番外篇：GNN模型及实现细节

吃草的牛牛
做平凡事，成放心人

> 本文属于对论文《The Graph Neural Network Model》中GNN模型及实现细节的讲解，模型例子基于论文所述的**子图匹配任务**，本文讲述**输入数据的结构、GNN模型实现细节、模型优化**等等。本文任务暂未实现代码，不过模型的Pytorch实现可以参见初探GNN：《The Graph Neural Network Model》。

输入数据

对于子图匹配任务，就是**事先给定一个子图**作为匹配的对象，然后在其它的图中找到**是否包含这样的一个子图**。数据示例如下

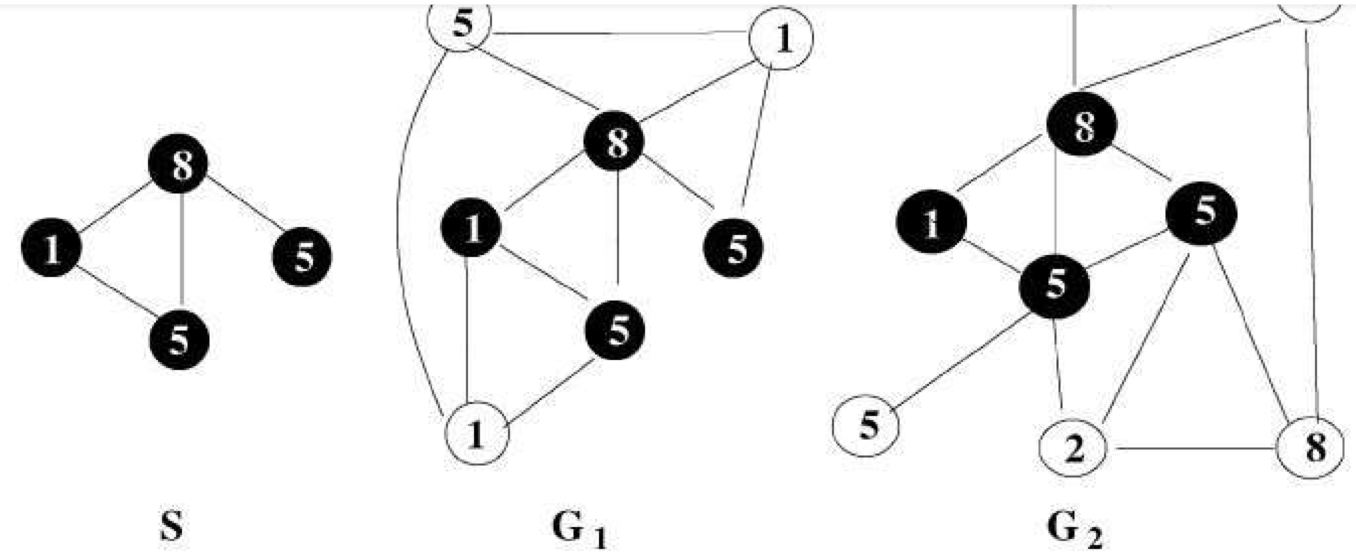


Fig. 4. Two graphs G_1 and G_2 that contain a subgraph S. The numbers inside the nodes represent the labels. The function τ to be learned is $\tau(G_i, n_{i,j}) = 1$, if $n_{i,j}$ is a black node, and $\tau(G_i, n_{i,j}) = -1$, if $n_{i,j}$ is a white node.

最左边的 $\mathbf{S}$ 为给定的子图(subgraph)，右边的两个图 $\mathbf{G}_1$ 和 $\mathbf{G}_2$ 为给定的两个graph数据，**节点中的数字用于标识节点，数字相同的节点属于同种类型的节点**(可以理解为特征相同的节点)，论文**子图匹配的任务**就是**给定子图 $\mathbf{S}$ ，在新的graph上对每个节点进行分类，如果该节点属于子图的一部分，那么为 1 ，否则为 -1** 。在上方的图示中，右边的两个graph的黑色节点由于属于子图 $\mathbf{S}$ ，所以这些黑色节点的标签为 1 ，其余的白色节点的标签为 -1 。总结如下：

- **输入(Input)**：子图 $\mathbf{S}$ 、graph数据 $\mathbf{G}_1, \mathbf{G}_2 \ldots \mathbf{G}_n$ 。对于所有graph，节点只有10种，使用数字0-9进行区分。
- **标签(Label)**：对于graph($\mathbf{G}$)来说，对于其中的某一节点 v ，如果该节点属于子图 $\mathbf{S}$ ，那么节点 v 的标签为 1 ，否则为 -1

$$\text{label}_v = \begin{cases} 1 & \text{if v belongs to } \mathbf{S} \\ -1 & \text{otherwise} \end{cases}$$

- **任务(Task)**：给定一个graph，模型预测该graph中每个节点的标签(1或-1)，即是否属于子图。

模型细节

GNN模型与其它模型的区别在于，**GNN是对每一个节点单独地进行处理和操作，忽略了graph整体的结构，对于GNN的输出，也是在每个节点单独进行、单独输出**，所以，这种模型对于输入graph整体的拓扑结构并不在乎。

> GNN的任务可以分为node-focused和graph-focused，对于node-focused任务，可以直接取每个节点的输出，对于graph-focused任务(比如图

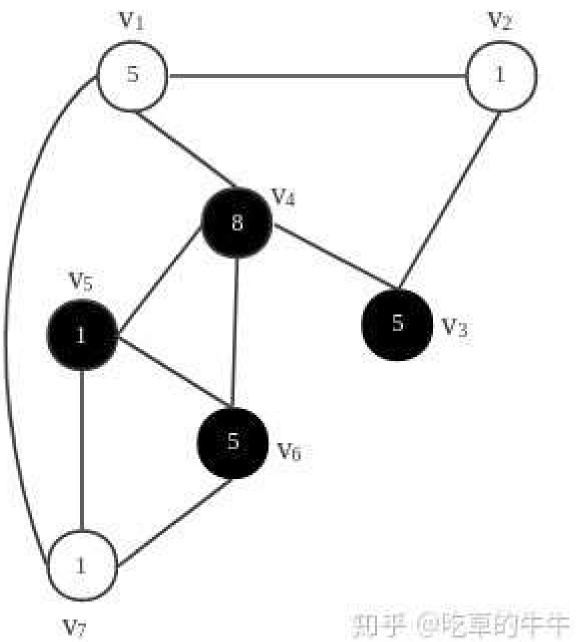
那么，现在考虑如何**对每个节点单独地进行操作**，只要定义好这样的一个操作，那么我们只需要对所有的节点分别进行同样的操作即可

变量定义

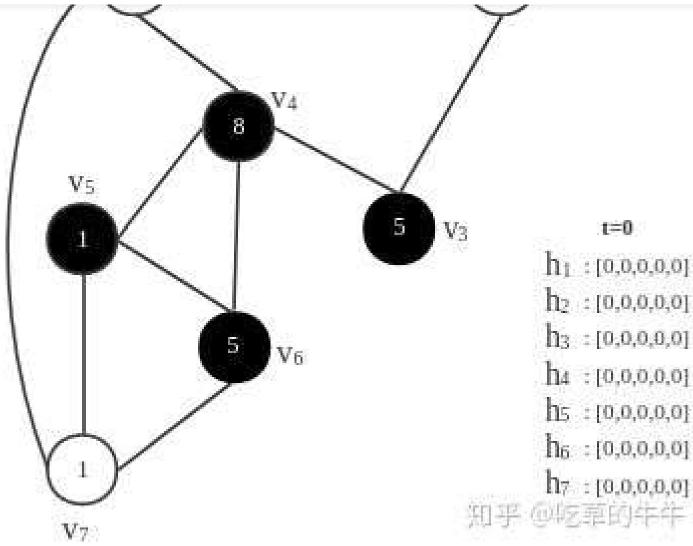
首先定义一下变量，对于图 G ，节点集合为 N ，边集合为 E

- $\mathbf{x_v}$ ：节点 v 特征向量，维度为 $\mathbb{R}^{D_V}$
- $\mathbf{h_v}$ ：节点 v 的状态向量，维度为 $\mathbb{R}^{D_S}$
- $\mathbf{x_{(v_1,v_2)}}$ ：边（v_1,v_2）的特征向量，维度为 $\mathbb{R}^{D_E}$
- $ne[v]$ ：节点 v 邻居节点集合
- $co[v]$ ：节点 v 相连的所有边集合

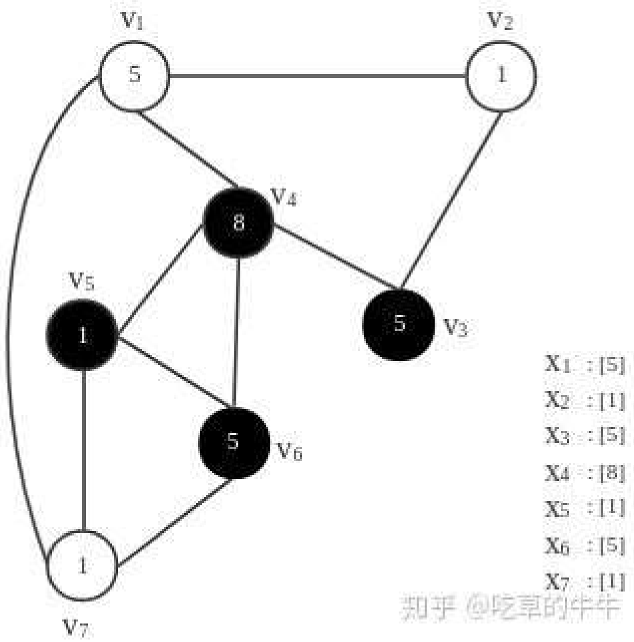
现在以第一张图中的 $\mathbf{G}_1$ 为例，说明以上变量的含义，如下图



- 对于**节点的状态向量**，由于节点初始都没有状态，所以令所有节点的初始状态 $\mathbf{h}_v^{t=0}$ 为全0，即
$$\mathbf{h}_v^0=\underbrace{[0,0,0,\ldots,0]}_{R^{D_S}},v\in V$$
  论文中对状态向量的维度设置为5，所以，每个节点的状态向量如下图，在t=0时刻，所有节点的状态向量都为0。



- **对于特征向量**，使用**节点中的数字表示节点特征**，那么，对于节点 v_1 ，特征向量 $\mathbf{x_1}$ 可以表示为 $\text{array}([5])$ ，对于节点 v_2 ，特征向量可以表示为 $\text{array}([1])$ ，对于节点 v_3 ，与节点 v_1 的特征向量相同，为 $\text{array}([5])$ ，以此类推。
  由此，得到所有节点的特征向量如下，右下角列出所有节点的特征向量



> 这里同样可以使用one-hot向量来表示节点的特征向量，比如，假设总共有10个特征，那么向量长度为10，节点特征向量就是节点数字对应index为1，其它位置为0的向量，即节点 v_1 的特征向量为 $\mathbf{x_1} = \underbrace{[0,0,0,0,1,0,0,0,0,0]}_{10}$ 。

- 对于**边的特征向量**，由于该子图匹配任务不需要边的特征，所以可以忽略边的特征向量或者令边的特征向量为全0。
- 对于**邻居节点**，以节点 v_1 为例，节点 v_1 的邻居节

节点操作涉及到两个方面，一方面，由于刚开始节点的状态初始化全为0，所以需要对节点状态进行更新，通过迭代的方式达到稳定状态，因此，对于节点 v 需要一个**节点状态更新函数** f_w^v ，使得节点的状态由 $\mathbf{h}_v^t \rightarrow \mathbf{h}_v^{t+1}$ ，由此更新节点状态。

> **注**：这里的状态更新函数和节点有关，不同的节点使用不同的参数和函数。

另一方面，对于节点 v ，最终需要一个输出，因此，需要一个**节点输出函数** g_w^v ，用于得到该节点的输出 $\mathbf{o}_v$ 。

> **注**：这里的节点输出函数 g_w^v 同样和节点有关。

- 考虑节点的**状态转化函数** f_w^v ，对于节点 v

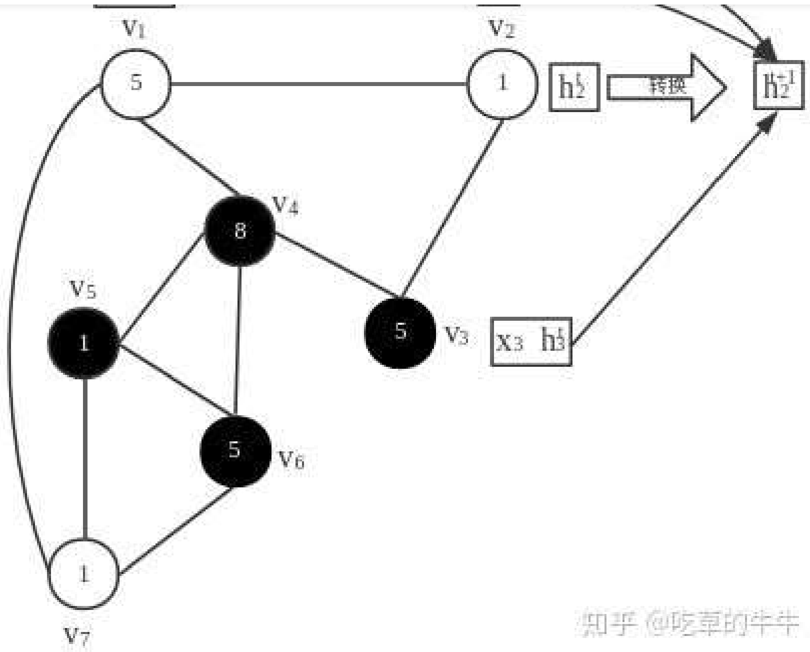
$$\mathbf{h}_v^{t+1} = f_w^v(\mathbf{x}_v, \mathbf{x}_{co[v]}, \mathbf{x}_{ne[v]}, \mathbf{h}_{ne[v]}^t)$$

  根据以上公式，可以看出，为了更新 v 的状态，需要如下变量
  => 该节点 v 自身的特征向量 $\mathbf{x}_v$ ，即节点中的*数字*，对于特定节点，**只有一个**这样的向量
  => 该节点 v 相邻边的特征向量 $\mathbf{x}_{co[v]}$ ，该任务中为全0或者不考虑，对于特定节点，**有co[n]个**这样的向量
  => 该节点 v 相邻节点的特征向量 $\mathbf{x}_{ne[v]}$ ，即相邻节点中的*数字*，对于特定节点，**有ne[n]个**这样的向量。
  => 该节点 v 相邻节点在t时刻的状态向量 $\mathbf{h}_{ne[v]}^t$ ，即相邻节点的状态，对于特定节点，**有ne[n]个**这样的向量

  现在的关键点在于，**如何设计转化函数 f_w^v ，使得该函数与相邻边co[n]的个数以及相邻节点ne[n]的个数无关**，最简单的想法是，对其周围的节点分别使用一个函数 h_w^v 进行转换，然后直接进行求和，即

$$\mathbf{h}_v^{t+1} = \sum_{u \in ne[v]} h_w^v(\mathbf{x}_v, \mathbf{x}_{(v,u)}, \mathbf{x}_u, \mathbf{h}_u^t)$$

  比如，在t时刻，更新节点 v_2 的状态向量 $\mathbf{h}_{v_2}^t$ 为 $\mathbf{h}_{v_2}^{t+1}$ ，这个过程的信息流动如下图



其中，**黑色带箭头实线**表示信息的流动，$\Rightarrow$ **转换**表示节点 v_2 的隐藏状态由 $\mathbf{h}_{v_2}^t$ 转换到了 $\mathbf{h}_{v_2}^t$ 。可以看出，在使用了这样的求和方式之后，**对于任何节点，无论该节点的度(即邻居节点个数)为多少，始终不影响该函数的操作和输出。**

> **注**：这里更新节点状态的时候并没有使用到节点当前时刻的状态。

- 考虑节点的**输出函数** g_w^v ，由于是对每个节点单独使用、单独输出，所以直接使用多层神经网络(全连接)就可以直接输出，即

$$g_w^v(\mathbf{x}_v, \mathbf{h}_v^t) = W_v[\mathbf{x}_v, \mathbf{h}_v^t] + \mathbf{b}_v$$

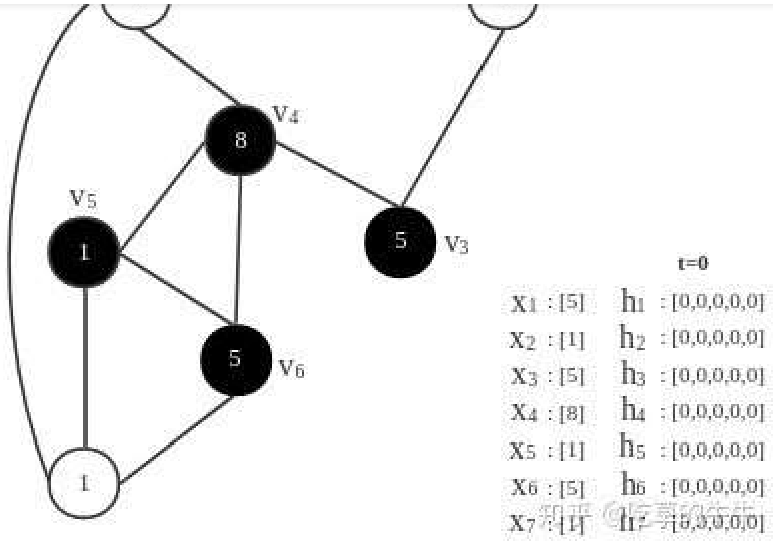
  值得注意的是，这里的输入仅仅是节点 v 自己的向量，输出的值为节点 v 的输出，所以**该函数的操作与图的拓扑结构和其它节点均无关。**

总结一下节点操作，首先，**函数 f_w^v 和 g_w^v 通过上述设计后，状态转换操作和输出操作与** 输入的graph的整体拓扑结构 **以及** 每一个节点周围的结构 **都无关**，对于 f_w^v ，使用**求和的方式**消除节点邻居数量的多样性，对于 g_w^v ，本身就与周围节点以及graph整体结构无关.

其次，**函数 f_w^v 和 g_w^v 的参数 w 都与操作的节点对象有关**，比如，对于节点 v_n ，函数的参数为 w_n ，**在该任务中，可以直接设置参数共享**，即对于所有的节点，函数 f_w^v 的 w 参数共享，可以写成 f_w ，函数 g_w^v 的 w 参数共享，可以写成 g_w 。

Forward过程

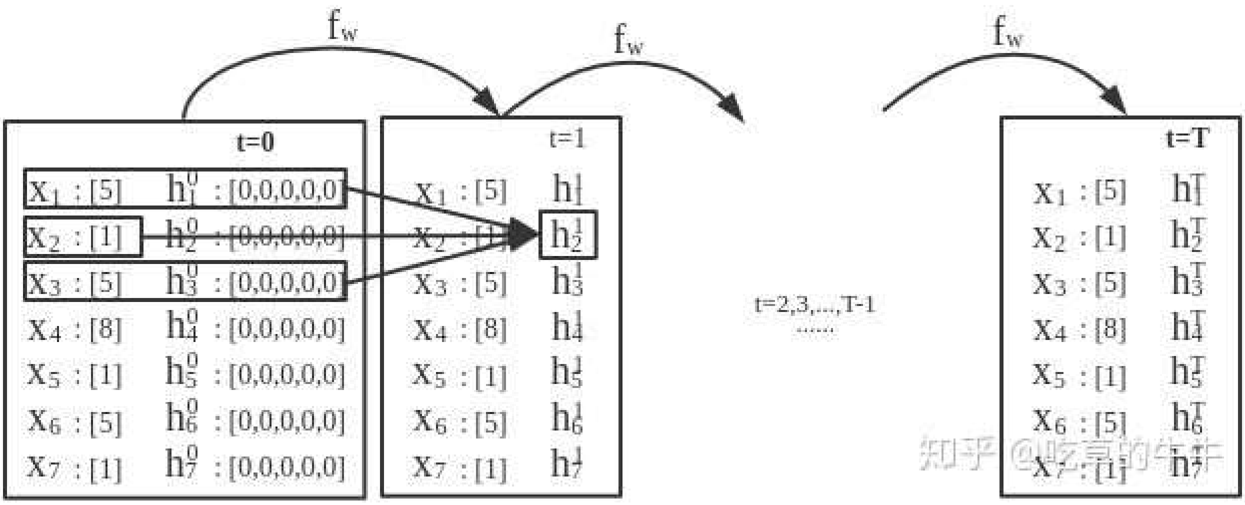
首先，假设现在**输入一个graph数据**(在实现的时候输入的是节点集合 N 以及邻接矩阵 A)，以及**对应每个节点的特征向量**，**初始化状态向量**，如下图



然后，对图中的每一个节点，使用公式

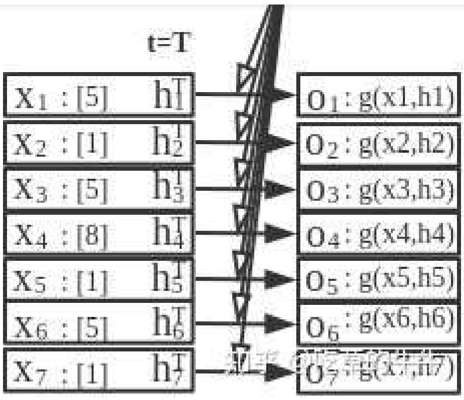
$$\mathbf{h}_v^{t+1} = \sum_{u \in ne[v]} h_w^v(\mathbf{x}_v, \mathbf{x}_{(v,u)}, \mathbf{x}_u, \mathbf{h}_u^t)$$

一直迭代T次，这里的T是一个超参数，需要预先设置，按照迭代时刻展开，变量的变化过程如下



图中标出了t=0时刻到t=1时刻节点 v_2 的隐藏状态由 $\mathbf{h}_2^0$ 转化为 $\mathbf{h}_2^1$ 的信息流动方向，其它节点类似。每一个大方框表示在对应时刻的各个节点变量的值，可以看到，**节点的特征向量 $\mathbf{x}_v$ 在迭代过程中始终是不变的，而状态向量 $\mathbf{h}_v^t$ 是一直变化的，在迭代 T 次之后，状态向量会接近于函数的不动点。**

在迭代次后，得到了第T时刻每个节点的状态向量，然后对每个节点应用函数 g_w ，得到每个节点的输出 $\mathbf{o}_v$ ，该过程如下图



可以看出，**每一个节点的输出，就是将T时刻对应节点的特征向量 $\mathbf{x}_v$ 和状态向量 $\mathbf{h}_v^T$ 使用函数 g_w 变换后的输出。**

得到了节点的真实输出 $\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_7$ 之后，按照之前的节点标签 $\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_7$ ，使用回归的思想求得最终的loss

$$\text{loss} = \frac{1}{7}\sum_{i=1}^{7}(\mathbf{o}_i - \mathbf{t}_i)^2 + \lambda||\theta||_2$$

总结一下，forward过程输入了一个graph数据，以及对应节点特征和状态，然后使用迭代的方式**分别对所有的节点单独应用 f_w 函数**，在这个过程中，节点特征向量不变，状态向量在迭代过程中不断更新；迭代T次后，取T时刻的节点状态，**对所有的节点分别单独应用 g_w 函数**，每个节点都会得到一个输出，然后与节点标签比较求得最终的loss。

Backward过程

在forward过程中求得loss之后，按照反向传播的步骤可以求得参数的梯度，然后使用梯度下降法进行优化，值得注意的是，**在所有涉及到的参数中，哪些变量需要求出对应梯度**

- $\mathbf{x}_v$ 不需要梯度，因为对于输入特定的graph和节点，节点特征已经固定
- $\mathbf{x}_{(u,v)}$ 不需要梯度，同样边的特征也固定
- 函数 f 的参数 w_f 需要梯度，函数 g 的参数 w_g 需要梯度
- **$\mathbf{h}_v^0$ 不需要梯度， $\mathbf{h}_v^1, \ldots \mathbf{h}_v^T$ 需要梯度**

关键在于最后一个，首先 $\mathbf{h}_v^0$ 不需要梯度是**因为最初的 $\mathbf{h}_v^0$ 可以任意赋值，而且，迭代到T时刻的不动点与 $\mathbf{h}_v$ 的初值无关，因此，完全不需要对初值进行优化**，而对于 $\mathbf{h}_v^1, \ldots \mathbf{h}_v^T$ ，forward的迭代过程如下

$$\cdots$$

$$\mathbf{h}_v^{T-1} = f_w(\mathbf{h}_{ne[v]}^T, \ldots)$$

可以看出，由于参数 w 需要梯度，所以在迭代过程中的 $\mathbf{h}_v^1, \ldots \mathbf{h}_v^T$ 也需要梯度，这些梯度是用来求 w 的梯度服务的。

编辑于 2019-08-13

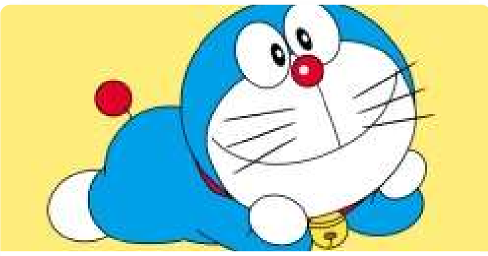
图神经网络（GNN） 深度学习（Deep Learning）

文章被以下专栏收录

GNN阅读合辑

已关注

推荐阅读



初探GNN：《The Graph Neural Network Model》

吃草的牛牛 发表于GNN阅读...

小白都能看懂的神经网络入快收下吧~

优达学城（Udacity）

6 条评论 切换为时间排序

写下你的评论...

赞同 12 6 条评论 分享