LEC4 Graph Representation Learning for Social Network Analysis

这节课的核心目的是讲解如何将社交网络中的“节点”（例如人）转换成计算机可以处理的“向量”（数字列表），以便进行机器学习任务。

1. 核心主题：图神经网络 (GNN) 与图表示学习

- **什么是 GNN？**

  图神经网络是一种深度学习模型，它的工作原理是**“聚合邻居信息”**。就像你通过一个人的朋友圈来了解这个人一样，GNN 通过聚合一个节点周围邻居的信息来学习该节点的特征。

- **发展历程：** 课件列出了从 2014 年的 **DeepWalk** 到 2017 年的 **GraphSAGE** 再到后来的 GIN 的发展时间线。

Remarks: The term *Graph Neural Network* was first introduced by Gori et al. (2005) and further consolidated by Scarselli et al. (2009); modern “graph convolution” variants became popular later (e.g., Bruna et al., 2013; Kipf & Welling, 2017).

DeepWalk

课件花了很大篇幅详细介绍 **DeepWalk** (Perozzi et al., 2014)，这是图学习领域的奠基性算法之一。

- **核心思想 ：**

  DeepWalk 借用了自然语言处理（NLP）的思想。

- 在 NLP 中，我们分析句子中的词序来理解词义。
- 在 DeepWalk 中，它通过**随机游走（Random Walks）**在图中生成一系列节点的序列。
- **类比：**
  - 图中的**随机游走路径** ≈ 自然语言中的**句子**。
  - 图中的**节点** ≈ 句子中的**单词**。
- 通过这种方式，它可以使用类似 Word2Vec 的技术来学习节点的向量表示（Embeddings）。

- **为什么要这么做？**

  现实世界的社交网络非常**稀疏**（Sparse），直接使用传统的统计方法很难处理。DeepWalk通过无监督学习，将稀疏的图结构转化为稠密的向量，捕捉了节点之间的“社区相似性”。

X

	Talkative	Outgoing	Sociable
v_1	0.4	0.3	0.2
v_2	0.9	0.8	0.9
v_3	0.3	0.2	0.6
v_4	?	?	?

E.g., vector representing $v_1 = (\Phi_{1,1}, \Phi_{1,2}, \ldots, \Phi_{1,d})$

Embeddings based on network structure

$$X_E = \begin{bmatrix} \Phi(v_1) \\ \Phi(v_2) \\ \Phi(v_3) \\ \Phi(v_4) \end{bmatrix} = \begin{bmatrix} \phi_{1,1} & \phi_{1,2} & \cdots & \phi_{1,d} \\ \phi_{2,1} & \phi_{2,2} & \cdots & \phi_{2,d} \\ \phi_{3,1} & \phi_{3,2} & \cdots & \phi_{3,d} \\ \phi_{4,1} & \phi_{4,2} & \cdots & \phi_{4,d} \end{bmatrix}$$

3. 任务定义：关系分类 (Relational Classification)

- **问题定义 ：**

  目标是将社交网络中的成员分类（例如：预测谁会购买某样商品，或者谁是潜在的流失用户）。

- **传统 vs. 关系型 (Slide 8)：**
  - **传统分类：** 只看个人的特征（如年龄、性别），假设人与人之间是独立的（IID假设）。
  - **关系分类：** 不仅看个人特征，还利用**网络结构**（即你和谁连接）。如果你的朋友都喜欢某样东西，你可能也会喜欢。

4. 数学定义

- **图 (Graph):** 定义为 $G = (V, E)$，其中 V 是节点，E 是边。
- **嵌入矩阵 (Embedding Matrix X_E):** DeepWalk 的目标是学习一个矩阵，其中每一行代表一个节点的低维向量特征（Latent dimensions）。
- **随机游走公式:** W_{vi} 代表从节点 vi 开始的随机游走，下一步是从邻居中均匀随机选择的。

DeepWalk 的宏观逻辑 (The Logic)

这部分解释了为什么我们可以把“图”当成“语言”来处理。

1. 核心动机：幂律分布 (Power Law Distribution)

- 这是一个非常重要的理论基础。
  - 左图是 YouTube 社交网络中节点出现的频率，右图是维基百科文章中单词出现的频率。
  - **发现：** 它们都遵循**幂律分布**（Power-Law）。也就是说，极少数节点/单词出现频率极高（如“the”或大V用户），而绝大多数节点/单词出现频率很低。
  - **结论：** 既然图的结构统计规律和自然语言这么像，那直接拿处理语言的模型（Deep Learning for NLP）来处理图也是合理的！

2. 核心类比：随机游走 = 句子

- 这是 DeepWalk 的灵魂所在。
  - **语言模型：** 给定一串词，预测下一个词。
  - **DeepWalk：** 给定一串随机游走经过的节点，预测下一个节点。
  - **公式：** $Pr(v_i|\Phi(v_1), \Phi(v_2)...)$。意思是：根据前面的节点路径，计算出现节点 v_i 的概率。

3. 算法流程 (Algorithm)

- **(伪代码):**
  - **输入：** 一张图 G。
  - **外层循环：** 决定每个节点要做几次“起始点”（γ 次）。

- **内层循环：** 从某个节点出发，随机走 t 步（Walk Length）。这就生成了一个“句子”。
- **核心步骤：** 拿到这个“句子”后，扔给 **SkipGram** 模型去学习更新节点的向量（Update representations）。

SkipGram 模型详解 (The Engine)

这部分解释了 DeepWalk 内部的“引擎”——SkipGram 是如何工作的。

1. SkipGram 的目标

- **Slide 17:** 它的目标不是预测下一个词，而是**利用中心词预测上下文窗口（Context Window）内的邻居词**。
- **例子：** 如果句子是 "Man who passes the sentence"，窗口大小 $w = 2$。
  - 对于中心词 "passes"，它需要预测前面两个词 "Man", "who" 和后面两个词 "the", "sentence"。

2. 极其简化的例子："Happy Birthday"

- **Slide 19:** 假设我们的语料库（Corpus）只有几句 "Happy birthday to you"。
  - 词表（Vocabulary）只有 5 个词：Happy, birthday, to, you, Mary。
  - 我们想把每个词压缩成一个 **N=2 维**的向量（比如 `[0.1, 0.9]`）。

3. 生成训练样本 (关键步骤)

这是模型真正“吃”进去的数据格式。假设窗口大小 $w = 1$（只看左右各一个词）：

- **句子：** "Happy birthday to you"
- **中心词 "birthday"：**
  - 它的左边是 "Happy" $\rightarrow$ 生成正样本对：**(birthday, Happy)**
  - 它的右边是 "to" $\rightarrow$ 生成正样本对：**(birthday, to)**
- **训练目标：** 告诉神经网络，当输入是 "birthday" 时，输出 "Happy" 和 "to" 的概率应该要高。

4. 训练结果

- **输入：** 目标词（Target Word）的向量。

- **输出：** 上下文词（Context Word）的概率分布。
- **最终效果：**
  - 在图中，如果两个节点经常在随机游走中**同时出现**（也就是距离很近），SkipGram 就会强迫它们的**向量表示（Embeddings）在空间中非常接近**。

总结整个流程

1. **图 (Graph)** $\rightarrow$ 社交网络。
2. **随机游走 (Random Walk)** $\rightarrow$ 把图结构转化成一串串“节点序列”（类比“句子”）。
3. **SkipGram** $\rightarrow$ 拿着滑动窗口在这些序列上滑，构建 (中心节点, 邻居节点) 的训练对。
4. **优化 (Optimization)** $\rightarrow$ 调整向量，使得拥有相同邻居的节点，其向量也相似。
5. **结果** $\rightarrow$ 每个用户变成了一个向量，可以直接用来做分类或推荐。

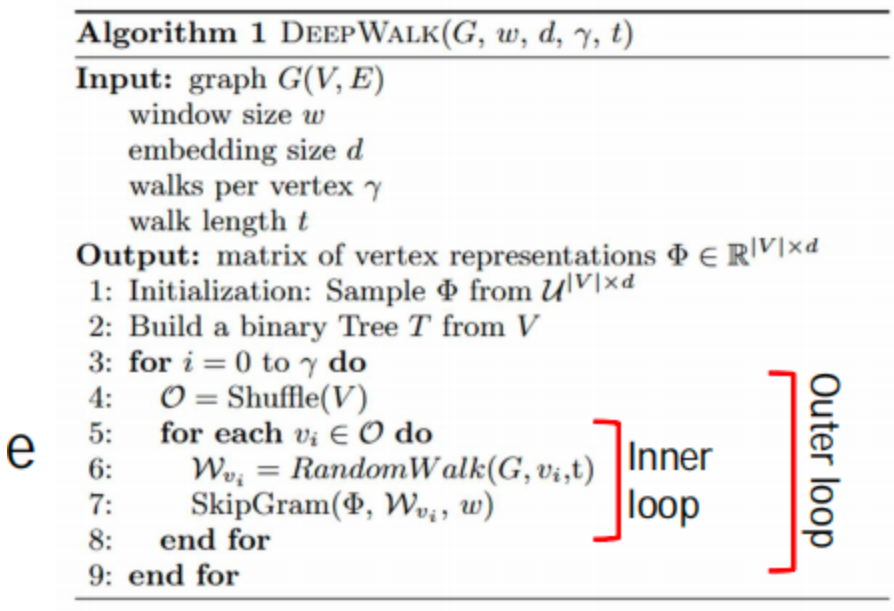
1. 理论依据：为什么图可以像语言一样处理？

这部分解释了 DeepWalk 能够成功的底层逻辑。

- **幂律分布 (Power Law Distribution) 的共性:**
  - 研究人员发现了一个有趣的现象：社交网络中节点出现的频率（有些是大V，大部分是普通人）与自然语言中单词出现的频率（有些是常用词，大部分是生僻词），都遵循**幂律分布**。
  - 左图展示了 YouTube 社交图谱，右图展示了维基百科文本，两者的分布曲线惊人地相似。
  - **结论：** 既然数学分布特征一致，那么用于处理语言的模型（Language Modeling）完全可以直接拿来处理图结构。
- **语言模型的目标:**
  - 在 NLP 中，目标是最大化在一个序列中看到某个词的概率：$Pr(w_n|w_0, ..., w_{n-1})$。
  - 在 DeepWalk 中，随机游走序列就是“句子”，目标变成了：给定一个节点，预测它周围可能出现的邻居节点。

2. 核心算法流程 (Algorithm Workflow)

这是 DeepWalk 的具体执行“食谱”，对应课件中的伪代码。



整个算法分为两个主要循环：

1. **外层循环 (Outer Loop):** 指定每个节点要做多少次“游走发起人”（参数 γ）。
2. **内层循环 (Inner Loop):** 遍历图中的每一个节点 v_i。
    - **Step A - 生成游走 (Random Walk):** 从节点 v_i 出发，随机跳跃 t 步，生成一个序列 W_{vi}。例如：`[v1, v2, v4, v2...]`。
    - **Step B - 滑动窗口 (SkipGram):** 拿着一个长度为 w 的窗口在这个序列上滑动。
    - **Step C - 更新参数:** 使用梯度下降更新节点的向量表示，使得中心节点能更好地预测上下文节点。

3. 关键优化：分层 Softmax (Hierarchical Softmax)

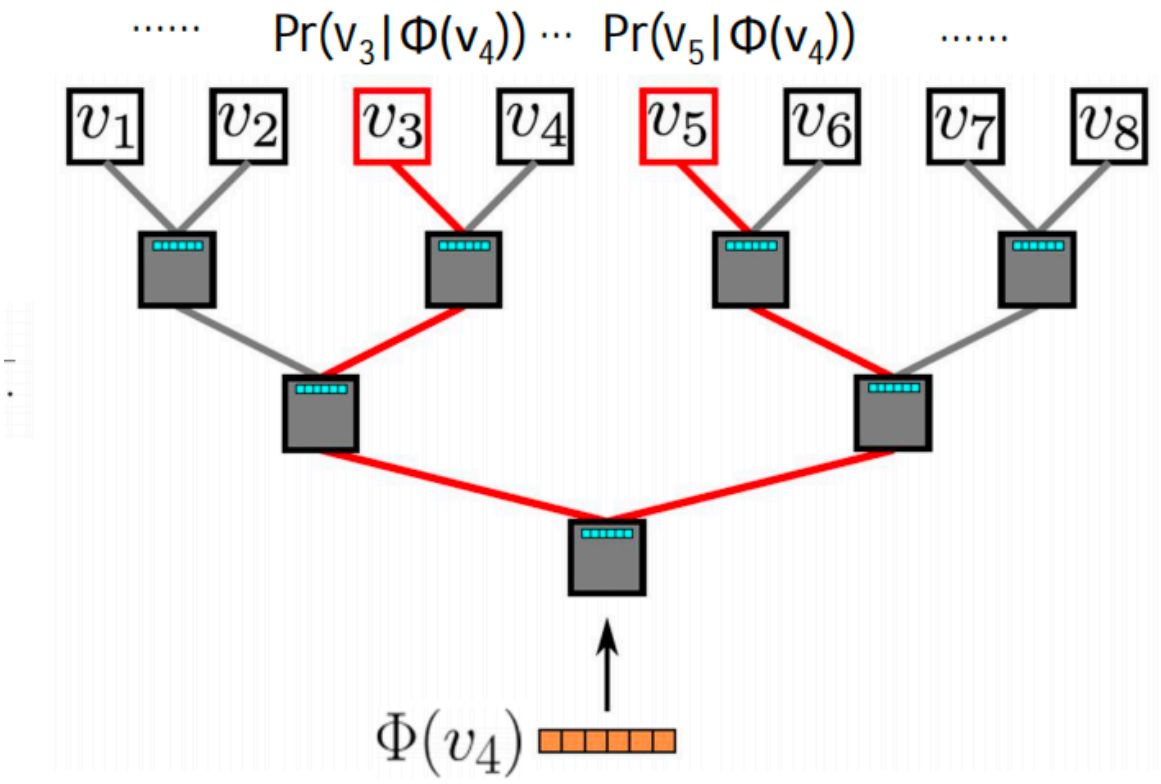
这是一个非常硬核的工程优化细节，解决了计算效率问题。

- **问题:**

  在计算概率时，如果图有 100 万个节点，每次计算都要做 100 万次分类预测（分母要是所有节点的指数和），计算量太大，根本算不动。

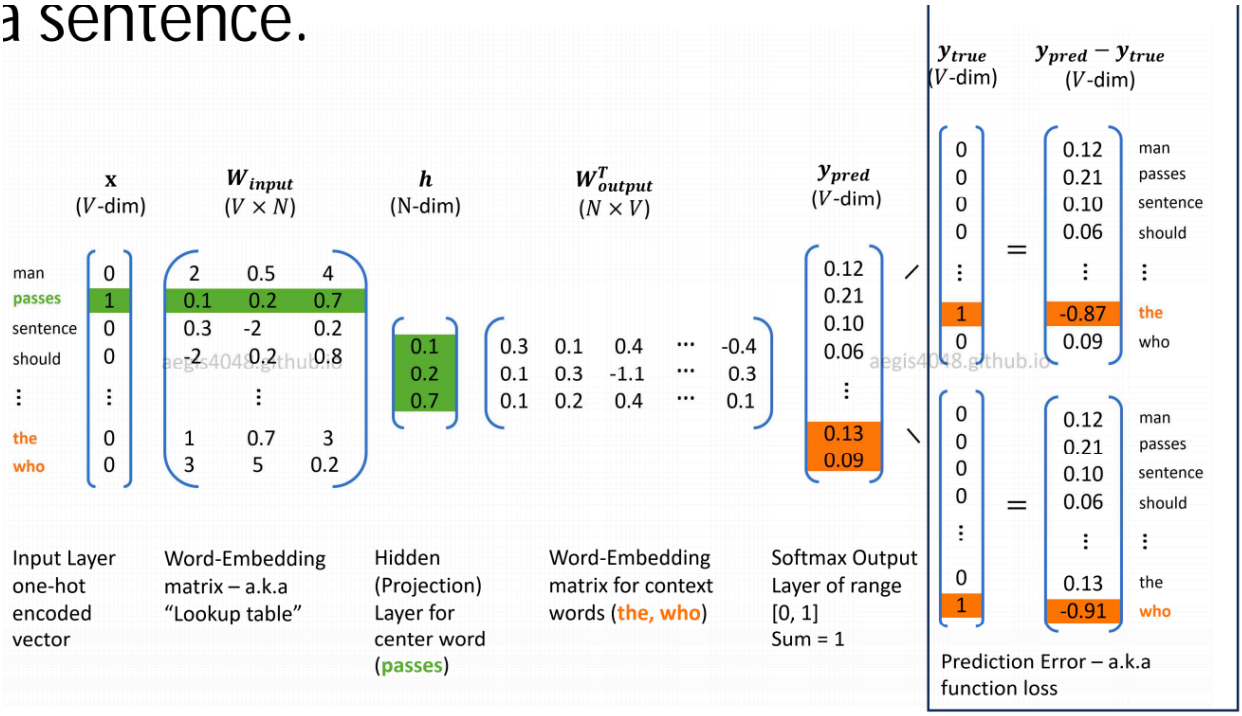
- **解决方案:**

  DeepWalk 使用了**二叉树 (Binary Tree)** 结构。



  - 它把所有节点挂在树的叶子上。
  - 预测一个节点 v_3 不再是“从 100 万个里挑 1 个”，而是变成了“在树上走路径”。
  - 例如，从根节点开始，向左走还是向右走？如果是向左，概率是多少？
  - **效果：** 计算复杂度从线性的 $O(|V|)$ 降到了对数的 $O(\log |V|)$。这使得 DeepWalk 能够处理像 YouTube 这种拥有百万级节点的超大网络。

4. 简易算例演示 (Simplified Example)



课件用了一个只有 4 个节点 (v_1, v_2, v_3, v_4) 的微型图来演示全过程。

- **初始化 (Initialization):**
  - 设定嵌入维度 $d = 2$（每个节点用两个数字表示）。
  - 初始矩阵 Φ 可能全为 0 或随机数。
- **生成游走 (Step 2):**
  - 生成了 4 条随机游走路径，长度 $t = 6$。
  - 例如第一条路径 $W_{v3} = [v_3, v_2, v_1, v_3, v_2, v_4]$。
- **训练更新 (Step 3):**
  - 利用 SkipGram 处理这些路径。
  - 图中展示了具体的误差反向传播过程：根据预测值 (y_{pred}) 和真实值 (y_{true}) 的差异（Error），去调整矩阵 Φ 中的数值（橙色列）。
- **最终结果:**
  - 你会得到一个矩阵 X_E。

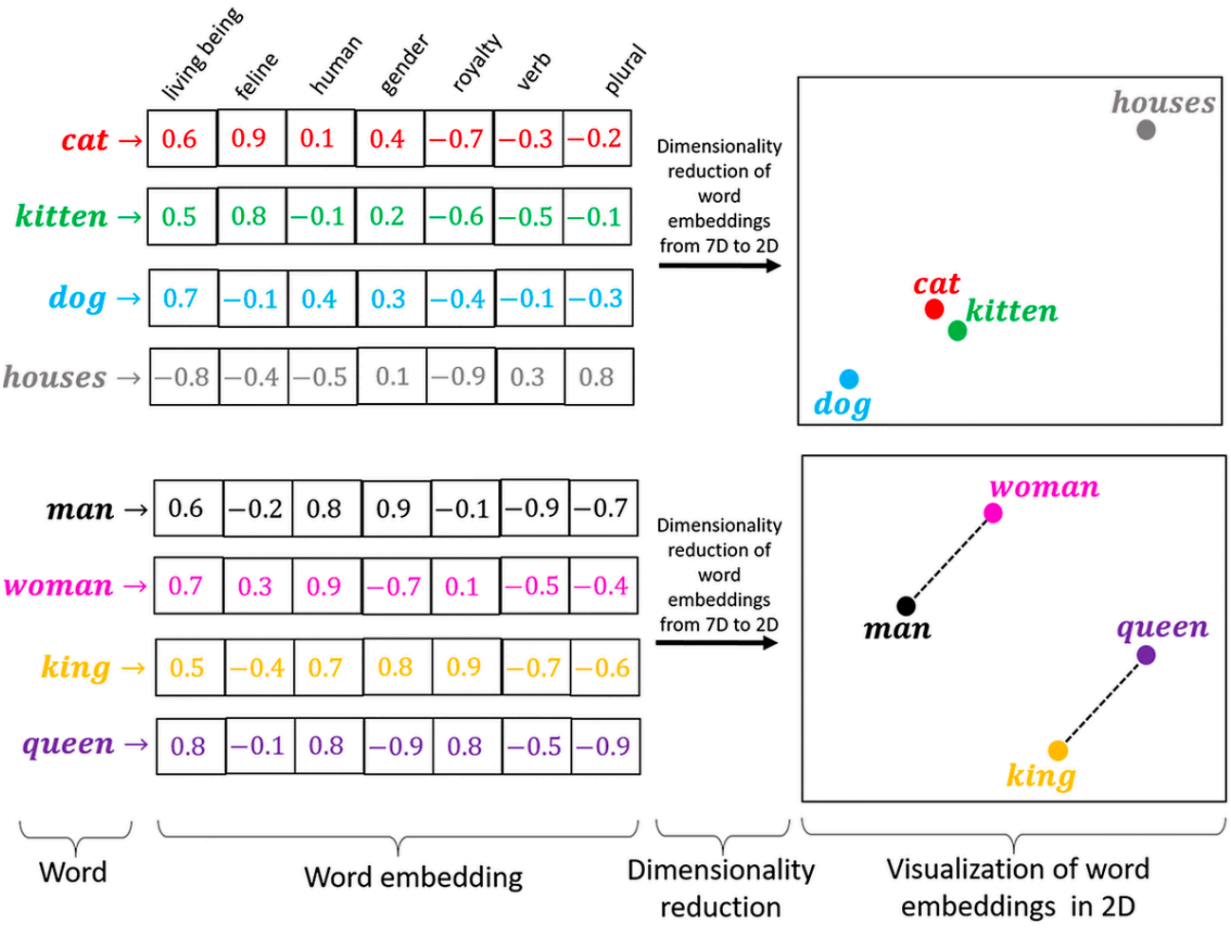
- 注意看结果矩阵：v_4 和 v_2 的向量非常接近（例如 0.89 vs 0.92），这说明它们在随机游走中经常一起出现，属于同一个“社区”。

总结

这套课件完整地讲述了 DeepWalk 如何将**图的拓扑结构**转化为**向量空间**。

1. **输入：** 稀疏的图（只有连接关系）。
2. **过程：** 随机游走 $\rightarrow$ 序列 $\rightarrow$ SkipGram (配合分层 Softmax 加速)。
3. **输出：** 稠密的向量矩阵（捕捉了谁和谁是邻居、谁和谁相似的信息）。

最后，课件提到了在 **YouTube 网络**上的实验，DeepWalk 能够有效处理 114 万节点和 299 万条边的巨大网络，而传统的谱聚类（Spectral Clustering）方法因为计算量太大直接失败了。



1. DeepWalk 的战绩与“死穴”

DeepWalk 的高光时刻：YouTube 实验

课件首先展示了 DeepWalk 在 YouTube 社交网络（114万节点）上的实验结果：

- **极度稀疏数据的胜利：** 即使只有 **1%** 的节点有标签（Labeled Data），DeepWalk 的分类准确率（Micro-F1）依然很高（约 38%），且随着标签数据增加稳步提升。
- **碾压传统方法：** 相比之下，传统的谱聚类（Spectral Clustering）等方法要么准确率低，要么因为计算量过大在这么大的图上直接运行失败。

DeepWalk 的致命缺陷：静态图限制

然而，Slide 36 指出了 DeepWalk 在现实世界（如 YouTube、Facebook）中遇到的巨大挑战：

- **新节点问题 (New Node Problem)：** 现实网络是动态的，每分钟都有新用户注册或新视频上传。
- **重训成本高：** DeepWalk 学习的是一个固定的“查找表”（Embedding Matrix）。如果来了一个新节点，表中没有它的位置，DeepWalk 无法直接生成它的向量，必须重新训练整个模型或进行昂贵的增量更新。

2. 核心概念转折：直推式 vs. 归纳式

为了解决上述问题，课件引入了两个极其重要的机器学习概念：

- **直推式学习 (Transductive Learning) —— 如 DeepWalk:**
  - **特点：** 训练时必须看到**所有**节点。
  - **比喻：** 就像“背答案”。模型记住了训练集中每个人的特征，但如果考试出了一道没见过的新题（新节点），它就懵了。
  - **产出：** 针对特定节点的特定向量。
- **归纳式学习 (Inductive Learning) —— 如 GraphSAGE:**
  - **特点：** 学习的是**规则**（即聚合函数）。
  - **比喻：** 就像“学公式”。模型学会了如何解题。即使遇到没见过的新节点，只要套用公式（看看它的邻居是谁），就能算出它的向量。

- **产出：** 一个可以应用于任何节点的**函数**（Aggregator）。

3. GraphSAGE：图采样与聚合 (Sample and Aggregate)

核心直觉："Tell me who your friends are..."

GraphSAGE 的名字来源于 **S**ample（采样）和 **aggre**gate（聚合）。它的核心思想非常直观：

- 一个节点的特征不应该只由它自己决定，还应该由它周围的邻居决定。
- **图示解释：** 想要生成节点 A 的向量，GraphSAGE 会查看它的邻居 B、C、D。把 B、C、D 的信息"聚合"（比如取平均值），然后和 A 自己的信息结合，形成 A 的新特征。

算法流程 (Forward Propagation)

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

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 $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$; Set the initial node embedding to be the node's feature vector

for $k = 1...K$ **do** Iterate over all the k-hop neighbourhoods

  for $v \in \mathcal{V}$ **do** Iterate through all the nodes in the graph

    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$; Run the aggregate-update cycle for every node in this k-th iteration

    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$

  end

  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$ Normalize the node embedding

end

$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

https://medium.com/analytics-vidhya/ohmygraphs-graphsage-and-inductive-representation-learning-ea26d2835331

GraphSAGE 生成向量的具体数学步骤：

1. **初始化：** 每个节点的初始向量就是它自身的特征（比如用户的个人资料文本）。
2. **聚合 (Aggregate):** 对于每一层 k（代表第几跳邻居）：
   - $h_{N(v)}^{k-1}$: 收集节点 v 所有邻居在上一层的特征，并通过聚合函数（如 Mean, LSTM, Pool）融合成一个向量。
3. **拼接与更新 (Concat & Update):**

- $h_v^k \leftarrow \sigma(W^k \cdot \mathrm{CONCAT}(h_v^{k-1}, h_{N(v)}^{k-1}))$
- 把“节点自己上一层的特征”和“聚合来的邻居特征”拼在一起，乘上一个权重矩阵 W，再过一个激活函数 σ。

4. **归一化 (Normalize):** 把向量长度归一化，保持数值稳定。

总结

这部分内容完成了一个从**“基于查找表”（DeepWalk）到“基于图神经网络”**（GraphSAGE）的范式转移。

- **DeepWalk** 适合**静态、中小型**的图，它关注图的**拓扑结构**。
- **GraphSAGE** 适合**动态、超大型**的图（包含节点特征），它关注如何利用**邻居信息**来推导节点特征，真正实现了 GNN 的“泛化能力”。

我们可以把 GraphSAGE 的工作流程分为三个核心部分来理解：**输入准备**、**前向传播（生成向量）**、以及**反向传播（模型训练）**。

1. 核心机制：前向传播 (Forward Propagation)

这是 GraphSAGE 生成节点向量的具体步骤，也就是课件中提到的“算法”部分。它的逻辑非常像我们在剥洋葱，一层一层地向外获取信息。

- **第 0 层 (Layer-0):**
  - 每个节点的初始向量就是它本身的**原始特征**（Input Features x_v）。比如用户的文本简介、图片特征等。
- **第 k 层 (Layer-k):**
  - **步骤 A - 聚合 (Aggregate):**
    - 这是 GNN 的精髓。对于节点 v，模型把它的所有邻居 $N(v)$ 在上一层 ($k-1$) 的特征拿过来，通过一个聚合函数（AGGREGATE）揉成一个新的向量 $h_{N(v)}^k$。
    - *直觉：* 这一步相当于在问：“我的朋友们都在想什么？”
  - **步骤 B - 拼接 (Concat):**

    - 把“聚合来的邻居信息” ($h_{N(v)}^k$) 和“节点自己上一层的信息” (h_v^{k-1}) 拼在一起。
    - *直觉：* 既要听朋友的，也不能忘了自己原本是谁。
  - **步骤 C - 变换 (Transform):**
    - 把拼接后的长向量乘以一个权重矩阵 W^k，然后通过一个非线性激活函数（如 ReLU 或 Sigmoid σ）。这就得到了节点在第 k 层的新表示。
  - **步骤 D - 归一化 (Normalize):**
    - 最后把向量长度归一化（除以二范数），保证数值稳定。

2. 三种聚合器 (Aggregator Architectures)

GraphSAGE 并不限制你具体怎么“聚合”邻居信息，课件介绍了三种方法：

1. **平均聚合 (Mean Aggregator):**
   - 简单地把所有邻居的向量取平均值。这是最常用的。
2. **LSTM 聚合 (LSTM Aggregator):**
   - 把邻居当成一个序列（Sequence）来读。
   - *优点：* 表达能力强，能捕捉复杂依赖。
   - *缺点：* 邻居本来是没有顺序的（图是无序的），强行排序可能会有问题，且速度慢。
3. **池化聚合 (Pooling Aggregator) - "Highlight Reel":**
   - 先让每个邻居过一个神经网络，然后对每个特征维度取**最大值 (Max Pooling)**。
   - *课件中的数值例子:*
     - 节点 v 有三个邻居，向量分别是 `[0.5, 1, 0.2]`, `[0, 1, 1]`, `[0, 1, 0.4]`。
     - Max Pooling 就像挑“高光时刻”：第一维最大是 0.5，第二维最大是 1，第三维最大是 1。
     - 聚合结果为 `[0.5, 1.0, 1.0]`（课件示例中可能有笔误或特定上下文，Slide 53 显示结果为 `[0.5, 1.0, 0.5]`，这通常意味着取的是各维度的最大值）。
   - *直觉：* 这能捕捉到邻居中最显著的特征（例如，只要有一个朋友是“黑客”，你就有“黑客圈”的属性）。

3. 如何训练：图损失函数 (Graph-Based Loss Function)

既然 GraphSAGE 是无监督学习（Unsupervised），它怎么知道自己学得对不对呢？

它使用了一个基于**负采样 (Negative Sampling)** 的损失函数：

$$J_G(z_u) = -\log(\sigma(z_u^T z_v)) - Q \cdot \mathbb{E}_{v_n} \log(\sigma(-z_u^T z_{v_n}))$$

这个公式看起来复杂，其实含义很简单：

- **前半部分 (Positive Sample):**
  - z_u 和 z_v 是邻居（随机游走中共同出现）。
  - 我们希望它们的向量点积（Similarity）越大越好。
  - $\sigma(z_u^T z_v) \approx 1 \rightarrow -\log(1) = 0$ (损失最小)。
- **后半部分 (Negative Sample):**
  - z_{v_n} 是随机抽取的“路人甲”（非邻居）。
  - 我们希望它们**不**相似（推开）。
  - 由于公式里有个负号 $-z_u^T z_{v_n}$，我们希望这个值大，意味着原始相似度要尽可能小（负数）。

总结

这节课完整地展示了 GraphSAGE 的数学美感：

1. **输入：** 利用节点本身的特征（这是 DeepWalk 做不到的）。
2. **过程：** 通过“聚合-拼接-变换”的流水线，把邻居的信息一层层传导过来。
3. **训练：** 利用“拉近邻居，推开路人”的损失函数来优化权重矩阵 W。

最终，通过训练好的 W 和聚合函数，哪怕来了一个全新的节点，我们只要看看它的特征和它的邻居，就能立刻算出它的 Embedding，完美解决了动态图的问题。

1. 终极对决：Reddit 社区预测实验

为了验证 GraphSAGE 是否真的解决了“新节点”问题，课件展示了一个非常经典的实验：**预测 Reddit 帖子属于哪个版块**（比如是属于 r/Science 还是 r/Sports）。

- **实验设置 (Setup):**
  - **数据：** 23万个帖子，如果同一个用户在两个帖子下都评论过，这两个帖子就连一条边。
  - **特征：** 帖子的标题和评论内容的文本特征（GloVe 向量）。
  - **关键点——时间切分：** 训练集是前 20 天的帖子，**测试集是剩下的新帖子**。这意味着模型必须对它**从未见过**的节点进行预测（Inductive Setting）。
- **实验结果 (The Result):**
  - **DeepWalk (惨败):** F1分数只有 **0.324**。
    - *原因：* DeepWalk 是"直推式"的，它只记住了前 20 天的图结构。面对新来的帖子，它无法生成向量，只能瞎猜。
  - **GraphSAGE (完胜):** 各种变体（GCN, Mean, LSTM, Pool）的 F1 分数都超过了 **0.90**。
    - *原因：* GraphSAGE 学会了"如何聚合邻居"的规则。新帖子来了，只要看看它的邻居是谁，套用规则就能算出向量。
- **采样效率:** 右下角的图表显示，采样邻居数量在 **25** 个左右时性价比最高。再增加邻居数量，准确率提升不明显，但计算时间会线性增加（Diminishing returns）。

2. DeepWalk vs. GraphSAGE：全方位对比

Slide 57 提供了一个非常棒的总结表，这通常是考试重点：

特性	DeepWalk	GraphSAGE
模型类型	浅层嵌入 (Shallow Embedding) *就像查字典*	深度生成式 (Deep Generative) *就像学公式*
处理新节点	**Transductive (直推式):** 必须重新训练才能处理新节点。	**Inductive (归纳式):** 可以直接为从未见过的节点生成向量。
节点特征	**忽略特征:** 只利用图的结构（谁连着谁）。	**利用特征:** 显式地聚合节点的文本/图像特征。
最佳场景	小型、静态的图。	大型、动态的图（如社交媒体 feed）。

3. GNN 的家族进化史 (2017-2019)

Slide 58 梳理了图神经网络的发展脉络，解释了每一代模型是为了解决什么问题而诞生的：

1. **GCN (2017):**
    - 开山之作，但它是 Transductive 的，需要把整张图放进显存训练，难以扩展到大图。
2. **GraphSAGE (2017):**
    - 解决了**扩展性**和**动态图**问题。引入了“采样”机制。
    - *缺点：* 它认为所有邻居的重要性是一样的（除非用 LSTM，否则主要是取平均或最大值）。
3. **GAT (Graph Attention Networks, 2018):**
    - 引入了**注意力机制 (Attention)**。
    - *核心：* “不是所有邻居都平等”。比如预测你的论文方向，你的导师节点应该比你的舍友节点权重更大。
4. **GIN (Graph Isomorphism Network, 2019):**
    - 解决了**表达能力**问题。证明了 GCN 和 GraphSAGE 在区分某些特定图结构时能力不足，GIN 使用 Sum Aggregator 达到了理论上的最强表达力。

4. 未来展望：Graph Transformers (2020-Present)

最后一页（Image 2）指出了 GNN 目前的瓶颈和最新的解决方案。

- **传统 GNN 的痛点：**
    - GNN 依赖“消息传递”。如果想获取距离你 10 跳（Hops）以外的节点信息，就需要堆叠 10 层网络。
    - **过度平滑 (Oversmoothing):** 层数多了之后，所有节点的向量趋同，变得无法区分（Wash out）。
- **解决方案：Graph Transformers**
    - 借用 NLP 中 Transformer 的**全局注意力 (Global Attention)** 机制。
    - **All-to-All:** 允许图中的任意一个节点直接关注到图中的任意另一个节点，无论它们在空间上距离多远。
    - *代表作：* Graphormer（在分子属性预测等科学领域表现SOTA）。

总结

这节课从 DeepWalk 的**随机游走**开始，讲到了 GraphSAGE 的**邻居聚合**，最后以 Graph Transformer 的**全局注意力**结束。这不仅是一节算法课，也是一部图深度学习从“以前怎么做”到“现在怎么做”再到“未来怎么做”的进化史。