over parameters $\theta^{(N)}$ in the last layer

$$\frac{\partial L(y^*, f(x, \theta))}{\partial \theta^{(N)}} = \frac{\partial L(y^*, f(x, \theta))}{\partial f(x, \theta)} \frac{\partial f^{(N)}(\ldots, \theta^{(N)})}{\partial \theta^{(N)}}.$$

Then it uses the chain rule of calculus, so that the result can be reused when calculating the gradient for layer $N - 1$:

$$\frac{\partial L(y^*, f(x, \theta))}{\partial \theta^{(N-1)}} = \frac{\partial L(y^*, f(x, \theta))}{\partial \theta^{(N)}} \frac{\partial f^{(N-1)}(\ldots, \theta^{(N-1)})}{\partial \theta^{(N-1)}}.$$

Then the technique is applied repeatedly backwards the network until the first layer, and the gradients are used to update the parameters using gradient descent.

Computational cost has been a bottleneck in the development of DNNs, which is alleviated since the application of high performance hardware especially GPUs (graphics processing units). Nevertheless, since back-propagation requires a backward sequential update to all layers, the scalability is still limited. Another challenge of DNN is that a large amount of training data is usually needed to well-train a DNN. Our proposed framework attempts to address these problems.

Although the theoretical understanding of deep neural networks is still largely lacking, it is being approached from the perspective of statistical physics. For example, to understand the training behavior of DNNs, the landscape of the loss functions of DNNs is analysed, in direct analogy with the energy landscape of repulsive, finite-range ellipses [Gei19]. A phase transition between the regimes of over-parameterized and under-parameterized DNNs is identified, similar to the jamming transition [Ber11] in the repulsive ellipse system. Around the critical point of the phase transition, the Hessian of the loss function is singular and
the variance of the output of the DNN is high. This high variance is associated with the poor generalization (testing) performance of DNNs around the critical point [Gei20]. It is shown that, by taking an ensemble averaging of outputs of independently trained DNNs, the variance is reduced and the generalization performance is greatly improved.

1.2.2 Generative Adversarial Networks

The invention of generative adversarial network (GAN) [Goo14] further enables DNN to serve as a generative model, which is able to generate random realizations corresponding to the sample distribution in the training data set. GAN architecture has been greatly progressed since its invention and successfully applied in numerous data generation tasks, for example, realistic image generation [Rad15; Bro19] especially facial images [Kar18], and music generation [Yan17]. A GAN consists of two DNNs, a generator network and a discriminator network competing against each other during training. The generator network $G$ takes a random vector $z$ from a certain distribution as the input, and produces a sample $G(z) \sim p_{model}$, usually called a fake sample. The discriminator network $D$ takes either the fake sample $G(z)$ or the real sample $x \sim p_{data}$ from true data distribution as the input, and outputs the probability that the sample is real. The optimization objective of the generator is to generate fake samples that is identified as real, i.e. $D(G(z)) = 1$, while the objective of discriminator is to distinguish between real samples and fake samples, ideally $D(x) = 1$ and $D(G(z)) = 0$, as long as $p_{model} \neq p_{data}$. The training framework can be expressed as $G$ and $D$ playing a minimax game:

$$\min_G \max_D \mathbb{E}_{x \sim p_{data}}[\log(D(x))] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]$$

It is shown that the objective function measures the Jensen-Shannon (JS) divergence between $p_{data}$ and $p_{model}$ [Goo14]. If the objective is modified to measure the very relevant Kullback-Leibler (KL) divergence, minimizing it amounts to maximizing the log-likelihood
of real data on $p_{\text{model}}$. KL divergence and JS divergence between distributions $p$ and $q$ are defined as follows:

$$D_{KL}(p||q) = \int_x p(x) \log \frac{p(x)}{q(x)} \, dx$$

$$D_{JS}(p||q) = \frac{1}{2} D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2} D_{KL}(q||\frac{p+q}{2})$$

The GAN is meant to solve the problem of optimizing deep generative models, whose likelihood functions are generally intractable and hard to approximate [Goo14]. Besides, GAN’s success relates to the use of JS divergence, since minimizing JS divergence has much better convergence property than KL divergence, and minimizing KL divergence is not practical [Hus15]. Unfortunately, however, training standard GAN is still slow and unstable, and convergence is not guaranteed. One of the reasons is the difficulty of reaching Nash Equilibrium for the game between $G$ and $D$, which is greatly affected by things like the learning rate and the balance between $G$ and $D$. Another reason is that JS divergence can be discontinuous and does not effectively measure the divergence when two distributions are disjoint [Arj17].

Wasserstein GAN (WGAN) is then proposed to replace the JS divergence by the Wasserstein distance [Arj17]. Wasserstein distance is also known as Earth Mover’s distance, representing the cost of the optimal transport plan to transform a distribution to the other. It has the good properties of being continuous and differentiable, and it allows the training of $D$ to overpower that of $G$, since $D$ is always able to yield a meaningful measurement, therefore making the training more stable. The loss function to implement WGAN is as follows:

$$\min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}} [D(x)] - \mathbb{E}_{z \sim p_z} [D(G(z))],$$

where $D$ has the constraint of being a 1-Lipschitz function, meaning:

$$|D(x_1) - D(x_2)| \leq |x_1 - x_2|.$$
Several methods are proposed to enforce the Lipschitz constraint, including weight clipping [Arj17] and gradient penalty [Gul17]. Although they show much better stability and convergence speed in training than the standard GAN, these different implementations of WGAN still suffer from instability to some extent, and the problem of implementing the Lipschitz constraint is open to further investigation.

Current studies of GAN architectures are usually limited to using a DNN as the generator. In our proposed framework, we use a graphical model called Deep Belief Network instead, which we introduce next. From this perspective, our work can be interpreted as a new generalization of GAN, and it may provide different data representation capabilities.

### 1.2.3 Deep Belief Networks

There are a class of energy-based probabilistic models that follows the form of Boltzmann distribution:

\[
p(x) = \frac{1}{Z} e^{-E(x)},
\]

where \(E(x)\) is the energy function of the configuration \(x\), and \(Z\) is the partition function.

The restricted Boltzmann Machine (RBM) [Smo86] is one of the most popular graphical models the class of energy-based models. The units/neurons in a RBM form a bipartite graph. Each side of the graph has a group of unconnected neurons, called visible units and hidden units respectively. The energy function for RBM is defined as:

\[
E(v, h) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_i \sum_j v_i w_{i,j} h_j,
\]

where \(v\) and \(h\) are the states of visible units and hidden (or latent) units, \(a\) and \(b\) being bias parameters, and \(w_{i,j}\) the weight (interaction) between visible and hidden units. Although it is proven that the partition function of RBM is intractable [Lon10], the conditional probabilities \(P(h|v)\) and \(P(v|h)\) are possible to calculate. In standard RBM, both visible and hidden
units are binary-valued, which makes the computation of conditional probability simple. For example, the activation probability of hidden units, given the visible layer, writes:

\[ P(h_j = 1|v) = \sigma(b_j + \sum_i w_{i,j}v_i), \]

and similarly:

\[ P(v_i = 1|h) = \sigma(a_i + \sum_j w_{i,j}h_j), \]

where \( \sigma \) stands for the sigmoid function. Given the state of visible units, one can then sample from a Bernoulli distribution for each hidden unit, and vice versa. Note that an RBM in this form is similar to a linear layer in DNN with sigmoid activation function. However there are some differences. For example, units (or neurons) in RBM are random variables, while neurons in DNN are deterministic. Also RBM is reversible, but DNN is usually feed-forward.

RBM can be trained on data that represent visible unit states, with maximum likelihood. However the exact gradient of the log-likelihood function is intractable, due to the partition function. The most popular training algorithm is contrastive divergence (CD) from Hinton [Hin06b], where the gradient for log-likelihood function is approximated by [Fis10]:

\[
CD_k(\theta, v^{(0)}) = - \sum_h p(h|v^{(0)}) \frac{\partial E(v^{(0)}, h)}{\partial \theta} + \sum_h p(h|v^{(k)}) \frac{\partial E(v^{(k)}, h)}{\partial \theta},
\]

where \( E \) is the energy function for RBM, \( v^{(0)} \) is the visible state given by training data and \( v^{(k)} \) is the visible state after \( k \) steps of Gibbs sampling between visible units and hidden units.

The use of CD enables effective and practical training of RBMs. However, it is showed that contrastive divergence for RBM is biased [CP05] and it converges to points different from maximum likelihood estimator. More recently it is found that CD may cause the log-likelihood to diverge when data distribution is difficult for RBM to learn [Fis10], and the same problems exist in variants of CD such as Persistent CD (PCD) [Tie08] and Fast PCD.
Deep Belief Network (DBN) [Hin06a] is a graphical model that consists of a stack of RBMs, where the hidden layer of a RBM is at the same time the visible layer of an adjacent RBM. A deep hierarchical structure of a DBN resembles that of a DNN and allows higher representation capability than RBMs, and the DBN is shown to be an universal approximator [LR10]. However the training of DBN is also harder. The common way is to pre-train every two layers greedily as a RBM, from one end of the DBN to another, then fine-tune the weights using, for example, supervised learning of a DNN.

This way of training the DBN relies on the sequential training of RBMs, which means it has the same limitation in terms of scalability and shares the reliability issue of training RBM using CD type of algorithms. Also a global fine-tuning is usually required as layers are trained in a greedy way. In the dissertation, we propose and explore a novel adversarial training framework for DBN, which has the potential to address all these problems and opens many promising directions to improve DBN training and design.

1.3 Overview

We explore the topic of graphical models in two areas, network modelling in community detection and general data representation in deep belief networks, with the focus on model design and training framework respectively.

In chapter 2, we define and propose a solution to a general community detection problem in multiplex networks. Multiplex networks are natural representations for cases when multiple types of relations exist, such as social networks. We consider the situation when the same communities may or may not exist across multiple layers, and the goal is to take the advantage if they do. We propose a generative graphical model for communities, by extending the stochastic block model from single layer networks to multiplex networks. Our model does not require any prior knowledge about which nodes belong to the same
community in different layers. To achieve that, we develop a constraint that only allows compatible community labels of nodes across two layers, and integrate it into the generative model. Then a belief propagation algorithm is used to estimate the marginal probabilities of community label for each node. By a set of experiments, we show the effectiveness of the proposed solution in improving community detectability.

In chapter 3, we concentrate on the training of deep belief networks. We developed a framework in light of generative adversarial networks, by replacing the generator with a DBN. The proposed framework addresses the limitations of traditional DBN training and neural network training, in that it generalizes the maximum likelihood framework and does not require back-propagation. We apply a variance reduction technique to improve the stability and convergence speed of our algorithm. With a number of experiments, we demonstrate viability of adversarial training of DBN, and the advantage of our approach against neural network especially with small and sparse data sets.

In chapter 4, we conclude the methods and algorithms developed in chapter 2 and 3, as well as their significance, in terms of comparisons against relative methods. We also highlight potential benefits we observed and discuss the possible future directions based on our work.

In the appendix, we present the mathematical proof for the equivalence between the well partitioned property (WPP) and the set of conditions given by the $f_{check}$ function, which are key concepts introduced in chapter 2.
CHAPTER 2

COMMUNITY DETECTION AND IMPROVED DETECTABILITY IN MULTIPLEX NETWORKS

2.1 Introduction

A multiplex network structure is a comprehensive representation of real world networks considering that it allows for multiple kinds of relations, and encodes them separately. The

multilayer nature of these networks substantially changes their structure and dynamics [Bia13; DD13; Car13; Boc14; DD15a; Mah16] in comparison to single layer representations [Kar11; Wan13]. Despite being one of the main topics of network science for over a decade, the community detection problem has only recently been more closely studied in the context of multilayer networks [Muc10; DD15a; Loe15; Wil17; VC16; Tay16; Sta16; Afs16; Pau17; DB17]. Community detection in multiplex networks has found numerous applications, such as dynamics [Pal07] and multi-relation [Sze10; Hua16b] in social networks, evolution of granular force networks [Pap16], and cognitive states of brain networks [Tel16].

It is generally advantageous for community detection to decompose an ordinary network into multiple layers based on additional attributes, and to create a multiplex network for individual layers to potentially unravel entangled structures, such as overlapping communities. It can, however, be difficult to reach this goal with a usually limited knowledge. Carelessly breaking a network into layers can be problematic since it can either retain overlapping communities in a single layer, or decrease detectability of certain communities by breaking them up and distributing them over multiple layers, leading to redundant communities and reduced edge density. This problem arises in many real world multiplex networks. For example, in the networks of protein-genetic interactions, each type of genetic interaction may be used to define a layer, but it is shown to return highly redundant layers, which require recombination [DD15b].

Some recent studies consider the redundancy phenomenon in multiplex networks [DD15b; Tay16; Sta16] and try to simplify it by further aggregating the redundant layers. Domenico et al. [DD15b] utilize tools from quantum information to identify redundant layers and aggregate them hierarchically, thus simplifying the structure. They discovered that many real world multiplex networks, including protein-genetic interactions, social networks, economical and transportation systems, can be significantly simplified by their proposed technique. Taylor et al. [Tay16] showed that the detectability of community structure is significantly improved by aggregating layers generated from the same stochastic
block model (SBM), which is a popular probabilistic generative model for describing nodes' group memberships [Wan87]. Stanley et al. [Sta16] proposed a specific multilayer SBM which partitions layers into sets called strata, each described by a single SBM. Layers in a stratum are treated as multiple realizations of the same community structure, thus improving community detection accuracy. A drawback of layer aggregation is that completely consistent community structure between layers is required and needs to be known a priori, otherwise different communities may overlap when aggregated into a single layer, as shown in Fig. 2.1. Domenico et al. [DD15a] used the concept of modular flow to show that aggregating layers into a single layer may obscure actual organizations, and that highly overlapping communities exist in some real-world networks. While many algorithms are proposed for overlapping community detection in single layer networks [Pal05; Esq11; Pso11; Yan13a; Yan13b; Ngu15; Gam16], the performances remain mediocre due to the loss of layer information.

Inspired by these works, we consider a general multiplex SBM that allows layers to be "partially" redundant, in which case layers may share one or more common communities, and have different ones at the same time (lower row in Fig. 2.1). Our goal is to improve detectability by leveraging the consensus communities without assuming any two layers to belong to the exact same SBM. This not only achieves higher accuracy, but improves detectability of weak consensus communities as well, by combining their information from different layers, which are otherwise too noisy to be detectable individually. Since our model potentially generates a heterogeneous community structure across layers, our method provides a way to detect overlapping communities at theoretically optimal accuracy, when they can be allocated to different layers.

Our method originates from an application of belief propagation algorithm to community detection, as first developed by Decelle et al. [Dec11a; Dec11b]. Belief propagation is one algorithms in the Bayesian inference framework, which in turn, is known to yield optimal estimates of communities for a network generated by the underlying SBM [Dec11a].
Decelle et al. studied detectability transition, and identified a phase transition point in the parameter space, where all community detection algorithms fail. Since then, some extending works using belief propagation have been reported [New16b; Gha16; Zha16; Kaw17]. Ghasemian et al. [Gha16] extended this method to temporal networks, introducing Dynamic Stochastic Block Model, where nodes gradually change connections and their community memberships over time. While not intended for general temporal networks, our multiplex network model in contrast to [Gha16], addresses networks that typically encode multiple relations through layers, and the members of a given community remain unchanged irrespectively of the layer the latter occurs in. Our model also does not enforce a temporal order of the layers as in [Gha16]. Despite aiming for different types of multiplex networks, a simplified version of [Gha16] is used as a comparison with our model, in presence of homogeneous and heterogeneous community structures. We show that in different situations, both method show their own strength.

The outline of the paper is as follows. In Sec. 2.2.1, we define the problem of community detection and information fusion in multilayer networks. In Sec. 2.2.2, we present our proposed stochastic model and a simpler model for comparison. In Sec. 2.2.3, we review the belief propagation algorithm and explain the implementation on the proposed model. In Sec. 2.3, we show multiple experimental results of the proposed model and discuss its evaluation in detail and its comparison with the simpler model.

2.2 Problem and method

2.2.1 Problem Description

An informed description of our problem of interest is the following: suppose that a multiplex network \( W = (V, E(1), E(2), ..., E(L)) \) is given where \( V = \{v_1, v_2, ..., v_N\} \) is the set of \( N \) nodes and \( E(l) \) is the set of edges on \( V \) at the \( l \)-th layer. We are to identify a collection \( C = \{C_1, C_2, ..., C_q\} \) of node communities, where \( C_i \subseteq V \) corresponds to a dense subgraph in at
Figure 2.1 Demonstration of potential benefit (upper row) and caveat (lower one) of aggregating multiplex layers. In both rows, left hand side shows the adjacency matrices of two multiplex layers, and right hand side shows that of the aggregated layer by adding the two adjacency matrices. The upper row shows a better community quality of the aggregated network than single layer, while the lower row shows a obscured community structure (notice in the lower row that the consistent community between two layers still gets enhanced quality).

least one layer. Although our problem admits overlapping communities, we assume that the co-occurring communities in each layer are disjoint. Each community may also appear in multiple layers, in which case the resulting data multiplicity can be used to improve community detectability by improving the signal to noise ratio (SNR). However, since the occurrence pattern of the communities is not a priori known, fusing multiple observations of the same community is not straightforward. For a large part of this paper, we assume that the number $q$ of communities is known. However in Section 2.3.3, we briefly discuss the impact of an incorrect choice of $q$ and possible remedies.

2.2.2 Bayesian Solution by Stochastic Modeling

We adopt a Bayesian approach by providing a stochastic generative model for the observed multiplex network, expressed by a likelihood function $P(W \mid C)$, as well as a prior distribution $P(C)$ on communities. Then, the maximum a-posteriori (MAP) estimate of the
communities is obtained by maximizing the a-posteriori distribution, computed according to the Bayes rule:

$$\hat{C}_{\text{ML}} = \arg \max_C P(C \mid W) = \arg \max_C \frac{P(W \mid C)P(C)}{P(W)},$$

where $P(W) = \sum_{C'} P(W \mid C')P(C')$ is a scaling constant and can be eliminated from optimization. Our generative model utilizes the stochastic block model (SBM), explained in Section 2.2.2.1, which is widely expressed in terms of node-community labeling. For this reason, we provide an alternative representation of the communities by community labeling $T = \{t_i(l)\}$ of the nodes $i$ at different layers $l$. Since, there is a correspondence between possible communities $C$ and the labeling $T$, the generative model $P(W \mid C)$ and the prior $P(C)$ can be equivalently expressed in terms of the labeling as $P(W \mid T)$ and $P(T)$, respectively. We carefully explain this approach, and the resulting stochastic model is given in Section 2.2.2.2. We can similarly obtain the MAP estimate $\hat{T}_{\text{MAP}}$ of $T$ and find its corresponding set of communities, which coincides with $\hat{C}_{\text{ML}}$, but we resort to a well-known alternative approach, for numerical feasibility. In this approach, we first calculate the marginal probability distribution $p_{i,l}(\alpha) = P(t_i(l) = \alpha \mid W)$ of the labels $\alpha$ of a single node $i$ in a single layer $l$. This is given by

$$P(t_i(l) = \alpha \mid W) = \sum_{T \mid t_i(l) = \alpha} P(T \mid W),$$

where we recall that the posterior distribution $P(T \mid W)$ is calculated by Bayes rule as

$$P(T \mid W) = \frac{P(W \mid T)P(T)}{\sum_{T'} P(W \mid T')P(T')}.$$  

Next, we obtain the maximum marginal a-posteriori probability (MMAP) label estimates $\hat{T}_{\text{MMAP}} = \{\hat{t}_{i,\text{MMAP}}(l)\}$ by individually maximizing the resulting posterior marginal distribu-
tions $p_{i,t}(\alpha)$ for every node:

$$\hat{r}_{t,MMAP}(l) = \arg \max_{\alpha} p_{i,t}(\alpha),$$

from which the corresponding community estimates $\hat{C}_{MMAP}$ can be easily obtained. It is shown in [Dec11a] that $\hat{C}_{MMAP}$ is an optimal estimate of the original assignment for large networks with the SBM, which is often slightly better than the MAP estimate $\hat{C}_{MAP}$ (ground state) in terms of the number of correct assignments.

Numerical efficiency of the above approach depends on the computation of marginal distributions $p_{i,t}(\alpha)$, which is difficult to perform directly. For example, the denominator in Eq. (2.2), known as the partition function, cannot be exactly calculated unless the system is extremely small or approximate approaches such as Gibbs sampling are used. In Sec. 2.2.3, we use a computationally more efficient variational method called belief propagation (BP), which gives the exact marginals $p_{i,t}(\alpha)$ as an approximation of the partition function by a product of marginals, and leads to an efficient implementation of the above approach. We next discuss the generative model in detail.

### 2.2.2.1 Stochastic Block Model in Single-Layer Network

Stochastic block model (SBM) is commonly used to describe non-overlapping community structures of a single layer network, and plays an important role in our model. Hence, we explain it first. As a generative model, it includes the following parameters: the number of communities $q$, the fraction of the size of each community $\{n_a\}$, the affinity matrix $p = \{p_{ab}\}$ showing the probability of an edge between nodes in communities $a$ and $b$, and the community assignment $t_i \in \{1, ..., q\}$ for each node $i$.

A single-layer network is generated from SBM by first assigning to each node one of the community labels $t_i$. The probability of a node being assigned to a community label $a$ is proportional to the size $n_a$ of the community. Then, a pair $(i, j)$ are connected ($A_{ij} = 1$ in
the adjacency matrix) with probability \( p_{t_{ij}} \) independently of other pairs. According to the SBM [Wan87], if the size of a community is large enough, the community will appear as a block with high probability in the adjacency matrix, under suitable ordering of the nodes.

In benchmark tests, it is common to set \( p_{ab} = p_{in} \) if \( a = b \), and \( p_{ab} = p_{out} \) if \( a \neq b \). The constants \( p_{in} \) and \( p_{out} \) are selected such that the fraction \( \epsilon = \frac{p_{out}}{p_{in}} \) is between 0 to 1, so as to control the community quality in the generated network. \( \epsilon = 0 \) means no connections between two different communities, which represents a high quality community structure. A high \( \epsilon \) value (\( \epsilon \approx 1 \)) means that the connection densities inside and outside the blocks are not significantly distinct, usually reflecting a noisy and weak community structure.

### 2.2.2.2 Generalization to Multiplex Networks

Now we generalize SBM to multiplex networks. The idea behind our generative model for multiplex networks is that the same community may appear in multiple layers. Each layer \( l \) takes a subset of a collection of communities \( C \), denoted by \( H_l \subseteq C \). If communities \( C_a, C_b \in H_l \) and \( a \neq b \) (here \( a \) and \( b \) are community labels), it is required that \( C_a \cap C_b = \emptyset \), meaning that overlapping communities are not allowed in any layer. Also, we assume that when a community \( C_a \) exists in multiple layers, it refers to the same group of nodes, so that the definition of \( C_a \) is independent of the layers. We call these requirements Well Partitioned Property (WPP), and it is an interlayer constraint. WPP has real world relevance a good case being the social network. We can build a multiplex social network using different types of relations, such as contacts, collegial interaction, common interests, etc., in order to disentangle overlapping community structures. However, communities may exist across multiple layers, e.g. a group of close friends may be reflected as the same community in both the rock music network and the soccer fan network. Meanwhile, in these two layers, other people may form inconsistent community structures. In conclusion, we want to build a model, such that only if consistent communities exist between layers, they will be matched and fused.
Under WPP, we may define the community label vector \( t(l) = (t_1(l), t_2(l), \ldots, t_N(l)) \) for all \( N \) nodes in layer \( l \) similarly to the single-layer model in Sec. 2.2.2.1:

\[
t_i(l) = \begin{cases} 
a & i \in C_a, \ C_a \in H_l, \\
\emptyset & \text{otherwise.}
\end{cases}
\]

The community \( C_a \) can be easily recovered from the labels by collecting every node labeled by \( a \):

\[
C_a = \{ v_i \mid \exists l, \ t_i(l) = a \}
\]

The generative model proceeds as follows: the community label vector \( t(l) \) for nodes in layer \( l \) is generated from SBM parameters, under the interlayer constraint WPP. The adjacency matrix of layer \( l \) is then independently generated as an ordinary SBM. We propose to formulate the probability of a multiplex network \( \{W(l)\} \) and community labels \( \{t(l)\} \), conditioned on a set of SBM parameters as,

**Proposed model:**

\[
P(\{W(l)\}, \{t(l)\} | p, q, \{n_a\}) = \frac{1}{Z} \prod_{(i,j) \in (l,l')} f_{\text{check}}(t_i(l), t_j(l), t_i(l'), t_j(l')) \\
\times \prod_{l=1}^{L} \prod_{(i,j) \in E(l)} p_{t_i(l),t_j(l)} \prod_{(i,j) \notin E(l)} (1 - p_{t_i(l),t_j(l)}) \prod_i n_{t_i(l)}.
\]

(2.3)

In the following, we break down the formulation and explain each component. We start with a factorized form of the likelihood function, assuming the set of parameters
\[ \theta = \{p, q, \{n_a\}\} \] given,

\[
P(\{W(l)\}, \{t(l)\}|\theta) = P(\{W(l)\}|\{t(l)\}, \theta) P(\{t(l)\}|\theta),
\]

(2.4)

\[
= P(\{t(l)\}|\theta) \prod_{l=1}^{L} P(W(l)|t(l), \theta),
\]

where \(W(l) = (V, E(l))\), \(l = 1, \ldots, L\) is a multiplex layer.

If we look at the product term, \(P(W(l)|t(l), \theta)\) is the probability that a layer \(l\) of the network being generated by a community structure \(t(l)\). Same as the single-layer SBM, introduced in \([Dec11a]\),

\[
P(W(l)|t(l), \theta) = \prod_{(i,j) \in E(l)} p_{t_i(l), t_j(l)} \prod_{(i,j)\notin E(l)} (1 - p_{t_i(l), t_j(l)}).
\]

(2.5)

The other term, \(P(\{t(l)\}|\theta)\), is the probability distribution over all community patterns satisfying the interlayer constraints from WPP. We express these constraints by a product of local indicator functions \(f_{\text{check}}\) over the associated community assignment labels \(t_i(l)\).

Therefore if at least one of the indicator functions is zero (local WPP condition is not satisfied), \(P(\{t(l)\}|\theta)\) will be zero. Specifically, we propose the distribution of community patterns as:

\[
P(\{t(l)\}|\theta) = \frac{1}{Z} \prod_{i,l} n_{t_i(l)} \prod_{i(l), j(l), i(l'), j(l')} f_{\text{check}}(t_i(l), t_j(l), t_i(l'), t_j(l')).
\]

(2.6)

where \(Z\) is a suitable normalization constant. The local constraint \(f_{\text{check}}\) is an indicator function of the state (community label) of the copies of 2 nodes \(i, j\) in 2 different layers, \(l, l'\) (\(i(l)\) means node \(i\) in layer \(l\)). The function \(f_{\text{check}}\) checks whether the associated labels satisfy WPP, and \(f_{\text{check}}\) equals one if the following occurs, and is zero otherwise:
Assume \[
\begin{cases}
t_i(l) = \alpha \\
t_j(l) = \beta
\end{cases},
\]
If \( \alpha = \beta \), then \[
\begin{cases}
t_i(l') = t_j(l') = \alpha \\
\text{or} \\
t_i(l') \neq \alpha
\end{cases},
\]
If \( \alpha \neq \beta \), then \[
\begin{cases}
t_i(l') \neq \beta \\
t_j(l') \neq \alpha
\end{cases}.
\]

This set of conditions summarize when the labels of two nodes satisfy WPP, as we will discuss in detail next. In practice, given a certain number of communities \( q \), we can build a list of all possible combinations that satisfy the above constraint and set \( f_{\text{check}} = 1 \). Therefore, the process of evaluating the function \( f_{\text{check}} \) by verifying the above constraint, can be significantly simplified by storing a look-up table. The look-up table is simple to build for moderate \( q \) with a complexity of \( q^4 \), and only needs to be computed once for a certain \( q \) value.

2.2.2.3 Characterizing WPP

We are able to proof that a multi-layer community structure \( (C, \{H_l\}) \) satisfies WPP, if and only if, for the labels of every pair of nodes and every two layers, the value of function \( f_{\text{check}} \) equals one and hence \( P(\{t(l)\} \mid \theta) = 1 \). The general proof is in the appendix.

Here we show a simple example to demonstrate one of the constraints.

Fig. 2.2 shows a situation where community structures in two layers are different (each connected component in a layer is a community). According to the connectivity patterns, we observe that \( t_i(1) = t_j(1) \) and \( t_i(2) \neq t_j(2) \) as \( i, j \) are in the same community in layer 1,
while they are in different communities in the second layer. We conclude that, in the second layer, neither node \( i \) nor node \( j \) can be assigned to the same community as the one in the first layer, and hence at least 3 communities are required for a correct assignment. This simple intuition is reflected in the definition of \( f_{\text{check}} \) (the case of \( \alpha = \beta \)), where \( l = 1, l' = 2 \).

We observe that the constraints in \( f_{\text{check}} \), when utilized in a Bayesian learning algorithm, ensure that distinct communities in different layers will not be assigned the same label and not be confounded as one community, so that the structural information will not be mixed up and obscured. This is, according to our example, due to the fact that assigning the same labels to unequal communities will lead to violation of constraints, and make corresponding \( f_{\text{check}} \) functions zero and consequently a zero-value posterior distribution \( P(\{W(l)\}, \{t(l)\})|\theta \). Another role of the constraints is to equally assign consistent communities in different layers, and fuse the structural information to improve detectability. This is illustrated in our example, depicted in Fig. 2.3, where only the community for node \( i \) is consistent between two layers, and our goal is to assign to the copies of node \( i \) the same community label. Notice that in total, 4 communities are involved in this example. If we set \( q = 4 \), any community assignment with \( t_i(1) \neq t_i(2) \) will violate the \( f_{\text{check}} \) constraints, which

\[ l = 1 \quad \begin{array}{c}
\includegraphics[width=0.5\textwidth]{figure2.2.png}
\end{array} \quad l = 2 \]

\textbf{Figure 2.2} An example of a two-layer network with different community structures.
in turn will force \( t_i(1) = t_i(2) \) in the Bayesian learning algorithm. For example, let \( t_i(1) = a_1 \) and \( t_i(2) = a_2 \). Since \( t_i(1) \neq t_j(1) \), according to the constraint where \( \alpha \neq \beta \), we know that \( t_i(1) \neq t_j(2) \) and \( t_i(2) \neq t_j(1) \), and we let \( t_j(1) = a_3 \) and \( t_j(2) = a_4 \), and therefore \( t_k(1) = a_3 \).

Similarly using the same constraint, we know \( t_i(1) \neq t_k(2) \). We derive that \( t_k(2) \neq a_1 \), and due to the community structure in layer 2, \( t_k(2) \neq a_2, a_4 \). Then again using the constraint of \( \alpha \neq \beta \) on \( t_j(2) \neq t_k(2) \), we derive that \( t_k(2) \neq t_j(1) = a_3 \). We find out that \( t_k(2) \) is not able to choose from any of the four community labels without violating the constraints. However if we set \( q = 5 \), we can find community assignments with \( t_i(1) \neq t_i(2) \) while satisfying the constraints (for example \( t_i(1) = a_1, t_j(1) = t_k(1) = a_2, t_i(2) = a_3, t_j(2) = a_4, t_k(2) = a_5 \)), in which case, the communities for node \( i \) in the two layers will be independently treated and detectability cannot be improved. This also demonstrates the important role of the number \( q \) of communities as a design parameter. Although we may not know \textit{a priori} the actual number of communities, this number can be estimated [Dec11a]. We will discuss later (in Sec. 2.3.3) how the number of communities affects detection results.

![Figure 2.3 An example of a two-layer network with partially consistent and partially different community structures. Each ellipse represents a community.](image-url)
2.2.2.4 A Prototypical Multiplex Model

To discuss the performance of our proposed approach in Sec. 2.2.3.2, we present a simpler "correlated model" without overlapping communities, but with variable and correlated ones in different layers. The correlated model is similar to the DSBM (Dynamic Stochastic Black Model) introduced in [Gha16]. This model achieves the best performance when the communities in different layers are the same, since layer consistency is used as prior knowledge, much like the layer aggregation method in [Tay16]. However, the presence of such a strong prior information is not always realistic, and this model only serves as an oracle bound for our proposed model as in Eq. (2.3).

We now modify the above SBM model to a correlated multilayer structure, following the same Bayesian description as in Eqs. (2.4) and (2.5), nevertheless different from our model in Eq. (2.3), in that the community assignment prior is instead given by:

$$ P([t(l)]|\theta) = \prod_{i(l),i(l')} f(t_i(l), t_i(l')), \quad (2.7) $$

where $f(\cdot)$ is a factor function for the correlation of community assignment of the same node $i$ in two layers, indicating the probability of different $(t_i(l), t_i(l'))$ combinations:

$$ f(t_i(l), t_i(l')) = \begin{cases} 
  p_{same}, & t_i(l) = t_i(l') \\
  1 - p_{same}, & t_i(l) \neq t_i(l') 
\end{cases} $$

where $p_{same} \in [0, 1]$ is the probability of consistent community labels between the same node in two layers. In a special case, if we constrain the number of both layers and communities to 2, when $p_{same} > 0.5$, node labels between two layers are correlated, when $p_{same} < 0.5$, anti-correlated, and when $p_{same} = 0.5$, uncorrelated. Note that when $p_{same} < 1$, it allows the same community label to correspond to different sets of nodes in different layers. For $q > 2$, $p_{same} = 0.5$ is still the threshold above which the communities become correlated, but then
\( f(t_i(l), t_i(l')) \) needs to be normalized to be the real probability. Similarly to Eq. (2.3), we propose the following Bayesian model:

\[
P(\{W(l)\}, \{t(l)\}|p, q, \{n_a\}) = \frac{1}{Z} \prod_{i(l), i'(l')} f(t_i(l), t_i(l')) \\
\times \prod_{l=1}^L \left[ \prod_{(i,j) \in E(l)} p_{t_i(l), t_j(l)} \prod_{(i,j) \not\in E(l)} (1 - p_{t_i(l), t_j(l)}) \prod_i n_{t_i(l)} \right].
\] (2.8)

Unlike WPP, this model assumes variable communities and correlation between community assignments of a single node between layers. This may be too ideal relative to Eq. (2.3), since it adds to the model some privileged prior knowledge which is uncommon in real scenarios. We will later compare the model in Eq. (2.8) with the constrained multiplex model proposed in Eq. (2.3).

### 2.2.3 BP algorithm for multilayer community detection

Belief Propagation is an efficient message-passing method for inference problems. Message-passing appears in various contexts, and with various references, such as sum-product algorithm, belief propagation, Kalman filter and cavity method which is used to compute phase diagrams of spin glass systems [Méz87]. Yedidia et al. [Yed03; Yed05] gave a detailed introduction to Belief Propagation and its connection to free energy.

We use the BP algorithm for calculating the marginal posterior distributions \( p_{t_i,l}(\alpha) \) as explained in Section 2.2.2. To that end, we will represent our model in Eq. (2.3) as a factor graph. A factor graph is composed of factor nodes and variable nodes. Each variable node corresponds to an actual node in our multiplex network. A factor node corresponds to a factor in Eq. (2.3). In a tree-like Bayesian network, each factor can also be interpreted as a conditional probability distribution \( p(x_i|\text{Parent}(x_i)) \). Here \( x_i \) corresponds to a variable node and \( \text{Parent}(x_i) \) denotes its parent nodes [Yed03]. A factor node is connected to its
contributing variables, therefore connecting a variable node and all its parent variable nodes. As seen in Eq. (2.3), two types of factor nodes arise in our case: constraint \(f_{\text{check}}\) nodes, connected to four variable nodes, and the remaining SBM nodes, connected to two variables (See Fig. 2.4).

In BP, "messages" are reciprocally sent between variable nodes and factor nodes. These messages are a set of equations about the estimates of the conditional marginals. These equations are self-consistent in the sense that they will converge to a consistent solution upon repeatedly iterating. On factor graphs, messages \(m_{i \rightarrow a}(x_i)\) from variable nodes \(i\) to factor nodes \(a\) are different from the reversed ones \(m_{a \rightarrow i}(x_a)\) and are given by:

\[
\begin{align*}
    m_{i \rightarrow a}(x_i) & := \prod_{c \in N(i) \setminus a} m_{c \rightarrow i}(x_i) \\
    m_{a \rightarrow i}(x_i) & := \sum_{x_a \setminus x_i} f_a(x_a) \prod_{j \in N(a) \setminus i} m_{j \rightarrow a(x_j)},
\end{align*}
\]

where \(N(i) \setminus a\) denotes the neighbors of the variable node \(i\) except \(a\), and \(x_a \setminus x_i\) denotes the neighbors of the factor node \(a\) except node \(i\). Basically, a variable-to-factor message is proportional to the product of all other incoming messages to the variable node, while a factor-to-variable message is the posterior marginal distribution of the variable based on the individual factor, and assuming other incoming messages to the factor as independent priors.

The computational complexity of BP is low. To obtain a marginal probability distribution of an objective node in graphs with no loops, one starts from all the leaves and uses all messages only once, toward the objective node. In practice, one starts with random initial messages, and let them update iteratively, until they converge to a fixed point, or until they meet a stopping criterion. Hence, for a fixed number of iterations, the computation time is \(O(|E|)\). In a generated sparse graph where we fix the average degree, the computation time is \(O(N)\). After convergence, the marginal distribution (also called belief) of a node can be calculated using all incoming messages:
While, in the presence of cycles, messages may theoretically require infinite iterations to converge, BP has been observed to perform well in graphs that are locally tree-like even if they have many loops[Dec11a]. Notice that in loopy graphs, the order of message passing is arbitrary and often heuristic.

### 2.2.3.1 Message Passing for Single-Layer SBM

For single layer networks, ordinary SBM is used to describe community structures. In [Dec11a] it is shown that since each factor is exactly connected to two variables, the two steps in Eq. (2.9) can be combined to yield a single node-to-node message passing step as follows:

\[ m_{t_i}^{i ightarrow j} = \frac{1}{Z_{i ightarrow j}} n_{t_i} \prod_{k \in N(i) \setminus j} \left( \sum_{t_k} c_{t_k t_i} A_{k i} (1 - c_{t_k t_i})^{1 - A_{k i}} m_{t_k}^{k ightarrow i} \right), \]

where \( Z_{i ightarrow j} \) is a normalization constant. \( n_{t_i} \) is the fraction of the size of the community \( t_i \) (assigned to node \( i \)), which represents local evidence for node \( i \). \( c_{t_k t_i} \) is the rescaled connection probability between nodes in communities \( t_k \) and \( t_i \) respectively i.e., \( c_{ab} = N p_{ab} \) (\( N \) is the number of nodes). Finally, \( A_{ik} \) is an element of the adjacency matrix of the network.

Decelle et al. [Dec11a] further use the following mean field approximation to simplify the influence from unconnected nodes,

\[ m_{t_i}^{i ightarrow j} = \frac{1}{Z_{i ightarrow j}} n_{t_i} e^{-h_{t_i}} \prod_{k \in N(i) \setminus j} \left( \sum_{t_k} c_{t_k t_i} m_{t_k}^{k ightarrow i} \right), \]

where \( h \) is an external field, and expressed as,

\[ h_{t_i} = \frac{1}{N} \sum_{k} \sum_{t_k} c_{t_k t_i} b_{t_k}^k. \]
Here $b^k_{tk}$ is the belief at node $k$ for community label $t_k$, corresponding to our objective in Eq. (2.1). The belief at node $i$ is written as,

$$b^i_{t_i} = \frac{1}{Z^i_{t_i}} n_{t_i} e^{-h_{t_i}} \prod_{k \in N(i)} \left( \sum_{t_k} c_{t_k t_i} m^k_{tk} \right).$$

Clearly, Eq. (2.11) bears a similar structure to a combination of the two steps in Eq. (2.9). Note that the inner summation part in parentheses in Eq. (2.11) is in the form of a message from a factor node to a variable node i.e., the second line in Eq. (2.9), while the outside product manifests message passing in the first line of Eq. (2.9) from a variable node to a factor node. We observe that the message (2.11), being from variable $i$ to variable $j$, essentially bypasses the factor node lying between these two variable nodes, hence further reducing complexity.

### 2.2.3.2 Multiplex Network as a Message Passing Model

Since the interlayer constraint function in Eq. (2.6) is defined by 4 variable nodes rather than pairwise interaction, we can no longer combine the two messages in Eq. (2.9) and directly write inter-layer messages between variable nodes. We instead opt to explicitly write inter-layer messages from variable nodes to factor nodes. For the sake of consistency, we do the same for intralayer messages. The factor graph is illustrated in Fig. 2.4. The message update equations are shown below.

**Proposed update equations:**

**Intra-layer message:**

$$m_{t_i}^{i \rightarrow a(l)} = \frac{1}{Z_{t_i \rightarrow a(l)}} n_{t_i(l)} e^{-h_{t_i(l)}} \prod_{d \in N_{intra}(l) \setminus a} \left( \sum_{t_d} c_{t_d t_i(l)} m_{td}^{d \rightarrow i(l)} \right)$$

$$\times \prod_{c \in N_{inter}(l)} \left( \sum_{t_j(l), t_j(l')} f_{check}(t_i(l), t_i(l'), t_j(l), t_j(l')) \prod_{k \in N_{inter}(c) \setminus i} m_{tk}^{k \rightarrow c} \right).$$

(2.12)
Inter-layer message:

\[ m_{ti}^{i\rightarrow c} = \frac{1}{Z_{i\rightarrow c}} n_{ti} e^{-h_{ti(l)}} \prod_{d \in N_{\text{intra}(i)}} \left( \sum_{td} c_{tdi} m_{td}^{d\rightarrow i} \right) \times \prod_{c \in N_{\text{inter}(i)} \setminus c} \left( \sum_{t_j(l), t_{i'}(l'), t_{j'}(l')} f_{\text{check}}(t_i(l), t_{i'}(l'), t_j(l), t_{j'}(l')) \prod_{k \in N_{\text{inter}(c)} \setminus i} m_{tk}^{k\rightarrow c'} \right) \]

(2.13)

where \( N_{\text{intra}(i)} \) represents intra-layer neighbors of node \( i \), and \( N_{\text{inter}(i)} \) the inter-layer ones (the constraint-checking factors). \( h_{ti(l)} \) is the external field in layer \( l \), referring to the single layer version in Eq. (2.11).

To calculate the node belief:

\[ b_{ti}^i(l) = \frac{1}{Z_i(l)} n_{ti(l)} e^{-h_{ti(l)}} \prod_{d \in N_{\text{intra}(i)}} \left( \sum_{td} c_{tdi} m_{td}^{d\rightarrow i(l)} \right) \times \prod_{c \in N_{\text{inter}(i)}} \left( \sum_{t_j(l), t_{i'}(l'), t_{j'}(l')} f_{\text{check}}(t_i(l), t_{i'}(l'), t_j(l), t_{j'}(l')) \prod_{k \in N_{\text{inter}(c)} \setminus i(l)} m_{tk}^{k\rightarrow c} \right) \]

(2.14)

Note that the marginal posteriors are given by the beliefs as \( p_i(\alpha) = b_{ti(l)}^{i(\alpha)}(l) \). For experimental purposes, and clarity, we write down the belief propagation equations for a two layer network, similarly for the following model. The associated resulting message passing algorithm is shown below as a pseudocode. The "for" loops, which update the messages, can be easily executed in a parallel or distributed fashion for large networks. In our experiments, a serial version of the algorithm is implemented. In each step, one edge is randomly selected without replacement and the corresponding message is updated, which influences the following updates of other edges.

2.2.3.3 An Oracle Limit: Correlated Variable Communities

We now show the message passing expression for the correlated-community model in Eq. (2.8). The message paths are illustrated in Fig. 2.5, highlighting inter-layer messages and
Figure 2.4 An example of a factor graph for our model. A circle is a variable node, and a square is a factor node. There are two types of factor nodes. One is within each layer, \( f_a(t_i(l), t_j(l)) = c_{t_i(l), t_j(l)} \), representing the likelihood of node \( i \) having label \( t_i(l) \) and node \( j \) having label \( t_j(l) \). Another is between layers, \( f_{\text{check}}(t_i(l, t_i(l'), t_j(l, t_j(l'))) \), representing the local constraints of community labels.

Intra-layer ones. Since every factor node connects only two variable nodes, we can bypass the factor nodes and write messages between variable nodes as in the figure.

**Proposed update equations:**

**Intra-layer message:**

\[
m_{t_i}^{i\rightarrow j(l)} = \frac{1}{Z_{i\rightarrow j(l)}} n_{t_i(l)} e^{-h_{t_i(l)}} \prod_{k \in N(i(l)) \setminus j(l)} \left( \sum_{t_k} c_{t_k t_i(l)} m_{t_k}^{k\rightarrow i(l)} \right) \times \sum_{t_j(l')} f(t_i(l), t_j(l')) m_{t_j}^{i(l')\rightarrow i(l)},
\]

**Inter-layer message:**

\[
m_{t_i}^{l(l')\rightarrow i(l)} = \frac{1}{Z_{l(l')\rightarrow i(l)}} n_{t_i(l')} e^{-h_{t_i(l')}} \prod_{k \in N(i(l'))} \left( \sum_{t_k} c_{t_k t_i(l')} m_{t_k}^{k\rightarrow i(l')} \right).
\]
Algorithm 1 BP for constrained multiplex networks

1: Initialize belief vector for each node in each layer
2: Compute initial messages and field $h$(more detail)
3: while $t < t_{\text{max}}$ and conv$>$criterium do
4:     conv=0; t=t+1
5:     for layer $l$ from 1 to $L$ do
6:         for every directed edge $i \rightarrow j$ in layer $l$ do
7:             Update message $m^{i-j}$ according to Eq. (2.12)
8:         Update message $m^{i-c}$ according to Eq. (2.13)
9:         for every node $i$ in layer $l$ do
10:             Update belief $b^i(l)$ according to Eq. (2.14)
11:     Update field $h(l)$ in layer $l$
12:     $\text{conv} = \sum |m_{\text{new}} - m_{\text{old}}|
13:     for every ordered pair of layers $l$ and $l'$ do
14:         for every ordered pair of nodes $i$ and $j$ do
15:             Update message from $i$ in layer $l$ to the constraint factor node between $i$ and
16:             $j$, $m^{i-c}(l)$ according to Eq. (2.13)
17:     Compute group assignment
18:     Compute accuracy

Figure 2.5 The red arrow shows the interlayer message from $i(l')$ to $i(l)$.
2.3 Detectability transition of constrained multiplex networks

2.3.1 Homogeneous multiplex network

In this section, we report the results of the Bayesian method in Section 2.2.2 with the message-passing algorithms developed in Sec. 2.2.3.2 and 2.2.3.3. We set the experimental scenario to consist of a 2-layer network with 200 nodes, where each layer is randomly generated according to a SBM. The nodes are partitioned into two communities of equal size, which are present in both layers. This is a result of the probability having the same labels between two layers, i.e. \( p_{\text{same}} = 0 \) or 1 in the correlated model, all the while simultaneously satisfying the WPP. For each algorithm, we observe community detectability transition by varying \( \epsilon = p_{\text{out}}/p_{\text{in}} \) in the SBM. The transition is quantitatively characterized by a normalized agreement score \( Q \in [0, 1] \) (referring to "agreement" in [Dec11a]),

\[
Q([[t^*_i(l)], \{\hat{t}_i(l)\}) = \max_{\pi} \frac{\frac{1}{N} \sum_i \delta_{t^*_i(l),\pi(\hat{t}_i(l))} - \max_a n_a}{1 - \max_a n_a},
\]

where \( \{t^*_i(l)\} \) is the ground truth community labels, \( \pi \) is one of the permutations of estimated community labels \( \{\hat{t}_i(l)\} \), and \( \max_a n_a \) is the size of the largest community. The term \( \frac{1}{N} \sum_i \delta_{t^*_i(l),\pi(\hat{t}_i(l))} \) is called agreement score and represents the overlap between estimated community labels and ground truth.

For the correlated-community model in Sec. 2.2.2.4, we observe transitions curves under various value of \( p_{\text{same}} \) in the algorithm in Sec. 2.2.3.3. In Fig. 2.6, for \( p_{\text{same}} = 0.5 \), detectability transition is similar to that in a single layer (red dash line) [Dec11a], because we are practically treating them as independent layers. Except for \( p_{\text{same}} = 0 \) or 1, high correlation (such as \( p_{\text{same}} = 0.9 \)) or anti-correlation (such as \( p_{\text{same}} = 0.1 \)) between labels increases detectability significantly. We conjecture that the poor performance for \( p_{\text{same}} = 0 \) or 1 is due to its low tolerance of wrong intermediate label, leading to a lower chance of convergence to the correct fixed point. The fluctuation of the \( p_{\text{same}} = 0 \) or 1 curves also
indicates that the convergence is not stable in these cases, especially considering the loopy factor graph.

Figure 2.6 Detectability transition curves for various $p_{\text{same}}$ ranging from 0 to 1. The slower the normalized agreement score $Q$ [Dec11a] drops down, the better the detectability. Data points for $p_{\text{same}} = 0, 0.1, 0.5, 0.9, 1$ are connected by solid lines. The results are averaged over 100 experiments. The error bars represent standard errors.

This naive assumption that all nodes in different layers have correlated community labels is, however, the same as directly connecting corresponding nodes in two layers without any further weight adjustment over messages. In this case, all nodes in each layer are assumed as uniformly correlated. This assumption from the correlated model is reasonable for certain types of multiplex networks such as temporal networks. However, to account for heterogeneous structure, and a more realistic case of unknown prior knowledge of consistent communities, it will be more suitable to use our generative model with label constraint.
Fig. 2.6 shows that a two-layer network is enough to exhibit the strength of the correlated model. To directly compare the constrained multiplex model in Sec. 2.2.2.2 with the correlated one in Sec. 2.2.2.4, we follow the same experimental setting as in Fig. 2.6, and test both methods on the homogeneous double layer network. We make sure that each layer has the same community structure and is independently generated by the same SBM parameters: 200 nodes which are divided into two equal communities. Note that we do not generate the network from the correlated model, although the correlated model fits it.

We vary $\epsilon = p_{out}/p_{in}$ to observe the detectability transitions. The result is shown in Fig. 2.7, where we include the transition curve for a single layer (red line) as a reference. Similarly to the correlated model (blue line), the constrained model (black line with circle marks) fails around similar $\epsilon$ values. They both perform much better (fail for larger $\epsilon$) than a single layer.

Note that in the correlated model, we know a priori that the community labels are correlated between two layers. In the constrained model, we, however, do not specifically have that prior knowledge. Just by enforcing WPP constraints and limiting the number of communities to 2, we can still achieve a similar performance improvement. This is beneficial for real world networks, since in practice we often have limited prior information about consistent communities. Indeed, in this experiment, this prior knowledge may also be inferred in the correlated model, setting interlayer correlation as a parameter and using the EM algorithm [Dem77]. However, in more complex cases where, for example, community structure in two layers can not be simply described by a single correlation parameter, the correlated model will face difficulty, as we will show in the next section.

One may suspect that as long as the blocks are consistent, the detectability can be automatically improved regardless of such correlation being available to the model. This is clearly not the case for the correlated model as in Fig. 2.6, since setting $p_{same} = 0.5$, does not include correlation in the model, and the performance is poorer and similar to a single layer setting.
Figure 2.7 Detectability transition curves for a single layer network, a correlated double layer network and a double layer network with constraint. The synthetic network is a two-layer network, where each layer is independently generated by the same SBM model such that 200 nodes form two equal communities. Nodes have the same community labels in layer 1 and 2, while the edges are different across layers, so information between two layers can be easily combined. $p_{same}$ is 0.9, only used as the inference parameter for the correlated model. The results are averaged over 30 experiments. Error bars represent the standard error of the experiments.
2.3.2 Heterogeneous multiplex network

The constrained model being the only model that naturally generates heterogeneous networks, shows the advantage over the correlated model or single layer networks. In the following we compare the community detection performance between the constrained model and the correlated model on heterogeneous networks. We construct a double layer network of 200 nodes, with $\epsilon \in [0, 1]$. An example of the synthetic network is shown in Fig. 2.8. In the first layer, the first 100 nodes form a community and the remaining 100 nodes are assigned to another community. In the second layer the first 100 nodes still form a community but the remaining 100 nodes are divided into two equal communities. By limiting the total number of communities to $q = 4$, we expect the belief of the first 100 nodes in both layers to converge to the same label, and the remaining 100 nodes in two layers to converge to three different labels (refer to Fig. 2.8). We refer to this expected result as WPP-satisfying labels and other results as error.

![Figure 2.8](image)

**Figure 2.8** An example ($\epsilon = 0.2$) of the heterogeneous network generated to test the constraint multiplex model. There are in total four distinct communities. First 100 nodes in two layers form the same community, while the rest 100 nodes form three different communities in two layers.
We performed 100 independent trials of tests using both models, and count the fraction of the tests that result in WPP-satisfying labels. As in Fig. 2.9, when $\epsilon \in [0, 0.4]$, our constrained model yields WPP-satisfying labels in some of the trials, while the correlated model is able to achieve that only for $\epsilon \in [0, 0.1]$. Also, the constrained model has significantly higher likelihood to yield correct labels, i.e., has the messages converge to the correct point, when $\epsilon \in [0, 0.4]$. Note that in this experiment, for each layer, we do not limit the number of communities to the correct value (i.e. two communities for layer 1 and three for layer 2), which means each node in a layer will freely choose from 4 different labels. If we detect communities independently in two layers, which corresponds to setting no constraint, the chance of WPP-satisfying labels is no more than $4!/(P_2^4 \times P_3^4) = 1/12$, where $P_k^n$ is $k$-permutation of $n$. Our result does show an advantage in identifying consistent communities in heterogeneous networks, while the correlated model is unsuitable for this task. The detection error may be attributed to local minima which violate the constraint.
(WPP) to some degree, with, however, sufficient resilience for the messages to converge. In practice, we can run the algorithm multiple times and choose the results that more likely converged to a correct point.

Figure 2.10 Detectability transition curves for constrained model and correlated model on heterogeneous networks. $p_{same}$ is 0.9 for the correlated model. The results are averaged over top 20 trials where most node labels satisfy WPP. Error bars represent the standard error of the trials. Refer to Fig. 2.8 for synthetic network setup.

In Fig. 2.10, for both constrained model and correlated model, we examine the agreement score $\frac{1}{N} \sum_i \delta_\epsilon(t_i, \pi(i_t(t)))$ between prediction and ground truth. That is because in this more complex experiment, it is not as straightforward to define a normalized agreement score $Q$ as in Fig. 2.6 and Fig. 2.7. As stated above, not every trial will converge to the correct point, we therefore select for both models the top 20 trials that satisfy WPP better (without using ground truth information). Specifically, for each trial we count how many pairs of nodes satisfy WPP locally, by calculating $f_{check}$ function over the inferred labels of pairs of nodes. We observe in Fig. 2.10 that for $\epsilon \in [0.1, 0.2]$, the agreement score of the constrained
model is remarkably higher than the correlated model. The performance advantage benefits from a high fraction of WPP-satisfying results using the constrained model for $\epsilon \in [0.1, 0.2]$, as shown in Fig. 2.9. When this benefit vanishes, for $\epsilon \in [0.3, 0.5]$, the constrained model gets similar or worse agreement score than the correlated model. Note that again, the proposed constrained model does not utilize the knowledge that the first 100 nodes have correlated community labels, while the correlated model is supplied with this prior information. The reason of the better performance for $\epsilon \in [0.1, 0.2]$ is that, the constrained model manages to fuse information for the first 100 nodes in two layers, meanwhile leaving the remaining 100 nodes intact, while the correlated model tends to unify the entire community structure in the two layers, hence corrupting the remaining 100 nodes. The poorer performance of the constrained model in the noisier $\epsilon \in [0.3, 0.5]$ range, we suspect, is due to an optimization in stability caused by many more constraints and factor nodes in the graphical model. On the other hand, the correlated model has a simpler form and is less susceptible to the stability issue.

In this section, we have compared our constrained model with a basic correlated model, and results show a higher modeling capability of the constrained model, in presence of heterogeneous community structures. Although the correlated model is simpler, the assumption of a uniform label correlation between two layers does not naturally generate multiplex networks with diverse relations, where only a portion of communities are correlated or consistent. Hence, the correlated model (similar to [Gha16]) is more appropriate for smoothly evolving temporal networks, and the constrained model we proposed is typically suitable for multiplex networks with different types of relations, where the layers are not necessarily uniformly correlated. In principle, the basic correlated model can be extended so that different nodes can have their own interlayer correlation, and the flexibility of the correlated model can be much greater. However, we expect inference difficulty for such model, given the significantly larger number of free parameters, unless the parameters are properly constrained. Such model design will require nontrivial work and is interesting for
future works.

Since the goal of the proposed algorithm is that of fusing consistent communities across layers in general networks and of improving detectability, we are not aware of a directly comparable algorithm that is designed for the exact same goal. Nevertheless, we provide in passing a comparison with a popular multilayer community detection algorithm, Genlouvain [Jut11], which maximizes a multilayer modularity function. For the same experiments in this section, when $\epsilon = 0.2$, Genlouvain converges correctly only 3 out of 100 trials, while our proposed algorithm has over 40% success rate. Genlouvain performs similarly to the correlated model in this particular test. The reason is that Genlouvain requires interlayer coupling parameters, which, when not given, and can only be assumed to be uniform. In contrast, our proposed constrained model implicitly infers interlayer coupling through $f_{\text{check}}$ factor nodes.

### 2.3.3 Impact of a known number of communities $q$

For a single layer network, any $q$ that is larger than or equal to the actual value will fit the model well. For example, by setting $q = 3$ while performing the BP algorithm in a network generated from SBM with 2 communities, we are allowing each node to choose from 3 distinct community labels. However when the messages have converged, generally most nodes will tend to choose from only 2 of the labels, leaving one barely used. Therefore, the general practice is to opt for a larger $q$, until the free energy of the model stops decreasing [Dec11a].

This is in contrast to the constrained multiplex networks. In the experiment of a homogeneous multiplex network, only $q = 2$ gives the best performance according to the detectability transition curve. To show this effect, we generate such a 2-layer, 2-community network, with high noise $\epsilon = 0.35$. (The noise is so high that when we perform BP algorithm on one of the layers with $q = 3$, the community detection is affected and all 3 labels may have a significant presence among nodes, making the decision of $q$ difficult.) Then we run
the algorithm with $q$ being 2, 3, and 4.

$$q = 2$$

$$q = 3$$

$$q = 4$$

Figure 2.11 Probabilities of nodes being assigned certain community labels (node beliefs) at $q = 2, 3$ and 4. Various community labels are represented by different colors and markers. The lower right image shows the noisy double layer network used in this test.

As shown in Fig. 2.11, the performance is getting poorer as $q$ increases. Specifically, at $q = 2$, most nodes have close-to-one probability of some label, and the selected labels match well among two layers. For $q = 3$, the labels still tend to match across layers, but for nodes from 101 to 200, two labels are competing with each other (blue circles and yellow asterisks). For $q = 4$, even the labels are not correctly matched. This is because the constraint factors, more specifically WPP, allows the same communities in two layers to be assigned different labels when $q > 2$. We therefore cannot combine their information to increase the signal-to-noise ratio. The fact that using the correct $q$ will give a distinctive
performance, also enables us to more reliably select $q$.

2.3.4 Practical considerations and more layers

A common challenge in belief propagation algorithm for general graphical model is the presence of a fair number of short loops. Specifically, in our model, the interlayer factor nodes introduce many short loops in our factor graph, both within layer and between layers. These short loops result in a quick convergence to undesirable points, and message update equations become more approximate, due to the influence of $f_{\text{check}}$ being overly amplified. To cope with this, we slightly modify message update equations. Specifically, instead of making the product over all incoming messages from neighboring interlayer factor nodes $N_{\text{inter}}(i)$, we sample and multiply a fraction $N_{\text{sample}}$ of incoming interlayer messages, which also conveniently reduces the computational load. Meanwhile, we can also change the values of $f_{\text{check}}$ function from \{0, 1\} to, for example, \{0.2, 0.8\}, to relax the constraint. By applying these modifications, we observe a more reliable and stable convergence to the correct point in our experiments. In Figure 2.12, we find multiple combinations of learning parameters $f_{\text{check}}$ value and $N_{\text{sample}}$, where the constrained model has over 50% chance to converge to the correct point. These points form a continuous band in the parameter space.

Generally for networks with $N$ node, $L$ layers and $E$ edges per layer, the number of message passing per epoch is $2EL + 2(N^2 - N)(L^2 - L)$, which is dense. For multiplex networks with $L > 2$ layers, our original idea needs in total $(L^2 - L)/2$ different interlayer factor nodes between pairs of layers, since we do not assume sequential layers. Viewing from the scale of layers, messages between all pairs of layers form high level loops, making it even more difficult to converge correctly. We address this difficulty by adopting the idea of alternating projection. Specifically, in each iteration, we optimize messages in every two layers at a time, while freezing other layers, until all pairs are updated. In this way, we break the high level loops among the layers, and decompose the problem $(L > 2)$ into several subproblems.
Figure 2.12 Ratio of trials that satisfy WPP constraint after convergence, in the parameter space of $f_{\text{check}}$ and $N_{\text{sample}}$. We identify a band in the parameter space where the constrained model have over half the chance to reach the correct point. The numerical experiment includes 50 trials for each point, and uses the same setup as the heterogeneous two-layer experiment in Section 2.3.2, when $\epsilon = 0.2$.

($L = 2$), which are more studied and have better convergence behavior. Another possibility to reduce the complexity is to incorporate this complex structure into a single factor node, an extended constraint function $f_{\text{check}}(t_1(1), ..., t_1(L), t_j(1), ..., t_j(L))$ that covers all the layers at once, instead of just two layers, so the number of interlayer messages will scale linearly to the number of layers. We show an experiment of a 3-layer heterogeneous network to compare these two strategies. The 3-layer network has 90 nodes in each layer, including 5 different communities in total, while a common one exists between layer 1 and 2, and between layer 2 and 3 respectively. We find that only optimizing two layers at a time has a significant advantage in improving the speed and chance of convergence to a correct point (48 correct convergence out of 100 trials). Similar to the experiment in Section 2.3.2, the
correlated model will fail to deal with such heterogeneous structures.

2.4 Conclusions

We developed a belief propagation algorithm for community detection in general multiplex networks. We considered a case where community labels of nodes are constrained by natural properties. This case corresponds to a potentially heterogeneous community structure for different layers, a likely scenario for real-world networks. As a comparison, we also considered a correlated model where community labels are uniformly correlated across the layers, for homogeneous multiplex networks. Relying on Bayesian inference, our method is theoretically optimal for networks described by our proposed probability model. For the correlated model, combining information from two layers significantly improves detectability due to the additional prior information. More importantly, for the label constrained model, we showed that using just label WPP constraints and limiting the number of communities, we can achieve a similar performance improvement as that of the correlated model, without rather restrictive prior assumptions. Furthermore, the constrained model is able to assign correct labels to heterogeneous community structures, and achieve a much better detection accuracy than the correlated model over some parameter space. This is especially beneficial for detecting sparse and noisy communities in multiplex networks, such as social networks and biological neural networks. Our current constrained model assumes a homogeneous structure within each community. For networks with specific topologies, we can apply modified SBM in our model, such as degree-corrected SBM for social networks [Kar11; New16b]. Future directions also include improving factor graph design and interlayer message passing efficiency, and applications to real world networks, with the proper numerical efficiencies.
3.1 Introduction

An essential problem in statistical machine learning (ML) is to model a given data set as a collection of independent samples from an underlying probability distribution. This distribution is generally referred to as a generative model. Representing and training generative models has a long and fruitful history and is popular in different mathematical modeling disciplines related to ML. The advent of deep learning and its associated methodology, have in recent years resulted in remarkable advances in the area of inference. Generative Adversarial Networks (GANs) are among the chief examples, and have gained enormous attention in different domains of application, including automatic translation [Yu17; Iso17], image generation [Goo14; Ree16; Arj17] and super-resolution [Led17]. Deep Belief Networks
(DBNs) are another set of highly popular examples with a wide range of application in acoustics [Moh12; Lee09b], computer vision [Nai09; Teh01] and others.

Within the class of generative models, the notion of generative network presents a marked difference with modern ML techniques, such as GANs and DBNs, and other conventional methods. In a nutshell, a generative network consists of a randomized computational unit, which is able to generate random realizations from a wide variety of distributions. For instance, GANs utilize a standard deep neural network (DNN) that we refer to as a neural generator, fed with a random sample from a fixed distribution. On the other hand, DBNs consist of a Markov chain of random vectors with the last vector in the chain as the output and the others as hidden features. In contrast to conventional techniques, generative networks in GANs and DBNs encode the generative models in an implicit way, whereby the desired probability distribution may be computationally unfeasible, but can still be statistically sampled efficiently. This fundamental difference leads to an incredible potential in representing highly complex generative models, with however, a radical paradigm shift in the training methodology. In this respect, GANs provide a novel and numerically efficient training approach, relying on an adversarial learning framework and the stochastic gradient descent (SGD) technique in back-propagation.

It has been observed that in many cases, DBNs have remarkable advantages over the neural generators in GANs. A motivating example, considered in this work, is modeling the recorded activities of biological neurons from the visual cortex area of a mouse brain under visual stimulation. While neural generators inherit the limiting properties of neural networks, such as continuity and differentiability, DBNs enjoy much more versatile statistical properties, including sparsity and less severe regularity [Sri12; Bou08]. For modeling biological neurons, sparsity considerably limits standard GANs’ performance. Relying on a symmetric probabilistic relation between different layers, some DBNs can also be used in a reverse order, i.e. by feeding the data at the output layer and reversely generating the hidden features. DBNs based on Restricted Boltzmann Machines (RBMs) are prominent examples
of such reversible networks. This unique property yields an efficient method for feature extraction, which has been exploited in different applications such as data completion and denoising [Lee09a; Lee09b]. It has also been used in supervised learning problems such as classification, by learning a joint generative model for the data and the labels and feeding the data to generate the labels [Hin06b; Lee09a; Che15]. Using reversible DBNs, as we later argue in this paper, also allows us to symmetrically capture the relation of stimuli and neural activities, such that one of them can be inferred from the other. In addition, when given a small amount of training data, DBNs have the potential of a better statistical performance than DNNs, on account of their Bayesian nature. This turns out to be crucial to modeling neural activities as these recorded data are often in short supply.

Despite their numerous advantages, DBNs are less popular than GANs in practice, especially when highly deep structures are required to represent complex models. One reason is that unlike neural generators, the existing training techniques for DBNs are based on layer-wise Gibbs sampling and/or variational methods, resulting in a substantially slower convergence rate than GANs. Moreover, the formulation of GANs admits various inference principles, as exemplified by the Wasserstein GAN (WGAN) architecture [Arj17], while DBNs are generally trained on the basis of the Maximum Likelihood (MaL) principle, which can become numerically unstable in many practical situations [Arj17]. Furthermore, the various well known regularizers for neural generators such as pooling layers and normalization, are not readily used in DBNs as of now. Our goal in this paper is to address the afore-mentioned issues by endowing DBNs with similar training methodologies available for GANs, while avoiding the conventional layer-wise training based on the MaL principle. The result may also be interpreted as a new generation of GANs with more flexible DBNs as their generators. We show that applying the adversarial learning approach to DBNs leads to a numerically more efficient algorithm than GANs, since unlike the back-propagation algorithm in GANs, our training approach is parallelizable over different layers. The main contributions of this paper can be summarized as follows:
• Inspired by GANs, we develop an adversarial training framework for DBNs with superior numerical properties, including scalability through parallelization and compatibility with the acceleration and adaptive learning rate schemes.

• Unlike existing approaches, our framework can address the most generic form of DBNs, creating a potential to incorporate similar regularization units as DNNs, such as pooling and normalization. Focusing on the standard RBM-based DBNs and convolutional DBNs, we leave the details of further generalizations to a future study.

• Based on our framework, we develop algorithms that train DBNs under different metrics than the MaL principle, such as the Wasserstein distance.

• We consider a number of illustrative experiments with the MNIST handwritten digits dataset [LeC98] as well as the aforementioned biological neural activities.

3.1.1 Related Literature

DBNs belong to a broader family of Bayesian networks [Nie09; Nea04]. The most popular form of DBNs are for binary variables and generalize the "shallow" architecture of Restricted Boltzmann Machines (RBMs) [Smo86; Hin06a; Ack85]. The most efficient methods of training RBMs, such the contrastive divergence method are based on Monte Carlo (MC) sampling and variational Bayesian techniques [Tie08; Sut10; Hin12]. Similar techniques are used in a layer-wise fashion for training DBNs consisting of multiple layers of RBMs [Ben07; Bou08; Hin06b]. Another approach for training DBNs is based on the variational lower bound (a.k.a Evidence Lower Bound ELBO) [Mni14]. Modifications of DBNs are also considered in the literature. In [Teh01] and [Nai10] for example, the application of DBNs to non-binary variables is discussed by respectively introducing binomial and rectified linear (ReLU) units. To impose shift invariance, convolutional DBNs are introduced in [Lee09a]. Using DBNs for modeling neural activities has also been considered in [Lee08; Zhe14]. Compared to DBNs, GANs and neural generators belong to a more recent literature. The original
idea of GANs stems from the original work in [Goo14]. Different variations of GANs, such as WGAN [Arj17] and Deep Convolutional GANs (DC-GANs) [Rad15] are highly popular in the literature. Conditional GANs are introduced in [Mir14] for modeling the relation of two variables such as images and labels [Gau14; Iso17]. It is worth noting that using adversarial learning for training neural generators is not limited to GANs. For example, [Mak15] introduces a training method based on the probabilistic auto encoder architecture used in the so-called Variational Auto Encoders (VAEs) [Kin14] and adversarial learning.

3.2 Mathematical Background

3.2.1 Problem Formulation: Training Deep Belief Networks

Theorem Given an observed data set \( \bar{X} = \{x_n\}_{n=1}^N \) with \( N \) data points \( x_n \) from a data space (domain) \( \mathcal{X} \), we are to estimate a probability measure \( \mu \) on \( \mathcal{X} \), with \( \bar{X} \) as its set of independent random samples. DBNs address this problem by generating a random variable with a desired distribution \( \mu \). To this end, multiple layers \( h^1, h^2, \ldots, h^L \) of random variables are considered. In the RBM-based DBNs, the \( l^{th} \) layer \( h^l = (h^l_1, h^l_2, \ldots, h^l_{d^l}) \) is a random \( d^l \)-dimensional binary vector and the joint probability density function\(^2\) (p.d.f) of the layers is written as

\[
\log p(h^1, h^2, \ldots, h^L) = \log C + \sum_{l=1}^{L-1} \sum_{(i,j) \in [d_l] \times [d_{l+1}]} h^{l+1}_i w^l_{ij} h^l_j + \sum_{l=1}^{L} \sum_{i \in [d_l]} b_i h^l_i, \tag{3.1}
\]

where \( w^l_{ij}, b^l_i \) are a set of weights and \( C \) is a proper normalization constant. The output layer \( h^L \) thus reflects the desired distribution, while the layers \( \{h^l\} \) form a Markov chain. This is apparent in the RBM-based formulation in (3.1) as the pdf can be factored by terms including only adjacent layers. We may write the joint distribution in the "forward" form

\(^1\)To define such a measure, we naturally assume that \( \mathcal{X} \) is also equipped with a proper sigma algebra. Indeed, we are practically concerned only with the case where \( \mathcal{X} = \mathbb{R}^m \) is the space of \( m \)-dimensional real vectors with the standard Borel sigma algebra.

\(^2\)For simplicity, we interchangeably use the terms probability distribution and probability mass.
\[ p(h^1, h^2, \ldots, h^L) = p_1(h^1)p_2(h^2 | h^1) \ldots p_L(h^L | h^{L-1}), \]
where

\[ \log p_i(h^1) = \log C_i + \sum_{i \in d_1} b_i^1 h_i^1, \]
and with a proper choice of the constant \( C_1 \), is the marginal distribution of the input layer \( h^1 \)
and

\[ \log p_l(h^l | h^{l-1}) = \log C_l + \sum_{(i, j) \in [d_{l-1}] \times [d_l]} h_i^l w_{ij}^{l-1} h_j^{l-1} + \sum_{i \in d_l} b_i^l h_i^l \]
are the transitional probabilities between the \( l, l-1 \) layers, with a suitable normalization constant \( C_l \). This forward representation enable us to conveniently sample the output of DBNs by first sampling the input (by \( p_1 \)) and by successively sampling the next layers (by \( p_l \)), given the realizations of the previous layers, until reaching the output. We observe that in the forward representation, the elements \( h_i^1 \) of the input layer are independent. Conditioned on their previous layer, the elements of the next layers are also independent. As the variables in (3.1) are binary, the constants \( C_l \) can also be explicitly calculated, resulting in logistic functions for the probability of individual elements in the first layer as well as the conditional probabilities of the subsequent layers:

\[ p_1(h_i^1 = 1) = \sigma(b_i), \quad p_l(h_i^l = 1 | h^{l-1}) = \sigma \left( b_i^l + \sum_{j \in [d_{l-1}]} w_{ij} h_j^{l-1} \right), \quad (3.2) \]
where \( \sigma(x) = 1/(1 + e^{-x}) \) is the logistic function. Once a DBN is trained, it can also be used in a backward way by factorizing the joint pdf as \( p_L(h^L)p_{L-1}(h^{L-1} | h^L) \ldots p_1(h^1 | h^2) \), where with an abuse of notation we also denote the backward transitional probabilities by \( p_l \). For the RBM-based model in (3.1), this factorization can be easily carried out, leading to similar expressions of (3.2), which allows us to reversely sample a set of hidden features from a given data point at the output layer.

DBNs are trained based on the Maximum Likelihood (MaL) principle. For a given set \( \theta = \{ \{ w_{ij}^l \}, \{ b_i^l \} \} \) of weights, denote the marginal distribution of the output \( h^L \) by \( p_0(h^L) \).
Then the MaL principle leads to the following optimization problem for training DBNs:

$$\min_{\theta} - \sum_{n=1}^{N} \log p_{\theta}(h^L = x_n). \quad (3.3)$$

The major difficulty in (3.3) is the calculation of the term $p_{\theta}(.)$, and its derivative is extremely difficult to calculate and making gradient-based optimization techniques not directly applicable. To overcome this difficulty, and make DBNs training more viable, we exploit in this paper, the GANs methodology to lift the numerical difficulties of the MaL-based optimization framework.

### 3.2.2 Proposed Method: Deep Adversarial Belief Networks

For training the DBNs, we adopt a similar solution to GANs. We consider the empirical measure $\hat{\mu} = \frac{1}{N} \sum_n \delta_{x_n}$ of the data set $\bar{X}$, where $\delta_x$ denotes Dirac’s delta measure at point $x \in \mathcal{X}$ and take the solution of the following optimization problem:

$$\min_{\mu \in \mathcal{M}} D(\hat{\mu}, \mu), \quad (3.4)$$

where $D$ is a positive distance or divergence function between two measures, and $\mathcal{M}$ is the set of probability measures $p_{\theta}(h^L)$ on $\mathcal{X}$ generated by the DBNs in (3.1). We observe that (3.4) generalizes the MaL framework in (3.3), since the latter is obtained as a special case, by letting $D$ be the Kullback Leibler divergence, i.e. $D(\hat{\mu}, \mu) = KL(\hat{\mu} \| \mu)$. More generally and similarly to GANs, we consider those distance (or divergence) functions $D$ that can be written as

$$D(\hat{\mu}, \mu) = \max_{f \in \mathcal{F}} E_{X \sim \hat{\mu}}[\phi(f(X))] + E_{Y \sim \mu}[\psi(f(Y))], \quad (3.5)$$

where $\mathcal{F}$ is a family of real-valued functions on $\mathcal{X}$, known as the discriminators. Furthermore, $\phi, \psi$ are two real functions and the notation $E_{X \sim \hat{\mu}}[\cdot], E_{Y \sim \mu}[\cdot]$ implies that the variables $X, Y$ in the arguments of expectation are respectively distributed according to $\hat{\mu}, \mu$. The
original GAN formulation uses $\phi(y) = \log(y)$ and $\psi(y) = \log(1 - y)$ with $\mathcal{F}$ as the set of all measurable functions, corresponding to the Jensen-Shannon divergence. The WGAN formalism is obtained by taking $\phi(x) = x$, $\psi(x) = -x$ and $\mathcal{F}$ as the set of all 1-Lipschitz functions, which leads to the Wasserstein distance between measures. The MaL framework in (3.3) can also be obtained by setting $\phi(x) = \log(x)$ and $\psi(x) = -x$. In practice, the discriminator $f \in \mathcal{F}$ is limited to the family of deep neural networks (DNNs) with a suitable fixed architecture. In this case, we denote the discriminator $f$ by $f_\rho$ where $\rho$ denotes the set of weights in the neural network at hand. Plugging (3.5) into (3.4) and using the above-mentioned specifications of the discriminator, we obtain the following optimization framework:

$$
\min_{\theta} \max_\rho \mathbb{E}_{X \sim \hat{\mu}}[\phi(f_\rho(X))] + \mathbb{E}_{Y \sim \mu=p_\theta}[\psi(f_\rho(Y))]. 
$$

(3.6)

Our proposed technique for training DBNs is hence entails solving the optimization problem in (3.6) to obtain the set $\theta$ of parameters of the underlying DBN. As we shortly elaborate, the stochastic gradient method provides a practical scheme for this purpose. We also observe that (3.6) bears a similar adversarial interpretation to GANs: As the loss function reflects the objective of the discriminator in distinguishing the "real" samples $X$ from the "fake" ones $Y$, the goal of the DBN ($p_\theta$) is to deceive the discriminator by counterfeiting "true samples" in the best possible way.

### 3.2.2.1 Algorithmic Details

The optimization problem in (3.6) can be solved by the SGD method: At each iteration $t = 1, 2, \ldots$ a set of samples (mini-batch) from either the data set $\bar{X}$ or the output $Y = h^t$ of the underlying DBN is randomly selected. The gradient $g^t$ with respect to both $\theta$ and $\rho$ of their corresponding term $\mathbb{E}[\phi(f_\rho(X))]$ or $\mathbb{E}[\psi(f_\rho(Y))]$ in the objective of (3.6), are estimated using the samples, and subsequently applied. When considering the $t^{\text{th}}$ iteration, if a set of $b$ samples $x_1^t, x_2^t, \ldots, x_b^t$ from the data set $\bar{X}$ is used, we adopt the standard procedure of
estimating the gradient by calculating the sample mean:

\[
\hat{g}_t = \left[ 0, \frac{1}{b} \sum_{i=1}^{b} \frac{\partial}{\partial \rho} \phi(f_\rho(x_i^t)) \right],
\]

where the estimate \( \hat{g}_t \) respectively includes the gradients with respect to \( \theta \) and \( \rho \) in the first and second entries. Note that the gradient with respect to \( \theta \) is zero in this case. The above solution is not applicable when the DBN samples \( y_1^t, y_2^t, \ldots, y_b^t \) are employed, since the relation of their corresponding term \( \mathbb{E}[\psi(f_\rho(Y))] \) to \( \theta \) is implicit in the underlying distribution \( p_\theta \). For this reason, we first express the exact gradient of this term with respect to \( \theta \) as\(^3\)

\[
\frac{\partial \mathbb{E}[\psi(f_\rho(Y))]}{\partial \theta} = \sum_h \psi(f_\rho(h^L)) p_\theta(h) \log p_\theta(h) = \mathbb{E}_{h \sim p_\theta} \left[ \psi(f_\rho(h^L)) \frac{\partial}{\partial \theta} \log p_\theta(h) \right],
\]

where \( h = (h^1, \ldots, h^L) \), and the notation \( p_\theta(h) \) is used to refer to the joint p.d.f in (3.1). We observe in (3.8) that the expected value on the right hand side is over all layers in \( h \), while the original expression on the left hand side is over the output \( Y = h^L \). Next, we estimate the right hand side by generating \( b \) samples \( h_1^t, h_2^t, \ldots, h_b^t \) of the entire network, where \( h_i^t = (h_1^{1,t}, h_2^{2,t}, \ldots, h_L^{L,t}) \) with \( y_i^t = h_i^{L,t} \) as the \( i^{th} \) sample of the output layer, and calculating the sample mean. This leads to the following expression for the gradient

\[
\hat{g}_t = \left[ \frac{1}{b} \sum_{i=1}^{b} \psi(f_\rho(y_i^t)) \frac{\partial \log p_\theta(h_i^t)}{\partial \theta}, \frac{1}{b} \sum_{i=1}^{b} \frac{\partial}{\partial \rho} \left( f_\rho(y_i^t) \right) \right].
\]

We notice that the term \( \frac{\partial \log p_\theta(h_i^t)}{\partial \theta} \) can be efficiently calculated on account of the Markovian properties of the DBNs, which allows us to efficiently express \( \log p_\theta(h_i^t) = \log p_1(h^1) + \sum_i \log p_i(h^i | h^{i-1}) \). This shows that the gradient of the variables at individual layers can

\(^3\)In the continuous variable case, the summation will be replaced by an integral, but the final expression remains unchanged.
be independently calculated in parallel, thus foregoing the back-propagation algorithm. This represents a great numerical advantage of adversarial DBNs over DNNs. We observe that for the RBM-based DBNs, calculating the term $\frac{\partial \log p_\theta(h^l)}{\partial \theta}$ amounts to differentiating the expressions in (3.2), which can be found in the standard literature of DBNs [Hin06b], and is hence skipped herein for space sake. Once the elements of the stochastic gradient $\hat{g}$ are calculated based on either (3.7) or (3.9), they are applied to their corresponding parameters with a suitable learning rate.

3.2.2.2 Extensions

Our training method by (3.7) and (3.9) enables us to extend the existing framework of DBNs in multiple respects:

**Modifying Mal Principle:** We can easily alter our training principle by modifying the pair of functions $\phi, \psi$. In particular, we consider the Wasserstein metric and the JS divergence in our next experiments, which are popular choices in the GAN literature.

**Non-RBM Layers:** As seen, our training technique is applicable to any DBN, such as (3.2), for which the derivative of the forward representation is simple to compute. For example, we may simply obtain the convolutional DBNs by replacing the linear terms $\sum w_{ij} h_{j}^{l-1}$ in (3.2) with a convolution. The resulting expressions and derivatives are similar to those in [Lee09a] and are hence skipped, we nevertheless use the resulting algorithm in our experiments. Further operations such as normalization factors and pooling can also be incorporated in the description of the transitional probabilities $p_i(h^l | h^{l-1})$ in (3.2), and their adoption is postponed to a future work as they will impact the reversibility property.

**Accelerated Learning:** Another advantage of our training methodology is that it admits standard techniques in optimization algorithms, such as acceleration and adaptive step size to improve convergence. We examine some of these approaches in our experiments.
3.2.2.3 Variance Reduction for Stabilization

While the fore-proposed method enables parallel weight updates, in contrast to the sequential back propagation for neural networks and contrastive divergence [Hin06b] for traditional DBN training, the mitigation in Eq. 3.8 is known to increase the variance of the estimated gradient. Variance reduction will be important when training with high dimensionality data and deep networks. Rebar [Tuc17] is a unbiased variance reduction technique for the REINFORCE gradient estimator as in Eq. 3.8. It reduces variance by using a continuous relaxation of discrete variables to construct a control variate, and the continuous relaxation can be efficiently estimated by a reparameterization procedure.

In our case, in lieu of just using the sample mean to estimate the gradient as in Eq. 3.9, we can apply the following equation using Rebar [Tuc17]:

$$\mathbb{E}_{h \sim p_\theta} \left[ \psi(f_\rho(h^L)) \frac{\partial}{\partial \theta} \log p_\theta(h) \right] = \mathbb{E}_{u, v} \left[ \left( \psi(f_\rho(h^L)) - \eta \psi(f_\rho(\sigma(\tilde{Z}^L))) \right) \frac{\partial}{\partial \theta} \log p_\theta(h) + \eta \frac{\partial}{\partial \theta} \psi(f_\rho(\sigma(Z^L))) - \eta \frac{\partial}{\partial \theta} \psi(f_\rho(\sigma(\tilde{Z}^L))), \right]$$

where $Z^L = g(u, \omega) = \log(\omega/(1-\omega)) + \log(u/(1-u))$ follows the Gumbel distribution. $\omega = p(h^L)$ is given by the last layer of DBN and $u \sim \text{Uniform}[0, 1]$. With a sigmoid function, $\sigma(Z^L)$ continuously relaxes the Bernoulli distribution of $h^L$. $\tilde{Z}^L = g(v, \omega) h^L$ that follows Gumbel distribution conditioned on $h^L$, and is calculated by replacing the sampling step in each layer by $\sigma(Z^L)$, to ensure continuity while passing through the DBN. $\eta$ is a scaling coefficient for the control variate. We apply Rebar in our training framework, and it significantly improves the convergence rate of the training.
3.3 Experiments

In this section, we test the viability of our proposed training algorithm, under various settings, by a way of two groups of numerical experiments. The first group concerns the application of DBNs to a computer vision problem, namely the MNIST dataset, containing 60,000 labeled samples of gray-scale hand-written digits for training and 10,000 more for testing. The goal is to check if DBN is able to generate MNIST-like digits and also help with classification. The second group investigates DBNs for modeling neural activities of the visual cortex as well as synthetic neural data.

3.3.1 Generation of MNIST-Like Digits

The goal of our first experiment is to generate synthetic handwritten digits by a DBN, with or without control over the generated digit. For this experiment, we use the dataset of [Sal08], containing 1797 samples of 8 by 8 cropped MNIST images, further binarized by thresholding the original grey scale images. In the first part of this experiment, a DBN is adversarially trained by (3.7) and (3.9) in an unsupervised way, i.e. by feeding the image samples as the last layer $h_L$ and discarding the labels. Sampling the resulting DBN generates handwritten instances with no control over the underlying digit. In the second part, another DBN is trained in a supervised way (with ground truth labels) to gain control over the generated digit. For this purpose, we adopt a similar approach to the conditional GAN structure [Mir14] by treating a pair of label and image as a data point $x_k$, which are respectively fed to the first ($h^1$) and last ($h^L$) layer of a DBN. The discriminator $f_p$ assumes this pair as an input and (3.7) and (3.9) are similarly used. The output of the discriminator in the two parts of our experiment can be interpreted as the likelihood of the input samples following the distribution of the training images, either unconditionally or given a label.

In both parts of our experiment, the discriminator is a 2-layer densely connected neural network, whose input vector length, the hidden layer, and the output layer respectively are
The output layer has a sigmoid activation function. In the first part, we employ a densely connected 3-layer DBN as in (3.2), and each with 64 units. For the second part, we add an input layer of length 10, corresponding to the one-hot encoding (converting categorical integers to a binary vector) of the ten digit labels.

The diagram of the DBN and the discriminator are shown in Fig. 3.1. Samples of the generated images in the two experiments are also shown in Fig. 3.1. Note that the last layer of the resulting DBNs generates binary images of the digits, while the values of the conditional distribution $p_L(h^L | h^{L-1})$ of the pixels of the last layer given the previous layer in (3.2), can be used as the grey scale images of the original MNIST digits, before thresholding. We show the results of the conditional distribution of the last layer in Fig. 3.1. We observe that the generated digits are well distributed and resemble the real digits. Moreover, in the bottom plot, each row is conditioned on a certain label, which shows that the generation of the digits is to a large extent, associated with the labels, while the digits maintain variability.
3.3.1.1 Training by Wasserstein distance

We also conducted the first part of our experiment using the WGAN formalism. Specifically, we apply weight clipping [Arj17] on the discriminator, remove the sigmoid function at the end of the discriminator, and change the loss function elements to $\phi(x) = x$, $\psi(x) = -x$ in (3.7) and (3.9) according to the Wasserstein distance. We show in Fig. 3.2 some generated samples of the digits using Wasserstein DBN, which demonstrates that training DBNs with different metrics than the KL divergence leads to different distributional properties of the generated images. In this experiment, we also explore different adaptive learning rate strategies. The convergence properties of three different optimizers, namely SGD, RMSprop [Tie12] and Adam [Kin15] are depicted on the right hand side of Fig. 3.2. As seen, SGD leads to highest variance and slowest convergence, while Adam results in the most suitable saddle point solution of (3.6).
3.3.1.2 High resolution MNIST Digits generation with variance reduction

Training on higher dimensional data is more difficult to converge due to high variance of the gradient estimator, even when using the Wasserstein distance. To train a DBN on the original 28 x 28 MNIST dataset, we apply the variance reduction technique Rebar [Tuc17] in our framework. Rebar is able to achieve a low variance, un-biased estimator for the gradient of our objective function as in Eq. 3.8, for which we used a sample mean to approximate. The idea is using Gumbel-softmax procedure to relax discrete distributions throughout DBN, which can be estimated efficiently by reparameterization, thereby lifting the bias of the estimator by using it as a control variate.

In Fig. 3.3, various digits with different styles are generated, while the curve of loss function is kept at a low variance. In Fig. 3.4, Rebar improves the convergence rate by at least an order of magnitude, while all the other training settings are consistent.

3.3.2 Classification of MNIST

In this experiment, we use the DBN formalism for classification of MNIST images. For this purpose, we train a conditional convolutional DBN on the original MNIST dataset using the proposed framework. This is similar to the second part of our previous experiment.
Figure 3.4 Loss function and discriminator scores of adversarial DBN without variance reduction (top) and with Rebar (bottom), as well as respective output distributions, in 10000 iterations.
Table 3.1 Classification accuracies when number of training samples is 100, 1000 and 60000 respectively, averaged over 100 trials, with standard deviation in the parentheses. Compared with CNN, the use of DBN as pretraining shows better generalizability for small training set.

<table>
<thead>
<tr>
<th>training set size</th>
<th>DBN + CNN</th>
<th>CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>77.772% (1.501%)</td>
<td>70.037% (6.32%)</td>
</tr>
<tr>
<td>1000</td>
<td>96.081% (0.622%)</td>
<td>94.625% (0.355%)</td>
</tr>
<tr>
<td>60000</td>
<td>99.140% (0.096%)</td>
<td>98.906% (0.092%)</td>
</tr>
</tbody>
</table>

with digits generation, with a slight difference that the DBN takes images as a non-binary input and outputs a vector of length 10. We repeat this experiment by binarizing the entire MNIST dataset, but since the two results are similar, we only present one set of results. The images are still used as the conditional inputs to the discriminator, as in Fig. 3.1. The DBN includes 4 convolutional layers, with the following number of filters [32,32,16,10], and their corresponding sizes [11,11,5,4]. In the discriminator architecture, we first feed the conditional image to a CNN to generate a feature vector of length 64. We subsequently concatenate the feature vector with the DBN’s output (generated labels) and pass the result through a two layer linear neural network.

Since generative models are not specialized for classification, their training is usually followed by a fine tuning stage, where their weights are updated by backpropagation as a conventional CNN [Lee09b]. We use the first layer of a trained convolutional DBN as the pretrained weights for the first layer filters of a CNN, and fine-tune the weights using backpropagation. The classification performance is compared with direct training of a CNN classifier of the same structure. The resulting accuracy for different sizes of the training set are listed in Table 3.1. This shows that CNN initialized by DBN outperforms normal CNN for a small training set and has better generalizability. For the CNN without DBN pre-training and a small size of the training set, the result is highly unstable. In contrast, CNN initialized by DBN exhibits a considerably more consistent performance. The reported accuracy for CNN is the average of 100 runs.
3.3.3 Modeling Neural Activities

Modelling neural spikes is a natural application of DBNs [Lee08]. The power of DBNs in addressing the limitations of DNNs, such as sparsity and reduced datasets, is demonstrated in this task. They naturally generate binary outputs, which can be easily interpreted as neural spikes. In this experiment, we show that a DBN is capable of modeling sparse spike signals.

3.3.3.1 Neuronal spike generation

In this experiment, we use a dataset recorded by a two-photon calcium imaging system capturing large scale neural activities [Sti16; Hua18]. The activities of individual neurons are simultaneously recorded as time series in the primary visual cortex (V1) and the antero-lateral (AL) areas of the visual cortex of an awake mouse. Top 50 neurons whose activities are most correlated across 20 trials are selected for modeling, and their binary spike trains are obtained by applying a standard deconvolution technique to the recorded fluorescence time series.

We model the spikes of the 50 neurons independent of the visual stimuli with a four-layer dense DBN. The number of units per layer are [128, 128, 128, 50]. The discriminator is a two-layer dense neural network with 64 neurons in the hidden layer. The training procedure is similar to the first part of the experiment in Section 3.3.1. In Fig. 3.5, the firing probabilities for individual neurons, given by the DBN are depicted, which exhibit high resemblance to the real firing rates, despite the fact that the overall firing rate is very low (about 1 spike per 100 frames). We also verify that the log-likelihood of the real data on DBN increases during training. The likelihood of the output layer of the DBN is estimated by sampling $p_L(h^L \mid h^{L-1})$, the distribution of the last layer conditioned on the penultimate layer, and averaging, amounting to the total probability rule.
3.3.3.2 DBN Data Generation: Spike Prediction

To demonstrate the potential of our DBN, we proceed to simulate and model a neural system using the conditional DBN, and examine its the performance with two experiments. Specifically, we assume in a physically connected neuronal network, if a subset of neurons are stimulated, other neurons will fire spikes (or not) according to a specific probability distribution. To assess the capability of the DBN model and our proposed training method, we opt to use a separate fixed DBN to generate synthetic data, primarily due to our limited access to sufficient real neuronal data. In the first experiment, the input layer represents a set of neurons that can be stimulated, and the output layer serves as a set of observed neurons. The state of each layer $h^l$ is a binary vector $[h_1^l, h_2^l, ..., h_N^l]$, meaning whether each of the $N$ neurons fires a spike. We subsequently use the synthetic data to train DBNs and neural networks for comparison. In the second experiment, we increase the input layer to higher dimensions with real-valued data to simulate visual stimuli.

In the first experiment, the specific steps to generate synthetic data are as follows: We construct a DBN with 20 input neurons, 3 hidden layers with 16 neurons to immitate a highly nonlinear system, and 10 output neurons. In hidden and output layers, the "firing" probability of each neuron is in the form of $\sigma(x_i^l/\tau - b)$, where $x_i^l = \sum_j w_{ij}h_j^{l-1}$. We can control the firing sparseness by $b$ and randomness by $\tau$. A set of sparse input vectors for
DBN is generated, such that $\sum_i h_1^i \leq 5$, i.e. no more than 5 out of 20 neurons fire at the same time. For each input $h_1^i$, the DBN generates an output binary vector $h^L_1$ at the last layer $L$, as well as the conditional distribution $p_L(h^L | h^{L-1})$. Only binary inputs $h_1^i$ and outputs $h^L_1$ are used in training, as we can not observe the probability but spikes in neurons. The data set is split into a training, validation and testing set of up to 10000, 1700 and 10000 samples respectively.

Since the synthetic data is generated by a known DBN, for a given input $h_1^i$, we can estimate the underlying conditional distribution $p(h^L | h_1^i)$, by sampling and averaging over $p_L(h^L | h^{L-1})$. We subsequently use it as the ground truth to compare with the predicted distribution $\hat{p}(h^L | h_1^i)$ from the learned model. Note that we use spike data in training, but directly use probability in testing for better evaluation accuracy. In terms of evaluation metrics, we calculate the correlation coefficient between $p(h^L | h_1^i)$ and $\hat{p}(h^L | h_1^i)$ for each $h_1^i$. For each output neuron $i$, we also calculate the correlation coefficient between the sequence of firing probabilities $[p(h^L_i | h_1^i), p(h^L_i | h_1^i), ..., p(h^L_i | h_1^{\text{sample}})]$ and the predicted ones. To check the magnitude differences between real and predicted probabilities, we calculate the Kullback–Leibler divergence and Jensen-Shannon divergence between Bernoulli distributions $p(h^L_i | h_1^i)$ and $\hat{p}(h^L_i | h_1^i)$ by following equations:

$$D_{KL}(p||q) = p \log \frac{p}{q} + (1-p) \log \frac{1-p}{1-q}$$
$$D_{JS}(p||q) = \frac{1}{2}D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2}D_{KL}(q||\frac{p+q}{2}).$$

The trained DBN has one hidden layer of 32 neurons and a neural network of the same structure, is trained to compare with. The evaluation metrics are shown in Fig. 3.6 as a function of sizes of the training set, to show the advantage of DBN with a small training set.

In the second experiment, we increase the input dimensionality to 64 to simulate visual stimuli of 8 by 8 images. The values in the input layer are now uniformly distributed between 0 and 1, instead of being binary. Other settings remain the same as the first experiment.
Figure 3.6 Mean correlation coefficient between $p(h^2|h^1)$ and $\hat{p}(h^2|h^1)$ (A), mean correlation coefficient between true and predicted neural firing probability sequences (B) and mean Kullback–Leibler divergence and Jensen-Shannon divergence between $p(h^1_i|h^1)$ and $\hat{p}(h^1_i|h^1)$ (C, D). Input has binary values and a dimensionality of 20, to simulate neural stimulation. Each data point is averaged over 50 trials, with the error bar being the standard error.
Figure 3.7 Mean correlation coefficient between $p(h^L|h^1)$ and $\hat{p}(h^L|h^1)$ (A), mean correlation coefficient between true and predicted neural firing probability sequences (B) and mean Kullback–Leibler divergence and Jensen-Shannon divergence between $p(h^L|h^1)$ and $\hat{p}(h^L|h^1)$ (C,D). Input has random values between 0 and 1 and a dimensionality of 64, to simulate visual stimuli of 8 by 8 images. Each data point is averaged over 50 trials, with the error bar being the standard error.
From both experiments, we observe DBN being comparable with NN in predicting neural spikes, but showing clear advantages when the training set is small, which is consistent with the results in our MNIST classification experiment.

We notice that in Fig. 3.7, for DBN, the largest training set seems to lead to less accurate predictions. By checking the training record for all trials, we find that it is caused by failure of convergence or numerical instability in some of the trials. This problem seems to happen less frequently with smaller training set. We conjecture that the minimax game in GAN framework is highly sensitive to gradient singularity caused by difficult samples. In this sense, the formulation of GAN may be ill-posed.

Similar to Fig. 3.1, where the loss function and discriminator scores start to oscillate heavily even after the generator and the discriminator seem to have reached a relatively stable stage, we find that in many failure cases in the experiments of Fig. 3.7, numerical instability happens after a seemingly convergence is reached. This may be related to the fact that a Nash equilibrium of GAN is hard to achieve if the generator and the discriminator are updated by gradient descent.

From this perspective, to potentially address such stability problems, we can work on updating schedules and weight optimizers. For example, we may consider optimizing the discriminator several times while the generator is updated once. This approach may avoid occasional bad samples to cause extreme discriminator outputs, and also tune the dynamics of the minimax game.

Another source of the unstable gradient may be the loss function. The JS divergence implemented in standard GAN can be non-differentiable. While the Wasserstein distance in WGAN is theoretically smooth, a perfect implementation of it is still lacking. The discriminator network is supposed to choose from the family of all Lipschitz continuous functions. However current methods limit the choices of functions or do not guarantee the Lipschitz continuity. Therefore, a better implementation of WGAN will also help with the instability issue.
3.4 Conclusion

In this work, we proposed an adversarial training framework for DBNs. The experiments verify that the DBN is able to be trained in the adversarial framework, and our method performs well under different structures and settings, including GAN, WGAN, conditional GAN and various optimizers. We also show the advantage of DBN over neural network in the scenario of small training sets. The development of this method opens a promising way to train complex DBNs and improve scalability.
4.1 Community Detection

4.1.1 Conclusion

In this dissertation, we first introduced the application of graphical models to the problem of community detection in multiplex networks. We propose a stochastic block model for multiplex networks that improves detectability for noisy community structures, in cases when the same community can be found across multiple layers. We design a WPP constraint into the standard SBM so it can generate communities according to our problem assumption. In contrast to some other methods that aim for a similar goal, our model does not presume explicit correlation of structures between layers, but a more generic assumption
that communities do not overlap in the same layer and may appear in multiple layers. Therefore, it is suitable for both homogeneous and heterogeneous multiplex networks. Experiments are performed to demonstrate the effectiveness of the model.

### 4.1.2 Future Works

A direct implication of this work, is that we are now able to improve community detectability in general multiplex networks, therefore the communities that are too weak to be detected in a single layer network, may become detectable. Furthermore, our method covers more real world scenarios than the method of layer aggregation. With our method, we expect new and more accurate community structures can be found in networks where WPP is a reasonable assumption, such as social networks and protein-genetic interaction networks.

Our method is compatible and can be combined with other probabilistic models, to be even more practical for real-world networks. For example, we can consider incorporating degree-corrected SBM to describe specific topology in social networks [Kar11; New16b]. For more general networks, the WPP constraint can be generalized to non-multiplex multilayer networks. Numerical performance is also worth improving, and that includes the design of factor graph and message passing techniques.

For more general implications to the field of community detection, we show how to incorporate complex constraint in the problem of community detection by graphical modelling, and it is possible to solve such models with belief propagation. That encourages other types of constraints to be developed for specific types of community structures and problem settings.
4.2 Deep Adversarial Belief Networks

4.2.1 Conclusion

We proposed the Deep Adversarial Belief Network (DABN), a novel adversarial training framework for the Deep Belief Network (DBN). By replacing the generator of a generative adversarial network by a DBN, we utilize the discriminator to measure the distributional distance between DBN outputs and training samples. Compared to the existing DBN training technique that attempts to approximately maximizing log-likelihood (i.e. minimizing the KL divergence), DABN is flexible in choosing the metrics to minimize, such as Wasserstein distance, which may have better convergence behavior in practice. Furthermore, another advantage of DABN is that it is able to update layers in parallel, thus having better scalability, in contrast to sequential training of DBN layers in existing methods. Experiments under various settings show the viability of training DBN with an adversarial framework. We also applied variance reduction technique to further improve the training speed.

While deep neural networks have great success in many areas, they usually require large amount of training data to get good model generalizability. On contrary, DBN is more suitable when training set is small. Using the proposed framework, we demonstrate training a DBN as a conditional generator of MNIST images, and show that the convolutional DBN provides a better weight initialization for CNN as a MNIST classifier, especially for a small training set. With synthetic sparse neural spike data, DBNs are trained to predict neural spikes conditioned on the input. DBN has comparable performance with neural networks, but noticeable advantages in small data situations.

4.2.2 Future Works

The development of our framework opens many interesting directions for future works. First of all, it makes possible more general definition of DBN and a higher flexibility of DBN
structure design. Especially, it does not require any backward sampling or propagation during training, which frees the DBN from being required to be reversible in every layer, and allows non-RBM modules to be used in the model. For example, we may consider borrowing techniques from modern development of deep neural networks, such as pooling and batch normalization etc.

Also, it is now possible to use a DBN as the generator of GAN. That means, under the framework of GAN, some types of data (such as sparse or discrete data) which it is hard for a neural network to represent, now have the DBN as another option.

The use of GAN framework reliefs the training of DBN from the maximum likelihood principle, which can be problematic in convergence. Other metrics to measure the divergence between distributions can be explored, including better ways to implement WGAN, especially considering that differences between DBN and DNN may lead to different suitable metrics. Nevertheless, DABN can still benefit from the wide and active study in the field of GAN. For example, the training stability of GAN can be improved by penalizing the discriminator [Tha19] and such work can be directly utilized to the training of DBN through our framework.

Another interesting future direction is the application, especially for small and discrete data modelling. For example, neural spike train data is naturally modelled by DBN with binary units. High variance and randomness of neural signal also support the use of a probabilistic model. To better model neural data, we may explore more dedicated DBN structures, such as 3D convolution and recurrent structures for dynamics modelling.
BIBLIOGRAPHY


Lon10 Long, P. M. & Servedio, R. A. “Restricted boltzmann machines are hard to approximately evaluate or simulate”. *Proceedings of the 27th International Conference*


In this section, we mathematically prove our previous claim that the local constraints in Section 2.2.2 are equivalent to the WPP. Take an $L$–layered multiplex network consisting of a sequence of graphs $G_l = (V, E_l)$ for $l = 1, 2, \ldots, L$ with the same set of $N$ nodes $V = \{v_1, v_2, \ldots, v_N\}$ and different set of edges $E_l \subseteq V \times V$. According to WPP, the definition of the communities is independent of the layers. Hence, we consider a family $\mathcal{C} = \{C_1, C_2, \ldots, C_Q\}$ of $Q$ subsets $C_q \subset V$ of nodes as the communities. The presence of communities at different layers is represented by a community structure $\mathcal{S}$, which is a sequence $\mathcal{S} = (S_1, S_2, \ldots, S_L)$, where $S_l \subset \mathcal{C}$ is the subset of clusters being present at the $l^{th}$ layer. Now, we may define WPP in the following way:
**Definition 1.** A triple $(V, C, S)$ of community structures is said to satisfy the well partitioned property (WPP). If,

$$\forall l \in [L], C_a, C_b \in S_l; \quad C_a \cap C_b \neq \emptyset \rightarrow C_a = C_b$$  \hspace{1cm} (A.1)

We further make the following definition:

**Definition 2.** A triple $(V, C, S)$ of community structures is said to be "observable" if each community $C_a \in C$ appears in at least one layer, i.e. $C_a \in S_l$ for some $l \in [L]$.

We take the community assignments $t_i(l)$ as defined in Section 2.2.2. Then, we have the following theorem:

**Theorem 1.**

1. A set of community assignments $t_i(l) \in [Q] \cup \{\emptyset\}$ for $i \in [N]$ and $l \in [L]$ corresponds to a community structure $(V, C, S)$ satisfying WPP if and only if:

$$\forall i, j \in [N], l, k \in [L];$$

$$(t_i(l) = t_j(k) \neq \emptyset) \rightarrow (t_i(k) = t_j(k))$$  \hspace{1cm} (A.2)

2. The community assignment uniquely identifies observable community structures.

**Proof.** For the “only if” part in part 1, take $t_i(l) \in [Q] \cup \{\emptyset\}$ as the natural labeling of some layered complex $(V, C, \mathcal{R})$. For any arbitrary given indices $i, j, l, k$, if $t_i(l) = t_j(k) \neq \emptyset$, we have that $t_i(l) = t_j(k) = q$ for some $q \in [Q]$. This means that $C_q \in H_k$ and $v_i \in C_q$. Hence, from definition we have $t_i(k) = q$, which proves the “only if” part. For the “if” part in part 1, take a labeling satisfying the condition above. For each $q \in [Q]$ define

$$C_q = \{v_i \mid \exists l \in [L]; t_i(l) = q\}$$

$$S_l = \{C_q \mid \exists i \in [n]; t_i(l) = q\}$$  \hspace{1cm} (A.3)
as in Section 2.2.2. Now, we show that \( C = \{ C_q \neq \emptyset : q \in Q \} \) and \( \mathcal{S} = (S_1, S_2, \ldots, S_L) \) satisfy WPP complex with \( \{ t_i(l) \} \) is its natural labeling. Take two communities \( C_a, C_b \in H_l \) where \( C_a \cap C_b \neq \emptyset \). Then, we can take \( m \in C_a \cap C_b \). Moreover, by definition there exist nodes \( i, j \) such that \( t_i(l) = a \) and \( t_j(l) = b \). Since \( m \in C_a \), there exists a layer \( k \) such that \( t_m(k) = a = t_i(l) \). From the assumption in (A.2), we get that \( t_m(l) = a \). On the other hand, \( m \in C_b \) implies with a similar approach that \( t_m(l) = b \). We conclude that \( a = b \), which shows that \( (V, \mathcal{C}, \mathcal{K}) \) satisfies WPP. Suppose that \( t_i(l) = q \). Then, by definition \( i \in C_q \) and \( C_q \in H_l \) which shows that \( t_i(l) \) corresponds to \( (V, \mathcal{C}, \mathcal{K}) \). This completes the proof of part 1.

For part 2, simply note that the relations in (A.3) hold for any assignment \( t_i(l) \) of an observable community structure. \( \square \)

In the above, we show that the WPP is equivalent to the constraint in (A.2). Now, we show that this set is equivalent to the set of constraints in Section 2.2.2.

**Theorem 2.** The set of constraints in (A.2) is equivalent to the set of constraints in Section 2.2.2 of the paper.

**Proof.** Let us first show that the constraints in 2.2.2 imply (A.2). Take arbitrary given indices \( i, j, l, k \) and suppose that \( t_i(l) = t_j(k) = a \neq \emptyset \). Note that from the constraints of 2.2.2 \( t_i(k) = a \), since otherwise it contradicts the last line of the constraints in Section 2.2.2. This proves (A.2). Now, let us prove the converse. Assuming (A.2), take again arbitrary given indices \( i, j, l, l' \) and denote \( t_i(l) = a \) and \( t_j(l) = \beta \). If \( a = \beta \), then the first line of constraints in Section 2.2.2 must hold since otherwise, exactly one of the two labels \( t_i(l'), t_j(l') \), say \( t_i(l') \) equals \( a \). Then, since \( t_i(l') = t_j(l) \), we have from (A.2) that \( t_j(l') = t_i(l') = a \), which is a contradiction. If \( a \neq \beta \), then assuming \( t_i(l') = \beta \) leads to \( t_i(l') = t_j(l) \), which according to (A.2) leads to \( t_i(l) = t_j(l) \), which is a contradiction. This shows that \( t_i(l') \neq \beta \). Similarly, we get \( t_j(l') \neq a \), which prove the second line of constraints in Section 2.2.2. This completes the proof. \( \square \)