图与机器学习

理论、模型与方法

黄涛
中山大学数学学院

My Work-Shop

在谈到如果要给后进的学弟学妹一个学习的方法的话，刘路回答到：“我只有一个想法，就是看淡分数，重在兴趣。”

序

图是重要的。

笔者
于康乐园

目　　录

第一部分

理论

第 1 章 绪论

1.1 图

图，一种常见的结构，可以用来表示一些现实中的数据或是表示数据之间的关系，图通常表示为由节点和边构成的二元组：$\mathcal{G} = (\mathcal{V}, \mathcal{E})$，其中 $\mathcal{V}$ 是节点的集合，而 $\mathcal{E}$ 代表链接节点集合中某两元素的边，也就是说：$\mathcal{E} \subseteq \mathcal{P}(V \times V)$，其中 $\mathcal{P}(\cdot)$ 代表幂集。这样的表示当然是方便理解的，但是当我们要用计算机处理和分析图时，这样的表示就过于抽象了，我们要注意到，在我们对包含 n 个元素的集合 $\mathcal{V}$ 中的元素赋予 1 到 n 的编号的时候 ($\mathcal{V} = \{a_1, \cdots, a_n\}$)，图是可以由一个邻接矩阵 $A \in \{0,1\}^{n \times n}$ 表示的：

$$A_{i,j} = \begin{cases} 1, & \text{if } a_i \sim a_j \\ 0, & \text{otherwise} \end{cases} \tag{1.1.1}$$

，其中 $a_i \sim a_j$ 代表 $(a_i, a_j) \in \mathcal{E}$，即图中存在点 a_i 与 a_j 的连接。

更一般的，当我们让每一个边 $e \in \mathcal{E}$ 可以拥有一个权重，我们可以给出一个更广义的图的定义：

定义 1.1. 图

由包含 n 个元素的节点的集合 $\mathcal{V}$ 和它的邻接矩阵 $A \in \mathbb{R}^{+n \times n}$ 所构成的二元组 $\mathcal{G} = (\mathcal{V}, A)$。♣

在这里，我们约束它的边的权重是非负的，因为在许多的场景中，负的权重没有实际意义。

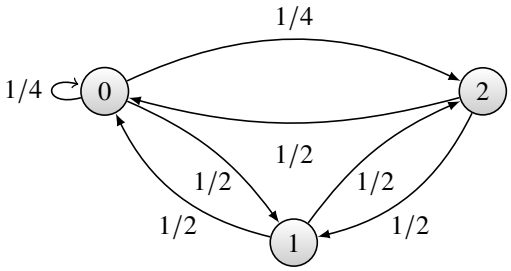

图 1.1: 一种图：状态转移图

一个经典的图的例子是图 1.1中所示的状态转移图，每一个节点代表一种状态，而节点间的边（有向）e_{ij} 表示从状态 a_i 到状态 a_j 的转移概率，其邻接矩阵可以表示为：

$$A = \begin{bmatrix} 1/4 & 1/2 & 1/4 \\ 1/2 & 0 & 1/2 \\ 1/2 & 1/2 & 0 \end{bmatrix}$$

这样的一个矩阵的表示上的意义大致如此，那么与矩阵密切相关的它的乘法上的意义呢？首先，这样的矩阵左乘一组 n 个向量组成的行向量组就是将一次图变换作用于这 n 个向量所表示的节点，例如将邻接矩阵乘以邻接矩阵就可以得到该图的长度为 2 的可达矩阵，因为邻接矩阵本身就是长度为 1 的可达矩阵，其 n 个行向量代表每个节点与图中所有节点的连接关系，即其与图中所有节点长度为 1 的所有节点的数量，作用一次图变换之后就可以得到这一节点与图中所有节点长度为 2 的边的数量。

1.2 拉普拉斯算子

为了分析图，一种保持图的性质的离散微分算子是重要的：

定义 1.2. 离散微分算子 (图)

$$\frac{\partial f}{\partial e_{ij}} := \sqrt{A_{i,j}}\,[f(j) - f(i)] \tag{1.2.1}$$

，而图上节点 i 的梯度的定义为向量：

定义 1.3. 梯度 (图)

$$\nabla_i f := \left[\left\{\frac{\partial f}{\partial e_{ij}}\right\}_{e_{ij}\in\mathcal{E}}\right] \tag{1.2.2}$$

进一步，我们定义：

定义 1.4. 局部变分 (图)

$$\begin{aligned}\|\nabla_i f\|_2 &:= \left[\sum_{e_{ij}\in\mathcal{E}}\left(\frac{\partial f}{\partial e_{ij}}\right)^2\right]^{\frac{1}{2}} \\ &= \left[\sum_{j\in\mathcal{N}_i} A_{i,j}\,[f(j) - f(i)]^2\right]^{\frac{1}{2}}\end{aligned} \tag{1.2.3}$$

，其中 $\mathcal{N}_i$ 代表节点 $v_i \in \mathcal{V}$ 的邻居集合。以及：

定义 1.5. 离散 p-迪利克雷形式

$$\begin{aligned}S_p(f) &:= \frac{1}{p}\sum_{i\in\mathcal{V}}\|\nabla_i f\|_2^p \\ &= \frac{1}{p}\sum_{i\in\mathcal{V}}\left[\sum_{j\in\mathcal{N}_i} A_{i,j}\,[f(j) - f(i)]^2\right]^{\frac{p}{2}}\end{aligned} \tag{1.2.4}$$

值得注意的是，$p = 2$ 时：

$$\begin{aligned}S_2(f) &= \frac{1}{2}\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i} A_{i,j}\,[f(j) - f(i)]^2 \\ &= \sum_{e_{ij}\in\mathcal{E}} A_{i,j}\,[f(j) - f(i)]^2 \\ &= f^T\mathcal{L}f\end{aligned} \tag{1.2.5}$$

1.3 谱图理论

我们有了图之后，我们更愿意去了解如何分析一个图，我们会很自然地想到用经典的线性代数上的运算去处理这种能够用矩阵表示的问题。矩阵问题的一个经典的研究方式就是研究该矩阵的特征值与特征代数，如矩阵的谱一般，图的谱被定义为：

定义 1.6. 谱（图）

图的邻接矩阵的特征值及其重数：

$$\begin{pmatrix} \lambda_1 & \cdots & \lambda_k \\ r_1 & \cdots & r_k \end{pmatrix}$$

对于无向图来说，其邻接矩阵是实对称阵，因而有：

$$\sum_{i=1}^{k} r_i = n \tag{1.3.1}$$

♣

第 2 章 图的构建

要研究图上的数据或是将原本的数据转变到图上进行研究，首要工作是从数据构建图 [1]。图由两个部分构成，节点和边，节点即为各个数据，或者称作特征，而节点之间是否连接以及连接的权重，则由数据之间的关联程度，亦即相似度决定。

数据之间的相似度通常有多种计算方法，以 n 维向量所表示的特征为例，有 L_2 距离和 *Cosine* 距离可以用于计算两特征之间的相似度。

在离散的图上，即 $A \in \{0,1\}^{n\times n}$ 的图上，通常使用 *Cosine* 相似度：

定义 2.1. *Cosine* **相似度**

即两特征之间的夹角的余弦值：

$$s_{i,j}^{Cosine} = \frac{|x_i \cdot x_j|}{|x_i| \cdot |x_j|} \tag{2.0.1}$$

给出一个相似度的阈值 θ，如果两个节点之间的相似度大于这一阈值，则连接这两节点，否则不连接：

$$A_{i,j} = \begin{cases} 1, & \text{if } s_{i,j} > \theta \\ 0, & \text{otherwise} \end{cases} \tag{2.0.2}$$

而在连续的图上，则常使用 L_2 相似度;

定义 2.2. L_2 **相似度**

即两特征之间的 L_2 距离之相反数：

$$s_{i,j}^{L_2} = \exp\left(\frac{\|x_i - x_j\|^2}{\mu^2}\right) \tag{2.0.3}$$

其中 μ 是与数据分布有关的系数

如若两特征间的距离小于某一阈值，则连接两节点，否则不连接：

$$A_{i,j} = \begin{cases} s_{i,j}, & \text{if } \|x_i - x_j\|^2 < \theta \\ 0, & \text{otherwise} \end{cases} \tag{2.0.4}$$

受限于计算资源，当数据特别多的时候，我们虽然能够用稀疏矩阵去存储这些数据所构成的图的邻接矩阵，但是直接去计算或处理这一图还是相当困难的，这个时候就需要对图进行分割。

第 3 章 图上的变换

3.1 图上的傅里叶变换

要定义图上的卷积变换，首先我们要给出图上的傅里叶变换，即对于图的某一特征值 λ_ℓ，以及图的特征向量 χ_ℓ 和信号 $f \in \mathbb{R}^n$，傅里叶变换 $\hat{f}(\lambda_\ell)$ 定义为：

定义 3.1. 傅里叶变换（图）

$$\hat{f}(\lambda_\ell) = \langle \chi_\ell, f \rangle = \sum_{i=0}^{n-1} \chi_\ell(i) f(i) \tag{3.1.1}$$

♣

对应地，图上的逆傅里叶变换定义为：

定义 3.2. 逆傅里叶变换（图）

$$f(i) = \sum_{\ell=0}^{n-1} \hat{f}(\lambda_\ell) \chi_\ell(i) \tag{3.1.2}$$

♣

第二部分

模型

第 4 章 图卷积神经网络

4.1 GCN

[2] 中提出的 Graph Convolutional Network(GCN) 模型核心在于将图上的卷积近似为：

$$g_\theta \star x \approx \theta'_0 x + \theta_1 (L - I_n) x = \theta'_0 x + \theta_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x \tag{4.1.1}$$

并假设

$$\theta = \theta'_0 = -\theta'_1 \tag{4.1.2}$$

得到

$$g_\theta \star x \approx \theta \left(I_n - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) x \tag{4.1.3}$$

并且在实际计算时使用如下的归一化技巧：

$$I_n - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \longrightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \tag{4.1.4}$$

其中：

$$\tilde{A} = I_n + A \tag{4.1.5}$$

令：

$$\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \tag{4.1.6}$$

在图中的数据输入下一层全连接层之前，我们可以先对其做图上的卷积变换，这样以来每一层的计算公式便为：

$$H^{(k+1)} = f(H^{(k)}, A) = \sigma \left(\hat{A} H^{(k)} W^{(k)} \right) \tag{4.1.7}$$

但是值得注意的一点是，不同的工作对于 $\hat{A}$ 的选取是不同的，因为作者可以选择是否归一化，甚至可以在式4.1.2中假设不同的 θ 的取值，例如假设 $\theta'_0 = \theta'_1$，从而得到：

$$\hat{A}' = I_n + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \tag{4.1.8}$$

4.2 ANGPN

ANGPN（Adaptive Neighborhood Graph Propagation Network）[3] 是对图卷积神经网络的一种改进，通过在图的传播（Propagation）过程中使用在理论上更为优秀的计算方式，提升了图卷积神经网络的性能。GCN 计算公式为：

$$H^{(k+1)} = \sigma\big((I_n + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) H^{(k)} W^{(k)} \big) \tag{4.2.1}$$

可以将计算流程分解为：

$$F^{(k)} = (I_n + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})H^{(k)}H^{(k+1)} = \sigma\left(F^{(k)}W^{(k)}\right) \tag{4.2.2}$$

令 $\hat{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ ，可以将第一步的计算操作展开为

$$f_i = \sum_{j=1}^{n} \hat{A}_{ij}h_j + h_i \tag{4.2.3}$$

这里我们会发现，这里我们难以控制是自身的 feature 重要还是邻居的 feature 重要，于是作者：

$$f_i^{(t+1)} = \alpha \sum_{j=1}^{n} A_{ij}f_j^{(t)} + (1-\alpha)h_i \tag{4.2.4}$$

其中 $f_j^{(0)} = h_j$
矩阵形式可以写为;

$$F^{(t+1)} = \alpha AF^{(t)} + (1-\alpha)H \tag{4.2.5}$$

迭代无穷次后，可得到稳定值为：

$$F^* = (1-\alpha)(I_n - \alpha A)^{-1}H \tag{4.2.6}$$

要使得 F 收敛，就是等价于解决如下优化问题：

$$\min_F \ \mathrm{Tr}(F^T(I_n - A)F) + \mu\|F - H\|_F^2 \tag{4.2.7}$$

其中 $\alpha = \frac{1}{1+\mu}$ ，

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} = \sqrt{\mathrm{trace}(A^*A)} = \sqrt{\sum_{i=1}^{\min\{m,n\}}\sigma_i^2} \tag{4.2.8}$$

，这里 A^* 表示 A 的共轭转置，σ_i 是 A 的奇异值

一般来说，要计算图中节点间的相似度构成的邻接矩阵 A 使手工的而且是难以设计的，令 $D_{ij}^x = \|x_i - x_j\|_2$ ，再引入代表数据 i 和 j 之间的相似程度的参数 S_{ij} ，可以将上述优化拓展为：

$$\begin{aligned} min_{S,F} \ & \sum_{i,j=1}^{n} D_{ij}^x S_{ij} + \gamma\|S\|_F^2 + \\ & \beta\mathrm{Tr}(F^T(I_n - S)F) + \mu\|F - H\|_F^2 \\ & s.t. \ \sum_{j=1}^{n} S_{ij} = 1, S_{ij} \geq 0 \end{aligned} \tag{4.2.9}$$

要解决这一优化问题，可以迭代地求出一个近似解：第一步，先对于给定的 F ，求出最优的 S

$$\begin{aligned} \min_S & \sum_{i,j=1}^{n} D_{ij}^x S_{ij} + \gamma\|S\|_F^2 + \beta\mathrm{Tr}(F^T(I_n - S)F) \\ & s.t. \ \sum_{j=1}^{n} S_{ij} = 1, S_{ij} \geq 0 \end{aligned} \tag{4.2.10}$$

等价于

$$\begin{aligned} &\min_{S} \sum_{i,j=1}^{n} (D^x - \beta FF^T)_{ij} S_{ij} + \gamma \|S\|_F^2 \\ &s.t. \ \sum_{j=1}^{n} S_{ij} = 1, S_{ij} \geq 0 \end{aligned} \tag{4.2.11}$$

这一问题有一个闭合（解析）解：

$$S_{ij} = \max\left\{ -\frac{1}{2\gamma}(D^x - \beta FF^T)_{ij} + \eta, 0 \right\} \tag{4.2.12}$$

其中 $\eta = \frac{1}{k} + \frac{1}{2k\gamma} \sum_{j=1}^{k} D_{ij}^x$

其后再根据公式更新 F

Algorithm 1 ANGPN Propagation Layer

1: **输入:** 特征矩阵 $H^{(k)} \in \mathbb{R}^{n \times d_k}$, 距离矩阵 $D \in \mathbb{R}^{n \times n}$, 权重 $W^{(k)}$, 参数 γ, β 和 α, 最大迭代次数 T
2: **输出:** 特征矩阵 $H^{(k+1)}$
3: 令 $F = H^{(k)}$
4: 计算 η $\eta = \frac{1}{k} + \frac{1}{2k\gamma} \sum_{j=1}^{k} D_{ij}^x$
5: **for** $t = 1, 2 \cdots T$ **do**
6: 计算 S
$S_{ij} = \max\left\{ -\frac{1}{2\gamma}(D^x - \beta FF^T)_{ij} + \eta, 0 \right\}$
7: 计算 F
$F = (\alpha S + (1 - \alpha) I_n) H^{(k)}$
8: **end for**
9: 返回 $H^{(k+1)} = \sigma\left(FW^{(k)}\right)$

4.3 Maximum a Posteriori

计算图上的邻接矩阵，特别是复杂的数据之间的邻接矩阵是一个值得深入探究的问题。有别于 [3]，[4] 提出了一种不依赖欧式距离的邻接矩阵优化方法：

$$\vec{\hat{A}}_{\text{MAP}}(\mathbf{x}) = \arg\max_{\hat{\mathbf{A}}} \ f(\vec{x} \mid \hat{\mathbf{A}}) g(\hat{\mathbf{A}}) \tag{4.3.1}$$

其中 $f(\vec{x} \mid \hat{\mathbf{A}})$ 是相似度函数，而 $g(\hat{\mathbf{A}})$ 是 $\hat{\mathbf{A}}$ 的先验概率分布.

4.3.1 相似度函数

首先作者假设图 $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ 上的数据 $\vec{x} = (x_1, x_2, ..., x_n)^\top \in \mathbb{R}^n$ 都是 GMRF 上的，即满足以下概率密度函数：

$$\pi(\vec{x}) = (2\pi)^{-\frac{n}{2}} |\mathbf{Q}|^{\frac{1}{2}} \exp\left(-\frac{1}{2} (\vec{x} - \vec{\mu})^\top \mathbf{Q} (\vec{x} - \vec{\mu}) \right) \tag{4.3.2}$$

，并且 $Q_{i,j} \neq 0 \iff \{i, j\} \in \mathcal{E}, \forall i \neq j$ 其中 μ 是均值，$\mathbf{Q} > 0$ 是精度矩阵 (precision matrix) 由于邻接矩阵代表了图上数据的关联性，与精度矩阵有关，作者假设：

$$\mathbf{Q} = \delta \mathcal{L} = \delta(\mathbf{I}_n - \hat{\mathbf{A}}) \tag{4.3.3}$$

因而相似度函数可以表示为：

$$f(\vec{x} \mid \hat{\mathbf{A}}) = \beta \exp\left(-\lambda_0 \vec{x}^\top (\mathbf{I}_n - \hat{\mathbf{A}}) \vec{x} \right) \tag{4.3.4}$$

其中 $\beta = (2\pi)^{-\frac{n}{2}} |\mathbf{Q}|^{\frac{1}{2}}$ 并且 $\lambda_0 = \frac{\delta}{2}$ 值得我们注意的是：

$$\vec{x}^\top \mathcal{L} \vec{x} = \sum_{i \sim j} a_{i,j} (x_i - x_j)^2 \tag{4.3.5}$$

其中 $a_{i,j}$ 代表 x_i 和 x_j 间边的权重，而 $i \sim j$ 表示点 i 和点 j 相连

4.3.2 先验概率分布

作者认为先验概率分布由稀疏性约束 (sparsity constraint) $g_s(\hat{\mathbf{A}})$ 和性质约束 (property constraint) $g_p(\hat{\mathbf{A}})$ 两部分组成：

$$g(\hat{\mathbf{A}}) = g_s(\hat{\mathbf{A}})g_p(\hat{\mathbf{A}}) \tag{4.3.6}$$

其中稀疏性约束是要求 $\hat{\mathbf{A}}$ 变得更加稀疏：

$$g_s(\hat{\mathbf{A}}) = \exp\left(-\lambda_1\|\hat{\mathbf{A}}\|_1\right) \tag{4.3.7}$$

其中 $\|\cdot\|_1$ 是矩阵的 l_1-norm ，而 λ_1 是一个权重参数而作为图的邻接矩阵，$\hat{\mathbf{A}}$ 具有如下性质：

$$\hat{\mathbf{A}}^\top = \hat{\mathbf{A}} \tag{4.3.8}$$

$$\hat{\mathbf{A}}\mathbf{1} = \mathbf{1} \tag{4.3.9}$$

$$\mathrm{tr}(\hat{\mathbf{A}}) = 0 \tag{4.3.10}$$

因此性质约束为：

$$g_p(\hat{\mathbf{A}}) = \exp\left(-\lambda_2\|\hat{\mathbf{A}}^\top - \hat{\mathbf{A}}\|_F^2 - \lambda_3\|\hat{\mathbf{A}}\mathbf{1} - \mathbf{1}\|_F^2 \ -\lambda_4|\mathrm{tr}(\hat{\mathbf{A}})|^2\right) \tag{4.3.11}$$

4.3.3 优化

作者提出的优化目标便可以写为：

$$\begin{aligned}\max_{\hat{\mathbf{A}}} &\exp\left(-\lambda_0\vec{x}^\top(\mathbf{I}_n - \hat{\mathbf{A}})\vec{x}\right)\cdot\exp\left(-\lambda_1\|\hat{\mathbf{A}}\|_1\right)\cdot \\ &\exp\left(-\lambda_2\|\hat{\mathbf{A}}^\top - \hat{\mathbf{A}}\|_F^2 - \lambda_3\|\hat{\mathbf{A}}\mathbf{1} - \mathbf{1}\|_F^2 - \lambda_4|\mathrm{tr}(\hat{\mathbf{A}})|^2\right)\end{aligned} \tag{4.3.12}$$

等价于：

$$\begin{aligned}\min_{\hat{\mathbf{A}}} &\lambda_0\vec{x}^\top(\mathbf{I}_n - \hat{\mathbf{A}})\vec{x} + \lambda_1\|\hat{\mathbf{A}}\|_1 + \\ &\lambda_2\|\hat{\mathbf{A}}^\top - \hat{\mathbf{A}}\|_F^2 + \lambda_3\|\hat{\mathbf{A}}\mathbf{1} - \mathbf{1}\|_F^2 + \lambda_4|\mathrm{tr}(\hat{\mathbf{A}})|^2\end{aligned} \tag{4.3.13}$$

最终的 loss 也由三部分组成:

$$\mathcal{L}_{\mathrm{GL}} = \mathcal{L}_{\mathrm{smooth}} + \mathcal{L}_{\mathrm{sparsity}} + \mathcal{L}_{\mathrm{properties}} \tag{4.3.14}$$

其中：

$$\mathcal{L}_{\mathrm{smooth}} = \lambda_0\|\mathbf{x}^\top(\mathbf{I}_n - \hat{\mathbf{A}}_{\mathrm{out}})\mathbf{x}\|_2^2 \tag{4.3.15}$$

$$\mathcal{L}_{\mathrm{sparsity}} = \lambda_1\|\hat{\mathbf{A}}_{\mathrm{out}}\|_1 \tag{4.3.16}$$

$$\mathcal{L}_{\mathrm{properties}} = \lambda_2\|\mathbf{A}_{\mathrm{out}}^\top - \mathbf{A}_{\mathrm{out}}\|_2^2 + \lambda_3\|\hat{\mathbf{A}}_{\mathrm{out}}\mathbf{1} - \mathbf{1}\|_2^2 + \lambda_4|\mathrm{tr}(\hat{\mathbf{A}}_{\mathrm{out}})|^2 \tag{4.3.17}$$

值得注意的是，为了使得输出的邻接矩阵尽可能地稀疏，作者将 $\mathcal{L}_{\mathrm{sparsity}}$ 的梯度下降规则设置为：

$$\frac{\partial\mathcal{L}_{\mathrm{sparsity}}}{\partial\hat{\mathbf{A}}_{\mathrm{out}}} = \mathrm{sgn}(\hat{\mathbf{A}}_{\mathrm{out}}) \tag{4.3.18}$$

而最终的邻接矩阵会被设置为：

$$\hat{\mathbf{A}}_{\mathrm{out}} = \frac{1}{2}\left(\hat{\mathbf{A}}^\top + \hat{\mathbf{A}}\right) \tag{4.3.19}$$

4.4 DropEdge

图卷积神经网络通常非常容易过拟合，为了解决这一问题，训练真正深的图卷积神经网络，[5] 提出了一种称为 DropEdge 的方法。

$$\mathbf{A}_{drop} = \mathbf{A} - \mathbf{A}' \tag{4.4.1}$$

其中 $\mathbf{A}'$ 是由原邻接矩阵直接随机保留固定数量的边得到的。这一操作算是算是图上的 Dropout。关于原来的 GCN 为啥不能变深，论文也说的很清楚：如若一个 GCN 有无穷个层，按如下公式更新：

$$\mathbf{H}^{(l+1)} = \hat{\mathbf{A}}\mathbf{H}^{(l)} \tag{4.4.2}$$

那么：

$$\lim_{l\to\infty} \mathbf{H}^{(l)} = \lim_{l\to\infty} \hat{\mathbf{A}}^l\mathbf{X} = \pi \tag{4.4.3}$$

其中 $\pi_{i,j} = \frac{d_j}{2|\mathcal{E}|}$ 与输入 $\mathbf{X}$ 无关。然后作者认为再使用了 DropGCN 计算得到的 $\mathbf{H}'$ 后：

$$|\mathbf{H}'^{(l)} - \pi'| \geq |\mathbf{H}^{(l)} - \pi| \tag{4.4.4}$$

就不会那么与输入无关了。

4.5 EGNN

对图上的节点做图卷积可以有效地做特征嵌入 (Embedding)，然而对图上的边做特征变换则更有助于分类或聚类，EGNN(Edge-Labeling Graph Neural Network [6]) 提出了一种对边进行变换的方法。

对于一个图，EGNN 的边 $\mathbf{e}_{ij} = \{e_{ijd}\}_{d=1}^2 \in [0,1]^2$ 是这样构建的

$$\mathbf{e}_{ij}^0 = \begin{cases} [1||0], & \text{if } v_i \sim v_j, \\ [0||1], & \text{if } v_i \neq v_j, \\ [0.5||0.5], & \text{otherwise,} \end{cases} \tag{4.5.1}$$

其中 $v_i \sim v_j$ 代表两者均已标注并且两者均属于同一类，而 $v_i \neq v_j$ 则代表两者均已标注且不属于同一类。节点按以下规则更新：

$$\mathbf{v}_i^\ell = f_v^\ell([\sum_j \tilde{e}_{ij1}^{\ell-1}\mathbf{v}_j^{\ell-1}||\sum_j \tilde{e}_{ij2}^{\ell-1}\mathbf{v}_j^{\ell-1}];\theta_v^\ell) \tag{4.5.2}$$

其中

$$\tilde{e}_{ijd} = \frac{e_{ijd}}{\sum_k e_{ikd}} \tag{4.5.3}$$

，而 f_v^ℓ 是特征转换网络（feature transformation network）。边则按照以下规则更新：

$$\bar{e}_{ij1}^\ell = \frac{f_e^\ell(\mathbf{v}_i^\ell, \mathbf{v}_j^\ell;\theta_e^\ell)e_{ij1}^{\ell-1}}{\sum_k f_e^\ell(\mathbf{v}_i^\ell, \mathbf{v}_k^\ell;\theta_e^\ell)e_{ik1}^{\ell-1}/(\sum_k e_{ik1}^{\ell-1})}, \tag{4.5.4}$$

$$\bar{e}_{ij2}^\ell = \frac{(1 - f_e^\ell(\mathbf{v}_i^\ell, \mathbf{v}_j^\ell;\theta_e^\ell))e_{ij2}^{\ell-1}}{\sum_k (1 - f_e^\ell(\mathbf{v}_i^\ell, \mathbf{v}_k^\ell;\theta_e^\ell))e_{ik2}^{\ell-1}/(\sum_k e_{ik2}^{\ell-1})}, \tag{4.5.5}$$

$$\mathbf{e}_{ij}^\ell = \frac{\bar{\mathbf{e}}_{ij}^\ell}{\|\bar{\mathbf{e}}_{ij}^\ell\|_1} \tag{4.5.6}$$

4.6 SGC

图上的卷积变换是 GCN 的核心，虽然这一操作并不是非常地复杂，但是不得不承认该操作是非常耗时的。Simple Graph Convolution (SGC)[7] 的作者认为

4.7 Attention-based Graph Neural Network

传统的图卷积神经网络的信息传递方法时固定不变的（每一层的传递的参数都是由输入确定的），Attention-based Graph Neural Network [8] 提出一种动态的传递方法，以在信息传递时用到当前层的特征之间的联系，而不仅仅是输入之间的联系，这样能够使得图卷积神经网络在传递信息时动态更新节点之间的联系，更好地发现输入之间的潜在联系。

信息的传播（propagation）层定义为

$$\tilde{X}^{(n)} = P^{(n)} X^{(n)} \tag{4.7.1}$$

$$P_{i,j}^{(n)} = \frac{\exp(\beta \cdot \cos(\mathbf{x}_i^{(n)}, \mathbf{x}_j^{(n)}))}{\sum_{k \in \mathcal{N}(i) \cup \{i\}} \exp(\beta \cdot \cos(\mathbf{x}_i^{(n)}, \mathbf{x}_k^{(n)}))} \tag{4.7.2}$$

，其中 $\mathcal{N}(i)$ 代表节点 i 的相邻节点的集合。

第三部分

应用

第 5 章 聚类

5.1 Attributed Graph Clustering

传统的基于图的聚类算法通常是先通过图对其上节点进行嵌入（embedding），之后再使用传统方法在嵌入后得到的特征上进行聚类，Attributed Graph Clustering[9] 在使用图卷积神经网络（GCN）使用一种注意力机制网络（attention network），相较于传统的两步（two-step）算法，本问题出的方法更直接地面对数据间的关联问题，而不会在嵌入这个地方绕个圈子。

每一层的输出表示为

$$z_i^{(l+1)} = \sigma(\sum_j \alpha_{ij} W z_j^{(l)}) \tag{5.1.1}$$

其中 $z^{(m)}$ 表示网络的第 m 层的输出，$\sigma(\cdot)$ 表示激活函数，α_{ij} 是注意力系数，由如下方法计算得出

$$\alpha_{ij} = \frac{\exp(c_{ij})}{\sum_j \exp(c_{ij})} \tag{5.1.2}$$

而系数 c_{ij} 可以看成由 2 层神经网络计算得出

$$c_{ij} = \mathbf{a}[Wx_i \| Wx_j] \tag{5.1.3}$$

向量 $\mathbf{a} \in R^{2m}$ 代表神经网络输出的权重。当然，该系数的计算还可以加入一定和图相关的约束，详见论文。

最后的 loss 有两部分重构（Reconstruction）loss

$$L_r = loss(A, \hat{A}) \tag{5.1.4}$$

其中

$$\hat{A}_{ij} = \text{sigmoid}(z_i^T z_j) \tag{5.1.5}$$

聚类（clusgtering）loss

$$L_c = D_{KL}(P \| Q) \tag{5.1.6}$$

参考文献

[1] SILVA T C, ZHAO L. Machine learning in complex networks: volume 2016[M]. [S.l.]: Springer, 2016.

[2] KIPF T N, WELLING M. Semi-supervised classification with graph convolutional networks[J]. arXiv preprint arXiv:1609.02907, 2016.

[3] JIANG B, WANG L, TANG J, et al. Semi-supervised learning with adaptive neighborhood graph propagation network[J]. ArXiv, 2019, abs/1908.05153.

[4] GAO X, HU W, GUO Z. Exploring graph learning for semi-supervised classification beyond euclidean data[J]. arXiv preprint arXiv:1904.10146, 2019.

[5] RONG Y, HUANG W, XU T, et al. The truly deep graph convolutional networks for node classification[J]. ArXiv, 2019, abs/1907.10903.

[6] KIM J, KIM T, KIM S, et al. Edge-labeling graph neural network for few-shot learning[Z]. [S.l.: s.n.], 2019.

[7] WU F, ZHANG T, DE SOUZA JR. A H, et al. Simplifying graph convolutional networks[Z]. [S.l.: s.n.], 2019.

[8] THEKUMPARAMPIL K K, WANG C, OH S, et al. Attention-based graph neural network for semi-supervised learning[Z]. [S.l.: s.n.], 2018.

[9] WANG C, PAN S, HU R, et al. Attributed graph clustering: A deep attentional embedding approach[Z]. [S.l.: s.n.], 2019.

附录 torch_geometric 使用

torch_geometric 是一个基于 PyTorch 的基于图的深度学习的包，本章节参考了Installation — pytorch_geometric 1.3.2 documentation。

A.1 安装

首先确保 torch 的版本在最新 (1.2.0)

```
$ python -c "import torch; print(torch.__version__)"
>>> 1.2.0
```

然后使用 pip 安装

```
$ pip install --verbose --no-cache-dir torch-scatter
$ pip install --verbose --no-cache-dir torch-sparse
$ pip install --verbose --no-cache-dir torch-cluster
$ pip install torch-geometric
```

A.2 例子

导入数据集

```
from torch_geometric.datasets import Planetoid
dataset = Planetoid(root='/tmp/Cora', name='Cora')
```

定义一个经典的两层 gcn

```
import torch
import torch.nn.functional as F
from torch_geometric.nn import GCNConv

class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = GCNConv(dataset.num_node_features, 16)
        self.conv2 = GCNConv(16, dataset.num_classes)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index

        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, training=self.training)
        x = self.conv2(x, edge_index)

        return F.log_softmax(x, dim=1)
```

训练!

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = Net().to(device)
data = dataset[0].to(device)
```

```
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)

model.train()
for epoch in range(200):
    optimizer.zero_grad()
    out = model(data)
    loss = F.nll_loss(out[data.train_mask], data.y[data.train_mask])
    loss.backward()
    optimizer.step()
```

接下来就是测试了

```
model.eval()
_, pred = model(data).max(dim=1)
correct = float (pred[data.test_mask].eq(data.y[data.test_mask]).sum().item())
acc = correct / data.test_mask.sum().item()
print('Accuracy: {:.4f}'.format(acc))
```

我的结果是 '0.7990'，你呢?