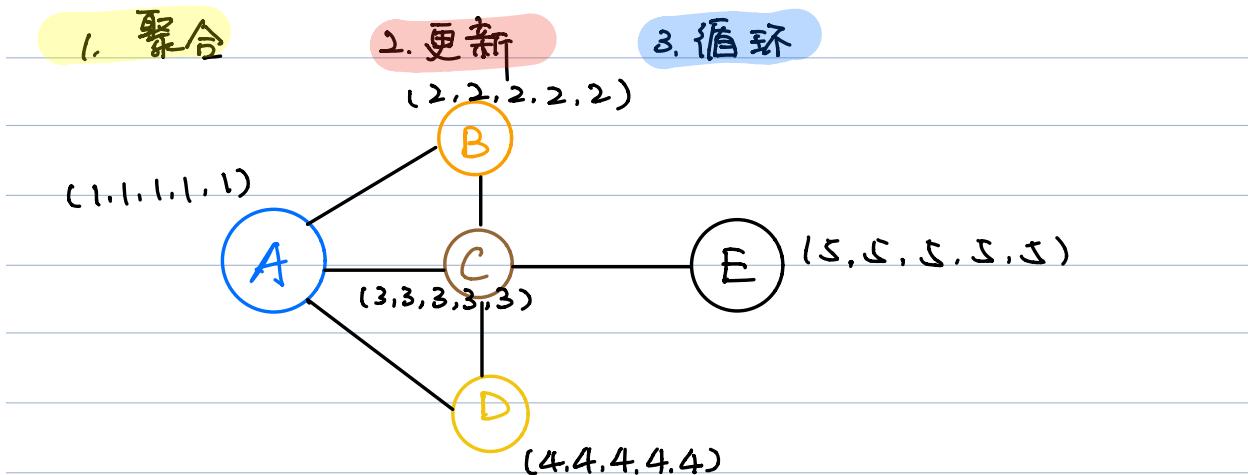


## GNN 流程



### 聚合:

经过一次聚合后得到的信息:

$a \text{ 乘 } b \text{ 的特征值}$

$$\text{邻居信息 } N = a * (2, 2, 2, 2, 2) + b * (3, 3, 3, 3, 3) \\ + c * (4, 4, 4, 4, 4)$$

邻居信息  $N$  是对自己的一个补充

### 更新:

$$A \text{ 的信息.} = \sigma(w((1,1,1,1,1) + \alpha * N))$$

$\sigma$  是激活函数 (relu, sigmoid 等等)

$w$  是模型需要训练的参数

### 循环

经过一次聚合后: A 有 B, C, D 的信息.

B 有 A, C 的信息.

C 有 A, B, D, E 的信息.

D 有 A, C 的信息,

E 有 C 的信息.

第二次聚合之后以此类推，以 A 节点为例

此时 A 聚合 C 的时候，C 中有上一层聚合到的 E 的信息。

所以此时 A 获得了二阶邻居 E 的特征

### GNN 的用途

通过聚合更新，到最后可以得到每个节点的表达。

也就是 feature，此时：

结点分类就可以直接拿去分类，算 loss，优化前面提到的 W

关联预测就最简单的办法：两个节点的特征一拼，拿去做分类，

一样的算 loss，优化

归根到底，GNN 就是个提取特征的方法

### GNN 和传统 NN 的区别

#### ① 对于图结构而言，并没有天然的顺序而言

- 标准的神经网络比如 CNN 和 RNN 不能适当地处理图结构输入（因为它们都需要节点的特征按照一定的顺序进行排列）

- GNN 采用在每个节点上分别传播 (propagate) 的方式进行学习，由此忽略了节点的顺序，相当于 GNN 的输出会随着输入的不同而不同

#### ② 图结构的边表示节点之间的依存关系

- 传统的神经网络中，依存关系是通过不同节点特征来间接表达节点之间的关系

- GNN 通过邻居节点的加权求和更新节点的隐藏状态。

## GNN 分类

- 图卷积网络 (Graph Convolutional Networks) 和  
图注意力网络 (Graph Attention Networks)  
⇒ 因为涉及传播步骤 (propagation step)
- 图的空域网络 (spatial-temporal networks)  
因为该模型通常用在动态图 (dynamic graph) 上
- 图的自编码 (auto-encoder) ⇒ 通常使用无监督学习  
(unsupervised)
- 图生成网络 (generative network)

## 数学表达式

GNN 的目标是学习得到一个状态的嵌入向量 (embedding)

$h_v \in R^s$ , 这个向量包含每个节点的邻居节点的信息

$h_v$ : 节点  $v$  的状态向量

$f$ : 局部转化函数 (local transition function), 这个函数在所有节点中共享, 并根据邻居节点的输入来更新节点状态

$g$ : 局部输出函数, 用于描述输出的产生方式

$o_v$ : 输出

节点  $v$  的状态向量  $\underline{h_v} = f(x_v, x_{co[v]}, h_{ne[v]}, x_{ne[v]})$

$x_v$ : 节点  $v$  的特征向量,  $x_{co[v]}$ : 节点  $v$  边的特征向量

$\underline{h_{ne[v]}}$ : 节点  $v$  邻居节点的状态向量  $x_{ne[v]}$ : 节点  $v$  邻居节点特征向量

$$O_v = g(h_v, x_v)$$

将所有的状态向量，输出向量，特征向量叠加起来分别使用矩阵  $H, O, X$  和  $X_N$  来表示，可以得到更加紧凑的表示：

$$H = F(H, X)$$

全局转化函数 (global transition function)

$$O = G(H, X_N)$$

全局输出函数 (global output function)

一个不动点

并且在  $F$  为收缩映

射下  $H$  有唯一的定  
义

GNN 使用如下的传统迭代方法来计算状态向量：

$$H^{t+1} = F(H^t, X)$$

$H$  的第  $t$  个迭代周期的张量

按照指数级速度收敛到最终的不动点解

$$\text{Loss} = \sum_{i=1}^P (t_i - O_i) \Rightarrow \text{使用梯度下降法}$$

步骤：

- 状态  $h^t$  按照迭代方程更新  $T$  个轮次，这时得到的  $H$  会接近不动点的解  $H(T) \approx H$

- 权重  $W$  的梯度从 loss 计算得到

- 权重  $W$  根据上一步中计算的梯度更新

## GNN 一般框架

### ① Message Passing Neural Network ( $MPNN = GNN + GCN$ )

- 种 general framework，用于在 graph 上进行监督学习：

- message passing phase 和 readout phase.

信息传递阶段就是前向传播阶段，该阶段运行T个steps，并通过 $M_t$ 数获取信息，通过 $U_t$ 数更新节点。

$$M_v^{t+1} = \sum_{w \in N_v} M_t(h_v^t, h_w^t, e_{vw})$$

$$h_v^{t+1} = U_t(h_v^t, M_v^{t+1})$$

节点v的状态向量

节点v到w的特征向量

readout阶段计算一个特征向量用于整个图的representation  
使用之数R实现

$$\hat{y} = R(\{h_v^T | v \in G\})$$

## (2) Non-local Neural Networks (NLNN)

使用神经网络获取长远依赖信息，一个位置的

non-local 操作使用所有 positions 的特征向量的加权和来作为该位置的信息。

$$h'_i = \frac{1}{C(h)} \sum_j f(h_i, h_j) g(h_j)$$

i: 输出位置(节点)的索引

j: 所有可能位置(节点)的索引

输入  $h_j$  的  
转换系数

得到一个 scalar，用于计算

节点i与节点j之间的关  
联度

· 线性转换系数， $g(h_i) = W_g h_i$  ( $W_g$  为可学习的权重矩阵)

· 高斯系数  $f(h_i, h_j) = e^{h_i^\top h_j}$

$$\Theta(h_i) = W_\Theta h_i \quad \Phi(h_j) = W_\Phi h_j \quad C(h) = \sum_j f(h_i, h_j)$$

- Dot product. 对于函数  $f$  使用点积相似度

$$f(h_i, h_j) = \Theta(h_i)^T \phi(h_j)$$

在这种情况下,  $C(h) = N$ , 其中  $N$  为位置(节点)的个数

- Concatenation. 即

$$f(h_i, h_j) = \text{ReLU}(w_f^T [\Theta(h_i) \| \phi(h_j)])$$

其中,  $w_f$  是一个权重向量, 用于将一个 vector 映射成一个 scalar,  $C(h) = N$

### ③ Graph Networks (MPGG + NLNN + ...)

图的定义.  $G = (u, H, E)$

$H = \{h_i\}_{i=1:N^v}$  表示节点集合,  $h_i$  为节点的属性向量

$u$ : 全局属性

$E = \{(e_k, r_k, s_k)\}_{k=1:N^e}$  为边的集合

$e_k$ : 边的属性向量

$r_k$ : receiver node 索引

$s_k$ : sender node 索引

GN block 的结构: 三个 update functions ( $\phi$ )

和三个 aggregation functions ( $\rho$ )

$$e'_k = \phi^e(e_k, h_{rk}, h_{sk}, u), \quad \bar{e}'_i = \rho^{e \rightarrow h}(E'_i)$$

$$h'_i = \phi^h(\bar{e}'_i, h_i, u) \quad \bar{e}' = \rho^{e \rightarrow h}(E')$$

$$u' = \phi^u(\bar{e}', \bar{h}', u) \quad \bar{h}' = \rho^{h \rightarrow u}(H')$$

其中  $E'_i = \{(e'_k, r_k, s_k)\}_{r_k=i, k=1:N^e}, H' = \{h'_i\}_{i=1:N^v}$

$$E' = \bigcup E'_i = \{(e'_k, r_k, s_k)\}_{k=1:N^e}$$

图神经网络设计基于三个基本原则：

flexible representations

configurable within-block structure

composable multi-block architectures.