Representation Learning on Graphs：Methods and Applications

William L. Hamilton、*Rex Ying*、*Jure Leskovec*

Department of Science Standford University

论文地址：https://arxiv.org/abs/1709.05584v3

目录

摘要

图机器学习目前是一个十分重要且常见的任务，从药物设计到社交网络的关系推荐都能看到它的应用。目前这个领域最主要的挑战是:找到一种方法可以表示、编码图结构以至于后续可以很轻松的利用机器学习模型。传统上，机器学习方法依赖于用户定义的试探法来提取关于图结构信息的特征的编码（例如，度统计或者核函数）。近年来，使用基于深度学习和非线性降维的技术，自动学习将图形结构编码成低维嵌入的方法引起了一股热潮。在这里，我们给出一个关于图表示学习领域主要进展的概念性综述，包括基于矩阵分解的方法，基于随机游走的方法，以及图神经网络。

1.Introduction

Graph是一种普遍存在的数据结构。社交网络（Social networks）、分子结构图（molecular graph structures）、生物蛋白质图（biological protein-protein networks）、推荐系统（recommender systems）--所有这些领域都可以很轻松的被建模成图结构，这些图可以捕捉单个单元（节点）之间的联系（边）。

图不仅仅在结构化知识库中有用，也在线代机器学习中也扮演着非常重要的角色。许多机器学习应用程序试图使用图结构数据作为特征信息来进行预测或发现新的模式。例如，人们可能希望对蛋白质在生物相互作用图中的角色进行分类，预测人在社交网络中的角色，向社交网络中的用户推荐新朋友，或者预测现有药物分子的新的治疗应用。

图机器学习的中心问题是：找到一种方法把图结构的信息整合到机器学习的模型中去。例如，在社交网络中的链路预测，人们可能想要对节点之间的成对属性进行编码，比如关系的强度或者共同朋友的数量。又或者，要做节点分类任务的情况下，我们可能想要包含关于图中节点的全局位置或者局部邻域的结构信息（图1）。从机器学习的角度看，挑战在于：没有直接的方法将这种关于图结构的高维非欧几里得信息编码到特征空间中去。

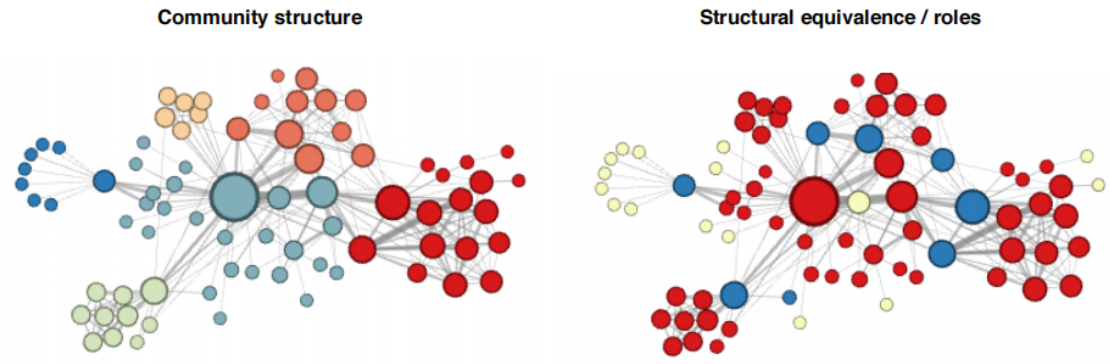


Figure 1: Two different views of a character-character interaction graph derived from the Les Misérables novel, where two nodes are connected if the corresponding characters interact. The coloring in the left figure emphasizes differences in the nodes' global positions in the graph: nodes have the same color if they belong to the same community, at a global level. In contrast, the coloring in the right figure denotes structural equivalence between nodes, or the fact that two nodes play similar roles in their local neighborhoods (*e.g.*, "bridging nodes" are colored blue). The colorings for both figures were generated using different settings of the node2vec node embedding method [28], described in Section 2. Reprinted from [28] with permission.[1]

为了从图中提取结构信息，传统的机器学习方法通常依赖于总结图统计（graph statistics，比方说，节点的度或者聚类系数），核函数（ kernel functions），或者精心设计的特征（engineered features）来测量局部邻域特征。然而，这些方法是有局限性的，因为这些手工 设计的特征不灵活，即它们不能在学习的过程中自适应（ hand-engineered features are inflexible），并且设计这些特征很耗时。

最近，出现了一种寻求学习表示的方法，可以编码图结构信息。这些表示学习方法背后的思想是学习一种映射，该映射可以将节点或者整个（子）图嵌入为低维向量空间R^d中的点。目标是优化映射，以便嵌入空间中的几何关系反映原始图形的结构。在优化嵌入空间后，学习的嵌入可以所谓下游机器学习任务的特征输入。表示学习方法和以前的工作之间的主要区别在于它们如何处理表示图结构的问题。先前的工作把这个问题视为预处理步骤，使用手工设计的统计来提取结构信息。而表示学习方法将这个问题视为机器学习任务本身，使用数据驱动的方法来学习嵌入。

这里我们回顾了近年来关于图表示学习的研究，我们在这里也用一个统一的框架去解释它们。我们重点回顾了最近在机器学习和数据挖掘邻域引起极大关注的方法，特别是那些可扩展到大规模图的方法（受到深度学习的启发）。

1.1 Notion and essential assumptions

我们假设，我们的表示学习算法的主要输入是一个无向图$G = (E, V)$,带有一个二元的邻接矩阵A.我们还假设这些方法可以利用节点属性$X^{m*|V|}$（比如，与节点有联系的文本数据或者元数据）。我们的目标是，利用包含在A和X中的信息，去把每个节点或子图映射到一个向量$Z \in R^d$中去,其中$d << |V|$的。

我们回顾的大多数方法，仅仅利用A和X中的信息，以无监督的方式去优化这种映射，而不了解特定的下游机器学习任务。但是我们也会讨论一些监督学习的表示方法，在监督学习的表示方法中模型利用分类或者回归标签来优化嵌入。

2 Embedding nodes

我们首先讨论节点嵌入的方法，其目标是将节点编码为低维向量，这些向量总结了节点在图中的位置信息和它们局部邻域图的结构信息。这些低维嵌入可被视为将节点编码或投影到潜在空间中，其中该潜在空间中的几何关系对应于原始图中的相互作用。图2展示了一个社交网络中嵌入的实例，其中二维节点嵌入捕捉了社交网络中隐含的社区结构。

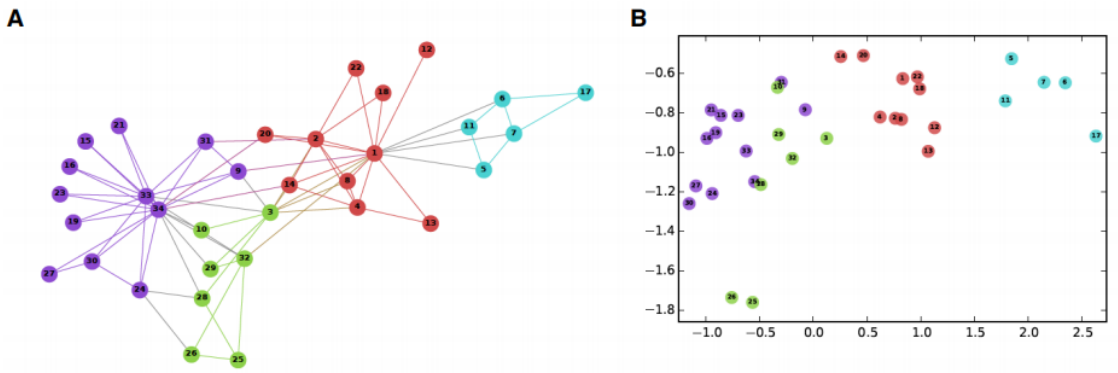


Figure 2: A, Graph structure of the Zachary Karate Club social network, where nodes are connected if the corresponding individuals are friends. The nodes are colored according to the different communities that exist in the network. **B**, Two-dimensional visualization of node embeddings generated from this graph using the DeepWalk method (Section 2.2.2) [47]. The distances between nodes in the embedding space reflect similarity in the original graph, and the node embeddings are spatially clustered according to the different color-coded communities. Reprinted with permission from [47, 49].

2.1 Overview of approaches: An encoder-decoder perspective

近年来，关于节点嵌入的研究激增，导致了符号、动机和概念模型的复杂性。因此，本文在讨论各种技术之前，我们首先提出一个统一的编码器-解码器框架。

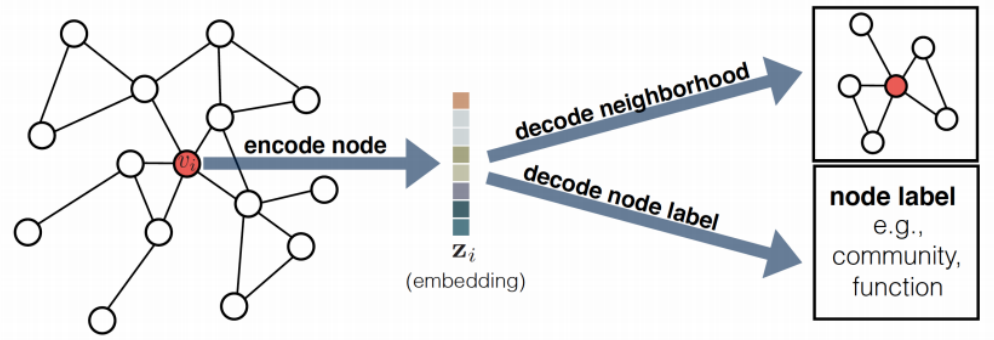


Figure 3: Overview of the encoder-decoder approach. First the encoder maps the node, v_i, to a low-dimensional vector embedding, $\mathbf{z}_i$, based on the node's position in the graph, its local neighborhood structure, and/or its attributes. Next, the decoder extracts user-specified information from the low-dimensional embedding; this might be information about v_i's local graph neighborhood (*e.g.*, the identity of its neighbors) or a classification label associated with v_i (*e.g.*, a community label). By jointly optimizing the encoder and decoder, the system learns to compress information about graph structure into the low-dimensional embedding space.

在这个框架中，我们围绕两个关键的映射函数来组织各种方法：`编码器encoder`，他将每个节点映射到一个低维向量或嵌入中；`解码器decoder`，它从学习到的嵌入中解码出关于图的结构信息（图3）。编码器解码器背后的思想是：如果我们能从编码得到的低维嵌入中解码高维图信息（比如图中节点的全局位置信息和局部邻域信息），那么这些嵌入应该包含下游机器学习任务所需的所有信息。

形式上编码器是一个函数：

$$ENC : V \rightarrow R^d,$$

将节点映射到向量嵌入$Z_i \in R^d$中，这里Z_i对应节点$v_i \in V$的嵌入。解码器也是一个函数，它接受一组节点的嵌入，并从嵌入中解码出用户指定的图统计信息。原则上，很多解码器都是可以的，然而绝大多数工作中使用的都是一个基本的成对解码器（*pairwise decoder*）：

$$DEC : R^d \times R^d \rightarrow R^+,$$

这将成对的节点嵌入映射到实值节点相似性度量，该度量量化了原始图中两个节点的相似性。

当我们将成对解码器应用于一对嵌入(z_i, z_j)时，我们得到了原始图中的v_i和v_j之间的一个重构，我们的目标就是优化编码器和解码器映射，以最小化该重构中的误差。从而：

$$DEC(ENC(v_i), ENC(v_j)) = DEC(z_i, z_j) \approx sG(v_i, v_j),$$

公式中，sG是用户定义的，基于图中节点间的相似性度量。换句话说，换句话说，我们希望优化我们的编码器-解码器模型，以便我们能够解码原始图sG(vi，vj)中来自低维节点嵌入zi和zj的成对节点相似性。

损失函数如下：

$$L = \sum_{(vi,vj)\in D} l(DEC(z_i, z_j), sG(v_i, v_j)),$$

其中l:R × R → R是用户指定的损失函数，它测量解码(即估计)的相似性值$DEC(z_i, z_j)$和$sG(v_i, v_j)$之间的差异。

一旦我们优化了编码器-解码器系统，我们就可以使用训练好的编码器来生成节点的嵌入，然后可以将其用作下游机器学习任务的特征输入。

采用这种编码器-解码器的观点，我们沿着以下四个方法组成部分来组织我们对各种节点嵌入方法的讨论：

1. A **pairwise similarity function** sG：$V \times V \rightarrow R^+$,定义在图G上，这个函数定义了图G中两节点间的相似度。
2. An **encoder function**，ENC生成节点的嵌入。这个函数包含许多可以训练的参数，可以在训练过程中得以优化。
3. A **decoder function**，DEC从生成的嵌入中重构成对的相似度值。通常不包含可以训练的参数。
4. A **loss function**，l,其确定如何评估成对重建的质量以训练模型，即如何将$DEC(z_i，z_j)$与真实的$sG(v_i，v_j)$值进行比较。

我们所展示的不同的节点嵌入技术，不同之处就在一它们对这四部分的定义不同。

2.1.1 Notes on optimization and implementation details

对优化和实施细节的说明：在大多数情况下，随机梯度下降用于优化，尽管一些算法允许通过矩阵分解的形式的解决方案。然而，请注意，我们在这里不关注优化算法，而是强调不同嵌入方法之间存在的高级差异，与优化方法的细节无关。

2.2 Shallow embedding approaches

大多数节点嵌入技术依赖于我们所说的浅层嵌入。对于这些浅层嵌入方法，将节点映射到嵌入向量的编码器在功能上只是一个“嵌入查找”（embedding lookup）：

$$ENC(v_i) = Zv_i,$$

这里，$Z \in R^{d\times|V|}$是一个包含所有节点嵌入向量的矩阵，v_i是一个one-hot indicator 向量，对于与Z中属于v_i那列。

表1总结了编码器-解码器框架中一些众所周知的浅层嵌入方法。表1强调了如何根据(I)它们的解码器功能，(ii)它们基于图的相似性度量，以及(iii)它们的损失函数来简洁地描述这些方法。

Table 1: A summary of some well-known shallow embedding embedding algorithms. Note that the decoders and similarity functions for the random-walk based methods are asymmetric, with the similarity function $p_{\mathcal{G}}(v_j|v_i)$ corresponding to the probability of visiting v_j on a fixed-length random walk starting from v_i.

Type	Method	Decoder	Similarity measure	Loss function (ℓ)
Matrix factorization	Laplacian Eigenmaps [4]	$\|\mathbf{z}_i - \mathbf{z}_j\|_2^2$	general	$\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) \cdot s_{\mathcal{G}}(v_i, v_j)$
	Graph Factorization [1]	$\mathbf{z}_i^\top \mathbf{z}_j$	$\mathbf{A}_{i,j}$	$\|\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\|_2^2$
	GraRep [9]	$\mathbf{z}_i^\top \mathbf{z}_j$	$\mathbf{A}_{i,j}, \mathbf{A}_{i,j}^2, ..., \mathbf{A}_{i,j}^k$	$\|\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\|_2^2$
	HOPE [45]	$\mathbf{z}_i^\top \mathbf{z}_j$	general	$\|\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\|_2^2$
Random walk	DeepWalk [47]	$\frac{e^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{k\in\mathcal{V}} e^{\mathbf{z}_i^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v_j\|v_i)$	$-s_{\mathcal{G}}(v_i, v_j) \log(\text{DEC}(\mathbf{z}_i, \mathbf{z}_j))$
	node2vec [28]	$\frac{e^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{k\in\mathcal{V}} e^{\mathbf{z}_i^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v_j\|v_i)$ (biased)	$-s_{\mathcal{G}}(v_i, v_j) \log(\text{DEC}(\mathbf{z}_i, \mathbf{z}_j))$

下面两节更详细地描述了这些方法，区分了基于矩阵分解的方法(第2.2.1节)和基于随机漫步的更近的方法(第2.2.2节)。

2.2.1 Factorization-based approaches

学习节点表示的早期方法主要集中在矩阵分解方法上，这直接受到经典降维技术的启发。

Laplacian eigenmaps. 编码器定义如下：

$$DEC(z_i, z_j) = ||z_i - z_j||_2^2$$

并且其中损失函数根据节点对在图中的相似性对节点对进行加权：

$$L = \sum_{(vi,vj)\in D} DEC(z_i, z_j) \cdot sG(v_i, v_j),$$

Inner-product methods. 基于成对内积解码器的嵌入方法：

$$DEC(z_i, z_j) = z_i^T z_j,$$

这里，两个节点之间的关系强度与它们嵌入的点积成正比。图因式分解(GF)算法4[1]、GraRep [9]和HOPE [45]都属于这一类。特别是，所有这三种方法都使用内积解码器，即均方误差损失：

$$L = \sum_{(v_i,v_j)\in D} ||DEC(z_i, z_j) - sG(v_i, v_j)||_2^2,$$

它们的不同之处主要在于所使用的节点相似性度量，即它们如何定义$sG(v_i, v_j)$。

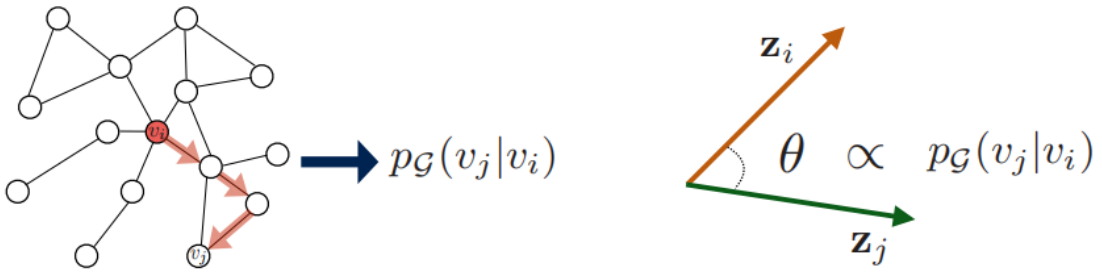
在本节中，我们将这些方法称为矩阵分解方法，因为在所有节点上求平均值，它们优化了以下形式的损失函数：

$$L \approx ||Z^T Z - S||_2^2,$$

其中，S是包含成对相似性度量(即，Si，j，sG(vi，vj))的矩阵，Z是节点嵌入的矩阵。直观地说，这些方法的目标仅仅是学习每个节点的嵌入，使得所学习的嵌入向量之间的内积近似于节点相似性的某种确定性度量。

2.2.2 Random walk approaches

最近许多成功的方法也属于浅层嵌入，它们学习节点嵌入的方法基于随机游走。它们的关键创新是优化节点嵌入，这样，如果节点倾向于在图中的短随机游走中同时出现，它们就有相似的嵌入(图4)。



1. Run random walks to obtain co-occurrence statistics.

2. Optimize embeddings based on co-occurrence statistics.

Figure 4: The random-walk based methods sample a large number of fixed-length random walks starting from each node, v_i. The embedding vectors are then optimized so that the dot-product, or angle, between two embeddings, $\mathbf{z}_i$ and $\mathbf{z}_j$, is (roughly) proportional to the probability of visiting v_j on a fixed-length random walk starting from v_i.

DeepWalk and node2vec. 像上面描述的矩阵分解方法一样，DeepWalk和node2vec依赖于浅层嵌入，并使用基于内积的解码器。然而，这些方法不是试图解码确定性的节点相似性度量，而是优化嵌入来编码随机行走的统计数据。这些方法背后的基本思想是学习嵌入，以便：

$$\begin{aligned} \text{DEC}(\mathbf{z}_i, \mathbf{z}_j) &\triangleq \frac{e^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{v_k \in \mathcal{V}} e^{\mathbf{z}_i^\top \mathbf{z}_k}} \\ &\approx p_{\mathcal{G},T}(v_j|v_i), \end{aligned} \tag{10}$$

这里，$p_{G,T}(v_j|v_i)$是在以v_i为起点步长为T的随机游走路径中v_j出现的概率。通常T在2~10之间取值。注意$p_{G,T}(v_j|v_i)$与上文的相似性度量不同，$p_{G,T}(v_j|v_i)$是随机且不对称的。

除了算法上的差异，DeepWalk和node2vec之间的主要区别在于node2vec允许随机游走的灵活定义，而DeepWalk在图上使用简单的无偏随机漫步。

Large-scale information network embeddings (LINE). 这是另一个非常成功的浅层嵌入方法。并不基于随机游走，该方法结合了两个编码器-解码器目标，分别优化“一阶”和“二阶”节点相似性。一阶目标使用基于sigmoid函数的解码器；二阶编码器-解码器目标是类似，但考虑了两跳相邻邻域，并使用了与等式（10）相同的编码器

HARP: Extending random-walk embeddings via graph pre-processing.

Additional variants of the random-walk idea.

2.3 Generalized encoder-decoder architectures

到目前为止，我们回顾的所有节点嵌入方法都是浅层嵌入方法，其中编码器只是一个嵌入查找(等式5)。然而，这些浅层嵌入方法独立地为每个节点训练唯一的嵌入向量，这导致了许多缺点：

1. 在编码器中，节点间不能共享参数

2. 浅层嵌入的方法无法利用节点的属性
3. 浅层嵌入的方法是典型的*transductive*，*transductive*和*inductive*的区别在于我们想要预测的样本，是不是我们在训练的过程中见（用）过。

现在一些基于深度神经网络的方法已经解决了上面的几个问题，这些方法使用一个更加复杂的编码器，依赖于图的属性和结构信息。

2.3.1 Neighborhood autoencoder methods

深度神经图表示和结构深度网络嵌入使用深度神经网络直接将图结构结合到编码器中，从而解决了第一个上面浅层嵌入方法的第一个问题。这些方法背后的思想是，使用自动编码器--众所周知的深度学习方法--来压缩关于节点局部邻域的信息。（图6）

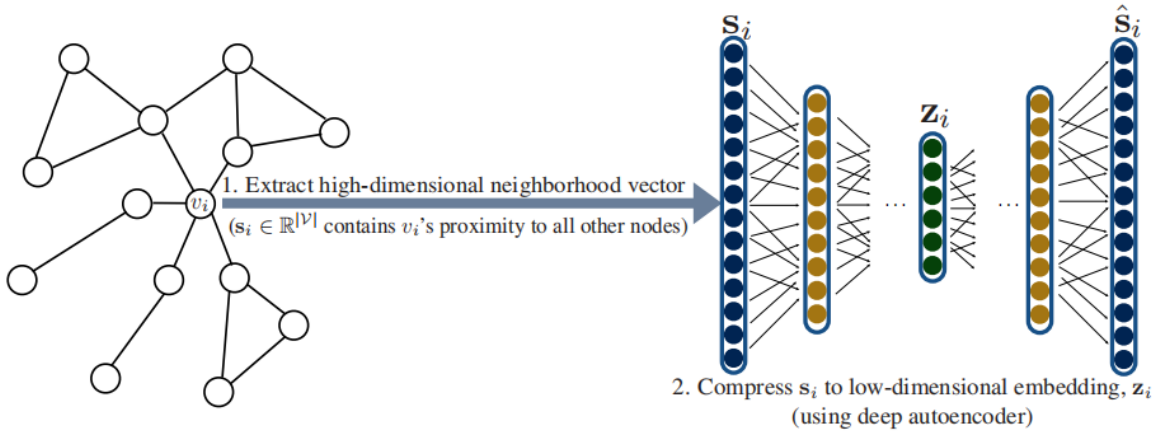


Figure 6: To generate an embedding for a node, v_i, the neighborhood autoencoder approaches first extract a high-dimensional neighborhood vector $\mathbf{s}_i \in \mathbb{R}^{|\mathcal{V}|}$, which summarizes v_i's similarity to all other nodes in the graph. The $\mathbf{s}_i$ vector is then fed through a deep autoencoder to reduce its dimensionality, producing the low-dimensional $\mathbf{z}_i$ embedding.

2.3.2 Neighborhood aggregation and convolutional encoder

许多节点嵌入的方法都旨在解决浅层嵌入和通过设计编码器的自动编码方法（这种方法仅仅依赖于一个节点的局部领域而不依赖整个图结构）的限制。这些方法背后的思想就是：通过从它局部邻居节点聚合信息来生成节点的嵌入。（图7）

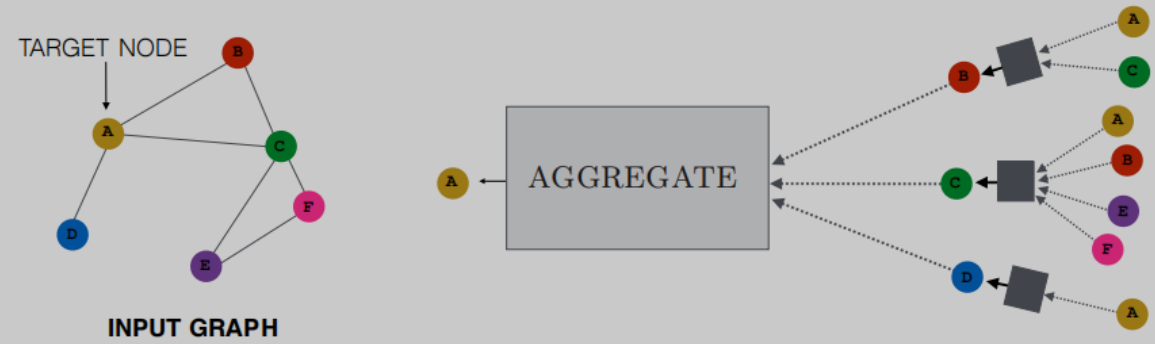


Figure 7: Overview of encoding in the neighborhood aggregation methods. To generate an embedding for node A, the model aggregates messages from A's local graph neighbors (*i.e.*, B, C, and D), and in turn, the messages coming from these neighbors are based on information aggregated from their respective neighborhoods, and so on. A "depth-2" version of this idea is shown (*i.e.*, information is aggregated from a two-hop neighborhood around node A), but in principle these methods can be of an arbitrary depth. At the final "depth" or "layer" the initial messages are based on the input node attributes.

与上面讨论的方法不同，邻域聚合算法利用了节点的属性来生成嵌入。当没有属性数据时，利用图的统计数据作为属性利用。

编码阶段算法伪代码：

Algorithm 1: Neighborhood-aggregation encoder algorithm. Adapted from [29].

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\{\mathbf{W}^k, \forall k \in [1, K]\}$; non-linearity σ; differentiable aggregator functions $\{\text{AGGREGATE}_k, \forall k \in [1, K]\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output: Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{COMBINE}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \text{NORMALIZE}(\mathbf{h}_v^k), \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

注意在算法中有一组可以训练的参数W^k,这些参数是可以在节点中共享的。共享的参数可以提高效率，也提供正则化，同时也允许这些方法去生成在训练过程中没有出现过的节点的嵌入。

2.4 Incorporating task-specific supervision

基于编码器-解码器框架的方法默认是无监督的，例如，模型在成对的节点集上被优化或训练去重构成对的相似性度量值$sG(v_i, v_j)$，这仅仅依赖于图G。然而许多节点嵌入的方法，特别是2.3.2小节中提到的方法也可以融入特定任务的监督学习任务。特别地， 为了学习嵌入，这些方法通常结合节点分类任务的监督。考虑二分类问题：

$$\mathcal{L} = \sum_{v_i \in \mathcal{V}} y_i \log(\sigma(\text{ENC}(v_i)^\top \boldsymbol{\theta})) + (1 - y_i) \log(1 - \sigma(\text{ENC}(v_i)^\top \boldsymbol{\theta})). \tag{16}$$

2.5 Extensions to multi-modal graphs

虽然我们关注的是简单的无向图，但许多真实世界的图具有复杂的多模态或多层结构(例如，异构节点和边类型)，并且许多工作已经引入了应对这种异构性的策略。

2.5.1 Dealing with different node and edge types

许多图包含不同类型的节点和边。例如，推荐系统图由两个不同的层组成——用户和内容——而许多生物网络有不同的层，它们之间有不同的相互作用(例如，疾病、基因和药物)。

处理这个问题的一般策略是(1)对不同类型的节点使用不同的编码器和(2)用特定类型的参数扩展成对解码器。例如，在边类型变化的图形中，

$$\text{DEC}_\tau(\mathbf{z}_i, \mathbf{z}_j) = \mathbf{z}^\top \mathbf{A}_\tau \mathbf{z}, \tag{17}$$

公式17的（双线性）编码器会替代标准内积编码器。

2.5.2 Tying node embedding across layers

跨层节点嵌入

某些情况下，不同层之间存在相同节点的副本，这个时候跨层共享嵌入信息使很有益处的。

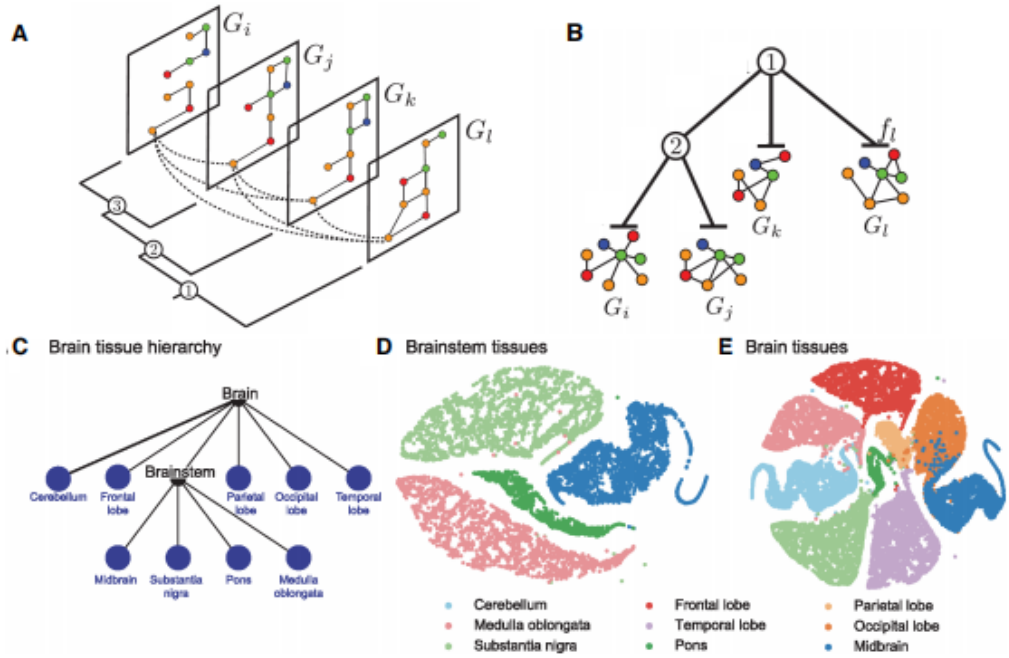


Figure 8: **A**, Example of a 4-layer graph, where the same nodes occur in multiple different layers. This multi-layer structure can be exploited to regularize learning at the different layers by requiring that the embeddings for the same node in different layers are similar to each other. **B**, Multi-layer graphs can exhibit hierarchical structure, where non-root layers in the hierarchy contain the union of the edges present in their child layers—*e.g.*, a biological interaction graph derived from the entire human brain contains the union of the interactions in the frontal and temporal lobes. This structure can be exploited by learning embeddings at various levels of the hierarchy, and only applying the regularization between layers that are in a parent-child relationship. **C-E**, Example application of multi-layer graph embedding to protein-protein interaction graphs derived from different brain tissues; **C** shows the hierarchy between the different tissue regions, while **D** and **E** visualize the protein embeddings generated at the brainstem and whole-brain layers. The embeddings were generated using the multi-layer OhmNet method and projected to two dimensions using t-SNE. Adapted from [61].

2.6 Embedding structural roles

到目前为止，我们回顾的所有方法都优化了节点嵌入，使得图中的邻近节点具有相似的嵌入。然而，在许多任务中，更重要的是学习与节点的结构角色相对应的表示，而不依赖于它们的全局图位置。第2.2.2节中介绍的第2种方法为这个问题提供了一种解决方案，因为格罗弗等人发现，对随机行走进行偏置可以使他们的模型更好地捕捉结构角色(图5)。然而，最近，Ribeiro等人和Donnat等人开发了专门设计来捕获结构角色的节点嵌入方法。

2.7 Applications of node embeddings

- **Visualization and pattern discovery.**可视化与模式发现
- **Clustering and community detection.**聚类与社区检测
- **Node classification and semi-supervised learning.**节点分类与半监督学习
- **Link prediction.**链接预测

3 Embedding subgraphs

上面我们回顾了节点级嵌入（表示）的方法，接下来，我们转向子图或者整个图的表示学习任务。这个任务的目标是：将一组边或节点集合编码进一个低维向量中。

子图的表示学习与图核的设计密切相关，图核定义了子图之间的距离度量。我们回顾的方法不同于传统的图形内核文献，主要在于我们寻求从数据中学习有用的表示，而不是通过内核函数预先指定特征表示。

3.1 Sets of node embeddings and convolutional approaches

有几种子图嵌入技术可以被视为卷积节点嵌入算法的直接扩展(在第2.3.2节中描述)。这些方法背后的基本思想是，它们将子图等同于节点嵌入集。他们使用卷积邻域聚合思想(即算法1)为节点生成嵌入，然后使用附加模块来聚合对应于子图的节点嵌入集。本节中不同方法之间的主要区别是它们如何聚合对应于子图的节点嵌入集。

3.1.1. Sum-based approaches

“convolutional molecular fingerprints”：

$$\mathbf{z}_{\mathcal{S}} = \sum_{v_i \in \mathcal{S}} \mathbf{z}_i, \tag{21}$$

where the embeddings, $\{\mathbf{z}_i, \forall v_i \in \mathcal{S}\}$, are generated using a variant of Algorithm 1.

3.1.2 Graph-coarsening approaches

堆叠卷积和“图粗化”层。 they stack convolutional and “graph coarsening” layers (similar to the HARP approach in Section 2.2.2).

3.1.3 Further variations

内伯特等人[44]和科尔尼等人[34]提出了卷积思想的其他变体。两者都主张使用替代方法来聚集对应于子图的节点嵌入集:科尔斯等人使用“模糊”直方图而不是总和来聚集节点集，并且他们还使用类似于[16]的边缘嵌入层。Neipart等人定义了节点上的排序——例如，使用问题特定的排序或通过使用现成的顶点着色算法——并且使用该排序，他们连接所有节点的嵌入，并且通过标准卷积神经网络架构馈送该连接的向量。

3.2 Graph neural networks （GNNs）

在最初的GNN框架[52]中，每个节点VII用随机嵌入h_i^0初始化，并且在GNN算法的每次迭代中，节点根据以下等式累积来自其邻居的输入:

$$\mathbf{h}_i^k = \sum_{v_j \in \mathcal{N}(v_i)} h(\mathbf{h}_j, \mathbf{x}_i, \mathbf{x}_j), \tag{24}$$

h是任意可微函数，形式为：$R^d \times R^m \times R^m \to R^d$.等式(24)以递归方式重复应用，直到嵌入收敛，并且必须特别小心以确保h是收缩映射。一旦嵌入在K次迭代后收敛，最终的输出嵌入计算为$z_{v_i} = g(h_i^K)$，其中g是g : Rd→ Rd形式的任意可微函数。

李等人[39]扩展和修改了框架，以使用门控递归单元和通过时间的反向传播[14]，这消除了运行方程(24)以收敛的需要。调整框架以使用现代递归单元也允许李等人利用节点属性进行初始化，并使用子图的中间嵌入的输出.

所有这些图神经网络方法原则上都可以用于节点级嵌入任务，尽管它们更经常用于子图级嵌入。为了计算子图嵌入，可以使用第3.1节中描述的任何聚合过程，但是斯卡塞利等人[52]还建议，聚合可以通过引入一个连接到目标子图中所有节点的“虚拟”超级节点来完成。

3.3 Applications of subgraph embeddings

subgraph classification, predict various properties of molecular graphs

4 Important open problems

- **Scalability**.
- **Decoding higher-order motifs**.
- **Modeling dynamic, temporal graphs**.
- **Reasoning about large sets of candidate subgraphs**.
- **Improving interpretability**.