用于社区检测的类深度自动编码器非负矩阵分解

研读笔记

中南大学彭汪祺，2021/6/27

1. 摘要

本次阅读的论文为 ***Deep Autoencoder-like Nonnegative Matrix Factorization for Community Detection***，同时阅读了这篇论文中涉及程度较深的几篇其它论文，这些论文均做了相关的笔记记录在文中。目标论文主要从以下几个方面来研读：

1、**论文所在的社区检测领域的基本背景**，以及我对社区检测的一些个人理解；

2、**前人在社区检测领域的研究成果**，这里我主要选择了传统模块化社区检测模型，以及 NMF 社区检测模型，并且阅读了相关的两篇论文 ***Modularity and community structure in networks***，***Overlapping Community Detection in Complex Networks using Symmetric Binary Matrix Factorization***，并且将对它们的理解写在了笔记中，目的是充分了解社区检测理论基础与常见处理模型；

3、**论文的预备知识**，这里主要通过阅读论文 ***educing the Dimensionality of Data with Neural Networks***，理解了自编码器的基本理论和数学原理，另外通过 Python 代码实践，初步掌握了 t-SNE 高维数据可视化工具的使用；

4、**正式研读了论文的主要工作**，理解其在原有社区检测基础上的创新点，DANMF 模型的数学推导、优化、训练过程，以及该模型的测试比较结果；

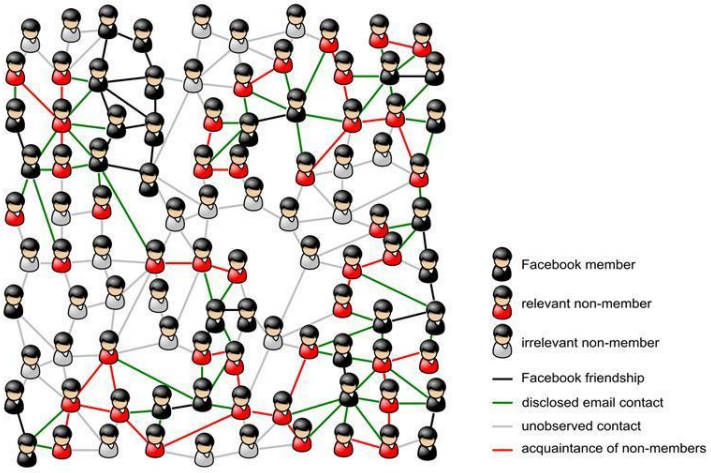
5、对论文提出的 DANMF 模型和优化算法进行**总结**；

6、**关于模型的进一步优化**，从论文 ***Anomaly Detection with Robust Deep Autoencoders*** 中受到启发，个人认为可以加入鲁棒深度自编码器，构建 RDANMF 模型，来增强模型对于原始网络数据的抗噪声干扰性能。

2. 背景

无论是自然界还是人类社会，都有许多复杂的交互系统，我们可以用复杂的网络来表征这些复杂系统，例如社交网络、协作网络、生物神经网络和蛋白质交互网络等等。

一个复杂的网络由多个社区组成，而一个社区由具有密集内部连接和外部连接的一组节点组成。我个人认为这可以用人的社交网络来具象解释，如下图：



该图体现了一个复杂的社交网络，网络中不同人之间的关系有亲疏之别，肉眼可见地形成了一个个社交圈，而得到这张原始的社区网络图后，我们更想推导出这个网络中哪些人属于哪些社交圈子，以及这个社交网络的组织模式等信息，那该如何分析出这个社交网络中隐藏的信息呢？

显然，为了能够深刻地分析一个复杂网络，我们需要采用各种算法对网络中的社区进行检测，检测出的社区分类越明显，我们越能接近这个网络的信息学本质。而社区与原始网络之间的映射关系，能够反映出这个网络中隐藏的信息。例如，如果我们能更加准确地对上图中的社交网络进行检测和分类，或者说检测得到的社交圈子差别越明显，我们将更容易推测出这个社交网络的结构与模式。

3. 前人研究成果

2.1 传统社区检测方法

最开始，研究人员一种传统的社区检测算法来分析复杂网络。传统社区检测法试图通过优化某些标准来找到最优的社区结构，例如模块性、标准化切割性等。

为了对传统社区检测方法有进一步的理解，我阅读了参考文献中关于利用模块性来进行社区检测的论文 ***Modularity and community structure in networks***, 这篇论文中，研究人员采用了**最佳模块化算法**来对一个网络进行了划分，从而检测到社区。**研究人员在论文中提出了一个用于判断模块化程度的参量——进行一次划分时，落在模块内的边数减去等效网络中随机边数的期望值。模块化参量越大，说明该模块（社区）内部连接越密集，外部连接越稀疏，划分效果就越好。将这个问题转化为一个模块化参数矩阵，通过不断调整该矩阵的特征向量，使其特征值最大，就得到了该网络最佳的模块化划分。**这里我们以对一个网络进行二分来作为例子，说明研究人员提出的模块化检测模型：

假定在一个网络中，A_{ij}代表节点 i 与 j 之间所连接的边的条数，k_i和k_j分别代表节点 i 与 j 的度，m 代表网络中边的总数，$S_i = \begin{cases} 1，i属于模块1 \\ -1，i属于模块2 \end{cases}$，根据上文所述，我们可以将模块化函数 Q 表示为：

$$Q=\frac{1}{4m}\sum_{ij}\left(A_{ij}-\frac{k_ik_j}{2m}\right)(s_is_j+1)=\frac{1}{4m}\sum_{ij}\left(A_{ij}-\frac{k_ik_j}{2m}\right)s_is_j$$

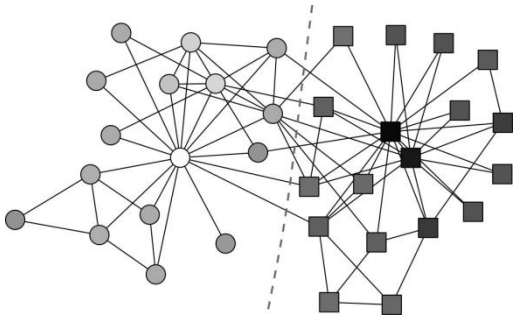
再令$B_{ij}=A_{ij}-\frac{k_ik_j}{2m}$，得到如下式子：

$$Q=\frac{1}{4m}\mathbf{s}^T\mathbf{B}\mathbf{s}$$

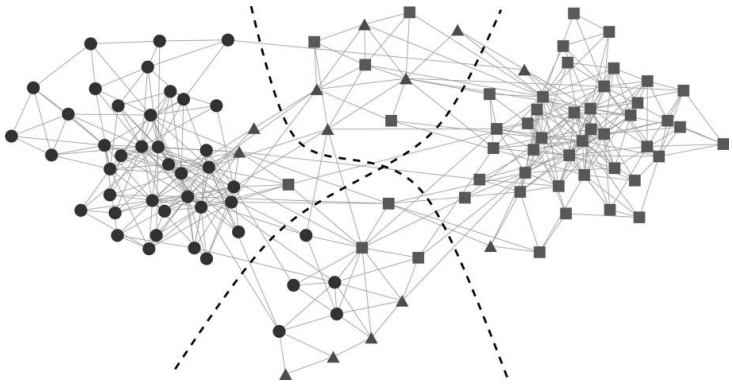
将 s 写成 B 的归一化特征向量u_i的线性组合 $s=\sum_{i=1}^{n}a_i\mathbf{u_i}$，$\beta_i$是 B 的关于$u_i$的特征值，可得 Q 的式子如下：

$$Q=\frac{1}{4m}\sum_i a_i\mathbf{u}_i^T\mathbf{B}\sum_j a_j\mathbf{u}_j=\frac{1}{4m}\sum_{i=1}^{n}\left(\mathbf{u}_i^T\cdot\mathbf{s}\right)^2\beta_i$$

这里就得到了 s 与β的关系式，通过不断调整 s（即网络划分），来使特征值最大，这样就得到了最佳的模块化社区检测。另外，可以从该论文中发现，度越大的节点，它的每次划分对整体结果的影响越大，因此网络中越中心化的节点，对于整个网络结构与社区划分结果的决定作用越强。



对于多个社区划分，研究者在两个社区划分的基础上，对每个模块重复上述划分过程，如果发现对该模块进一步的划分，对于整体的特征值有正向影响，则这次划分是有效的，如果划分的结果对于整体特征值的影响为零甚至负，则这次划分无效。多社区划分结果可以见下图：



但是，这些传统社区检测的算法中，更习惯于将一个节点划分到某个单独的社区中，很明显这是不合常理的，因为它没有考虑到社区重叠的情况。社区重叠的现象在社会中十分常见，例如，一个人可能既活跃在知乎社区，也活跃在豆瓣社区，将它单独划分到其中一个社区都是不恰当的，不利于我们对复杂虚拟社交网络进行分析。这是传统社区检测最大的问题。

2.2 NMF 社区检测方法

由于传统社区检测存在社区不可重叠的问题，近年来，非负矩阵分解（NMF）已被广泛采用，它对不重叠与重叠社区检测的适应性和可解释性更强。**基于 NMF 的社区检测方法将给定网络的领接矩阵近似分解为两个非负因子矩阵，即$\mathbf{A} \approx \mathbf{UV}$ $(\mathbf{U} \geq \mathbf{0}, \mathbf{V} \geq \mathbf{0})$，因子矩阵 V 的每一列即为属于不同社区的节点的倾向（即社区成员），而因子矩阵 U 可以被视为原始网络与社区成员空间之间的映射**。

为了对 NMF 社区检测有进一步的认识，我阅读了参考文献中关于采用对称二元矩阵分解来检测重叠社区的论文 ***Overlapping Community Detection in Complex Networks using Symmetric Binary Matrix Factorization***，针对传统社区检测方法在重叠社区检测方面的缺陷，该论文的研究者提出了对称二进制矩阵分解(SBMF)模型，该模型结合了模糊社区重叠检测和非模糊社区重叠检测模型的优点，可以明确地将社区成员资格分配给节点，并可以区分离群点和重叠节点。

模型的数学构建方法如下：

首先设定 $n \times n$ 的二进制矩阵 A，代表原始网络数据；$n \times c$ 的社区成员矩阵 U，其中存在关系$U_{it} = \begin{cases} 1，结点\ i\ 在社区\ t\ 中 \\ 0，结点\ i\ 不在社区\ t\ 中 \end{cases}$，这样就可以用矩阵表达一个节点位于多个社区中的情况。当然，存在少数孤立节点不位于任何社区中的情况。因此，有如下关系式：

$$\begin{cases} \sum_j U_{ij} > 1，\ 节点i对应多个社区 \\ \sum_j U_{ij} = 1，\ 节点i对应一个社区 \\ \sum_j U_{ij} = 0，\ 节点为孤立节点 \end{cases}$$

对于孤立节点，需要在优化模型中设定惩罚值，因此 SBMF 的优化模型如下所示：

$$\min_U \left\| A - UU^T \right\|_1 + \sum_i \left(1 - \Theta \left(\sum_j U_{ij} \right) \right)$$
$$\text{s.t. } U_{ij}^2 - U_{ij} = 0, i = 1, 2, \ldots, n, j = 1, 2, \ldots, c$$

孤立节点的存在会使惩罚值增大，优化的目的在于得到函数的最小值，因此必须得到最优的社区成员矩阵 U，使得 A 与 UU^T尽量接近，且孤立节点足够少。

然而，该优化模型规模大且不连续，求解较为困难。为了求得该优化模型中的矩阵 U，我们首先通过求解 SBMF模型初始化 U，然后通过求解以下更简单的无约束非线性编程，得到最优近似解：

$$\min_u \left\| A - \Theta(U - u)\Theta(U - u)^T \right\|_1 + \\ + \sum_i \left(1 - \sum_j \Theta(U - u)_{ij} \right)$$

此外，论文中还提出了一种利用修正分区密度推断适当数量社区的方法。采用传统社区检测方法中的模块法来推测社区数量误差较大，研究者创建了一个新的数学模型来推测不同社区数量下的分区密度：

设n_α为社区α中的节点总数，m_α为社区α中连接边总数，N 为所有社区及孤立节点规模之和。同时，因为同时属于多个社区的节点并不多见，因此这里设立一个惩罚值来抑制单节

点对应多社区的情况——$q_\alpha = max_{j\in\alpha} l_j$，其中$l_j$代表节点 j 所拥有的的社区标签数。论文中给出的模型为：

$$D = \frac{2}{N}\sum_{\alpha=1}^{c}\frac{n_\alpha}{q_\alpha}\frac{m_\alpha-(n_\alpha-1)}{(n_\alpha-2)(n_\alpha-1)}$$

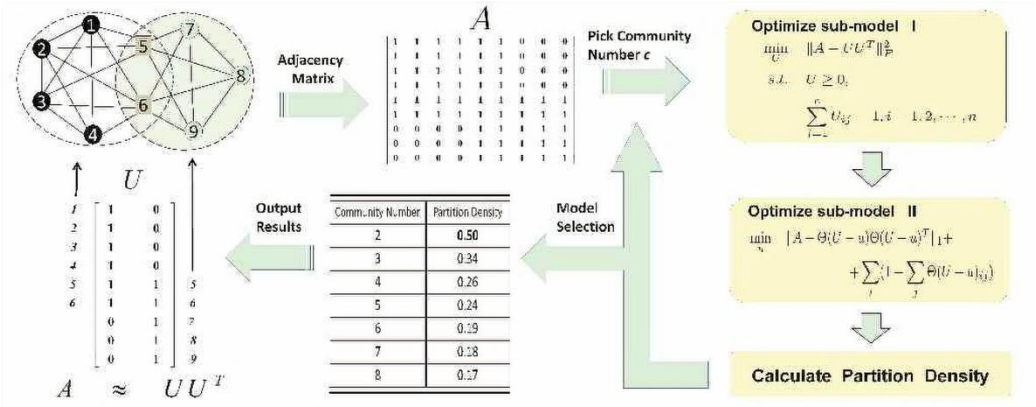
在这个数学模型中，每个社区的权重是由原始公式中社区中的边数给出的。这里使用节点的数量作为权值，同时对单节点对应多个社区的情况进行一定抑制，可以得到更好的结果。

综上，整个模型的运行过程为：

(1) 原始网络矩阵 A，通过 SBMF 优化模型，求解出社区成员矩阵 U；

(2) 分区密度模型对不同社区数目下的 U，进行分区密度评估，选出最合适的社区数；

(3) 得到社区成员矩阵 U，从而揭示复杂网络中的社区结构。



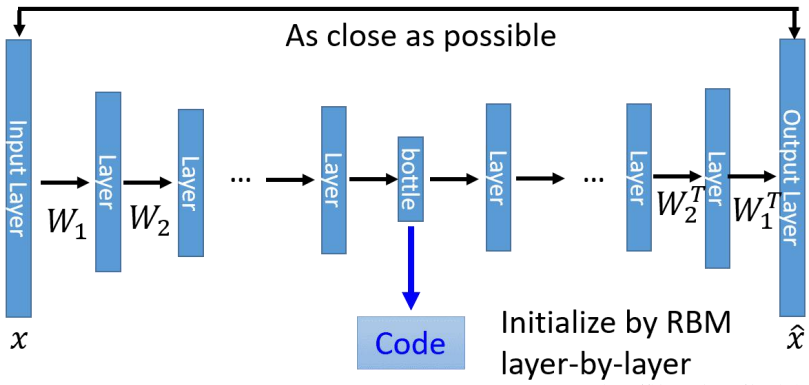
从这篇论文来看，相对于传统的模块化社区检测，SBMF 模型（NMF 模型中的一种）较为有效地解决了重叠社区带来的节点分配问题。

但是，NMF 模型也存在不足。从上述论文来看，NMF 模型只是简单地将一个复杂原始网络的邻接矩阵表示为两个矩阵之积，也就是用一个映射矩阵来表示这复杂网络背后的信息。显而易见，一个矩阵表示的信息量非常有限，真实世界中一个复杂网络背后的信息，是很难用一层映射来表示的，因此 NMF 只适合用于检测较为简单的、可能存在社区重叠的网络。

4. 论文预备知识

4.1 深度自编码器

深度自编码器是一种典型的无监督学习算法，它的基本思想是，**输入的数据乘以一个矩阵得到降维之后的结果，之后再将这个数据结果乘以之前权重矩阵的转置，可以恢复得到近似的原始数据。如果将这个降维-重构的过程通过多层矩阵来表示，那么这些矩阵的维度必定是层层缩小的，我们可以将这个降维-重构的过程视为编码-解码的过程——某个大维度的数据集，通过层层编码，被表示为一个小维度的编码，再通过层层解码，恢复为一个大维度数据，编码前与解码后的数据越接近，那么这个模型训练得越好，这个降维得到的结果就越有意义**，整个过程可以通过以下图片表示：



为了对深度自编码器背后的底层数学原理原理有进一步的理解，我阅读了 G. E. Hinton 和 R. R. Salakhutdinov 的论文 ***educing the Dimensionality of Data with Neural Networks***，这篇 paper 描述一种非线性的 PCA 的推广，利用一个自适应的、多层的编码网络，达到降维的目的。**首先对编码网络和解码网络进行随机初始权值，然后通过最小化原始数据和重构数据之间的差异，来训练这两个网络。然后利用链规则，先后通过解码器网络和编码器网络来传播误差，从而获得梯度，梯度用于对网络权值进行微调。**该论文提出了自编码器，并证明了它相对于 PCA 降维算法的优越性。

然而，优化多隐层的非线性自编码器的权重非常困难。使用大的初始权重，自编码器往往会陷入局部极小的情况；使用小的初始权重，前几层的梯度非常小，使得训练多隐层的自编码器变得不可行。针对这个问题，研究人员提出了逐层预训练的方法，这里采用二值向量的集合（比如图片）作为输入数据的例子，来解释逐层预训练的底层原理与过程。

二值向量的集合能够被限制玻尔兹曼向量机（RBM）的双层网络来建模，在 RBM 中，随机二值像素与随机二值特征探测子相连，随机二值像素相当于 RBM 的可见节点（visible units），而随机二值特征探测子相当于 RBM 中的隐层节点（hidden units）。这里设v_i和h_i代表像素 i 和特征 j 的二值状态，b_i和b_j是像素 i 和特征 j 的偏置，w_{ij}是像素 i 和特征 j 之间的权值，研究人员给出了一个可见层与隐藏层的联合分布（v，h）的能量函数：

$$E(v,h) = -\sum_{i\in \text{pixels}} b_i v_i - \sum_{j\in \text{features}} b_i h_i - \sum_{i,j} v_i h_i w_{ij}$$

可以通过调整权值和偏置的方式来提高训练图片的概率以此来降低图片的能量，并且提升相似"虚构"图片的能量。权重的变化由下式给出：

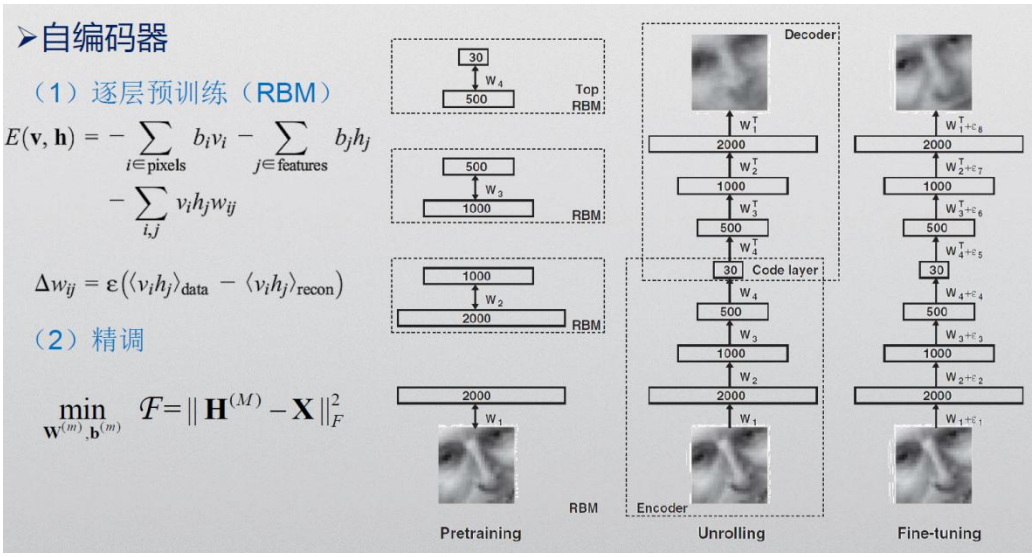
$$\Delta w_{ij} = \varepsilon\left(\langle v_i h_j\rangle_{\text{data}} - \langle v_i h_j\rangle_{\text{recon}}\right)$$

其中 ε 代表学习率，$\langle v_i h_j\rangle_{data}$代表当特征探测子由数据驱动时，像素 i 和特征探测子 j 在一起的时间分数，$\langle v_i h_j\rangle_{recon}$代表对应的相似虚构分数。

单层二值网络不足以模拟一组图片，因此训练完一层网络后，把这一层的激活值作为第二层学习中特征的输入值，不断逐层学习直至所有层都被训练完成为止。预训练完成后，模型将被展开成编码网络与解码网络。对于一个高维度的数据集来说，逐层预训练是逐步揭示低维非线性结构的有效方法。

预训练完成后，进入全局微调阶段，研究人员用确定性的概率以及对整个自编码网络的 BP 微调权重，来取代随机激活值，来达到进一步优化重建的目的。

该论文提出的自编码器整体运行框架包含逐层预训练与微调两部分，可以用下图来表示：



4.2 t-SNE 数据可视化工具

t-SNE 的主要用途是可视化和探索高维数据。它由 LaurensvanderMaatens GeoffreyHinton 开发。t-SNE 的主要目标是将多维数据集转换为低维数据集。相对于其他的降维算法，对于数据可视化而言 t-SNE 的效果最好。如果我们将 t-SNE 应用于 n 维数据，它将智能地将 n 维数据映射到 3 维甚至 2 维数据，并且原始数据的相对相似性非常好。与 PCA 一样，t-SNE 不是线性降维技术，它遵循非线性，这是它可以捕获高维数据的复杂流形结构的主要原因。

t-SNE 算法在 SNE 算法的基础上增加了两个改进：

1.把 SNE 修正为对称 t-SNE，提高了计算效率；

2.在低维空间中采用了 t 分布替换原来的高斯分布，解决了高维空间映射到低维空间所产生的拥挤问题，优化了 SNE 过于关注局部特征而忽略全局特征的问题。

在这篇论文中，**t-SNE 主要作为数据可视化工具来出现和使用，这里我采用了 Python 中的 sklearn 库，利用其中的 TSNE 来对 sklearn 库中 8*8=64 维的手写数字进行降维处理，使其显示在二维平面上，观察它的分类效果**。Python 代码的基本运行逻辑为：

1.下载 sklearn 库中的手写数字模型；

2.设定 t-SNE 算法的基本参数；

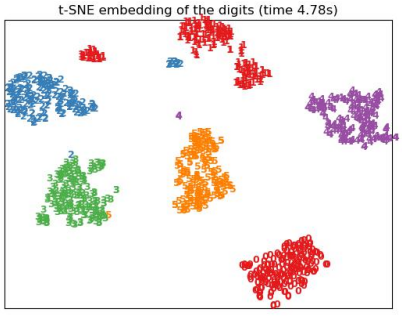
3.利用 t-SNE，将手写数字模型数据降维成二维数据；

4.画图显示降维得到的二维数据。

具体代码如下所示：

```python
from time import time
import numpy as np
import matplotlib.pyplot as plt

from sklearn import datasets
from sklearn.manifold import TSNE

def plot_tsne(data, label, title):
    x_min, x_max = np.min(data, 0), np.max(data, 0)
    data = (data - x_min) / (x_max - x_min)

    fig = plt.figure()
    ax = plt.subplot(111)
    for i in range(data.shape[0]):
        plt.text(data[i, 0], data[i, 1], str(label[i]),
                 color=plt.cm.Set1(label[i] / 10.),
                 fontdict={'weight': 'bold', 'size': 9})
    plt.xticks([])
    plt.yticks([])
    plt.title(title)
    return fig

def get_sklearn_data():
    digits = datasets.load_digits(n_class=6)  #载入手写数据集，8*8即64维的数据集，需要降维成2维
    data = digits.data
    label = digits.target
    n_samples, n_features = data.shape
    return data, label, n_samples, n_features

def main():
    data, label, n_samples, n_features = get_sklearn_data()
    print('Computing t-SNE embedding')
    tsne = TSNE(n_components=2, init='pca', random_state=0)  #维度为2，PCA初始化
    t0 = time()
    result = tsne.fit_transform(data)  #t-SNE对数据进行降维处理
    fig = plot_tsne(result, label,
                        't-SNE embedding of the digits (time %.2fs)'
                        % (time() - t0))
    plt.show(fig)


if __name__ == '__main__':
    main()
```

运行得到如下图像：



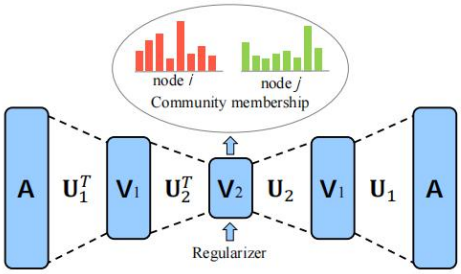
从代码实践中可以看出，t-SNE 可以将高维度的数据降维成低维度的数据来进行可视化，方便我们进行不同维度数据比较等任务，是一个优秀的数据可视化工具。

5. 论文的主要工作

5.1 用于社区检测的 DANMF 模型

5.1.1 概述

前面讨论了传统社区检测模型，NMF 社区检测模型和深度自动编码器，我研读的论文***Deep Autoencoder-like Nonnegative Matrix Factorization for Community Detection***，作者在这些模型的启发下，认为深度自编码器作为缩小原始数据底层抽象与顶层抽象之间差距的优秀方案，可以与 NMF 社区检测相结合，**一方面通过增加原始网络与社区成员空间之间的映射层数（类似于深度自动编码器），这样每个抽象层都可以捕获不同粒度级别的节点之间的相似性；另一方面，将编码器组件与解码器组件集成到统一的损失函数中，通过不断缩小编码前网络与解码后网络的差距，使 DANMF 继承了深度自编码器较强的学习能力**。整体模型如下图所示：



矩阵 A 代表原始网络，矩阵V_2代表最终得到的社区成员矩阵，矩阵U_i代表原始网络与社区成员矩阵之间的映射。**从左侧的原始网络 A 到中间的矩阵V_2，代表的是编码器组件的编码过程，而从矩阵V_2到右侧的网络 A，代表的是解码器组件将社区成员矩阵重构为原始网络的解码过程。**

5.1.2 数学模型建立

首先讨论在 NMF 中数学模型的建立。这里设 A 为原始网络邻接矩阵，该网络中含有 n 个节点，k 个社区，而 $n \times k$ 矩阵 U 的每一列代表对一个社区的描述，$k \times n$ 矩阵 V 的每一列代表一个节点对不同社区的关联贡献。因此，$U_{il}V_{lj}$代表第 l 个社区对边A_{ij}的影响，那么存在关系式$\widehat{A}_{ij} = \sum_{l=1}^{k} U_{il}V_{lj}$代表所有社区对于节点 i 与 j 的关系的影响总和。显然，为了模型应该尽可能与实际原始网络一致，故存在：

$$\min_{U,\mathbf{V}} \|\mathbf{A} - \mathbf{UV}\|_F^2, \text{ s.t. } \mathbf{U} \geq \mathbf{0}, \mathbf{V} \geq \mathbf{0}$$

矩阵 A 与矩阵 UV 之间的差矩阵的范数越小，那么说明非负矩阵 U 与 V 之积就越接近原始网络 A，模型就越准确。

在 NMF 的基础上，开始讨论深度 NMF 的数学模型。真实世界的原始网络是极为复杂的，靠 NMF 的单层映射无法概括复杂网络中隐藏的结构信息，这里将原始网络矩阵 A 分解为 p+1 个非负矩阵，如下所示：

$$\mathbf{A} \approx \mathbf{U}_1\mathbf{U}_2 \cdots \mathbf{U}_p\mathbf{V}_p$$

通过这种方法，每个抽象层都可以捕获不同粒度级别的节点之间的相似性，其中上一层与下一层的数学关系可以依次用下列式子表示：

$$
\begin{aligned}
\mathbf{V}_{p-1} &\approx \mathbf{U}_p\mathbf{V}_p \\
&\vdots \\
\mathbf{V}_2 &\approx \mathbf{U}_3\cdots\mathbf{U}_p\mathbf{V}_p \\
\mathbf{V}_1 &\approx \mathbf{U}_2\cdots\mathbf{U}_p\mathbf{V}_p
\end{aligned}
$$

通过 DNMF 可以得到一个更精确的社区成员矩阵V_p，上述的过程在自编码器中可以视为译码过程，那么我们表示译码组件中的损失函数如下：

$$
\begin{aligned}
&\min_{\mathbf{U}_i,\mathbf{V}_p} \mathcal{L}_{\mathrm{D}} = \|\mathbf{A}-\mathbf{U}_1\mathbf{U}_2\cdots\mathbf{U}_p\mathbf{V}_p\|_F^2 \\
&\text{s.t. } \mathbf{V}_p \geq \mathbf{0}, \mathbf{U}_i \geq \mathbf{0}, \forall i = 1,2,\cdots,p
\end{aligned}
$$

最后，在前面 NMF 和 DNMF 两个数学模型推导的基础上，我们还需要进一步考虑将编码器和译码器的损失函数统一为同一个损失函数。编码器的损失函数与译码器损失函数同理：

$$
\begin{aligned}
&\min_{\mathbf{U}_i,\mathbf{v}_p} \mathcal{L}_{\mathrm{E}} = \left\|\mathbf{V}_p - \mathbf{U}_p^T\cdots\mathbf{U}_2^T\mathbf{U}_1^T\mathbf{A}\right\|_F^2 \\
&\text{s.t. } \mathbf{V}_p \geq \mathbf{0}, \mathbf{U}_i \geq \mathbf{0}, \forall i = 1,2,\cdots,p
\end{aligned}
$$

此外，考虑到节点对的内在几何结构，还需要加入图正则化因子$\lambda\mathcal{L}_{reg} = \lambda tr(V_pLV_p^T)$，其中 λ 表示正则化参数，L 表示图拉普拉斯矩阵，由于是无向网络，这里的图拉普拉斯矩阵设为 $\mathrm{L}=\mathrm{D}-\mathrm{A}$，D 是一个对角矩阵。

综上，DANMF 的统一损失函数可以表示为：

$$
\begin{aligned}
\min_{\mathbf{U}_i,\mathbf{V}_p} \quad & \mathcal{L} = \mathcal{L}_{\mathrm{D}} + \mathcal{L}_{\mathrm{E}} + \lambda\mathcal{L}_{\mathrm{reg}} \\
& = \|\mathbf{A}-\mathbf{U}_1\mathbf{U}_2\cdots\mathbf{U}_p\mathbf{V}_p\|_F^2 + \left\|\mathbf{V}_p - \mathbf{U}_p^T\cdots\mathbf{U}_2^T\mathbf{U}_1^T\mathbf{A}\right\|_F^2 \\
& +\lambda\,\mathrm{tr}\left(\mathbf{V}_p\mathbf{L}\mathbf{V}_p^T\right), \\
\text{s.t. } & \mathbf{V}_p \geq \mathbf{0}, \mathbf{U}_i \geq \mathbf{0}, \forall i = 1,2,\cdots,p
\end{aligned}
$$

得到这个损失函数后，可以用于指导模型的训练，也就是对 DANMF 进行逐层预训练。逐层预训练由前面讲述的深度自编码器论文 *Reducing the Dimensionality of Data with Neural Networks* 提出，基本方法是从第一层开始，通过预训练网络权值将这一层损失函数降到最小，再用第一层的输出来作为第二层的输入，重复直至所有层都完成训练，使因子矩阵U_i和V_i有一个初始逼近，这个预训练过程可以大大缩短整个模型的训练时间。

5.2 模型的微调（优化）

所有层完成预训练后，从前面的深度自编码器论文 *Reducing the Dimensionality of Data with Neural Networks* 中我们知道，往往还需要进行一个全模型微调过程，这里也同样进行了全模型微调。

这里我们以U_i为对象，推导U_i在微调过程中的更新规则。

首先，保持U_i不变，将 DANMF 的损失函数改写为：

$$
\begin{aligned}
&\min_{\mathbf{U}_i} \mathcal{L}(\mathbf{U}_i) = \|\mathbf{A}-\mathbf{\Psi}_{i-1}\mathbf{U}_i\mathbf{\Phi}_{i+1}\mathbf{V}_p\|_F^2 \\
&+\left\|\mathbf{V}_p - \mathbf{\Phi}_{i+1}^T\mathbf{U}_i^T\mathbf{\Psi}_{i-1}^T\mathbf{A}\right\|_F^2 \\
&\text{s.t. } \mathbf{U}_i \geq \mathbf{0}
\end{aligned}
$$

其中$\Psi_{i-1} = U_1U_2\ldots U_{i-1}$，$\Phi_{i+1} = U_{i+1}\ldots U_{p-1}U_p$，为了求解该式，我们引入了一个拉格朗日乘子矩阵$\Theta_i$来对$U_i$施加非负约束，得到以下式子：

$$\min_{\mathbf{U}_i,\mathbf{\Theta}_i} \mathcal{L}(\mathbf{U}_i,\mathbf{\Theta}_i) = \|\mathbf{A} - \mathbf{\Psi}_{i-1}\mathbf{U}_i\mathbf{\Phi}_{i+1}\mathbf{V}_p\|_F^2$$
$$+ \left\|\mathbf{V}_p - \mathbf{\Phi}_{i+1}^T\mathbf{U}_i^T\mathbf{\Psi}_{i-1}^T\mathbf{A}\right\|_F^2 - \mathrm{tr}\left(\mathbf{\Theta}_i\mathbf{U}_i^T\right)$$

由拉格朗日极值定理可知，如果想求出该式子的极小值，那么$\mathcal{L}(U_i,\Theta_i)$对于$\Theta_i$的偏导数应该为 0。进一步通过 KKT 四大条件中的互补松弛条件：KKT 乘子（对偶变量）或不等式约束(g(x*)≤0)在极值处为零，我们可以推导出如下式子：

$$\mathbf{\Theta}_i \odot \mathbf{U}_i = \left(-4\mathbf{\Psi}_{i-1}^T\mathbf{A}\mathbf{V}_p^T\mathbf{\Phi}_{i+1}^T + 2\mathbf{\Pi}_i\right) \odot \mathbf{U}_i = \mathbf{0}$$

将该式加以变换，可以推导出关于U_i的更新规则数学式：

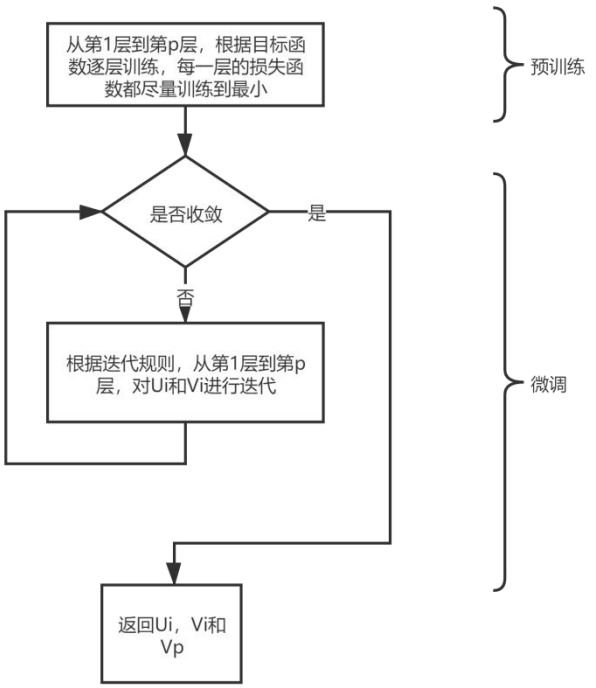
$$\mathbf{U}_i \leftarrow \mathbf{U}_i \odot \frac{2\mathbf{\Psi}_{i-1}^T\mathbf{A}\mathbf{V}_p^T\mathbf{\Phi}_{i+1}^T}{\mathbf{\Pi}_i}$$

同理，也可以推导出关于V_i的更新规则：

$$\mathbf{V}_i \leftarrow \mathbf{V}_i \odot \frac{2\mathbf{\Psi}_i^T\mathbf{A}}{\mathbf{\Psi}_i^T\mathbf{\Psi}_i\mathbf{V}_i + \mathbf{V}_i}$$

此外，论文中证明了更新规则的收敛性，并讨论了该优化算法的时间复杂度。在微调过程中，需要根据更新规则，循环对每一层的U_i和V_i进行更新优化，直至模型收敛为止。

因此，整个模型的建立过程包括逐层预训练和微调两部分，具体执行流程如下图所示：



5.3 模型测试与比较

5.3.1 不重叠社区检测

研究者采用了以下五种网络作为不重叠社区检测的数据集：

Email：一个涉及来自 42 个部门的 1005 名研究人员和 25571 个关系的通信网络。

Wiki：一个由来自 19 个类别和 17981 个边的 2405 个网页组成的文档网络。

Cora：一个具有 2708 个节点和 5429 个边的引文网络。每个节点都被分为 7 个类之一。

Citeseer：一个具有 3312 个节点和 4732 个边缘的引文网络。每个节点都被分为 6 个类之一。

Pubmed：一个具有 19717 个节点和 44338 条边的引文网络。每个节点被分为三个类中的一个。

针对这五种数据集，DANMF 设计了五种映射层配置，如下所示：

Dataset	n	Layers Configuration
Email	1005	1005-256-128-42
Wiki	2405	2405-256-128-19
Cora	2708	2708-256-64-7
Citeseer	3312	3312-256-64-6
Pubmed	19717	19717-512-64-3

从映射层配置中我们可以看出，DANMF 都是按照“原始网络节点-中间层-社区成员”来配置层次的，例如 Email 数据集中，存在 1005 个研究人员（网络节点），这些研究人员之间存在 25571 条通信关系；同时，这些研究人员从属于 42 个部门（社区），而每一个人都只从属于一个部门，不存在一个研究人员从属于多个部门或者不从属于任何部门的情况，也就是一个节点只对应一个社区，这是一个不重叠社区网络。

因此，在 Email 数据集中，DANMF 配置维度为 1005 的输入层，通过“-256-128-”的中间映射层，类深度自编码器将这 1005 个维度降成 42 个维度，也就是将 1005 名研究人员分配到 42 个对应的部门里去，而进行分配的精度，取决于模型训练和微调的结果。模型越好，则研究人员更有可能被分配到正确的部门中去。

首先是对 NMF、DNMF、DANMF 等算法之间误差率的比较，这里选用的是 Wiki 数据集，导入各个模型后，得到的错误率如下表：

Method	Encoder	Decoder	Encoder+Decoder
NMF	0.2326	0.0543	0.2869
ONMF	0.0043	0.0547	0.0590
PNMF	0.0000	445.58	445.58
NSED	0.0025	0.0547	0.0572
DNMF	0.2131	0.0546	0.2677
DANMF	0.0020	0.0541	0.0561

很明显，DANMF 的表现最优，**值得一提的是，DANMF 的误差率远远低于 DNMF，最重要的原因在于 DANMF 集成了编码器组件和解码器组件，这是论文的一大创新点，因为这两个组件能够在学习过程中相互引导，编码前的原始网络与解码重构后的网络进行对比，二者相差越小，模型训练越精确，DANMF 继承了深度自编码器的优点，可以明显提高模型的准确度。**

同时，DANMF 还就 ARI、NMI、ACC 三个指数与其它模型进行了比较，指数越大，说明模型越好。测试结果如下：

Table 3: Performance evaluation based on ARI

Method	Email	Wiki	Cora	Citeseer	Pubmed
NMF	0.4989	0.1195	0.2145	0.0590	0.0978
ONMF	0.4832	0.1233	0.1964	0.0825	0.1589
PNMF	0.4641	0.1151	0.1863	0.0801	0.0967
BNMF	0.3545	0.1705	0.1812	0.0838	0.0872
BigClam	0.2478	0.0217	0.0306	0.0283	0.0258
HNMF	0.2079	0.1448	0.1113	0.0262	0.0360
NSED	0.5215	0.1253	0.1782	0.0866	0.1258
LINE	0.3325	0.1344	0.1271	0.0278	0.1017
Node2Vec	0.4195	0.1621	0.1063	0.0182	0.0170
MNMF	0.0041	0.0016	0.0002	0.0007	0.0001
DNMF	0.5256	0.1341	0.2452	0.0990	0.1185
DANMF	0.5521	0.1628	0.3194	0.1343	0.2563

Table 4: Performance evaluation based on NMI

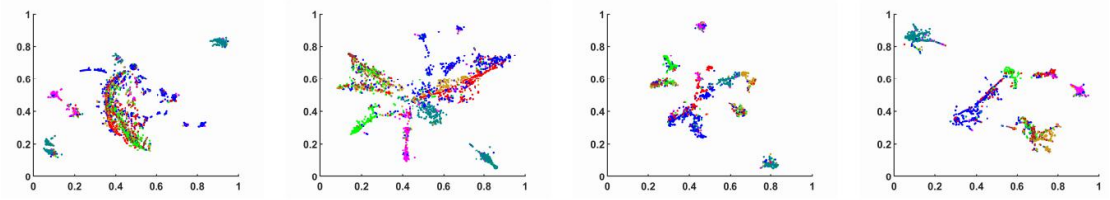
Method	Email	Wiki	Cora	Citeseer	Pubmed
NMF	0.6751	0.2673	0.2851	0.1319	0.1606
ONMF	0.6734	0.2607	0.2416	0.1423	0.1582
PNMF	0.6770	0.2684	0.2893	0.1355	0.1511
BNMF	0.5960	0.2903	0.2521	0.0835	0.0714
BigClam	0.5796	0.2722	0.1864	0.0735	0.0291
HNMF	0.5146	0.2959	0.1425	0.0312	0.0311
NSED	0.6845	0.2659	0.2928	0.1492	0.1729
LINE	0.6393	0.2772	0.2376	0.0573	0.1357
Node2Vec	0.6784	0.3331	0.1978	0.0486	0.0635
MNMF	0.2138	0.0274	0.0035	0.0031	0.0002
DNMF	0.6850	0.2798	0.3572	0.1582	0.1709
DANMF	0.6943	0.3406	0.4114	0.1831	0.2221

Table 5: Performance evaluation based on ACC

Method	Email	Wiki	Cora	Citeseer	Pubmed
NMF	0.5851	0.3027	0.4103	0.3074	0.5133
ONMF	0.5761	0.3069	0.3811	0.3330	0.5575
PNMF	0.5791	0.3052	0.4029	0.3451	0.5073
BNMF	0.4299	0.3751	0.4191	0.3324	0.5110
BigClam	0.4768	0.2545	0.3781	0.3046	0.3978
HNMF	0.3463	0.3518	0.3903	0.2569	0.4128
NSED	0.6179	0.2981	0.4234	0.3448	0.5201
LINE	0.4657	0.3289	0.4044	0.3019	0.4990
Node2Vec	0.5244	0.3568	0.3674	0.2521	0.4067
MNMF	0.1075	0.0886	0.1647	0.1890	0.3397
DNMF	0.6199	0.3543	0.4849	0.3635	0.5389
DANMF	0.6358	0.4112	0.5499	0.4242	0.6393

结果是令人振奋的，DANMF 模型在绝大部分数据集中的指标为最优，这有力地说明了 DANMF 在不重叠的社区检测中的优越性。

同时，研究人员借助 t-SNE 数据可视化工具，将 DANMF 在 Cora 数据集训练过程中，每一级的社区成员矩阵V_i输入 t-SNE 中，一律降成二维数据后显示在平面中，可以清楚地看到它们的离散情况：



(a) Cora - before training　(b) Cora - layer 1　(c) Cora - layer 2　(d) Cora - layer 3

从 t-SNE 降维得到的数据中，我们可以明显看到，每一级检测到的社区，其分类越来越明显和准确，可分辨性越来越强，说明模型得到的社区成员矩阵越精确，社区检测效果越好。

5.3.2 重叠社区检测

由于真实可信的重叠社区网络数据集很少公开可用，这里研究人员采用 LRF 工具包来生成具有重叠社区结构的合成网络数据集。将数据集运用到模型中进行社区检测后，采用 ONMI 度量来衡量检测结果，指数越大，检测效果越好。关于社区中混合参数分别为 0.1 和 0.3 的结果如下图所示：

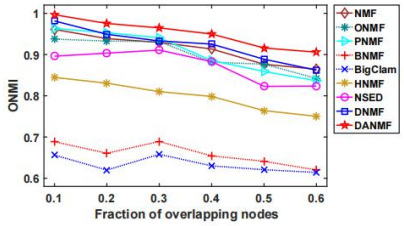


Figure 7: ONMI on LFR benchmarks with $\mu = 0.1$.

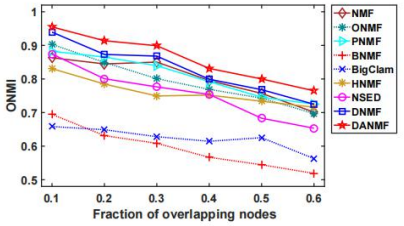


Figure 8: ONMI on LFR benchmarks with $\mu = 0.3$.

我们可以从图中总结出三个点：

(1) DANMF 在重叠社区检测中优于其它社区检测模型；

(2) 随着重叠节点比例的上升，所有模型社区检测的精度都在下降；

(3) 混合参数为 0.3 的情况下，社区检测的精度明显低于混合参数为 0.1 的情况，因此社区重叠越严重，对社区检测的负面影响也越大。

6. 总结

论文作者在基于传统社区检测方法和 NMF 社区检测方法的基础上，从深度自编码器中得到启发，将深度自编码器与 NMF 结合在一起，提出了 DANMF 模型。**DANMF 模型最大的创新点和优点在于，一方面它通过增加原始网络与社区成员空间之间的映射层数（类似于深度自动编码器），这样每个抽象层都可以捕获不同粒度级别的节点之间的相似性；另一方面，将编码器组件与解码器组件集成到统一的损失函数中，通过不断缩小编码与解码的误差，使 DANMF 继承了深度自编码器较强的学习能力。**同时，由于 DANMF 的训练难度较大，研究人员提出了一个优化算法，并采用**“逐层预训练+优化算法微调”**的训练模式，有效减小了模型的训练量。通过实验证明，DANMF 模型在社区检测精度上优于当前其它的社区检测方法，是一个很有意义的社区检测模型。

7. 模型改进建议——基于鲁棒的深度自编码器的异常检测

在论文中，对于 NML 社区检测方法，作者认为它虽然可以用于非重叠与重叠社区检测，但真实世界中的网络是非常复杂的，NML 仅用一层映射不能挖掘出足够的隐藏信息，这是 DANMF 模型为什么要引入深度自编码器的一个原因。读完论文后，我认为相对 NMF 模型，DANMF 显然更适用于真实世界，**但是我冒昧地认为，真实世界的网络除了非常复杂外，还存在另一个问题：我们所获得的的真实世界网络数据集未必是完全“干净”的，也就是说这些数据集会受到噪声和干扰的影响，这些来自各个方面的噪声干扰会“污染”数据集，我们根据这些被“污染”的数据集来训练得到的 DANMF 模型，想必会引起误差。**

因此，**我认为论文模型应该从基于鲁棒的深度自编码器的异常检测方面进行改进。**

我的这一想法来源于论文 ***Anomaly Detection with Robust Deep Autoencoders*** 的启发，我认为将深度自编码器改进为鲁棒深度自编码器后，可以过滤掉原始数据集中的噪声干扰，得到的模型称为 RDANMF，可能在真实世界网络会具有更好的社区检测性能。以下是关于该模型的数学推导：

假设在一个深度自编码器网络中，X 为原始输入数据，X 可以被分解为两个矩阵L_D和 S 之和，其中L_D是一个低秩矩阵，S 是一个稀疏矩阵，L_D代表没有受到噪声污染的“干净”的原始网络数据集（因为“干净”的原始网络内部有一定的结构信息，造成各行或列间是线性相关的），而 S 代表原始网络数据集中的噪声干扰（因为噪声矩阵是稀疏的），而如何从原始数据集中提取出L_D矩阵，是鲁棒深度自编码器的关键问题。这里设函数$E_\theta()$代表编码过程，函数$D_\theta()$代表解码过程，而衡量一个编码器网络精确度，需要比较编码前和解码后的网络差距，同时引入噪声干扰矩阵 S，作为惩罚值，因此得到一个数学模型如下：

$$\min_\theta \|L_D - D_\theta(E_\theta(L_D))\|_2 + \lambda\|S\|_0$$

$$\text{s.t. } X - L_D - S = 0$$

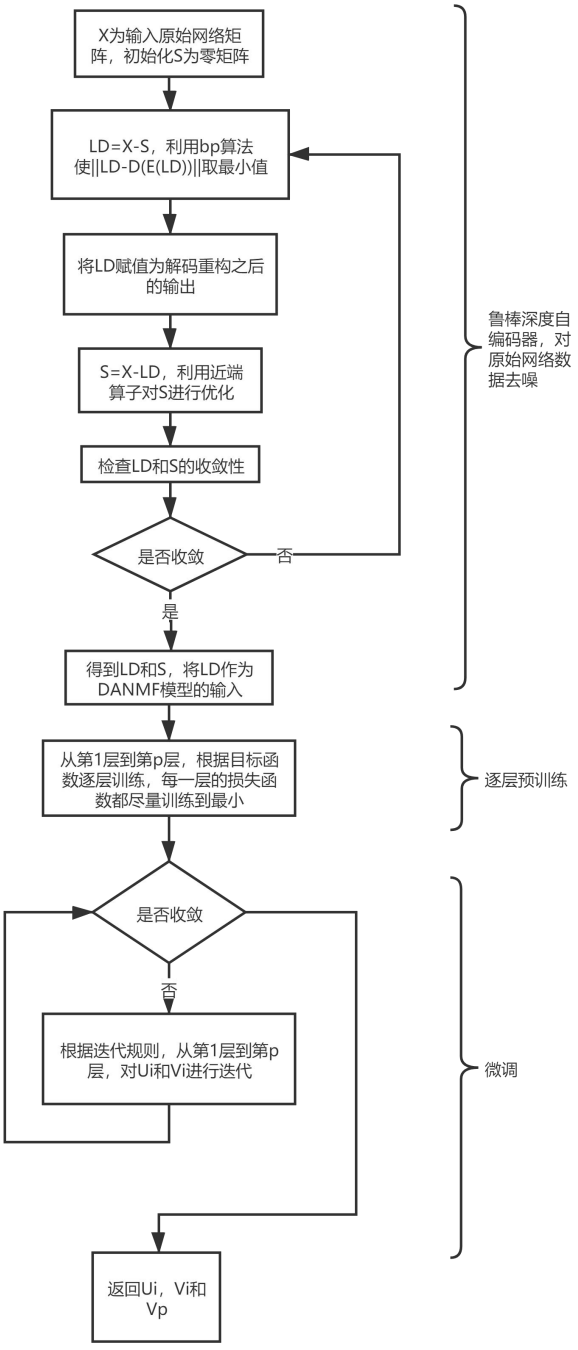
由于 rank 和$\mathcal{L}_0$范数在优化上存在非凸和非光滑特性，所以我们一般将它转换成求解以下一个松弛的凸优化问题：

$$\min_\theta \|L_D - D_\theta(E_\theta(L_D))\|_2 + \lambda\|S\|_1$$

$$\text{s.t. } X - L_D - S = 0$$

这里我们将输入数据 X 分为两部分，L_D和 S，这可以用类似 ADMM 的方法，通过交替优化来实现损失函数最小化。其中L_D是对自动编码器$D_\theta(E_\theta(L_D))$的输入，我们通过令 S 为固定值，采用反向传播最小化重建错误$||L_D - D_\theta(E_\theta(L_D))||_2$来训练这个自动编码器；另一方面，S 包含噪声和异常值，在式子中作为惩罚项，我们令L_D为固定值，利用近端梯度算子来使它最小化。在循环优化过程中L_D和 S 会互相投影。

基于以上的数学模型，再结合论文中的 DANMF 模型，我认为可以在 DANMF 进行正式训练前，先进行上述的鲁棒自编码器训练，从原始网络数据集中提取得到较为“干净”的数据集L_D，以L_D作为 DANMF 的输入数据。具体流程如下所示：


X为输入原始网络矩阵，初始化S为零矩阵
LD=X-S，利用bp算法使||LD-D(E(LD))||取最小值
将LD赋值为解码重构之后的输出
S=X-LD，利用近端算子对S进行优化
检查LD和S的收敛性
是否收敛
否
是
得到LD和S，将LD作为DANMF模型的输入
从第1层到第p层，根据目标函数逐层训练，每一层的损失函数都尽量训练到最小
是否收敛
否
根据迭代规则，从第1层到第p层，对Ui和Vi进行迭代
返回Ui，Vi和Vp
鲁棒深度自编码器，对原始网络数据去噪
逐层预训练
微调