# 4.10 图神经网络 （Graph Neural Networks，GNN）

## 4.10.1 图嵌入（graph embedding）

图（复杂网络）是一种用于描述和建模复杂系统（complex systems）的通用语言。通过量化的图分析可以更好的理解复杂网络，但是对于包含大量顶点和边及其属性的网络而言，通常需要使用图嵌入技术有效的将高维稀疏图转换为低维、密集和连续的（嵌入）向量空间，最大限度的保留图的结构属性，从而很容易在嵌入向量空间中使用标准度量量化原始复杂不规则空间中顶点的相似度[1][2]57。图嵌入的方法有 DeepWalk[3]、node2vec[4]、Line[5]、SDNE[6]和 struc2vec[7]等。

### 4.10.1.1 DeepWalk

Perozzi, B.等人[3]基于自然语言处理中词嵌入的 Word2Vec[8] 方法（可以查看*Transformer——自然语言处理*部分）提出了处理复杂网络的 DeepWalk， 该方法以一个图作为输入，通过给定深度（长度、次数）随机漫步（Random Walks）获得每个顶点随机邻接顶点，及随机邻接顶点的随机邻接顶点不断迭代深度的一个行走序列顶点列表来获取每个顶点的局部信息，将其视为自然语言处理中的语料库（corpus）用于学习复杂网络顶点（词）的嵌入向量/潜在表征（latent representations ）。

#### 1） 随机漫步（Random Walks）

以顶点vi​为源，其随机漫步的顶点（属性）列表记为Wvi​​，为一个随机过程，包含给定长度的随机变量 Wvi​1​,Wvi​2​,…,Wvi​k​。对于一个随机变量Wvi​k+1​，是从顶点vk​邻接顶点中随机选择的一个顶点。随机漫步已被用于内容推荐（content recommendation）、社区检测（community detection）等各类问题中的相似性度量。除了作为可以从网络中提取信息的基本工具，随机漫步能够执行并行计算，即可以同时探索一个图中的不同部分信息，提高计算效率；其次，依赖从随机漫步中获得的信息，可以在不需要全局重新计算的情况下适应图结构的微小变化，用新的随机漫步迭代更新已学习到的模型。

定义RandomWalker类，其deepwalk\_walk方法[9]实现了网络顶点随机漫步的算法，代码如下，

def deepwalk\_walk(self, walk\_length, start\_node):  
 walk = [start\_node]  
  
 while len(walk) < walk\_length:  
 cur = walk[-1]  
 cur\_nbrs = list(self.G.neighbors(cur))  
 if len(cur\_nbrs) > 0:  
 walk.append(random.choice(cur\_nbrs))  
 else:  
 break  
 return walk

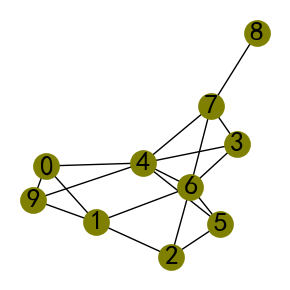
计算以[NetworkX](https://networkx.org/documentation/stable/index.html)①库提供的erdos\_renyi\_graph[10]随机图中顶点0的随机漫步为例，计算结果如下。

%load\_ext autoreload   
%autoreload 2  
import usda.network as usda\_network   
import usda.network.graph\_embedding as usda\_ge  
import usda.manifold as usda\_manifold  
  
import networkx as nx  
import pandas as pd  
import numpy as np  
from sklearn.manifold import TSNE  
import matplotlib.pyplot as plt  
  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score

C:\Users\richie\anaconda3\envs\rl\Lib\site-packages\paramiko\transport.py:219: CryptographyDeprecationWarning: Blowfish has been deprecated  
 "class": algorithms.Blowfish,

erdos\_renyi\_graph(n, p, seed=None, directed=False)方法中，参数n为顶点的数量；p为创建边的概率。

G\_er=nx.erdos\_renyi\_graph(10, 0.3, seed=1, directed=False)  
usda\_network.G\_drawing(G\_er)



仅计算了图中顶点0的随机漫步，随机选择了其邻接顶点4；而顶点4的随机邻接顶点为7，顶点7的随机邻接顶点为6，以此类推，完成指定长度的随机漫步。

G\_er\_randomwalk=usda\_ge.RandomWalker(G\_er)  
walk\_lst=G\_er\_randomwalk.deepwalk\_walk(walk\_length=10, start\_node=0)  
walk\_lst

[0, 4, 7, 6, 4, 5, 4, 5, 4, 3]

#### 2）从自然语言处理的词嵌入到复杂网络的图（顶点）嵌入

自然语言建模的目的是估计语料库中更容易构成具有意义短语或句子词汇组合的可能性。给出一个词汇组成的序列（句子或短语）W1n​=(w0​,w1​,⋯,wn​)，式中， wi​∈V（V是词汇表），希望对于所有训练的语料库最大化Pr(wn​∣w0​,w1​,⋯,wn−1​)。对应到复杂网络，将词汇对应到顶点，将词汇组成的句子对应到随机漫步表征局部网络结构特征的顶点序列上，则观测到顶点vi​的可能性的公式可表示为Pr(vi​∣(v1​,v2​,⋯,vi−1​))。

顶点嵌入向量的估计同词嵌入是要学习到可以表征网络结构顶点间的距离关系，而不仅仅是顶点共现（co-occurrences）的概率分布。因此引入一个映射函数Φ:v∈V↦R∣V∣×d，式中，Φ表示与图中每个顶点v相关的嵌入向量空间，并用一个∣V∣×d大小的矩阵表示，有Pr(vi​∣(Φ(v1​),Φ(v2​),⋯,Φ(vi−1​)))。在 Word2Vec 中，上述方法称之为 CBOW（Continuous Bag-of-Words），而提出的 Skip-gram 结构模型表现要好于 CBOW，其不是使用上下文语境来预测缺失的词，与其相反，由该词预测上下文中出现词的概率，对应到复杂网络，上述公式调整为Φminimize​−logPr({vi−w​,⋯,vi−1​,vi+1​,⋯,vi+w​}∣Φ(vi​))。给定顶点vi​，最大化其邻接顶点在随机漫步中的概率，可以用分类器（如逻辑回归 SoftMax函数）来学习这种后验分布。然而，使用逻辑回归建模将会产生大量（数量等于∣V∣）的标签，为了加快训练速度，可以使用 Hierarchical Softmax（H-Softmax） 方法近似概率分布。

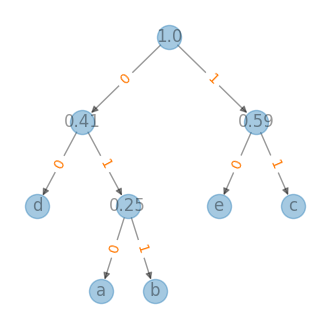
霍夫曼编码（Huffman Coding）

在计算机科学和信息理论中，David A. Huffman [11]提出了一种特殊类型的最优前缀码（prefix code），称为霍夫曼编码（Huffman Coding），通常用于无损数据压缩。在下述的试验中，提供了一组字符data，包括有['a', 'b', 'c', 'd', 'e']等5个字符，假设每个字符在语料库中出现的频率freq，为[0.10, 0.15, 0.30,0.16, 0.29]，和为1，那么对其进行霍夫曼编码结果如下图。从图中可以观察到霍夫曼编码的过程，将字母按照其权值排序，首先选择最小权值的两个字母a和b，构造其父顶点，权值为a和b之和 0.25；继续排序剩余的顶点加上新构造的父顶点，仍旧选择最小权值的两个顶点，d和新构造值为0.25的顶点构造其父顶点，以其和为父顶点的权值，即 0.41，以此类推，直至完成所有顶点的图构造，并最终存在一个根顶点。因为字符的频率和为1，因此根顶点的权值为1。而每个字符均位于叶顶点（节点）。且总共增加的新顶点数为。

如果将每个顶点的左分支标记为0，而右分支标记为1，则可以编码叶顶点，例如对于字符a，其霍夫曼编码结果为010；对于字符c，其霍夫曼编码结果为11。从编码结果可以发现，字符出现的频率高，则其编码长度会相对短；而字符出现的频率低，则其编码的长度相对长。如果使用二进制定长编码（给每个字符一个固定长度的编码），对于试验中的5个字符，需要至少长度为3个位数（）满足要求，那么总共位数为 15。而霍夫曼编码结果总位数为12，这要小于定长编码，因此可以有效的压缩数据。

data=['a', 'b', 'c', 'd', 'e']  
freq=[0.10, 0.15, 0.30,0.16, 0.29]  
size=len(data)  
  
root=usda\_network.HuffmanCodes(data, freq, size,verbose=False)  
usda\_network.draw\_tree(root,figsize=(4,4))  
  
huffman\_encoding\_dict=usda\_network.huffman\_encoding\_dict  
print(huffman\_encoding\_dict)

{'d': [0, 0], 'a': [0, 1, 0], 'b': [0, 1, 1], 'e': [1, 0], 'c': [1, 1]}



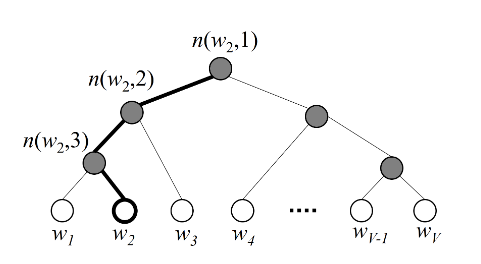
定义huffman\_encode()和huffman\_decode()方法，根据上述构造的霍夫曼编码树root，实现对任意给定字符组合的霍夫曼编码和解码。

s='badbed'  
s\_encoded=usda\_network.huffman\_encode(s,huffman\_encoding\_dict)  
print(s\_encoded)  
s\_decoded=usda\_network.huffman\_decode(s\_encoded,root)  
print(s\_decoded)

011010000111000  
badbed

* Hierarchical Softmax（H-Softmax）[12]

H-Softmax 是一种计算 softmax 的有效方法[13]，使用霍夫曼编码树编码词汇表中所有的词汇，如图[12]，其中白色圆圈为叶顶点，对应到各个词汇；黑色圆圈为内部（中间）顶点。每个词汇均有一个从根（root）到对应叶顶点的唯一路径，该路径用于估计由叶顶点表示的词汇的概率。例如对于词汇w\_2，用顶点路径表示的编码长度为L(w2​)=4。n(w,j)表示路径中第j个顶点。



在 H-Softmax 中，一个词汇的概率定义为， p(w=wO​)=∏j=1L(w)−1​σ([[n(w,j+1)=ch(n(w,j))]]⋅vn(w,j)′​Th)，式中， ch(n)为顶点n的左分支子顶点； vn(w,j)′​为内部顶点n(w,j)的嵌入向量/向量表示（vector representation）；h是隐藏层的输出值，在 Skip-gram 模型中， h=vwI​​，为词汇w的嵌入向量[14]。在 CBOW 模型中， h=C1​∑c=1C​vwc​​。因此在 H-Softmax 中，并没有输出词嵌入矩阵，而是为词内部顶点的嵌入向量vn(w,j)′​； [[x]]为一个特殊函数，定义为， [[x]]={1−1​ if x is true;  otherwise ​。

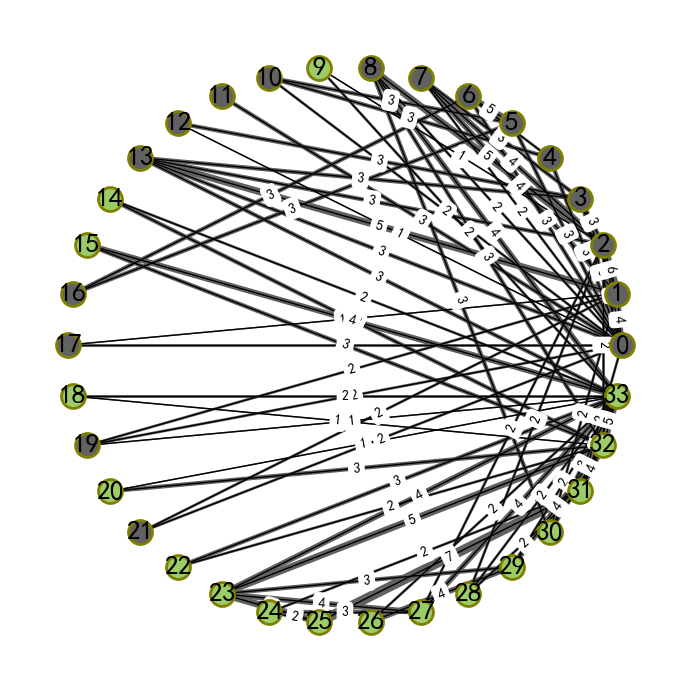
计算上图中w2​作为输出词的概率，该概率定义为从根顶点开始随机漫步，结束于该词对应的叶顶点。对于每个内部顶点（包括根顶点），需要分配向左或向右的概率。如果定义内部顶点n向左的概率为p(n, left )=σ(vn′T​⋅h)，由内部顶点的嵌入向量和该词的嵌入向量确定；因此，内部顶点n向右的概率为， p(n, right )=1−σ(vn′T​⋅h)=σ(−vn′T​⋅h)。因此按照上图给出的词w2​的路径，计算w2​作为输出词的概率为， p(w2​=wO​)=p(n(w2​,1), left )⋅p(n(w2​,2), left )⋅p(n(w2​,3), right )=σ(vn(w2​,1)′​Th)⋅σ(vn(w2​,2)′​Th)⋅σ(−vn(w2​,3)′​Th)，并且，∑i=1V​p(wi​=wO​)=1。

回到复杂网络，将顶点构造为霍夫曼编码树，对于顶点uk​∈V，计算Pr(uk​∣Φ(vj​))可以转化为最大化树中特定路径的概率。如果到顶点uk​的路径表示为一个树顶点的序列 (b0​,b1​,…,b⌈log∣V∣⌉​),(b0​=root,b⌈log∣V∣⌉​=uk​)，则， Pr(uk​∣Φ(vj​))=∏l=1⌈log∣V∣⌉​Pr(bl​∣Φ(vj​))，式中， Pr(bl​∣Φ(vj​))可以通过顶点bl​父顶点的二元分类器建模，从而降低了计算Pr(uk​∣Φ(vj​))的复杂度， 从O(∣V∣)降低到O(log∣V∣)。

#### 3）空手道俱乐部的 DeepWalk 图嵌入

下述的试验以NetworkX库提供的karate\_club\_graph图[15]为例计算图嵌入。该图总共有34个顶点，顶点有字段为club的属性，表示该节点是属于Mr. Hi还是Officer阵营。每个边赋予了一个权值，为基于该边相互作用顶点在上下文语境中出现的次数。

G\_karate\_club=nx.karate\_club\_graph()  
pos=nx.circular\_layout(G\_karate\_club)  
  
nodes\_attri=pd.concat({k: pd.DataFrame.from\_dict(v, orient='index') for k, v in dict(G\_karate\_club.nodes(data=True)).items()}, axis=0).reset\_index().rename(columns={'level\_0':'node','level\_1':'attri',0:'club'})  
nodes\_attri\_groups=nodes\_attri.groupby("club")["node"].apply(list).reset\_index()  
  
edge\_attributes=nx.get\_edge\_attributes(G\_karate\_club,'weight')  
usda\_network.G\_drawing(G\_karate\_club,  
 nodes=nodes\_attri\_groups.node.tolist(),  
 nodes\_color=['#616161','#9CCC65'],  
 pos=pos,figsize=(7,7),  
 routes=list(edge\_attributes.keys()),  
 edge\_widths=list(edge\_attributes.values()),  
 edge\_labels='weight',  
 edge\_colors=['#616161']\*len(edge\_attributes))



查看图的顶点和边属性。

print(list(G\_karate\_club.nodes(data=True))[17:21])  
print(list(G\_karate\_club.edges(data=True))[:3])

[(17, {'club': 'Mr. Hi'}), (18, {'club': 'Officer'}), (19, {'club': 'Mr. Hi'}), (20, {'club': 'Officer'})]  
[(0, 1, {'weight': 4}), (0, 2, {'weight': 5}), (0, 3, {'weight': 3})]

将顶点属性编码为整数值。

nodes\_attri['club\_int']=nodes\_attri.club.apply(lambda x:1 if x=='Officer' else 0)  
nodes\_attri[8:11]

|  | **node** | **attri** | **club** | **club\_int** |
| --- | --- | --- | --- | --- |
| **8** | 8 | club | Mr. Hi | 0 |
| **9** | 9 | club | Officer | 1 |
| **10** | 10 | club | Mr. Hi | 0 |

定义DeepWalk类实现 DeepWalk 图嵌入算法，核心使用gensim库的Word2Vec方法。配置嵌入向量的维度embed\_size(vector\_size)为 128 维；使用 Skip-gram 模型（sg=1）和 H-Softmax 算法（hs=1）；配置当前顶点到预测顶点间的最大距离window\_size（window）为10；并执行并行计算（workers=3）。

model\_deepwalk=usda\_ge.DeepWalk(G\_karate\_club, walk\_length=10, num\_walks=80, workers=1)  
model\_deepwalk.train(  
 window\_size=10,   
 iter=30,  
 sg=1, # Skip-gram  
 hs=1, # Hierarchical softmax  
 workers=3,  
 embed\_size=128,  
 );

Learning embedding vectors...  
Learning embedding vectors done!

查看计算结果，获得顶点的嵌入矩阵，维度为(顶点数，嵌入向量维数)。

embeddings=model\_deepwalk.get\_embeddings()  
embeddings\_array=np.array([embeddings[node] for node in G\_karate\_club.nodes])  
embeddings\_array.shape

(34, 128)

使用wv.most\_similar方法可以计算与给定顶点距离最近的多个顶点（由近及远排序）。对于顶点 0，基于上述图结构，与其最近的顶点为 4，距离值约为 0.746（距离度量使用 cosine similarity 算法）。

model\_deepwalk.w2v\_model.wv.most\_similar(positive=[0])

[(4, 0.7461870312690735),  
 (10, 0.690133273601532),  
 (3, 0.6589341759681702),  
 (6, 0.6216689348220825),  
 (5, 0.6070966124534607),  
 (12, 0.5991224050521851),  
 (16, 0.5978891849517822),  
 (1, 0.5969511866569519),  
 (11, 0.5932941436767578),  
 (21, 0.5916509628295898)]

可以直接使用wv.similarity方法计算给定两个顶点间的距离。

model\_deepwalk.w2v\_model.wv.similarity(0,4)

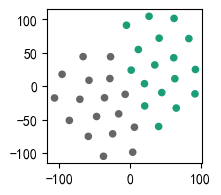
0.74618703

使用 t-SNE 算法降维图顶点嵌入矩阵，由 128 维降至 2 维打印查看顶点的分布情况。从结果来看，代表不同阵营的顶点能够很好的分离，表明了 DeepWalk 图嵌入方法可以将图顶点转化为连续的嵌入向量空间，分析图结构。

Y=usda\_manifold.tsne(embeddings\_array,2,max\_iter=1000,print\_interval=200,init='pca',random\_state=0,learning\_rate='auto')

Preprocessing the data using PCA...  
Computing pairwise distances...  
Mean value of sigma: 2.0342620 of 34...  
Iteration 1000: error is 0.348395

plt.figure(figsize=(2, 2), dpi=100)  
plt.scatter(Y[:, 0], Y[:, 1], 20, nodes\_attri.club\_int,cmap='Dark2')  
plt.show()

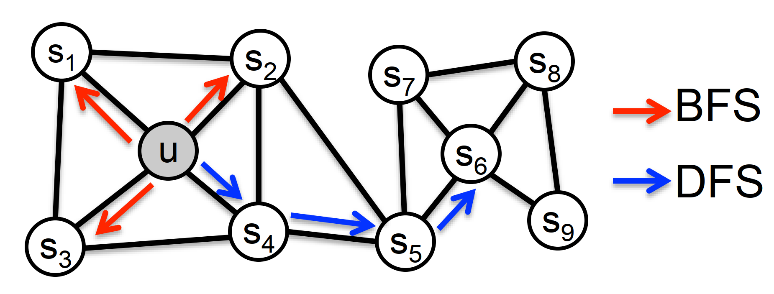


### 4.10.1.2 node2vec[4]

#### 1）BFS 和 DFS

下图[4]呈现了网络结构的两个特性，第一种特征是同质性（homophily），如顶点和顶点之间的关系，这几个顶点属于同一个紧密结合的社区，顶点之间边的数量（路的长度）趋小，而之间边的权值趋大；另一种特征是结构等效性（structural equivalence），如顶点和，这两个顶点并不是紧密相连，而是位于不同的社区中，但是表现出相似的结构特征，为各自所在社区中心顶点的角色，其顶点度均为4，与邻接顶点构成围长为3的圈有3~4个。

为了权衡同质性和结构等效性，Grover, A.[4]等人将源顶点邻域采样问题看作是一种局部搜索形式，提出了两种采样策略，广度优先采样（ Breadth-ﬁrst Sampling，BFS）和深度优先采样（Depth-ﬁrst Sampling，DFS）。BFS 倾向于采样源顶点邻域邻接的顶点，如U邻接顶点3S1​,S2​,S3​；DFS 倾向于由距离源顶点越来愈远的顶点依次采样组成，如U与S4​,S5​,S6​。



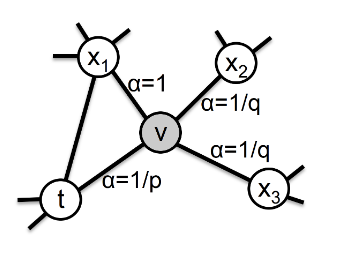
BFS 和 DFS 代表了同质性和结构等效性两种极端搜索的情境，并对学习到的图嵌入向量空间产生影响。而复杂网络的顶点预测任务经常在同质性和结构等效性之间变换。在同质性假设下，高度互连和属于相似簇或社区的顶点，对其估计的嵌入向量距离更近；而在结构等效性假设下，在网络中具有相似结构角色的顶点（不强调连通性，在网络中的顶点可能相距很远），对其估计的嵌入向量距离则更近。

#### 2）node2vec 采样策略

基于 BFS 和 DFS，Grover, A.等人设计了一个灵活的邻域采样策略，开发了一种灵活的有偏随机漫步（biased random walk），以 BFS 和 DFS 方式搜索邻域，可以在 BFS 和 DFS 之间平滑的进行插值。

给定一个源顶点，模拟一个给定长度的随机漫步。用ci​表示漫步中的第i个顶点，开始的顶点c0​=u，即为源顶点。顶点ci​由下述分布给出， otherwise P(ci​=x∣ci−1​=v)={Zπvx​​0​ if (v,x)∈E otherwise ​，式中，πvx​为顶点v和x之间的非归一化转移概率（unnormalized transition probability），Z是一个归一化常数。

如下图[4]，定义一个有两个参数p和q的2阶随机漫步，该随机漫步已经经过顶点t到达了顶点v，现在需要确定如何走下一步，是到x1​,x2​,x3​，还是返回t，因此需要计算顶点v的边 (v,x)的转移概率πvx​。设非归一化的转移概率πvx​=αpq​(t,x)⋅wvx​，有αpq​(t,x)=⎩⎨⎧​p1​1q1​​ if dtx​=0 if dtx​=1 if dtx​=2​，式中， dtx​表示顶点t和x之间最短路径距离，且dtx​属于集合{0,1,2}。参数p和q控制了随机漫步探索和离开源顶点u邻域的速度，并在探索过程中在 BFS 和 DFS 之间（近似）插值（interpolate）。



返回参数（Return parameter），：该参数用于控制随机漫步中，某一顶点即刻重访其上一顶点的概率，即当前顶点重访上一顶点的可能性（上图）。如果将其配置为一个较大的值（），则接下来的两步中，不太可能对已经访问过的顶点进行采样（除非随机漫步中的下一个顶点没有邻接顶点）。相反，如果该参数值很小（），这将导致随机漫步后退一步，使得漫步更倾向于在源顶点附近。返回参数（Return parameter）。

出入参数（In-out parameter），：该参数可以控制权衡在“内部的”（inward）和“外部的”（outward）顶点中搜索。如果，随机漫步倾向于选择顶点附近的顶点，近似 BFS 采样策略的同质性特征。相反，如果，随机漫步更倾向于访问离顶点更远的顶点，近似 DFS 采样策略的结构等效性的特征。

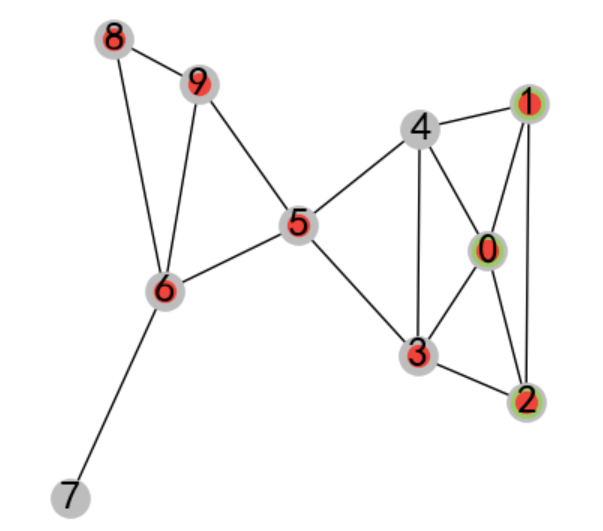
如果，随机漫步则等同于 DeepWalk 中的随机漫步。

下面的试验重构了 node2vec 论文中解释 BFS和 DFS的示例图，其中顶点 0 对应到；顶点 6 对应到。定义node2vec\_walk2方法[9]计算 node2vec。计算中多次运行比较获得如下结果，从图中可以观察到，含有红色的圆实现了 DFS 采样策略；而含有绿色的圆则实现了 BFS 采样策略。

G\_n2v=nx.Graph()  
G\_n2v.add\_edges\_from([(0,1),(0,2),(0,3),(0,4),(1,4),(1,2),(3,2),(3,4),(5,3),(5,4),(6,5),(6,7),(6,8),(6,9),(9,5),(9,8)])

pq=[(1,1),(1,10),(10,1)]  
walk\_lsts=[]  
for p,q in pq:  
 node2vec\_walk=usda\_ge.RandomWalker(G\_n2v,p=p, q=q, use\_rejection\_sampling=False)  
 node2vec\_walk.preprocess\_transition\_probs()  
 walk\_lst=node2vec\_walk.node2vec\_walk2(walk\_length=8, start\_node=0)  
 walk\_lsts.append(walk\_lst)  
print(walk\_lsts)   
  
usda\_network.G\_drawing(  
 G\_n2v,  
 node\_color='#BDBDBD',  
 edgecolors='#BDBDBD',  
 nodes=walk\_lsts[1:],  
 nodes\_color=['#9CCC65','#F44336'],  
 nodes\_size=[200,100],  
 figsize=(4,4),  
 )

[[0, 3, 0, 4, 3, 2, 0, 2], [0, 2, 0, 2, 1, 2, 0, 1], [0, 1, 2, 3, 5, 6, 9, 8]]



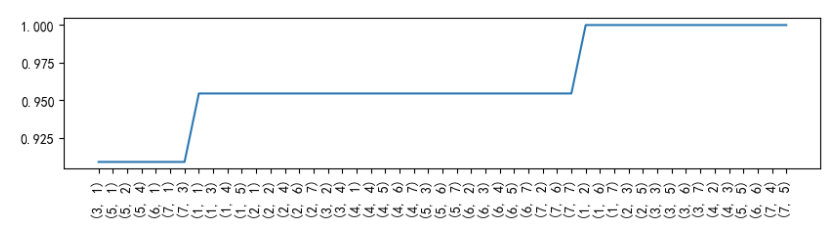
#### 3）空手道俱乐部的 node2vec 图嵌入

该试验包括两个部分，首先用 node2vec 方法计算图嵌入；然后用顶点的嵌入向量使用RandomForestClassifier随机森林分类算法建立分类预测模型，计算预测精度。比较多个值对计算结果建立分类预测模型的精度，可以发现当时，即为 DeepWalk 图嵌入算法所获得的精度并不是最高的，表明配置值合理的 node2vec 有偏随机漫步要优于 DeepWalk。

train\_mask=[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24]  
test\_mask=[0, 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 26, 27, 28, 29, 30, 31, 32, 33]  
labels=nodes\_attri.club\_int.to\_numpy()  
  
pq\_pairs=[(p,q) for p in range(1,8) for q in range(1,8)]  
  
pq\_scores={}  
for p,q in pq\_pairs:  
 model\_node2vec=usda\_ge.Node2Vec(G\_karate\_club, walk\_length=10, num\_walks=80,p=p, q=q, workers=1, use\_rejection\_sampling=1)  
 model\_node2vec.train(window\_size=10, iter=3,embed\_size=64);   
  
 clf=RandomForestClassifier(random\_state=0)  
 clf.fit(model\_node2vec.w2v\_model.wv[train\_mask], labels[train\_mask])  
   
 y\_pred=clf.predict(model\_node2vec.w2v\_model.wv[test\_mask])  
 acc=accuracy\_score(y\_pred, labels[test\_mask])  
  
 pq\_scores[(p,q)]=acc

Learning embedding vectors done!

pq\_scores\_sorted=sorted(pq\_scores.items(),key=lambda x:x[1])  
  
fig,ax=plt.subplots(figsize=(10,2))  
ax.plot([str(i[0]) for i in pq\_scores\_sorted], [i[1] for i in pq\_scores\_sorted], '-')  
ax.tick\_params(axis='x', labelrotation=90)



### 4.10.1.3 植物调查样方物种的图嵌入

[Countryside Survey 1978 vegetation plot data](https://ckan.publishing.service.gov.uk/dataset/countryside-survey-1978-vegetation-plot-data)②数据集包括于1978年在英国各地调查的 256 个 地块中存在的物种及其丰度（abundance）。该数据是在生态与水文中心（Centre for Ecology & Hydrology）管理的乡村调查（ Countryside Survey）长期监测项目下收集的。下载的数据集包括数据说明文件和两个数据文件，其中一个数据文件为地块的基本信息，其包含的字段如下：

1. YEAR - 调查年份；
2. SQUARE\_ID - 方阵标识；
3. PLOT\_ID - 地块标识；
4. PLOT\_TYPE - 地块类型；
5. COUNTRY ENV\_ZONE\_2007 - 环境分区编号（2007版）；
6. EZ\_DESC\_07 - 环境分区描述（2007版）。

另一个数据文件为调查地块中物种信息，其包含的字段如下：

1. YEAR - 调查年份；
2. SQUARE\_ID - 方阵标识；
3. PLOT\_ID - 地块标识；
4. AMALG\_PTYPE - 地块类型；
5. BRC\_NUMBER - 物种标识；
6. BRC\_NAMES - 种名；
7. TOTAL\_COVER - 占整个调查地块的百分比。

植物群落是在一个指定的地理单元内植物物种的集合或联合（ association）[16]，并形成一个与相邻不同植被类型斑块区分开来相对统一的斑块。每个植物群落的组成受到土壤类型、地形、气候和人为干扰的影响[17]。这里假设位于同一个调查样方中的植物物种更倾向于组成相对稳定的植物群落，从而对该数据集中植物调查样方（地块）物种构建图并计算图嵌入。构建图时，如果物种存在于同一个调查样方则建立边，并以占地百分比的差值作为边的一个属性，字段名为cover\_diff。如果具有边连接的两个物种存在于多个调查样方中，则计算其均值。

from itertools import combinations  
from collections import defaultdict  
from tqdm import tqdm  
from sklearn.preprocessing import LabelEncoder  
from statistics import mean   
from node2vec import Node2Vec  
import pandas as pd  
import networkx as nx

读取和查看植物调查数据。为了方便数据分析，使用[Sklearn](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html)③库的LabelEncoder方法对物种名称进行整数编码。

vegetation\_plot\_species\_list\_fn='../data/Countryside\_Survey\_1978\_vegetation\_plot\_data/data/Vegetation Plot - Species List 1978.csv'  
vegetation\_plot\_information\_fn='../data/Countryside\_Survey\_1978\_vegetation\_plot\_data/data/Vegetation Plot - Plot Information 1978.csv'

species\_list=pd.read\_csv(vegetation\_plot\_species\_list\_fn)  
le=LabelEncoder()  
species\_list['name\_encoder']=le.fit\_transform(species\_list.BRC\_NAMES)  
species\_list.tail(3)

|  | **YEAR** | **SQUARE\_ID** | **PLOT\_ID** | **AMALG\_PTYPE** | **BRC\_NUMBER** | **BRC\_NAMES** | **TOTAL\_COVER** | **name\_encoder** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **44492** | 1978 | WDGZFA | GKGCGUEPMT | X | 5505962.0 | Cladonia sp. | 1 | 153 |
| **44493** | 1978 | WDGZFA | GKGCGUEPMT | X | 9202136.0 | Vaccinium myrtillus | 1 | 643 |
| **44494** | 1978 | WDGZFA | GKGCGUEPMT | X | 9202138.0 | Vaccinium vitis-idaea | 1 | 646 |

按调查地块（PLOT\_ID字段）计算两两物种的覆盖占比差值。

plot\_species\_group=species\_list.groupby(by=['PLOT\_ID'])

plot\_species\_group\_pairs\_info={}  
species\_pairs\_percent\_cover\_diff\_dict\_list=[]  
  
for group in tqdm(plot\_species\_group):  
 name\_encoder=group[1].name\_encoder.to\_list()  
 species\_total\_cover=group[1].TOTAL\_COVER.to\_list()  
 species\_cover\_dict=pd.Series(group[1].TOTAL\_COVER.values,index=group[1].name\_encoder).to\_dict()  
 species\_pairs\_percent\_cover\_diff\_dict={tuple(sorted((i,j))):abs(species\_cover\_dict[i]-species\_cover\_dict[j]) for i,j in combinations(name\_encoder,2)}  
 species\_pairs\_percent\_cover\_diff\_dict\_list.append(species\_pairs\_percent\_cover\_diff\_dict)  
  
species\_pairs\_percent\_cover\_diff\_dict\_merged=defaultdict(list)  
for d in species\_pairs\_percent\_cover\_diff\_dict\_list:  
 for key, value in d.items():  
 species\_pairs\_percent\_cover\_diff\_dict\_merged[key].append(value)

100%|███████████████████████████████████████████| 2279/2279 [00:00<00:00, 2528.74it/s]

species\_pairs\_percent\_cover\_diff\_mean={k:mean(v) for k,v in species\_pairs\_percent\_cover\_diff\_dict\_merged.items()}

构建图，并保存至本地磁盘。

species\_encoder2name=pd.Series(species\_list.BRC\_NAMES.values,index=species\_list.name\_encoder).to\_dict()  
  
G\_species=nx.Graph()  
G\_species.add\_edges\_from(list(species\_pairs\_percent\_cover\_diff\_mean.keys()))  
nx.set\_edge\_attributes(G\_species, species\_pairs\_percent\_cover\_diff\_mean,'cover\_diff')  
nx.set\_node\_attributes(G\_species,species\_encoder2name,'name')  
  
nx.write\_gml(G\_species,'../data/g\_species.gml')

计算图嵌入。此次试验使用的node2vec算法是直接调用[Node2Vec](https://github.com/eliorc/node2vec)④库提供的方法。实验中配置参数p,q不同值，计算了两次图嵌入，可以发现不同的p,q值将会有不同的结果。因此在实际问题分析时，需要根据分析目的权衡同质性和结构等效性，分析群落结构性的特征。

G\_species=nx.read\_gml('../data/g\_species.gml')  
node2vec=Node2Vec(G\_species, dimensions=64, walk\_length=30, num\_walks=200, p=.1, q=10, workers=8)  
model\_pq=node2vec.fit(window=10, min\_count=1, batch\_words=4)

Computing transition probabilities: 100%|███████████| 675/675 [02:30<00:00, 4.49it/s]

用计算获得的图嵌入矩阵（向量），定义most\_similar\_species()函数，给定物种名返回与该物种在网络结构中相似，距离相近的物种列表。

def most\_similar\_species(species\_name,species\_encoder2name,model,num=10):  
 species\_name2encoder={v:k for k,v in species\_encoder2name.items()}  
 species\_similar=model.wv.most\_similar(species\_name2encoder[species\_name])  
 species\_similar\_nameNencoder=[(i[0],species\_encoder2name[int(i[0])],i[1]) for i in species\_similar]  
   
 species\_encoder=species\_name2encoder[species\_name]  
 return species\_similar\_nameNencoder,species\_encoder   
  
species\_similar\_nameNencoder\_pq,species\_encoder=most\_similar\_species('Avena fatua',species\_encoder2name,model\_pq)   
species\_similar\_nameNencoder\_pq

[('132', 'Centaurea nigra', 0.6480474472045898),  
 ('521', 'Rumex acetosa', 0.6247394680976868),  
 ('627', 'Trifolium pratense', 0.6161571741104126),  
 ('477', 'Prunella vulgaris', 0.5903562307357788),  
 ('98', 'Cardamine pratensis', 0.5866711139678955),  
 ('470', 'Potentilla erecta', 0.5834638476371765),  
 ('136', 'Cerastium fontanum', 0.583010196685791),  
 ('628', 'Trifolium repens', 0.5810055732727051),  
 ('605', 'Succisa pratensis', 0.57342928647995),  
 ('469', 'Potentilla anserina', 0.5671511888504028)]

计算给定物种和其相似物种之间的最短路径，例如对于物种 Avena fatua（60）到嵌入向量空间中距离最近的物种为 Centaurea nigra（132），但是在所有的调查地块中，这两个物种并没有同时出现，而是存在一个中间物种 Achillea millefolium。而对于相似物种 Trifolium pratense（627），则存在边，即存在同时出现的地块。

shortest\_path\_st\_pq={i[0]:[G\_species.has\_edge(int(species\_encoder),int(i[0])),nx.shortest\_path(G\_species,source=int(species\_encoder),target=int(i[0])) ] for i in species\_similar\_nameNencoder\_pq}  
shortest\_path\_st\_pq

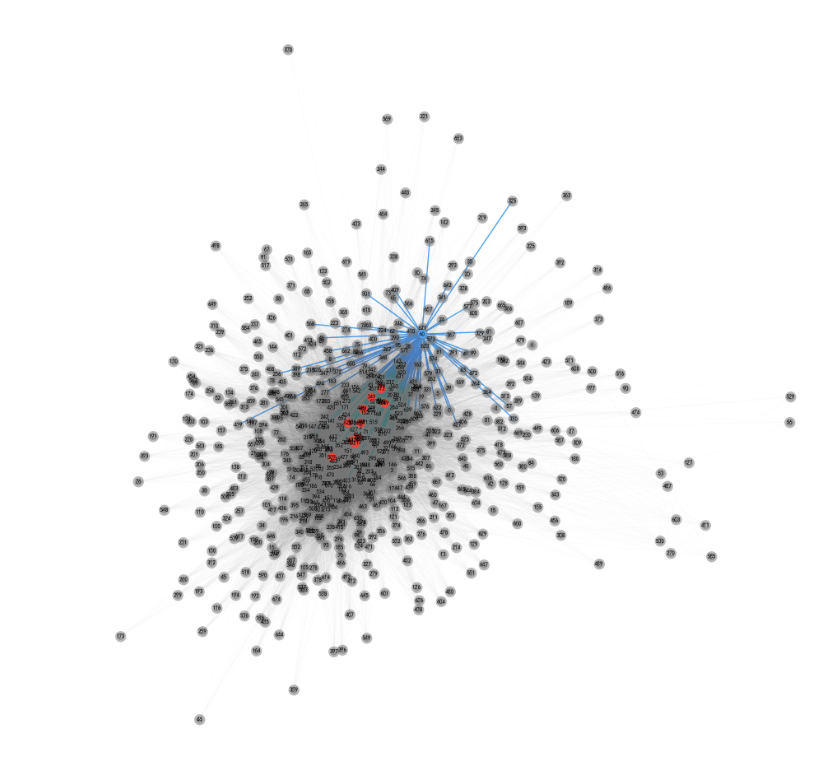
{'132': [False, [60, 2, 132]],  
 '521': [False, [60, 10, 521]],  
 '627': [True, [60, 627]],  
 '477': [False, [60, 12, 477]],  
 '98': [False, [60, 48, 98]],  
 '470': [False, [60, 183, 470]],  
 '136': [True, [60, 136]],  
 '628': [True, [60, 628]],  
 '605': [False, [60, 10, 605]],  
 '469': [False, [60, 12, 469]]}

species\_encoder2name[2]

'Achillea millefolium'

在图中显示给定物种及通过图嵌入计算获得的相似物种。

species\_edges=list(G\_species.edges(species\_encoder))  
  
similar\_nodes=[[species\_encoder],[int(i[0]) for i in species\_similar\_nameNencoder\_pq]]  
usda\_network.G\_drawing(G\_species,  
 figsize=(20,20),  
 edge\_widths=[2]\*len(species\_edges),  
 nodes=similar\_nodes,nodes\_color=['#42A5F5','#F44336'],  
 routes=species\_edges,  
 edge\_colors=['#42A5F5']\*len(species\_edges),  
 edge\_color=(117/255,117/255,117/255,.01),  
 font\_size=10,  
 node\_color=(117/255,117/255,117/255,.5),  
 edgecolors=(117/255,117/255,117/255,.5),  
 )



调整p,q值，再次计算图嵌入，观察计算结果发现，对于同一给定物种 Avena fatua（60）的相似物种发生的变化。

node2vec=Node2Vec(G\_species, dimensions=64, walk\_length=30, num\_walks=200, p=2, q=1, workers=8)  
model\_qp=node2vec.fit(window=10, min\_count=1, batch\_words=4)

Computing transition probabilities: 100%|███████████| 675/675 [02:32<00:00, 4.43it/s]

species\_similar\_nameNencoder\_qp,species\_encoder=most\_similar\_species('Avena fatua',species\_encoder2name,model\_qp)   
shortest\_path\_st={i[0]:[G\_species.has\_edge(int(species\_encoder),int(i[0])),nx.shortest\_path(G\_species,source=int(species\_encoder),target=int(i[0])) ] for i in species\_similar\_nameNencoder\_qp}  
shortest\_path\_st

{'315': [False, [60, 12, 315]],  
 '151': [False, [60, 2, 151]],  
 '319': [False, [60, 10, 319]],  
 '208': [False, [60, 2, 208]],  
 '240': [False, [60, 12, 240]],  
 '521': [False, [60, 10, 521]],  
 '493': [False, [60, 2, 493]],  
 '253': [False, [60, 12, 253]],  
 '175': [False, [60, 136, 175]],  
 '209': [True, [60, 209]]}

## 4.10.2 从 原始（vanilla） GNN 到 GCN、GAT、GraphSAGE 和 GIN [18,19]

阐述多种 GNN 网络模型时，为了便于理解算法，使用dash库定义gnn\_algorithms\_dash Web 应用程序[19]，通过交互操作更新图顶点表征（嵌入向量），观察值变化来分析算法的机制。

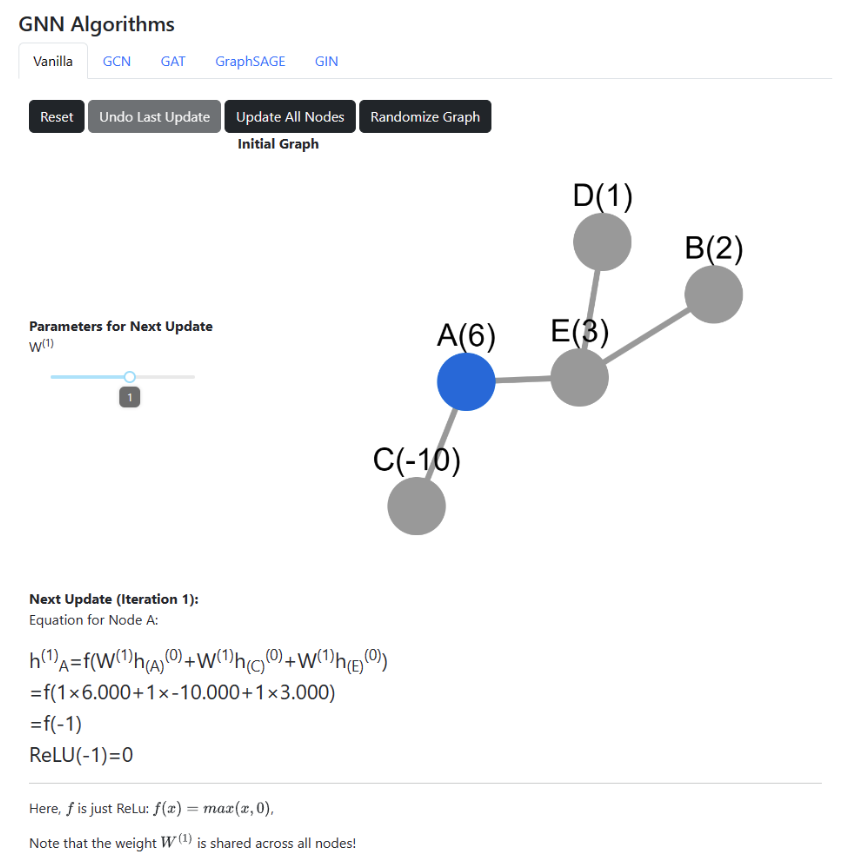
%load\_ext autoreload   
%autoreload 2   
import usda.network as usda\_nx  
  
import torch  
from torch\_geometric.datasets import FacebookPagePage,PPI,TUDataset  
from torch\_geometric.utils import to\_dense\_adj  
  
from sklearn.metrics import f1\_score  
from torch\_geometric.data import Batch  
from torch\_geometric.loader import DataLoader, NeighborLoader  
from torch\_geometric.nn import GraphSAGE  
from torch\_geometric.data import Data  
from torch\_geometric.utils import to\_dense\_batch  
from torch.nn import LSTM  
  
import networkx as nx

The autoreload extension is already loaded. To reload it, use:  
 %reload\_ext autoreload

usda\_nx.gnn\_algorithms\_dash()

### 4.10.2.1 vanilla (多层感知机，Multi-Layer Perceptron（MLP）) GNN（vGNN）

一个基本的神经网络层对应一个线性变换（linear transformation），公式表达为，式中，为顶点的输入向量（特征向量）；为权重。如果构建 MLP 直接学习特征向量（输出为顶点对应的一个分类），将忽略顶点之间边的联系，使得顶点间完全分离，网络学习不到图中顶点的语境，训练的模型预测精度必然不高。如果考虑顶点邻接顶点的特征，设NA​为顶点A的邻接顶点（含自身），则可以将图的线性层（graph linear layer）调整为， hA​=∑i∈NA​​xi​WT。下图简单示例中，假设图含有5个顶点，并仅包含一个特征（特征向量的维度为1），对于顶点A，结果为 -1，经激活函数（ReLU）非线性变换后结果为0。



将线性变换转换为矩阵表达有，式中为顶点的输入特征向量矩阵。同样将图线性层的上述公式转换为矩阵表达有H=A~TXWT，式中A~为包含自身的顶点邻接矩阵（方阵），为A~=A+I，式中A为邻接矩阵， I为单位矩阵（identity matrix）。用 0 表示两个顶点间不邻接，用 1 表示邻接。矩阵的对角线为顶点自身之间的关系，如果值为 1， 则包含自身。[PyG （PyTorch Geometric）](https://pytorch-geometric.readthedocs.io/en/latest/index.html)⑤是一个基于 [PyTorch](https://pytorch.org/)⑥构建的库，可以轻松编写和训练神经网络（Graph Neural Networks，GNNs），广泛用于与结构化数据相关的应用。

PyG 的 [datasets](https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html)⑦模块包含有大量图数据集，可以轻松编写和训练神经网络（Graph Neural Networks，GNNs），广泛用于与结构化数据相关的应用。PyG 的 [datasets](https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html" \t "_blank)⑦模块包含有大量图数据集，其中 Facebook 为经过验证页面-页面（page-page）的图数据集。顶点对应到官方的 Facebook 页面，边对应到互相点赞（likes）的站点。顶点特征为对站点的描述，任务是对站点类别进行多类别分类[20]。构建 vGNN 网络，用 Facebook 图数据集训练模型，完成分类任务。

下载和读取 Facebook 图数据集。

facebookPagePage\_fn='../data/pyg\_datasets'  
dataset=FacebookPagePage(root=facebookPagePage\_fn)

查看数据集相关信息。

data=dataset[0]  
print(f'number of graphs:{len(dataset)}\  
 \nnumber of nodes:{data.x.shape[0]}\  
 \nnumber of features:{dataset.num\_features}\  
 \nnumber of classes:{dataset.num\_classes}\  
 \nedges are directed:{data.is\_directed()}\  
 \ngraph has isolated nodes:{data.has\_isolated\_nodes()}\  
 \ngraph has loops:{data.has\_self\_loops()}')

number of graphs:1   
number of nodes:22470   
number of features:128   
number of classes:4   
edges are directed:False   
graph has isolated nodes:False   
graph has loops:True

查看顶点属性特征向量，和对应的站点类别。

print(data.x.shape,'\n',data.x,'\n',data.y)

torch.Size([22470, 128])   
 tensor([[-0.2626, -0.2765, -0.2624, ..., -0.2151, -0.3759, -0.2238],  
 [-0.2626, -0.2765, -0.2624, ..., -0.2151, -0.3641, -0.1286],  
 [-0.2626, -0.2651, -0.2624, ..., -0.2151, -0.3759, -0.2238],  
 ...,  
 [-0.2626, -0.2765, -0.2624, ..., -0.1804, -0.3721, -0.2226],  
 [-0.2626, -0.2765, -0.2624, ..., -0.2151, -0.3759, -0.2181],  
 [-0.2323, -0.2765, -0.2624, ..., -0.1959, -0.3759, -0.2213]])   
 tensor([0, 2, 1, ..., 2, 1, 0])

采用 PyG 库对图数据处理的方法，配置训练、验证和测试数据集区间。

data.train\_mask=range(18000)  
data.val\_mask=range(18001, 20000)  
data.test\_mask=range(20001, 22470)

用PyG的to\_dense\_adj方法计算顶点邻接矩阵，并配置对角线值为 1，即包含顶点自身。

adjacency=to\_dense\_adj(data.edge\_index)[0]  
adjacency+=torch.eye(len(adjacency))  
print(adjacency.shape)  
adjacency

torch.Size([22470, 22470])  
  
  
  
  
  
tensor([[1., 0., 0., ..., 0., 0., 0.],  
 [0., 1., 0., ..., 0., 0., 0.],  
 [0., 0., 1., ..., 0., 0., 0.],  
 ...,  
 [0., 0., 0., ..., 1., 0., 0.],  
 [0., 0., 0., ..., 0., 1., 0.],  
 [0., 0., 0., ..., 0., 0., 1.]])

查看 vGNN 网络层，定义有两个线性层，执行完一个线性层计算后，计算与顶点邻接矩阵的点积（点乘），代码为x = torch.sparse.mm(adjacency, x)，然后经激活函数（ReLU）非线性变换后，执行第2个线性层，最后用 SoftMax 函数分类，使用 PyTorch库的torch.nn.functional.log\_softmax方法。

gnn\_vanilla=usda\_nx.VanillaGNN(dataset.num\_features, 16, dataset.num\_classes)  
gnn\_vanilla

VanillaGNN(  
 (gnn1): VanillaGNNLayer(  
 (linear): Linear(in\_features=128, out\_features=16, bias=False)  
 )  
 (gnn2): VanillaGNNLayer(  
 (linear): Linear(in\_features=16, out\_features=4, bias=False)  
 )  
)

训练模型。预测精度为预测分类正确数的百分比。

gnn\_vanilla.fit(data,adjacency,epochs=1000,verbose=200)

Epoch 0 | Train Loss: 1.309 | Train Acc: 79.72% | Val Loss: 1.25 | Val Acc: 79.24%  
Epoch 200 | Train Loss: 0.324 | Train Acc: 91.18% | Val Loss: 0.36 | Val Acc: 90.80%  
Epoch 400 | Train Loss: 0.294 | Train Acc: 92.40% | Val Loss: 0.34 | Val Acc: 90.70%  
Epoch 600 | Train Loss: 0.241 | Train Acc: 93.25% | Val Loss: 0.32 | Val Acc: 91.10%  
Epoch 800 | Train Loss: 0.204 | Train Acc: 93.88% | Val Loss: 0.31 | Val Acc: 91.30%  
Epoch 1000 | Train Loss: 0.574 | Train Acc: 89.84% | Val Loss: 0.62 | Val Acc: 89.59%

用测试数据集查看多类别分类预测精度，结果为 88.62%。

acc=gnn\_vanilla.test(data,adjacency)  
print(f'\nGNN test accuracy: {acc\*100:.2f}%')

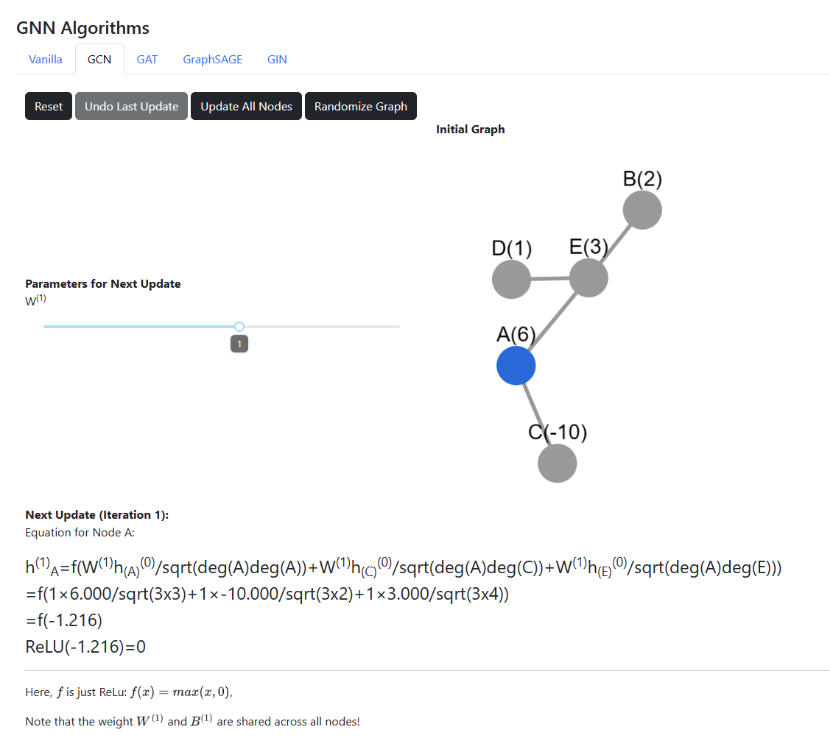
GNN test accuracy: 88.62%  
  
  
C:\Users\richie\anaconda3\envs\rl\Lib\site-packages\usda\network\\_gnn\_interpretation.py:325: FutureWarning:  
  
adjacency\_matrix will return a scipy.sparse array instead of a matrix in Networkx 3.0.  
  
C:\Users\richie\anaconda3\envs\rl\Lib\site-packages\scipy\sparse\\_index.py:143: SparseEfficiencyWarning:  
  
Changing the sparsity structure of a csr\_matrix is expensive. lil\_matrix is more efficient.

​

### 4.10.2.2 GCN（Graph Convolutional Networks）

基于卷积神经网络（Convolutional Neural Networks，CNNs），Thomas N. Kipf等人[21]提出了用于复杂网络的图卷积网络（Graph Convolutional Networks，GCNs）。不同于vGNN，仅是加权求和顶点与其邻接顶点的特征值，GCN考虑到当前顶点度，有hi​=deg(i)1​∑j∈Ni​​xj​Wτ，式中deg(i)为当前顶点度；对应矩阵的形式表示为H=D~−1A~XWT，式中， A~=A+I， A为邻接矩阵， I为单位矩阵，为包含当前顶点自身。D~−1=(D+I)−1，为顶点度倒数矩阵。Thomas N. Kipf等人注意到，具有更高顶点度，即具有更多邻接顶点的顶点的特征更容易传播。为了抵消这种影响，提出了H=D~−21​A~TD~−21​XWT，可以使得低顶点度的顶点分配有更高的权重。就单个顶点，GCN 的计算公式为hi​=∑j∈Ni​​deg(i)​deg(j)​1​xj​WT。

下图简单示例中，配置权重值（因此可以省略），对于顶点，其自身顶点度为3（有 2 个邻接顶点，并包含自身），其特征值为 6， 因此根据单个顶点的 GCN 公式有，6×3​×3​1​；对于A的邻接顶点E，其顶点度为4，特征值为 3，因此有3×13×43×3​×4​1​；同理，对于邻接顶点C有，−10×13×2−10×3​×2​1​。将上述各邻接顶点（含自身）结果求和后结果为 -1.216，经激活函数（ReLU）非线性变换后结果为0。



PyG集成有大量[卷积层（Convolutional Layers）](https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#convolutional-layers)⑧，其中GCNConv对应到Thomas N. Kipf等人提出的方法。使用GCNConv层定义 GCN，如下包含有两层GCNConv。同样使用 Facebook 图数据集进行试验。

gcn=usda\_nx.GCN(dataset.num\_features, 16, dataset.num\_classes)  
gcn

GCN(  
 (gcn1): GCNConv(128, 16)  
 (gcn2): GCNConv(16, 4)  
)

可以发现相对于 vGNN 模型，GCN 具有更快的收敛速度和相对较高的预测精度。

gcn.fit(data, epochs=1000,verbose=200)

Epoch 0 | Train Loss: 0.236 | Train Acc: 93.09% | Val Loss: 0.24 | Val Acc: 92.35%  
Epoch 200 | Train Loss: 0.153 | Train Acc: 95.67% | Val Loss: 0.20 | Val Acc: 93.75%  
Epoch 400 | Train Loss: 0.144 | Train Acc: 96.01% | Val Loss: 0.20 | Val Acc: 93.90%  
Epoch 600 | Train Loss: 0.141 | Train Acc: 96.04% | Val Loss: 0.19 | Val Acc: 94.00%  
Epoch 800 | Train Loss: 0.139 | Train Acc: 96.12% | Val Loss: 0.19 | Val Acc: 93.90%  
Epoch 1000 | Train Loss: 0.138 | Train Acc: 96.13% | Val Loss: 0.19 | Val Acc: 93.90%

用测试数据集查看多类别分类预测精度，结果为 93.16%。

acc=gcn.test(data)  
print(f'\nGCN test accuracy: {acc\*100:.2f}%\n')

GCN test accuracy: 93.16%

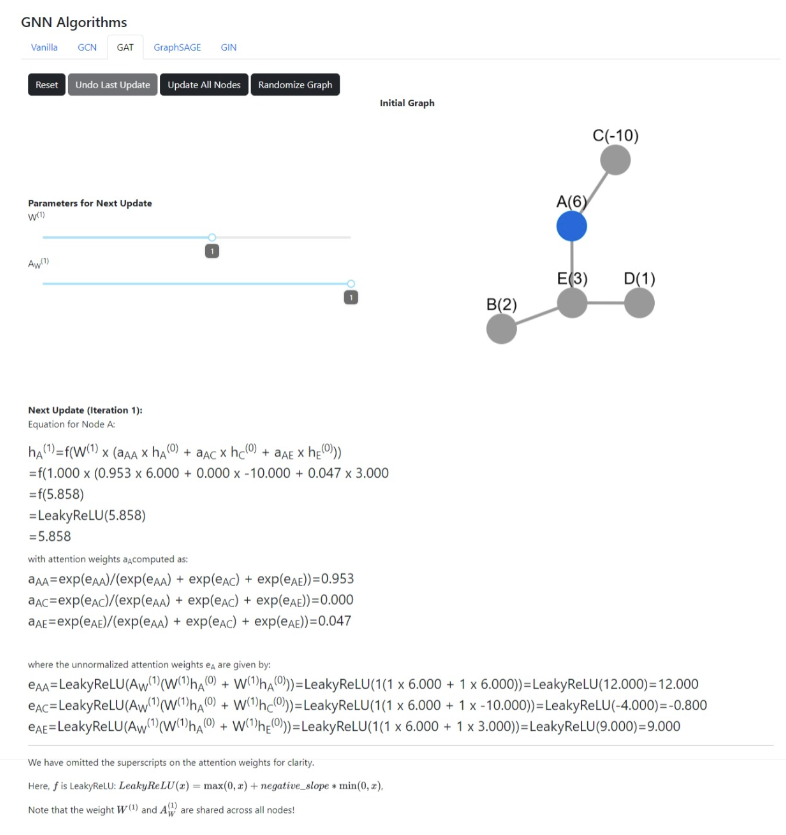
​

### 4.10.2.3 GAT（Graph Attention Networks）

在*从 RNN 到 Transformer 和 GPT，从自然语言处理到视觉模型*部分解释了 transformer 网络中的注意力机制，将该机制用于 GNN 建立 GAT（Graph Attention Networks）。相较于 GCN 考虑顶点度以区分邻接顶点少的顶点比其它顶点更重要，GAT 则引入了考虑顶点特征重要性的加权因子，为顶点间的注意力得分（attention score），对于顶点间的注意力得分，可记为，从而有，。该公式对应到下图顶点示例的计算部分，其中，分别为顶点到邻接顶点的注意力得分（含自身）。

注意力得分的计算包括线性变换（Linear transformation），应用激活函数（Activation function）和 Softmax 标准化（ Softmax normalization）。组合线性变换和激活函数可表示为，eij​=LeakyReLU(Wattτ​[Whi​∥Whj​])，对应到下图示例中eAA​,eAC​,eAE​的计算部分，式中， W为共享的权重矩阵； Watt​对应图中示例的AW​权重矩阵。对线性变换结果执行 LeakyReLU 激活函数，对 LeakyReLU 的解释可以查看*判别器（D）的层结构*部分，其公式表述为LeakyReLU(x)=max(0,x)+ negative\_slope  ∗min(0,x)，或 LeakyReLU  otherwise  LeakyReLU (x)={x, negative\_slope ×x,​ if x≥0 otherwise ​。LeakyReLU 的计算直接调用PyTorch库的torch.nn.LeakyReLU(negative\_slope=0.01, inplace=False)方法，在下图示例计算时，配置参数negative\_slope=0.2。

*如果要比较不同的注意力得分，则需要将其标准化，使用 Softmax 标准化方法，公式为*αij*​=softmax*j*​(*eij*​)=∑*k*N*i*​​exp(*eik*​)exp(*eij*​)​，对应图中*aAA*​,*aAC*​,*aAE*​部分的计算。实际计算时调用torch.nn.Softmax(dim=None)方法。*

**

由于自注意力（self-attention）可能并不稳定，因此 Vaswani, A. 等人[22]提出了多头注意力（Multi-head attention ），在 GAT 中则是将线性变换，应用激活函数和 Softmax 标准化过程重复多次（n）计算，而后取均值或只是连接（Concatenation），对于取均值可表示为， hi​=n1​∑k=1n​hik​=n1​∑k=1n​∑j∈Ni​​αijk​Wkxj​；对于连接可表示为， hi​=∥k=1n​hik​=∥k=1n​∑j∈Ni​​αijk​Wkxj​。实践中，为隐藏层时选择连接方式，当位于网络最后一层时选择均值方式。

Shaked Brody 等人[23]发现 GAT 计算的是一种非常有限的注意力：注意力得分排名在查询的顶点上是无条件的。为与更具表现力的动态注意力（dynamic attention）区分，将这种受限的注意力定义为静态注意力（static attention）。使用静态注意力机制的 GAT 不能表示简单图问题（ simple graph problems），在在受控问题（controlled problem）中甚至会阻碍训练数据拟合。为了消除静态注意力的限制，Shaked Brody 等人通过调整 GAT 的计算顺序提出了 GATv2。对 GAT 中计算eij​的公式进行调整，有eij​=Wattτ​LeakyReLU(W[hi​∥hj​])。PyG 库对则应提供了GATConv和GATv2Conv方法。

继续使用 Facebook 图数据集进行试验，计算过程和结果如下。

gat=usda\_nx.GATv2(dataset.num\_features, 32, dataset.num\_classes)  
gat

GATv2(  
 (gat1): GATv2Conv(128, 32, heads=8)  
 (gat2): GATv2Conv(256, 4, heads=1)  
)

gat.fit(data,epochs=1000,verbose=200)

Epoch 0 | Train Loss: 2.047 | Train Acc: 22.72% | Val Loss: 2.02 | Val Acc: 22.56%  
Epoch 200 | Train Loss: 0.331 | Train Acc: 89.11% | Val Loss: 0.33 | Val Acc: 89.39%  
Epoch 400 | Train Loss: 0.323 | Train Acc: 88.94% | Val Loss: 0.31 | Val Acc: 89.19%  
Epoch 600 | Train Loss: 0.339 | Train Acc: 88.44% | Val Loss: 0.34 | Val Acc: 88.29%  
Epoch 800 | Train Loss: 0.332 | Train Acc: 88.81% | Val Loss: 0.33 | Val Acc: 89.34%  
Epoch 1000 | Train Loss: 0.330 | Train Acc: 88.50% | Val Loss: 0.32 | Val Acc: 89.29%

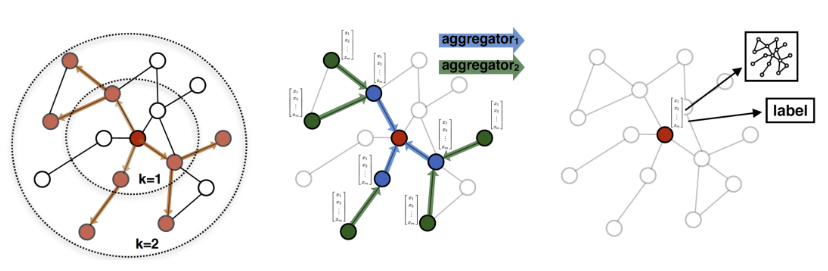
用测试数据集查看多类别分类预测精度，结果为 91.17%。

acc=gat.test(data)  
print(f'GAT test accuracy: {acc\*100:.2f}%')

GAT test accuracy: 91.17%

### 4.10.2.4 GraphSAGE

大型图中顶点的低维嵌入（low-dimensional embeddings）已经被证明在各种预测任务中非常有用，为了避免训练嵌入时包含图中所有的顶点（称为直推式，（transductive）），Hamilton, W.等人[24]提出了 GraphSAGE 算法，一个通用的归纳式（inductive）框架，利用顶点特征信息，从顶点的局部邻域中采样和聚合特征学习一个函数，不是为每个顶点训练一个不同的嵌入向量，而是训练一组聚合器函数（aggregator functions），从而有效的为之前未见过的数据生成顶点嵌入。其具体的过程如下图示[24]：



上图中，左图为邻域采样；中图为从邻域采样中聚合特征信息；右图为使用聚合的信息预测图的上下文（context）和类标。

* 邻域采样（Neighbor sampling）

聚合器函数从远离给定顶点不同跳数（hops）或搜索深度聚合信息。从上图左中可以观察 1 跳为顶点的邻接顶点，2 跳为邻接顶点的邻接顶点，以此类推。同时，为了避免对应跳数顶点数的指数级增长，采用邻域采样的方式降低顶点数。

* 聚合器架构（Aggregator Architectures）

聚合器函数作用于没有自然顺序的邻接顶点。理想情况，聚合器函数应是对称（对其输入的排列不变）且是可训练的，保持有较高的表示能力，从而保证神经网络模型可以被训练并应用于任意有序的顶点邻域特征集。论文中作者研究了三种聚合器函数，为均值、LSTM 和池化（Pooling）。

均值算子是计算邻域采样{huk−1​,∀u∈N(v)}中顶点向量的均值，基本等同于 GCN 的卷积传播规则，用公式表示为hvk​←σ(W⋅MEAN({hvk−1​}∪{huk−1​,∀u∈N(v)})；LSTM 聚合器基于 LSTM 架构（可以参考*从 RNN 到 Transformer 和 GPT，从自然语言处理到视觉模型*章节对 LSTM 的阐述），与均值聚合器相比，LSTM 聚合器理论上可以区分更多的图结构，从而生成更好的嵌入。但 LSTM 输入为一个序列，而顶点没有任何顺序，因此需要对顶点进行随机排序，例如在 PyG 中使用data.sort(sort\_by\_row=False)方法按照边索引的目标顶点排序等；池化聚合器，将邻域采样每个顶点的向量喂入全连接神经网络，而后用最大池化（max-pooling）聚合信息，公式表达为AGGREGATEkpool ​=max({σ(Wpool ​hui​k​+b),∀ui​∈N(v)})，式中， max表示取最大值，σ为非线性激活函数。原则上，在最大池化之前应用的函数可以是任意深度的多层感知器。

下图单个顶点的 GraphSAGE 更新演示使用 LSTM 聚合器函数。为了方便演示，配置 LSTM 的所有参数均为值1。通过按照边索引（edge\_index）的目标顶点排序，得到如下边索引结果data.edge\_index。按照边索引，使用PyG的to\_dense\_batch方法，按照排序后的边索引，可以得到图中每个顶点邻接顶点向量x\_dense。如果不足最大邻接顶点数，则用值0占位，例如对于顶点，其邻接顶点为，有[[-10.],[2.],[0.]]。

x=torch.tensor([[6],[2],[-10],[1],[3]]).to(torch.float)  
edge\_index=torch.tensor([[0, 0, 1, 2, 3, 4, 4, 4],[2, 4, 4, 0, 4, 0, 1, 3]])  
data=Data(x=x, edge\_index=edge\_index)  
  
data=data.sort(sort\_by\_row=False)  
data.edge\_index

tensor([[2, 4, 4, 0, 4, 0, 1, 3],  
 [0, 0, 1, 2, 3, 4, 4, 4]])

x\_=torch.tensor([data.x[i] for i in data.edge\_index[0]]).reshape(-1,1)  
x\_dense,\_=to\_dense\_batch(x\_,data.edge\_index[1])  
x\_dense

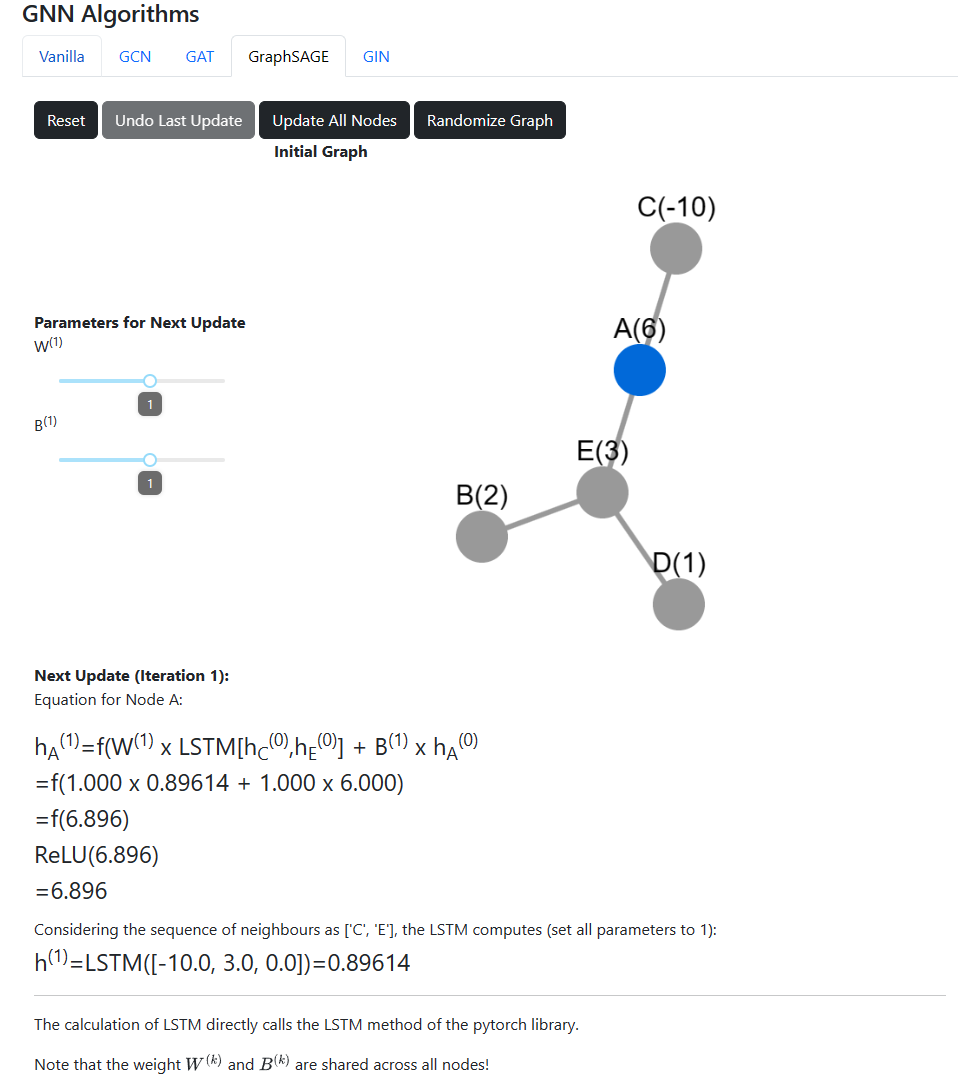
tensor([[[-10.],  
 [ 3.],  
 [ 0.]],  
  
 [[ 3.],  
 [ 0.],  
 [ 0.]],  
  
 [[ 6.],  
 [ 0.],  
 [ 0.]],  
  
 [[ 3.],  
 [ 0.],  
 [ 0.]],  
  
 [[ 6.],  
 [ 2.],  
 [ 1.]]])

用 LSTM 模型计算按边索引目标顶点排序的邻接顶点向量结果如下。

lstm=LSTM(1, 1, batch\_first=True)  
for para in list(lstm.parameters()):   
 para.data.fill\_(1.0)  
  
lstm\_out=lstm(x\_dense)[0][:,-1]   
lstm\_out

tensor([[0.8961],  
 [0.9393],  
 [0.9395],  
 [0.9393],  
 [0.9756]], grad\_fn=<SelectBackward0>)

完成聚合器函数（LSTM）计算后，对应演示后续计算公式表示为hi′​=σ(Whi​+B⋅LSTMj∈Ni​​(hj​))。



直推式只能生成固定图的嵌入，不能泛化到不可见的顶点或图。而基于邻域采样，GraphSAGE 被设计成在局部使用修剪后的计算图进行预测的归纳式架构，从而可用于具有相同特征模式的任何计算图。使用PyG提供的蛋白质间相互作用网络图数据集PPI[25]，包含定位基因（positional gene）集，基序基因（motif gene）集，作为顶点特征的免疫特征（immunological signatures）数据（总共有 50 个），并将基因本体集（gene ontology）作为标签（总共 121 个）。直接迁移 *Hands-On Graph Neural Networks Using Python*[18]137示例代码计算过程和结果如下。

查看 GPU。

print(torch.\_\_version\_\_)  
device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')  
device

2.1.0+cu118  
  
  
  
  
  
device(type='cuda')

下载数据至本地磁盘。

# Load training, evaluation, and test sets  
root='../data/pyg\_datasets/PPI'  
train\_dataset= PPI(root=root, split='train')  
val\_dataset = PPI(root=root, split='val')  
test\_dataset = PPI(root=root, split='test')  
  
print(len(train\_dataset))  
train\_dataset[0]

20  
  
  
  
  
  
Data(x=[1767, 50], edge\_index=[2, 32318], y=[1767, 121])

训练数据集包括 20 个图，将这些所有图通过Batch.from\_data\_list方法将其描述为一个不连接的大图（统一成一个集合），并建立用于训练的数据加载器。

# Unify the training graphs and apply neighbor sampling  
train\_data = Batch.from\_data\_list(train\_dataset)  
train\_loader = NeighborLoader(train\_data, batch\_size=2048, shuffle=True, num\_neighbors=[20, 10], num\_workers=2, persistent\_workers=True)  
  
# Evaluation loaders (one datapoint corresponds to a graph)  
val\_loader = DataLoader(val\_dataset, batch\_size=2)  
test\_loader = DataLoader(test\_dataset, batch\_size=2)  
  
train\_data

DataBatch(x=[44906, 50], edge\_index=[2, 1226368], y=[44906, 121], batch=[44906], ptr=[21])

初始化 GraphSAGE 模型，损失和优化器。

model = GraphSAGE(  
 in\_channels=train\_dataset.num\_features,  
 hidden\_channels=512,  
 num\_layers=2,  
 out\_channels=train\_dataset.num\_classes,  
).to(device)  
  
criterion = torch.nn.BCEWithLogitsLoss()  
optimizer = torch.optim.Adam(model.parameters(), lr=0.005)  
  
model

GraphSAGE(50, 121, num\_layers=2)

定义模型训练函数。

def fit(loader):  
 model.train()  
  
 total\_loss = 0  
 for data in loader:  
 data = data.to(device)  
 optimizer.zero\_grad()  
 out = model(data.x, data.edge\_index)  
 loss = criterion(out, data.y)  
 total\_loss += loss.item() \* data.num\_graphs  
 loss.backward()  
 optimizer.step()  
 return total\_loss / len(loader.data)

定义精度计算函数，使用对应于准确率和召回率的调和平均值f1\_score（f1-分数）。

@torch.no\_grad()  
def test(loader):  
 model.eval()  
  
 data = next(iter(loader))  
 out = model(data.x.to(device), data.edge\_index.to(device))  
 preds = (out > 0).float().cpu()  
  
 y, pred = data.y.numpy(), preds.numpy()  
 return f1\_score(y, pred, average='micro') if pred.sum() > 0 else 0

训练模型。

for epoch in range(1001):  
 loss = fit(train\_loader)  
 val\_f1 = test(val\_loader)  
 if epoch % 200 == 0:  
 print(f'Epoch {epoch:>3} | Train Loss: {loss:.3f} | Val F1-score: {val\_f1:.4f}')  
  
print(f'Test F1-score: {test(test\_loader):.4f}')

Epoch 0 | Train Loss: 12.720 | Val F1-score: 0.4894  
Epoch 200 | Train Loss: 8.501 | Val F1-score: 0.8194  
Epoch 400 | Train Loss: 8.431 | Val F1-score: 0.8319  
Epoch 600 | Train Loss: 8.398 | Val F1-score: 0.8317  
Epoch 800 | Train Loss: 8.362 | Val F1-score: 0.8373  
Epoch 1000 | Train Loss: 8.368 | Val F1-score: 0.8341  
Test F1-score: 0.8570

查看已训练模型的预测精度。

print(f'Test F1 score: {test(test\_loader):.4f}')

Test F1 score: 0.8570

### 4.10.2.5 GIN（Graph Isomorphism Network）

#### 1）同构图的 Weisfeiler-Lehman（WL） 检验[26]

基于 Weisfeiler and Lehman, A.A. 等人[27]的 Weisfeiler-Lehman（WL）同构图检验的 1-维变体（1-dimensional variant），即朴素顶点求精（naive vertex reﬁnement），Shervashidze, N.等人[26]在图中迭代 1-维 WL检验，测试给定的图和是否同构，算法如下：

下：

**Algorithm 1**One iteration of the 1-dim. Weisfeiler-Lehman test of graph isomorphism

**1: Multiset-label determination**

• For *i*=0, set .*Mi*​(*v*):=*l*0​(*v*)=ℓ(*v*).

• For *i*>0, assign a multiset-label *Mi*​(*v*) to each node *v* in *G* and′*G*′ which consists of the multiset{*li*−1​(*u*)∣*u*∈N(*v*)}.

**2: Sorting each multiset**

• Sort elements in *Mi*​(*v*) in ascending order and concatenate them into a string *si*​(*v*).

• Add*li*−1​(*v*) as a prefix to *si*​(*v*) and call the resulting string *si*​(*v*).

**3: Label compression**

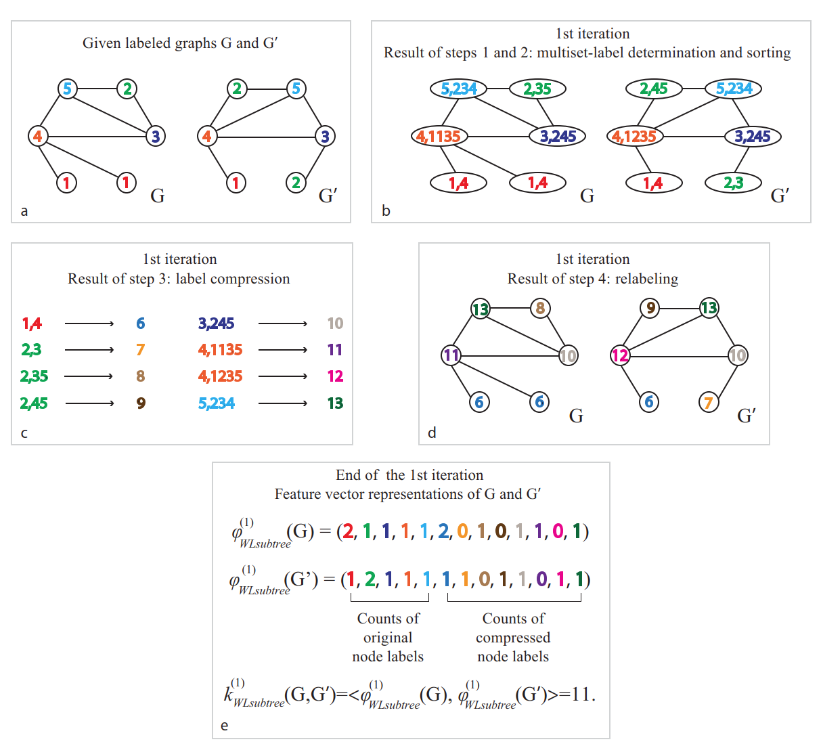
• Sort all of the strings *si*​(*v*) for all *v* from *G* and ′*G*′ in ascending order.

• Map each string *si*​(*v*) to a new compressed label, using a function  *f*:Σ∗→Σ such that *f*(*si*​(*v*))=*f*(*si*​(*w*)) if and only if *si*​(*v*)=*si*​(*w*).

**4: Relabeling**

• Set *li*​(*v*):=*f*(*si*​(*v*)) for all nodes in *G* and ′*G*′.

对应上述算法的图解如下[26]，



该算法的核心思想是将排序的邻接顶点标签增加到原顶点标签上，并将这些扩充的顶点标签压缩为短的标签，重复这个过程直至图的顶点标签集不同，或者迭代次数达到。例如图中h=1的 WL 子树核（subtree kernel）计算中，将{1,2,…,13}∈Σ视为字母，如果一个顶点具有标签 8，则意味着该顶点有一个高度为1的子树模式（subtree pattern）， 多集标签（Multiset-label）为2,35，对应根标签为 2，邻域标签为3和5。

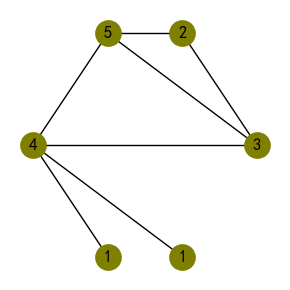
步骤 3（c） 中对多集进行基数排序（radix sort），并定义（如哈希函数（Hash function））把多集映射（压缩）到一组新的标签。多集排序保证了所有相同的字符串都映射到相同的新标签（数字）上。其中Σ必须足够大，确保f是内射的。在多次迭代中（图示例中仅需迭代一次），如果新创建的标签集在图G和′G′中不相同，说明二者不同构。如果n次迭代后，集合相同，则意味着图G和′G′同构或算法无法确定它们不是同构的。

步骤 4（e）中，计数了 WL 子树核的公共原始标签和映射标签。

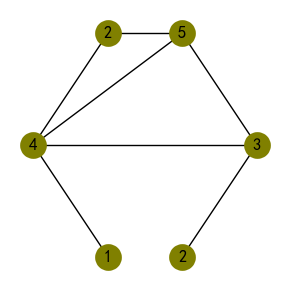
NetworkX库提供了weisfeiler\_lehman\_graph\_hash(G, edge\_attr=None, node\_attr=None, iterations=3, digest\_size=16)方法返回 WL 图哈希值。该函数迭代的对每个顶点的邻域进行聚合和散列（哈希），以更新顶点标签。返回值为输入图的哈希值对应的十六进制字符串。例如下述代码构建了上述示例图中的图（G1）和（G2），并计算了图的 WL 哈希值，结果不同，表明图和不同构。

G1=nx.Graph()  
G1.add\_edges\_from([(4,0),(4,1),(4,5),(4,3),(2,5),(2,3),(5,3)])  
nx.set\_node\_attributes(G1,{0:1,1:1,2:2,3:3,4:4,5:5},name='labels')  
  
G2=nx.Graph()  
G2.add\_edges\_from([(4,2),(4,1),(4,5),(4,3),(2,5),(2,3),(0,3)])  
nx.set\_node\_attributes(G2,{0:2,1:1,2:5,3:3,4:4,5:2},name='labels')  
  
pos={0:[2,0],1:[1,0],2:[2,2],3:[3,1],4:[0,1],5:[1,2]}

usda\_nx.G\_drawing(G1,node\_labels='labels',pos=pos)



usda\_nx.G\_drawing(G2,node\_labels='labels',pos=pos)



WL\_G1=nx.weisfeiler\_lehman\_graph\_hash(G1,node\_attr='labels',iterations=1)  
WL\_G2=nx.weisfeiler\_lehman\_graph\_hash(G2,node\_attr='labels',iterations=1)  
print(WL\_G1,WL\_G2)

965633870406c8813aa3c2daad107bbe 466a349290fb7ad16aa95c70014ab524

NetworkX库也提供了weisfeiler\_lehman\_subgraph\_hashes(G, edge\_attr=None, node\_attr=None, iterations=3, digest\_size=16)方法返回顶点为键子图哈希值的字典。从计算结果可以发现，当仅迭代1次时，即对应顶点列表的第1个值，图和的顶点3、1对应相同等，而图中的顶点5和图中的顶点2相同。

g1\_hashes=nx.weisfeiler\_lehman\_subgraph\_hashes(G1, iterations=2, digest\_size=8,node\_attr='labels')  
g2\_hashes=nx.weisfeiler\_lehman\_subgraph\_hashes(G2, iterations=2, digest\_size=8,node\_attr='labels')  
print(g1\_hashes)  
print(g2\_hashes)

{4: ['bb4d352c7c4fcc44', '6274bc44044b5a3f'], 0: ['5110adc459f44b27', 'e5e3a395c0f7e75d'], 1: ['5110adc459f44b27', 'e5e3a395c0f7e75d'], 5: ['6d4d91e01b12ba52', '5cdde0408b77fb12'], 3: ['ff34a88346b6b71f', '1380842ff2943f32'], 2: ['7d5a4600089dcd1b', '6117058e1c02d97f']}  
{4: ['15f9a158b7b4c4da', '01830c815cd29704'], 2: ['6d4d91e01b12ba52', '96603cec7f6af193'], 1: ['5110adc459f44b27', '5594b95736b6436f'], 5: ['48c13d17f6c77427', 'aec9546ac5c31b74'], 3: ['ff34a88346b6b71f', 'fcd5055cf24b328b'], 0: ['26808910ed0da6e0', '8a99dfce7ad1fff5']}

#### 2）Graph Isomorphism Network，GIN[28]

设图G=(V,E)，其顶点特征向量为Xv​(v∈V)。对于图顶点分类，其中每个顶点v∈V都有一个相对应的分类标签yv​，目标是学习顶点v的表征向量（representation vector）hv​，使得顶点v的分类标签可以由yv​=f(hv​)预测；对于图分类，给定一组图{G1​,…,GN​}⊆G和对应的分类标签{y1​,…,yN​}⊆Y，目标是学习一个表征向量hG​，可以用其预测整个图的分类，有yG​=g(hG​)。

GNNs 使用图结构和顶点特征学习顶点或整个图的表征向量或，遵循邻域聚合策略，通过聚合邻域顶点表征信息来迭代更新顶点的表征。经过k次聚合迭代，顶点的表征捕获到了其k-hop（k-跳）网络邻域的结构信息，第k层（迭代）的 GNN 可以表示为， av(k)​=AGGREGATE(k)({hu(k−1)​:u∈N(v)}),hv(k)​=COMBINE(k)(hv(k−1)​,av(k)​)，式中， hv(k)​是第k次迭代，顶点v的特征向量，并初始化hv(0)​=Xv​，而N(v)是顶点v的邻域顶点集合。AGGREGATE(k)(⋅) and COMBINE(k)(⋅)的选择是关键，GraphSAGE 就提到均值、LSTM 和池化（Pooling）等聚合器函数。

对于顶点分类，使用最后一次迭代顶点的表征hv(k)​进行预测；对于图分类， READOUT函数聚合最后一次迭代的顶点特征，得到整个图的表征hG​=READOUT({hv(K)​∣v∈G})。READOUT可以是一个简单的置换不变函数（permutation invariant function）（特征之间没有空间位置关系），例如求和函数或者为一个更复杂的图级池化函数（graph-level pooling function ）。

理想情况下，最强大的 GNN 可以将不同的图结构映射到嵌入空间（ embedding space）中的不同表征（ representations）来区分不同的图结构。然而，这种将任意两个不同图映射到不同嵌入的能力，意味着要解决具有挑战性的图同构问题，即希望同构图映射到相同的表征，而非同构图映射到不同的表征。同构图的 Weisfeiler-Lehman（WL） 检验为其提供了一种途径。

引理（Lemma）1，G1​和G2​为非同构的图，如果一个图神经网络A:G→Rd映射图G1​和G2​为不同的嵌入时，WL 图同构检验也判定图G1​和G2​不同构。

定理（Theorem）2， 设A:G→Rd为 GNN，迭代足够次数，如果满足如下条件， A将 WL 图同构检验判定为非同构的任意图G1​和G2​映射到不同的嵌入，

a） A用hv(k)​=ϕ(hv(k−1)​,f({hu(k−1)​:u∈N(v)}))迭代的聚合和更新顶点特征，式中，函数f作用于多集， ϕ为单射；

b）A图级readout函数，作于于顶点特征{hv(k)​}的多集，为单射。

Keyulu Xu 等人[28]开发的 GIN 简单架构的算法模型证明满足定理 2 中的条件，推广了 WL 检验，实现了 GNNs 的最大判别能力。为了对邻域聚合建模单射多集函数（ injective multiset functions），发展了“深度多集”（“deep multisets”）的理论，即用神经网络参数化通用多集函数。下一引理（3）陈述了求和聚合器是单射的，为多集上的通用函数（universal functions）。

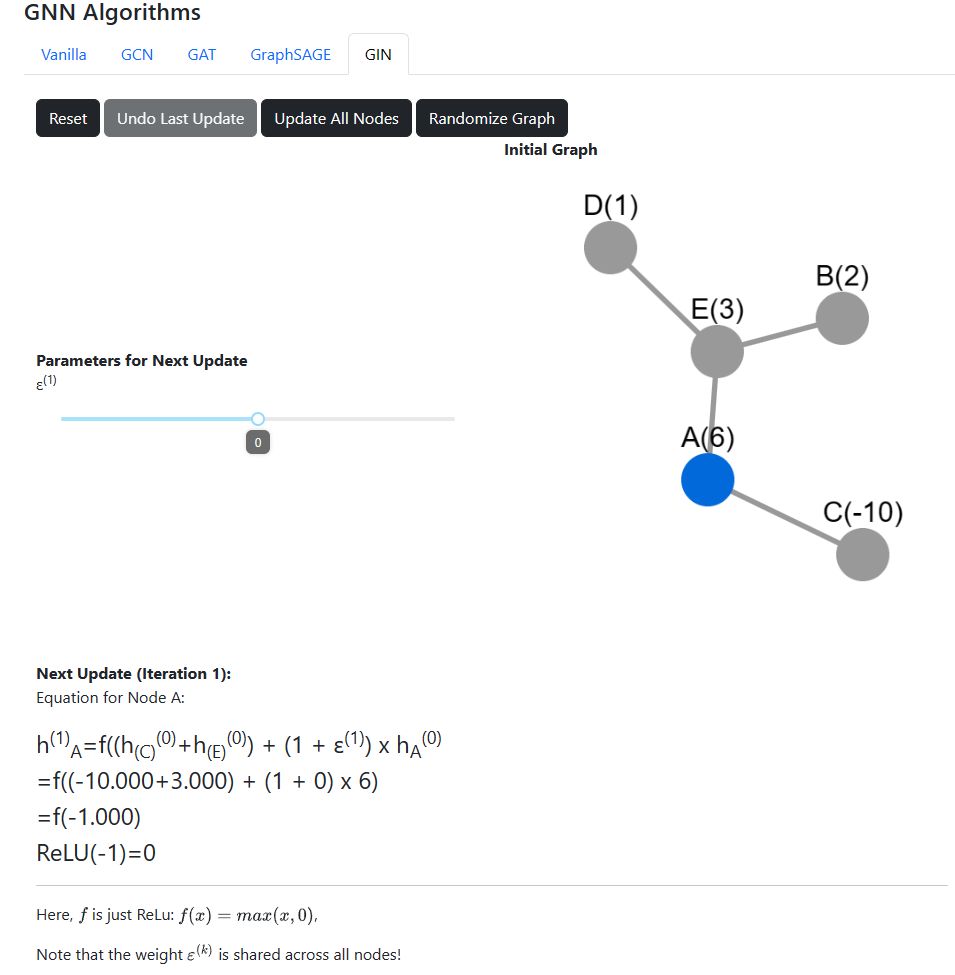
引理 3，假设X是可计算的（countable），存在一个函数f:X→Rn，使得h(X)=∑x∈X​f(x)对于每一个有界大小的多集X⊂X是唯一的。对于函数ϕ，任意多集函数g都可以分解为g(X)=ϕ(∑x∈X​f(x))。

推论（4）在诸多的聚合方案中提供了一个简单而具体的公式。

推论（Corollary）4，假设X是可计算的，存在一个函数f:X→Rn，使得对于ϵ无限多个选择，包括所有无理数， h(c,X)=(1+ϵ)⋅f(c)+∑x∈X​f(x)对每一对 (c,X)都是唯一的，式中c∈X和X⊂X是一个有界大小的多集。此外对于一些函数φ，任何函数g都可以分解为g(c,X)=φ((1+ϵ)⋅f(c)+∑x∈X​f(x))。

可以使用多层感知机（MLPs）学习推论（4）中的f和φ。因为 MLPs 可表示函数的组合，因此用 MLPs 对f(k+1)∘φ(k)建模。在第1次迭代中，输入的特征如果为独热编码，它们的和为单射，因此在求和之前不需要 MLPs。ϵ可以作为一个可学习的参数，也可以为固定的标量。定义 GIN 顶点表征的更新公式为， hv(k)​=MLP(k)((1+ϵ(k))⋅hv(k−1)​+∑u∈N(v)​hu(k−1)​)。

下述简单的演示案例应用了 GIN 顶点更新的上述公式，例如对于顶点A，聚合邻接顶点C和E，求和为−10+3=−7，进而与顶点当前表征hA​与 (1+ϵ)之积求和得到−7+(1+0)∗6=−1，经过 ReLU 激活函数的非线性变换得到结果值为0。



#### 3）GIN 图分类

PyG库的TUDataset提供了收集于多特蒙德工业大学（TU Dortmund University）多种多样的图核（graph kernel）基准数据集，使用其中的PROTEINS（蛋白质）数据集，应用 GIN 网络执行图分类任务。

首先下载数据集并打印相关信息。

root='../data/pyg\_datasets/TUDataset'  
dataset = TUDataset(root=root, name='PROTEINS').shuffle()

# Print information about the dataset  
print(f'Dataset: {dataset}')  
print('-----------------------')  
print(f'Number of graphs: {len(dataset)}')  
print(f'Number of nodes: {dataset[0].x.shape[0]}')  
print(f'Number of features: {dataset.num\_features}')  
print(f'Number of classes: {dataset.num\_classes}')

Dataset: PROTEINS(1113)  
-----------------------  
Number of graphs: 1113  
Number of nodes: 31  
Number of features: 3  
Number of classes: 2

切分数据集为训练、验证和测试数据集。

# Create training, validation, and test sets  
train\_dataset = dataset[:int(len(dataset)\*0.8)]  
val\_dataset = dataset[int(len(dataset)\*0.8):int(len(dataset)\*0.9)]  
test\_dataset = dataset[int(len(dataset)\*0.9):]  
# Create mini-batches  
train\_loader = DataLoader(train\_dataset, batch\_size=64, shuffle=True)  
val\_loader = DataLoader(val\_dataset, batch\_size=64, shuffle=True)  
test\_loader = DataLoader(test\_dataset, batch\_size=64, shuffle=True)

初始化 GIN 网络，该网络使用PyG库的GINConv方法构建。

gin = usda\_nx.GIN(dataset.num\_node\_features,dataset.num\_classes,dim\_h=32)  
gin

GIN(  
 (conv1): GINConv(nn=Sequential(  
 (0): Linear(in\_features=3, out\_features=32, bias=True)  
 (1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (2): ReLU()  
 (3): Linear(in\_features=32, out\_features=32, bias=True)  
 (4): ReLU()  
 ))  
 (conv2): GINConv(nn=Sequential(  
 (0): Linear(in\_features=32, out\_features=32, bias=True)  
 (1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (2): ReLU()  
 (3): Linear(in\_features=32, out\_features=32, bias=True)  
 (4): ReLU()  
 ))  
 (conv3): GINConv(nn=Sequential(  
 (0): Linear(in\_features=32, out\_features=32, bias=True)  
 (1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)  
 (2): ReLU()  
 (3): Linear(in\_features=32, out\_features=32, bias=True)  
 (4): ReLU()  
 ))  
 (lin1): Linear(in\_features=96, out\_features=96, bias=True)  
 (lin2): Linear(in\_features=96, out\_features=2, bias=True)  
)

训练网络模型。

gin = usda\_nx.gin\_train(gin, train\_loader,val\_loader,epochs=200,verbose=40)

Epoch 0 | Train Loss: 1.26 | Train Acc: 72.05% | Val Loss: 0.44 | Val Acc: 77.36%  
Epoch 40 | Train Loss: 0.45 | Train Acc: 78.74% | Val Loss: 0.41 | Val Acc: 84.24%  
Epoch 80 | Train Loss: 0.41 | Train Acc: 80.36% | Val Loss: 0.60 | Val Acc: 76.86%  
Epoch 120 | Train Loss: 0.39 | Train Acc: 81.26% | Val Loss: 0.56 | Val Acc: 78.42%  
Epoch 160 | Train Loss: 0.39 | Train Acc: 81.20% | Val Loss: 0.53 | Val Acc: 78.99%  
Epoch 200 | Train Loss: 0.37 | Train Acc: 82.40% | Val Loss: 0.53 | Val Acc: 78.99%

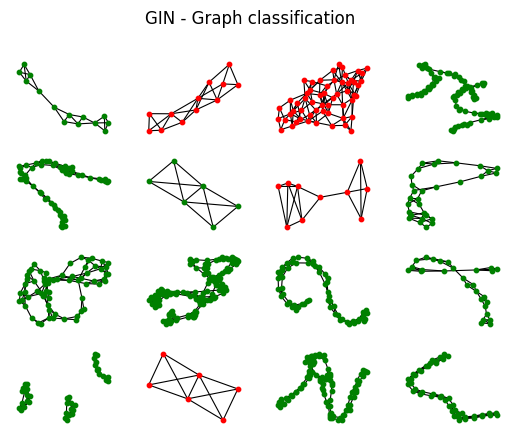
在测试数据集上计算精度。

test\_loss, test\_acc = usda\_nx.gin\_test(gin, test\_loader)  
print(f'Test Loss: {test\_loss:.2f} | Test Acc: {test\_acc\*100:.2f}%')

Test Loss: 0.64 | Test Acc: 77.34%

由训练的模型预测分类并打印结果（总共 2 类，红绿颜色区分）。

\_=usda\_nx.gin\_prediction\_plot(gin,dataset[-16:],figsize=(4,4))



## 4.10.3 VGAE（Variational Graph Auto-Encoder）连接（边）预测（Link prediction） 和植物调查样方物种间的关联

VGAE 和 GAE 均由Kipf, T. N. 和 Welling, M.[29]提出。变分图自编码器（ Variational Graph Auto-Encoders​，VGAE）是一种基于变分自编码器（Variational Auto-Encoder ，VAE）的一种用于图结构数据无监督学习框架，可利用潜在变量（latent variables）学习无向图可解释的潜在表征。

### 4.10.3.1 变分自编码器（Variational Auto-Encoder ，VAE）

变分自编码器（Variational Auto-Encoder ，VAE）是由Kingma, D. P. 和 Welling, M. [30]提出的一种神经网络结构，为概率生成模型（probabilistic generative models），是概率图模型（ probabilistic graphical models）和变分贝叶斯方法（variational Bayesian methods）家族的一部分。VAE 包括编码器（encoder）和解码器（decoder），编码器将特征（输入）向量映射到潜在空间（latent space），可以产生来自同一分布的多个不同样本。而解码器为编码器的逆过程，从潜在空间映射回输入空间，以生成数据点（data points）。

将神经网络用于概率编码器表示为 qϕ​(z∣x)，为生成模型pθ​(x,z)后验pθ​(z∣x)的近似。设潜在变量的先验为中心各向同性多元高斯（centered isotropic multivariate Gaussian）分布（多元正态分布），pθ​(z)=N(z;0,I)。注意到先验缺少参数，设pθ​(x∣z)为多元高斯，其分布参数用 MLP（具有单个隐藏层的全连接神经网络）从潜在空间z中计算 。假设真实的后验具有近似对角协方差的近似高斯形式， logqϕ​(z∣x(i))=logN(z;μ(i),σ2(i)I)，式中，μ(i),σ(i)为近似后验的均值和标准差，对应到 MLP 的输出。x(i)为数据点的非线性函数。ϕ为变分参数。

从下述试验的代码中可以观察到试验中的编码器网络结构被设计为，

(img\_2hid): Linear(in\_features=784, out\_features=200, bias=True)  
(hid\_2mu): Linear(in\_features=200, out\_features=20, bias=True)  
(hid\_2sigma): Linear(in\_features=200, out\_features=20, bias=True)

其中hid\_2mu层输出为均值，而hid\_2sigm层输出为标准差。

用z(i,l)=gϕ​(x(i),ϵ(l))=μ(i)+σ(i)⊙ϵ(l) where ϵ(l)∼N(0,I)公式从近似后验z(i,l)∼qϕ​(z∣x(i))中采样，式中⊙表示元素积（element-wise product）。对应下述代码书写为，

用表示元素积（element-wise product）。对应下述代码书写为，

def forward(self, x):  
 mu, sigma = self.encode(x)  
  
 # Sample from latent distribution from encoder  
 epsilon = torch.randn\_like(sigma)  
 z\_reparametrized = mu + sigma\*epsilon  
  
 x = self.decode(z\_reparametrized)  
 return x, mu, sigma

从潜在空间采样后z\_reparametrized，将其输入到解码器，解码器的网络结构如下：

(z\_2hid): Linear(in\_features=20, out\_features=200, bias=True)  
(hid\_2img): Linear(in\_features=200, out\_features=784, bias=True)

已知输入的特征向量和从潜在空间采样经解码器生成返回的特征向量，为了学习到较好的均值和标准差，其损失增加 KL 散度（Kullback-Leibler Divergence）一项，有∑j=1J​(1+log((σj(i)​)2)−(μj(i)​)2−(σj(i)​)2)+L1​∑l=1L​logpθ​(x(i)∣z(i,l)) where z(i,l)=μ(i)+σ(i)⊙ϵ(l) and ϵ(l)∼N(0,I)。对应的代码为，

# loss  
loss\_fn = nn.BCELoss(reduction="sum")  
reconst\_loss = loss\_fn(x\_reconst, x)  
kl\_div = - torch.sum(1 + torch.log(sigma.pow(2)) - mu.pow(2) - sigma.pow(2))  
  
# Backprop and optimize  
loss = reconst\_loss + kl\_div

试验时，为了观察学习到的潜在空间，首先配置Z\_DIM = 2，即学习到的均值和标准差的维度为2，用于在二维中打印观察。而后配置为Z\_DIM = 20用于实际的计算。

%load\_ext autoreload   
%autoreload 2   
import usda.network as usda\_nx  
import usda.utils as usda\_utils  
  
import torchvision.datasets as datasets   
from torch.utils.data import DataLoader   
from torchvision import transforms  
import torch.nn as nn  
import matplotlib.pyplot as plt  
import torch  
import numpy as np  
import torchvision

配置参数。

# Configuration  
device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')  
INPUT\_DIM = 784  
Z\_DIM = 2  
H\_DIM = 200  
NUM\_EPOCHS = 10  
BATCH\_SIZE = 32  
LR\_RATE = 3e-4

下载 MNIST 手写数字数据集。

dataset=datasets.MNIST(root="C:/Users/richie/omen\_richiebao/omen\_temp/dataset/", train=True, transform=transforms.ToTensor(), download=True)  
train\_loader=DataLoader(dataset=dataset, batch\_size=BATCH\_SIZE, shuffle=True)

初始化 VAE 模型、优化器和损失。

# Initialize model, optimizer, loss  
model = usda\_nx.VariationalAutoEncoder(INPUT\_DIM, Z\_DIM).to(device)  
optimizer = torch.optim.Adam(model.parameters(), lr=LR\_RATE)  
loss\_fn = nn.BCELoss(reduction="sum")

查看模型的网络结构，包含编码器和解码器。

model

VariationalAutoEncoder(  
 (img\_2hid): Linear(in\_features=784, out\_features=200, bias=True)  
 (hid\_2mu): Linear(in\_features=200, out\_features=2, bias=True)  
 (hid\_2sigma): Linear(in\_features=200, out\_features=2, bias=True)  
 (z\_2hid): Linear(in\_features=2, out\_features=200, bias=True)  
 (hid\_2img): Linear(in\_features=200, out\_features=784, bias=True)  
)

训练模型。

# Run training  
usda\_nx.vae\_train(NUM\_EPOCHS, model, optimizer, loss\_fn,train\_loader,INPUT\_DIM,device)

1875it [00:18, 98.98it/s, loss=5.79e+3]   
1875it [00:18, 101.03it/s, loss=5.06e+3]  
1875it [00:18, 101.94it/s, loss=5.28e+3]  
1875it [00:17, 104.55it/s, loss=5.73e+3]  
1875it [00:18, 103.89it/s, loss=5.4e+3]   
1875it [00:18, 102.00it/s, loss=5.79e+3]  
1875it [00:19, 94.77it/s, loss=5.83e+3]   
1875it [00:18, 103.85it/s, loss=5.23e+3]  
1875it [00:19, 96.91it/s, loss=5.32e+3]   
1875it [00:20, 92.01it/s, loss=5.26e+3]

从数据集中分别对应10个数字各提取一个样本用于训练模型编码器的输入以获得均值和标准差（用多元高斯分布（均值和标准差）表征的潜在空间）。

images = []  
idx = 0  
for x, y in dataset:  
 if y == idx:  
 images.append(x.to(device))  
 idx += 1  
 if idx == 10:  
 break  
  
encodings\_digit = []  
for d in range(10):  
 with torch.no\_grad():  
 mu, sigma = model.encode(images[d].view(1, 784))  
 encodings\_digit.append((mu, sigma))

用np.random.multivariate\_normal方法从多元高斯分布中采样数据点。

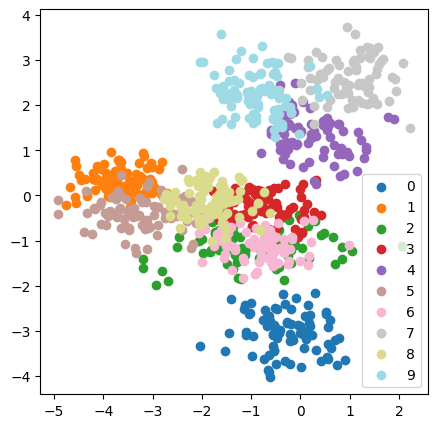
means=np.array([i[0][0].detach().cpu().numpy() for i in encodings\_digit])  
covs=np.array([i[1][0].detach().cpu().numpy() for i in encodings\_digit])  
covs\_diag=[np.diag(i) for i in covs]  
  
pts\_lst=[np.random.multivariate\_normal(mean, cov, size=80) for mean,cov in zip(means,covs\_diag)]

C:\Users\richie\AppData\Local\Temp\ipykernel\_28376\3622666592.py:5: RuntimeWarning:  
  
covariance is not symmetric positive-semidefinite.

​

打印数据点的分布，可以发现10个数字的分布开始显现出团聚的分离性。因为配置的潜在空间维度为2，而有10个数字，因此将维度配置为大于10的一个值将会提高不同数字的分离。

colors=usda\_utils.cmap2hex('tab20',10)  
fig, ax = plt.subplots(figsize=(5,5))  
for idx,pts in enumerate(pts\_lst):  
 ax.scatter(pts[:,0], pts[:,1],c=colors[idx],label=f'{idx}')  
ax.legend()  
plt.show()



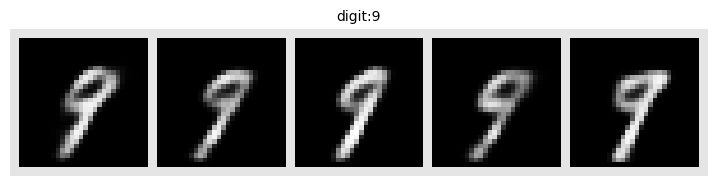
将潜在空间维度配置为20，训练模型。

Z\_DIM = 20  
  
model = usda\_nx.VariationalAutoEncoder(INPUT\_DIM, Z\_DIM).to(device)  
optimizer = torch.optim.Adam(model.parameters(), lr=LR\_RATE)  
loss\_fn = nn.BCELoss(reduction="sum")  
usda\_nx.vae\_train(NUM\_EPOCHS, model, optimizer, loss\_fn,train\_loader,INPUT\_DIM,device)

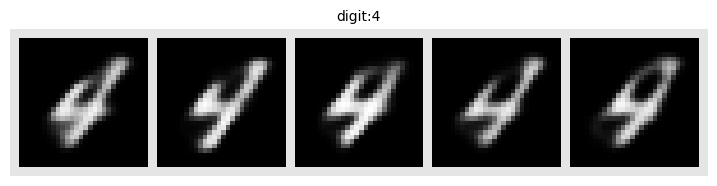
1875it [00:19, 96.93it/s, loss=5.15e+3]   
1875it [00:18, 99.45it/s, loss=4.78e+3]   
1875it [00:19, 94.36it/s, loss=4.48e+3]   
1875it [00:19, 97.07it/s, loss=4.53e+3]   
1875it [00:20, 93.59it/s, loss=4.03e+3]   
1875it [00:19, 94.85it/s, loss=4.58e+3]   
1875it [00:19, 98.16it/s, loss=4.44e+3]   
1875it [00:19, 94.29it/s, loss=4.41e+3]   
1875it [00:20, 93.64it/s, loss=4.17e+3]   
1875it [00:19, 94.44it/s, loss=4.04e+3]

用训练的模型生成数字。

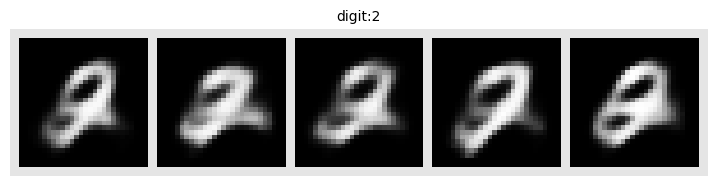
usda\_nx.vae\_digit\_inference(9,model,dataset,device, num\_examples=5,figsize=(9,9),fontsize=10)



usda\_nx.vae\_digit\_inference(4,model,dataset,device, num\_examples=5,figsize=(9,9),fontsize=10)



usda\_nx.vae\_digit\_inference(2,model,dataset,device, num\_examples=5,figsize=(9,9),fontsize=10)



### 4.10.3.2 VGAE[29]

用 GCN 作为 VGAE 的编码器，用简单的内积作为其解码器。

定义（Deﬁnitions），给定一个无向无权图G=(V,E)，有N=∣V∣个顶点。图G的邻接矩阵为A（并假设对角线元素值为1，即顶点与自身的邻接），和度矩阵为D。同时，随机潜在向量（latent variables）为zi​，归纳为N×F的矩阵Z。顶点特征归纳为N×D的矩阵X。

推理模型（Inference model），构建有两层 GCN 参数化的简单推理模型： q(Z∣X,A)=∏i=1N​q(zi​∣X,A), with q(zi​∣X,A)=N(zi​∣μi​,diag(σi2​))，式中，均值向量μi​的矩阵形式为μ=GCNμ​(X,A)；类似有标准差的自然对数， logσ=GCNσ​(X,A)。两层的 GCN 网络定义为GCN(X,A)=A~ReLU(A~XW0​)W1​，其权重矩阵为Wi​⋅GCNμ​(X,A)和GCNσ​(X,A)，共享第一层参数W0​⋅ReLU(⋅)=max(0,⋅)和A~=D−21​AD−21​，为对称归一化邻接矩阵。

生成模型（Generative model ），由潜在变量的内积给出：  
p(A∣Z)=∏i=1N​∏j=1N​p(Aij​∣zi​,zj​), with p(Aij​=1∣zi​,zj​)=σ(zi⊤​zj​)，式中， Aij​为A的元素，σ(⋅)为 Logistic Sigmoid 函数。

学习（Learning），就变分参数的Wi​优化变分下界L： L=Eq(Z∣X,A)​[logp(A∣Z)]−KL[q(Z∣X,A)∥p(Z)]，式中， KL[q(⋅)∥p(⋅)]为q(⋅)和p(⋅)之间的 KL 散度。并进一步取高斯先验p(Z)=∏i​p(zi​)=∏i​N(zi​∣0,I)。对于非常稀疏的邻接矩阵A，可以在L中重新加权Aij​=1的项或者Aij​=0的子样（ sub-sample）项。训练过程中执行全批梯度下降，并使用重参数化技巧（reparameterization trick）[30]进行训练。对于无特征的情况，只需放弃对X的依赖，将X替换为 GCN 中的单位矩阵。

GAE（Non-probabilistic graph auto-encoder）模型，对于 VGAE 模型的非概率变体，计算嵌入Z和重构的邻接矩阵A^有， A^=σ(ZZ⊤), with Z=GCN(X,A)。

使用前文中植物调查样方物种数据集，不过由植物调查样方物种的图嵌入计算转换为预测物种间具有联系的概率，即物种共存的可能性。顶点特征使用两个因素，一个为覆盖率，另一个为物种名称。覆盖率为连续数值，通过除以100将其缩放至[0,1]区间；而物种名称为分类数据，需要转换为数值后喂入模型，因为分类数据并无顺序关系，因此使用独热编码（One-Hot Encoding），而不是整数编码（Integer Encoding）[31]250。

在图构建时，配置阈值为 0.05（5%），只有大于等于该阈值覆盖率的物种才构建调查样方内物种间的边。

%load\_ext autoreload   
%autoreload 2   
import usda.network as usda\_nx  
  
import networkx as nx  
from itertools import combinations  
from torch\_geometric.utils.convert import from\_networkx  
import torch\_geometric.transforms as T  
import torch  
from torch\_geometric.data import Batch,Data  
from torch\_geometric.loader import DataLoader  
from tqdm import tqdm  
import pandas as pd  
from sklearn.preprocessing import OneHotEncoder,Normalizer,LabelEncoder  
import numpy as np

读取数据并构建图，顶点属性x为归一化的覆盖率和物种的独热编码。

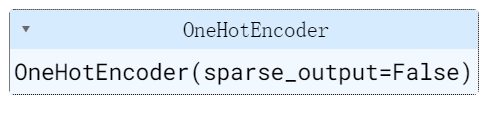
vegetation\_plot\_species\_list\_fn='../data/Countryside\_Survey\_1978\_vegetation\_plot\_data/data/Vegetation Plot - Species List 1978.csv'  
vegetation\_plot\_information\_fn='../data/Countryside\_Survey\_1978\_vegetation\_plot\_data/data/Vegetation Plot - Plot Information 1978.csv'

species\_list=pd.read\_csv(vegetation\_plot\_species\_list\_fn)  
le=LabelEncoder()  
species\_list['name\_encoder']=le.fit\_transform(species\_list.BRC\_NAMES)  
species\_list['total\_cover\_norm']=species\_list.TOTAL\_COVER.apply(lambda x:x/100)  
species\_list.tail(3)

|  | **YEAR** | **SQUARE\_ID** | **PLOT\_ID** | **AMALG\_PTYPE** | **BRC\_NUMBER** | **BRC\_NAMES** | **TOTAL\_COVER** | **name\_encoder** | **total\_cover\_norm** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **44492** | 1978 | WDGZFA | GKGCGUEPMT | X | 5505962.0 | Cladonia sp. | 1 | 153 | 0.01 |
| **44493** | 1978 | WDGZFA | GKGCGUEPMT | X | 9202136.0 | Vaccinium myrtillus | 1 | 643 | 0.01 |
| **44494** | 1978 | WDGZFA | GKGCGUEPMT | X | 9202138.0 | Vaccinium vitis-idaea | 1 | 646 | 0.01 |

建立独热编码模型，用于后续物种名称到独热编码的计算。

X\_names=np.expand\_dims(species\_list.BRC\_NAMES.values,1).astype(str)  
onehot\_encoder=OneHotEncoder(sparse\_output=False)  
onehot\_encoder.fit(X\_names)



按调查样方地块PLOT\_ID列分组数据。

plot\_species\_group=species\_list.groupby(by=['PLOT\_ID'])  
species\_encoder2name=pd.Series(species\_list.BRC\_NAMES.values,index=species\_list.name\_encoder).to\_dict()

依次构建各个调查样方内物种的图。

datas=[]  
  
for group in tqdm(plot\_species\_group):  
 species\_cover\_dict=pd.Series(group[1].total\_cover\_norm.values,index=group[1].name\_encoder).to\_dict()  
 nodes\_attri=[(k,{'x':np.concatenate((np.array([v]),onehot\_encoder.transform([[species\_encoder2name[k]]])[0])).astype(np.float32),'name':species\_encoder2name[k]}) for k,v in species\_cover\_dict.items()]  
 species\_cover\_dict\_threshold={k:v for k,v in species\_cover\_dict.items() if v >=0.05}  
 edges=list(combinations(list(species\_cover\_dict\_threshold.keys()), 2))  
 G=nx.Graph()  
 G.add\_nodes\_from(nodes\_attri)  
 G.add\_edges\_from(edges)  
 G\_pyg=from\_networkx(G)  
 datas.append(G\_pyg)

100%|█████████████████████████████████████████████| 2279/2279 [01:10<00:00, 32.20it/s]

使用PyG的Batch方法将所有调查样方的图描述为一个大的（不连接）图数据对象，并切分为训练、测试和验证数据集。

device=torch.device('cuda' if torch.cuda.is\_available() else 'cpu')  
  
transform=T.Compose([  
 T.ToDevice(device),  
 T.RandomLinkSplit(num\_val=0.05, num\_test=0.1, is\_undirected=True, split\_labels=True, add\_negative\_train\_samples=False),  
])  
  
batch=Batch.from\_data\_list(datas)  
batch=transform(batch)  
train\_data, val\_data, test\_data=batch

查看顶点的特征。

train\_data.x

tensor([[0.0100, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],  
 [0.0100, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],  
 [0.0100, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],  
 ...,  
 [0.0100, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],  
 [0.0100, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],  
 [0.0100, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000]],  
 device='cuda:0')

查看 VGAE 网络结构。编码器由多层 GCN （使用PgG提供的GCNConv层）组成，输出为潜在空间多元高斯分布的均值和标准差的自然对数；解码器由潜在变量的内积给出，即InnerProductDecoder()方法。从潜在空间中采样对应的代码为：

def reparametrize(self, mu: Tensor, logstd: Tensor) -> Tensor:  
 if self.training:  
 return mu + torch.randn\_like(logstd) \* torch.exp(logstd)  
 else:  
 return mu

损失部分的代码为：

z = self.model.encode(train\_data.x, train\_data.edge\_index)  
loss = self.model.recon\_loss(z, train\_data.pos\_edge\_label\_index) + (1 / train\_data.num\_nodes) \* self.model.kl\_loss()

损失包括两部分，一部分为recon\_loss，一部分为kl\_loss。其中recon\_loss部分对应代码如下：

def recon\_loss(self, z: Tensor, pos\_edge\_index: Tensor,  
 neg\_edge\_index: Optional[Tensor] = None) -> Tensor:  
 r"""Given latent variables :obj:`z`, computes the binary cross  
 entropy loss for positive edges :obj:`pos\_edge\_index` and negative  
 sampled edges.  
  
 Args:  
 z (torch.Tensor): The latent space :math:`\mathbf{Z}`.  
 pos\_edge\_index (torch.Tensor): The positive edges to train against.  
 neg\_edge\_index (torch.Tensor, optional): The negative edges to  
 train against. If not given, uses negative sampling to  
 calculate negative edges. (default: :obj:`None`)  
 """  
 pos\_loss = -torch.log(  
 self.decoder(z, pos\_edge\_index, sigmoid=True) + EPS).mean()  
  
 if neg\_edge\_index is None:  
 neg\_edge\_index = negative\_sampling(pos\_edge\_index, z.size(0))  
 neg\_loss = -torch.log(1 -  
 self.decoder(z, neg\_edge\_index, sigmoid=True) +  
 EPS).mean()  
  
 return pos\_loss + neg\_loss

kl\_loss部分对应代码为：

def kl\_loss(self, mu: Optional[Tensor] = None,  
 logstd: Optional[Tensor] = None) -> Tensor:  
 r"""Computes the KL loss, either for the passed arguments :obj:`mu`  
 and :obj:`logstd`, or based on latent variables from last encoding.  
  
 Args:  
 mu (torch.Tensor, optional): The latent space for :math:`\mu`. If  
 set to :obj:`None`, uses the last computation of :math:`\mu`.  
 (default: :obj:`None`)  
 logstd (torch.Tensor, optional): The latent space for  
 :math:`\log\sigma`. If set to :obj:`None`, uses the last  
 computation of :math:`\log\sigma^2`. (default: :obj:`None`)  
 """  
 mu = self.\_\_mu\_\_ if mu is None else mu  
 logstd = self.\_\_logstd\_\_ if logstd is None else logstd.clamp(  
 max=MAX\_LOGSTD)  
 return -0.5 \* torch.mean(  
 torch.sum(1 + 2 \* logstd - mu\*\*2 - logstd.exp()\*\*2, dim=1))

vgae\_model=usda\_nx.VGAE\_gnn(train\_data.num\_features, 16,lr=0.01,device=device)  
vgae\_model.model

VGAE(  
 (encoder): Encoder(  
 (conv1): GCNConv(676, 32)  
 (conv\_mu): GCNConv(32, 16)  
 (conv\_logstd): GCNConv(32, 16)  
 )  
 (decoder): InnerProductDecoder()  
)

训练模型。

vgae\_model.fit(train\_data, val\_data,epochs=301,verbose=50 )

Epoch 0 | Loss: 3.5018 | Val AUC: 0.6436 | Val AP: 0.5539  
Epoch 50 | Loss: 0.8684 | Val AUC: 0.9529 | Val AP: 0.9146  
Epoch 100 | Loss: 0.7979 | Val AUC: 0.9768 | Val AP: 0.9622  
Epoch 150 | Loss: 0.7788 | Val AUC: 0.9836 | Val AP: 0.9764  
Epoch 200 | Loss: 0.7793 | Val AUC: 0.9849 | Val AP: 0.9772  
Epoch 250 | Loss: 0.7628 | Val AUC: 0.9878 | Val AP: 0.9808  
Epoch 300 | Loss: 0.7584 | Val AUC: 0.9893 | Val AP: 0.9836

用测试数据集测试结果精度约为 0.98。

test\_auc, test\_ap = vgae\_model.test(test\_data)   
print(f'Test AUC: {test\_auc:.4f} | Test AP {test\_ap:.4f}')

Test AUC: 0.9858 | Test AP 0.9817

保存和读取已训练的模型。

vgae\_path='../models/vgae.pt'  
torch.save(vgae\_model.model,vgae\_path)

model\_vgae=torch.load(vgae\_path,map\_location=torch.device('cpu'))   
model\_vgae.eval();

从一个调查样方中随机提取三个物种构建两两之间均存在边的图，其中覆盖率随机给出，用已训练的模型预测物种间存在边的可能性如下。

edge\_index\_1=torch.tensor([[0, 0, 1],  
 [1, 2, 2]], dtype=torch.long)  
x\_1\_coverNID=[[ 20., 175], [ 10., 394], [ 50., 151]]  
x\_1=torch.tensor([np.concatenate((np.array([i[0]/100]),onehot\_encoder.transform([[species\_encoder2name[i[1]]]])[0])) for i in x\_1\_coverNID], dtype=torch.float32)  
data\_fake\_1=Data(x=x\_1, edge\_index=edge\_index\_1)  
  
encoder2name4print=lambda x:{i:species\_encoder2name[i] for i in [int(j[1]) for j in x]}  
print(encoder2name4print(x\_1\_coverNID))

{175: 'Cynosurus cristatus', 394: 'Nardus stricta', 151: 'Cirsium palustre'}

预测结果表明从一个调查样方中随机提取三个物种两两之间存在边的可能性极大，即可以估计为物种共存。

z=model\_vgae.encode(data\_fake\_1.x,data\_fake\_1.edge\_index)   
Ahat=torch.sigmoid(z @ z.T)  
Ahat

tensor([[0.9776, 0.9764, 0.9715],  
 [0.9764, 0.9911, 0.9931],  
 [0.9715, 0.9931, 0.9959]], grad\_fn=<SigmoidBackward0>)

下述则随机给出三个物种和各自假设的覆盖率，可以观察到 Oenanthe crocata 和 Papaver rhoeas 存在边的可能性为 0.7543，而其它物种间则只有约一半的几率存在边。

edge\_index\_2=torch.tensor([[0, 0, 1],  
 [1, 2, 2]], dtype=torch.long)  
x\_2\_coverNID=[[ 10., 200], [ 35., 399], [ 50., 410]]  
x\_2=torch.tensor([np.concatenate((np.array([i[0]/100]),onehot\_encoder.transform([[species\_encoder2name[i[1]]]])[0])) for i in x\_2\_coverNID], dtype=torch.float32)  
data\_fake\_2=Data(x=x\_2, edge\_index=edge\_index\_2)  
print(encoder2name4print(x\_2\_coverNID))  
  
z=model\_vgae.encode(data\_fake\_2.x,data\_fake\_2.edge\_index)   
Ahat=torch.sigmoid(z @ z.T)  
Ahat

{200: 'Eleocharis palustris', 399: 'Oenanthe crocata', 410: 'Papaver rhoeas'}  
  
  
  
  
  
tensor([[0.5781, 0.5348, 0.5069],  
 [0.5348, 0.6813, 0.7543],  
 [0.5069, 0.7543, 0.8557]], grad\_fn=<SigmoidBackward0>)

可以对测试数据集顶点间存在边的可能性进行估计。

z = model\_vgae.encode(test\_data.x.cpu(), test\_data.edge\_index.cpu())  
Ahat = torch.sigmoid(z @ z.T)

print(Ahat.shape)  
Ahat

torch.Size([44466, 44466])  
  
  
  
  
  
tensor([[0.5685, 0.5089, 0.5407, ..., 0.5687, 0.5447, 0.5137],  
 [0.5089, 0.5613, 0.5291, ..., 0.5539, 0.5306, 0.5130],  
 [0.5407, 0.5291, 0.5704, ..., 0.5681, 0.5449, 0.5162],  
 ...,  
 [0.5687, 0.5539, 0.5681, ..., 0.6827, 0.5785, 0.5481],  
 [0.5447, 0.5306, 0.5449, ..., 0.5785, 0.5827, 0.5303],  
 [0.5137, 0.5130, 0.5162, ..., 0.5481, 0.5303, 0.5526]],  
 grad\_fn=<SigmoidBackward0>)

### 4.10.3.3 生成图

NetworkX库提供有[多种类型生成图的函数](https://networkx.org/documentation/latest/reference/generators.html)⑨，例如各种类型随机图生成器，各类小型并有命名的图（如 Krackhardt Kite Social Network、Moebius-Kantor graph等）的生成器，著名社交网络生成器和用于研究社交网络的图形类生成器等。而基于 GNN 的图生成器更具表现力，用于深度图生成的网络架构主要有 GVAEs，GANs 和自回归模型（autoregressive models）等。上文用A^=σ(ZTZ)预测顶点间的链接，如果给一个概率的阈值，例如下述的 0.9，则可以根据邻接矩阵的概率生成图，大于0.9的顶点赋值为1，预示顶点间存在边；否则赋值为0，预示顶点间不存在边。

adj = torch.where((z @ z.T) > 0.9, 1, 0)  
adj[10000:,2000:]

tensor([[0, 0, 0, ..., 0, 0, 0],  
 [1, 1, 1, ..., 0, 0, 0],  
 [0, 0, 0, ..., 0, 0, 0],  
 ...,  
 [0, 0, 0, ..., 0, 0, 0],  
 [0, 0, 0, ..., 0, 0, 0],  
 [0, 0, 0, ..., 0, 0, 0]])

注释（Notes）：

① NetworkX，（<https://networkx.org/documentation/stable/index.html>）。

② Countryside Survey 1978 vegetation plot data，（<https://ckan.publishing.service.gov.uk/dataset/countryside-survey-1978-vegetation-plot-data>）。

③ Sklearn，（<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html>）。

④ Node2Vec，（<https://github.com/eliorc/node2vec>）。

⑤ PyG （PyTorch Geometric），（<https://pytorch-geometric.readthedocs.io/en/latest/index.html>）。

⑥ PyTorch，（<https://pytorch.org/>）。

⑦ PyG （PyTorch Geometric） datasets，（<https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html>）。

⑧ 卷积层（Convolutional Layers），（<https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#convolutional-layers>）。

⑨ NetworkX库多种类型生成图的函数，（[https://networkx.org/documentation/latest/reference/generators.html#](https://networkx.org/documentation/latest/reference/generators.html)）。

参考文献（References）:

[1] Xu, M. (2021). Understanding Graph Embedding Methods and Their Applications. SIAM Review, 63(4), 825–853. doi:10.1137/20M1386062.

[2] Abdel-Basset, M., Moustafa, N., Hawash, H., & Tari, Z. (2023). Responsible Graph Neural Networks (1st ed.). Chapman and Hall/CRC. https://doi.org/10.1201/9781003329701

[3] Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). DeepWalk: Online Learning of Social Representations. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 701–710). Association for Computing Machinery.

[4] Grover, A. & Leskovec, J. (2016). node2vec: Scalable feature learning for networks. Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining (p./pp. 855–864).

[5] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J. & Mei, Q. (2015). LINE: Large-scale Information Network Embedding. WWW (p./pp. 1067–1077), New York, NY, USA: ACM. ISBN: 978-1-4503-3469-3

[6] Shen, X. & Chung, F.-L. (2019). Deep Network Embedding for Graph Representation Learning in Signed Networks.. CoRR, abs/1901.01718.

[7] Ribeiro, L. F., Saverese, P. H. & Figueiredo, D. R. (2017). struc2vec: Learning Node Representations from Structural Identity. Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD ’17 (p./pp. 385–394), Halifax, NS, Canada: ACM Press. ISBN: 978-1-4503-4887-4

[8] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv [Cs.CL]. Retrieved from http://arxiv.org/abs/1301.3781

[9] GraphEmbedding, <https://github.com/shenweichen/GraphEmbedding>

[10] P.Erdős and A. Rényi, On Random Graphs, Publ. Math. 6, 290 (1959).

[11] Huffman, D. (1952). A Method for the Construction of Minimum-Redundancy Codes. Proceedings of the IRE, 40(9), 1098–1101. doi:10.1109/jrproc.1952.273898

[12] Rong, X. (2014). word2vec parameter learning explained. arXiv preprint arXiv:1411.2738.

[13] Morin, F., & Bengio, Y. (2005). Hierarchical Probabilistic Neural Network Language Model. In Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (pp. 246–252). PMLR.

[14] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. arXiv [Cs.CL]. Retrieved from http://arxiv.org/abs/1310.4546

[15] Zachary, Wayne W. “An Information Flow Model for Conflict and Fission in Small Groups.” Journal of Anthropological Research, 33, 452–473, (1977).

[16] Ornduff, R., Faber, P., & Keeler-Wolf, T. (2003). Introduction to California Plant Life. University of California Press.

[17] Plant community, <https://en.wikipedia.org/wiki/Plant_community>

[18] Labonne, M. (2023). Hands-On Graph Neural Networks Using Python (1st ed.). Packt Publishing. (Original work published 2023)

[19] Understanding Convolutions on Graphs, <https://distill.pub/2021/understanding-gnns/>.

[20] Benedek Rozemberczki, Carl Allen, Rik Sarkar, Multi-Scale attributed node embedding, Journal of Complex Networks, Volume 9, Issue 2, April 2021, cnab014, https://doi.org/10.1093/comnet/cnab014

[21] Thomas N. Kipf, & Max Welling (2017). Semi-Supervised Classification with Graph Convolutional Networks. In 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net.

[22] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention Is All You Need. arXiv [Cs.CL]. Retrieved from http://arxiv.org/abs/1706.03762

[23] Shaked Brody, Uri Alon, & Eran Yahav (2022). How Attentive are Graph Attention Networks? . In International Conference on Learning Representations.

[24] Hamilton, W., Ying, Z. & Leskovec, J. (2017). Inductive Representation Learning on Large Graphs. Advances in Neural Information Processing Systems, , 1024-1034.

[25] Marinka Zitnik, Jure Leskovec, Predicting multicellular function through multi-layer tissue networks, Bioinformatics, Volume 33, Issue 14, July 2017, Pages i190–i198, https://doi.org/10.1093/bioinformatics/btx252

[26] Shervashidze, N., Schweitzer, P., Leeuwen, E., Mehlhorn, K., & Borgwardt, K. (2011). Weisfeiler-Lehman Graph Kernels. J. Mach. Learn. Res., 12(null), 2539–2561.

[27] Weisfeiler and Lehman, A.A. (1968) A Reduction of a Graph to a Canonical Form and an Algebra Arising during This Reduction. Nauchno-Technicheskaya Informatsia, 9.

[28] Keyulu Xu, Weihua Hu, Jure Leskovec, & Stefanie Jegelka (2019). How Powerful are Graph Neural Networks?. In 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net.

[29] Kipf, T. N. & Welling, M. (2016). Variational Graph Auto-Encoders. arXiv:1611.07308 [cs, stat].

[30] Kingma, D. P. & Welling, M. (2014). Auto-Encoding Variational Bayes. 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings.

[31] Brownlee, J, Data Preparation for Machine Learning - Data Cleaning, Feature Selection, and Data. machine learning mastery, 2020.