Author = Song Jian

#### 阅读论文： The graph neural network model

|  |  |  |
| --- | --- | --- |
| @article{Scarselli2009,  author = {Scarselli, Franco and Gori, Marco and Tsoi, Ah Chung and Hagenbuchner, Markus and Monfardini, Gabriele},  journal = {IEEE Transactions on Neural Networks},  pages = {61--80},  title = {{The graph neural network model}},  year = {2009}  } | F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” IEEE Trans. Neural Networks, vol. 20, no. 1, pp. 61–80, 2009.  [文章下载链接](https://repository.hkbu.edu.hk/cgi/viewcontent.cgi?article=1000&context=vprd_ja) | 锡耶纳大学 |
|  | M. A. Khamsi, An Introduction to Metric Spaces and Fixed Point Theory. New York: Wiley, 2001.  不动点理论 |  |
| @article{Li2016,  author = {Li, Yujia and Zemel, Richard and Brockschmidt, Marc and Tarlow, Daniel},  journal = {4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings},  pages = {1--20},  title = {{Gated graph sequence neural networks}},  year = {2016}  } | Y. Li, R. Zemel, M. Brockschmidt, and D. Tarlow, “Gated graph sequence neural networks,” 4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc., no. 1, pp. 1–20, 2016.  [文章下载链接](https://arxiv.org/pdf/1511.05493.pdf) | 多伦多大学  微软研究院 |

#### 文档由以下几个部分组成：

* 文章简述
* 理论背景
* 论文实例
* 基于矩阵的GNN实现
* 代码部分

#### 文章简述

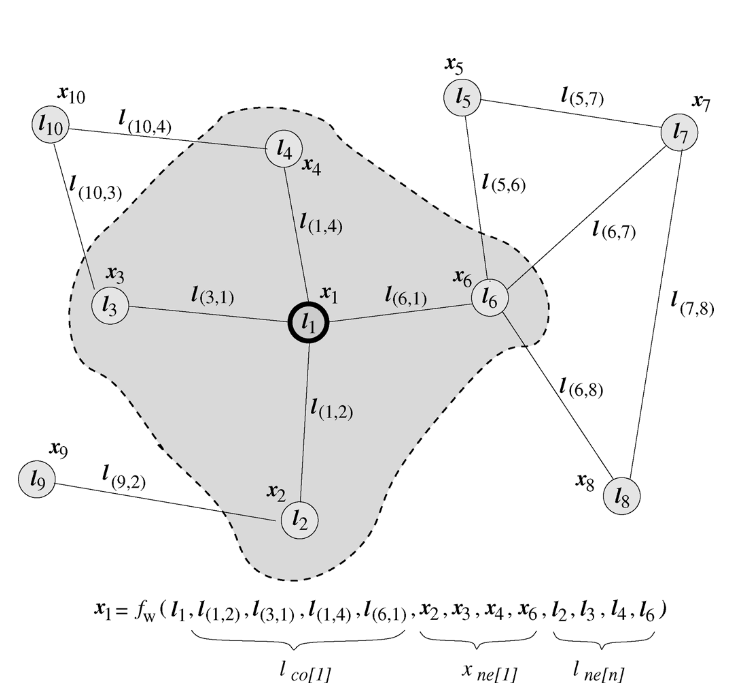
图神经网络是为了处理图论中的的图(Graph)，即非欧式图。

最早的图神经网络起源于2009年发表在IEEE上的文章The graph neural network model，理论基础是不动点理论。给定一张图G，每个节点以及边都有自己的特征， GNN的学习目标是获得每个结点的图感知的隐藏状态 ，这就意味着：对于每个节点，它的隐藏状态包含了来自邻居节点的信息。下面的公式表示了在某一时刻节点的隐藏状态(状态更新公式)

其中，为该节点的隐藏状态，为这个节点的特征，为相邻边的特征，为相邻节点的隐藏状态，为相邻节点的特征。

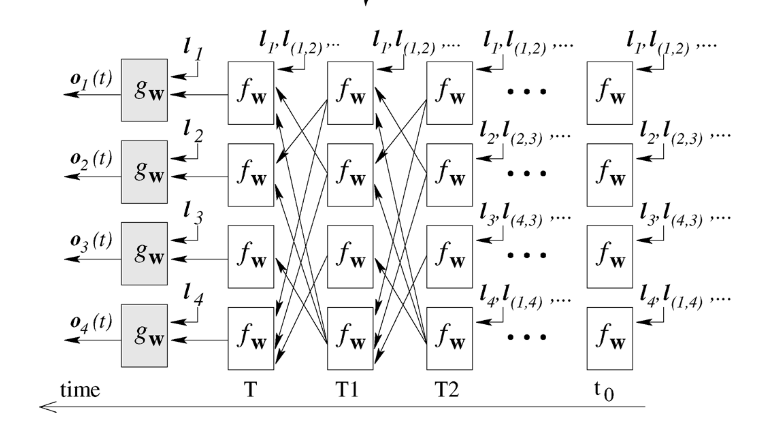
当每个节点的隐藏状态通过多次迭代收敛之后，是否收敛由两个时刻的范数差值是否小于一个阈值来判定。

用图形象的表示就是：



公式实例

之后可以通过一个目标函数来描述下游任务的。比如一个社交网络判断这个用户是不是僵尸账号。



注意图中的函数和函数，以及输入数据，中间相连的特点

#### 理论背景

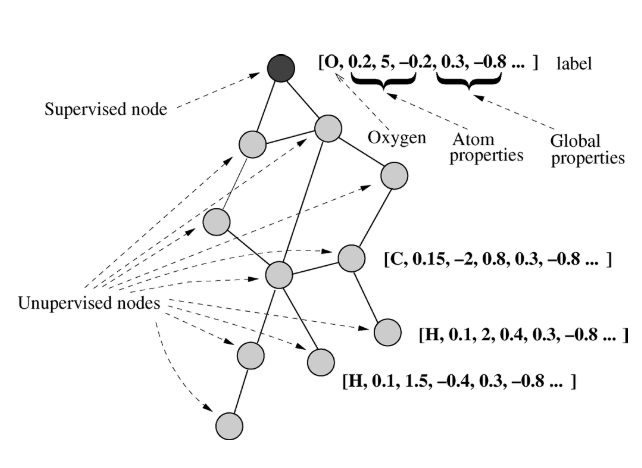
不动点理论是专指巴拿赫不动点定理(Banach's Fixed Point Theorem)。用表示多个

堆叠，表示多个g堆叠。图上所有节点的状态更新公式和目标公式可以写为：

不动点理论是，不论你输入的x是什么，经过F（F必须是contraction map）之后x都会迭代收敛于某一个固定点，因此称为不动点理论。

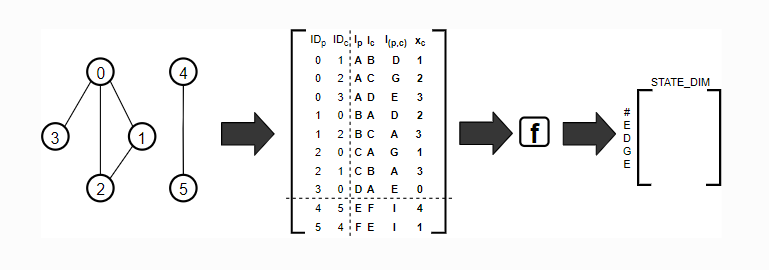
#### 论文实例

如判断化合物是否有害，一个分子可以看做是一张图，原子作为节点。在不断迭代得到根节点氧原子的收敛状态后在上面接一个前馈神经网络作为输出层(即)。



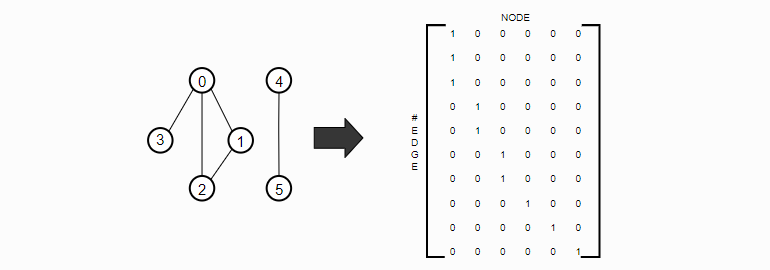
#### 基于矩阵的GNN实现

对于每一个图转化成矩阵形式，第一列表示父结点id，第二列表示子结点id，第三列表示父结点特征，第四列表示子结点特征，第五列表示边的特征，第六列表示子结点t时刻的状态。每一行对应一条边。



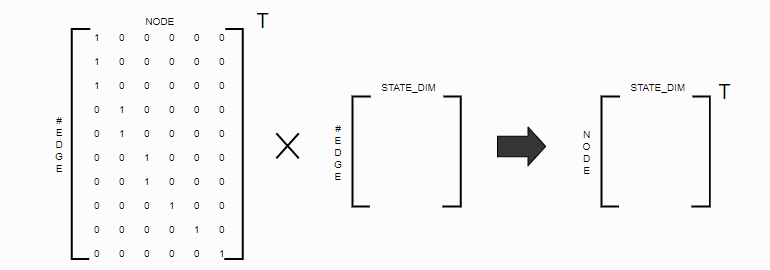
生成每个边对状态的贡献 1

为了聚合每个节点的状态，构造如下矩阵，行表示边，列表示节点，数值1表示是否以此节点作为边的父节点。



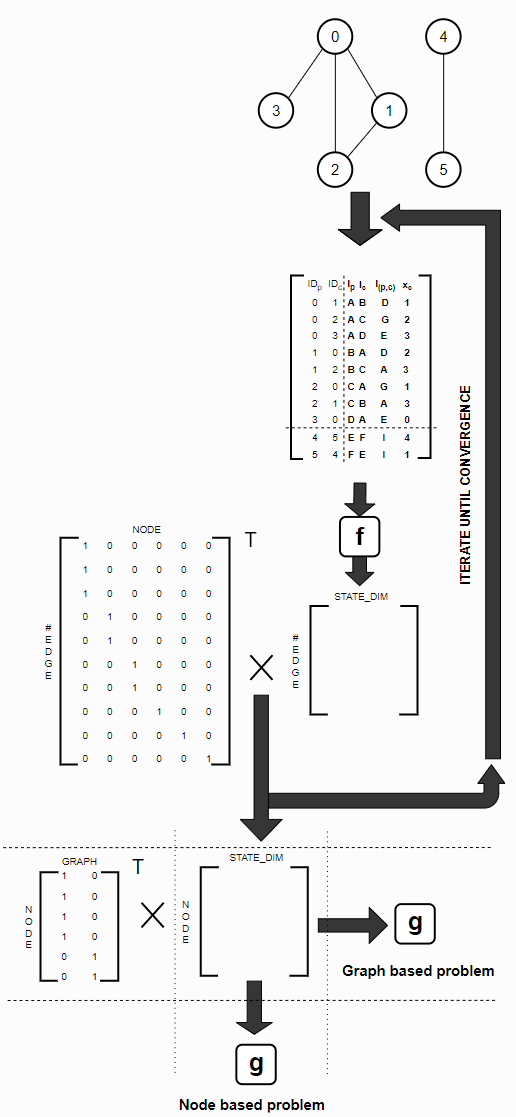
边和节点的关系矩阵 2

使用对每个节点的邻域进行编码的边缘节点矩阵执行矩阵乘法，因此获得了每个节点的状态聚合表示



聚合每条边的贡献 3

所以整个流程如下图所示：



#### 代码部分：

<https://github.com/Ivan0131/gnn_demo>

# reading train, validation dataset

data\_path = "./data"

set\_name = "sub\_15\_7\_200"

############# training set ################

inp, arcnode, nodegraph, nodein, labels, \_ = gnn\_utils.set\_load\_general(data\_path, "train", set\_name=set\_name)

inp = [a[:, 1:] for a in inp]

############ validation set #############

inp\_val, arcnode\_val, nodegraph\_val, nodein\_val, labels\_val, \_ = gnn\_utils.set\_load\_general(data\_path, "validation", set\_name=set\_name)

inp\_val = [a[:, 1:] for a in inp\_val]

inp是图1中的矩阵，arcnode是图2中的矩阵，但是inp也有区别，前两列是父节点和子节点的id，后面四列是父节点和子节点的四个特征。这里采用先去除第一列父节点id，因为运算用不到。Nodegraph表示节点属于哪个图，对整个图做预测的时候需要使用。Nodein是节点数量，labels是要预测的每个节点的标签。

def f\_w(inp):

    with tf.variable\_scope('State\_net'):

        layer1 = tf.layers.dense(inp, 5, activation=tf.nn.sigmoid)

        layer2 = tf.layers.dense(layer1, state\_dim, activation=tf.nn.sigmoid)

        return layer2

def g\_w(inp):

    with tf.variable\_scope('Output\_net'):

        layer1 = tf.layers.dense(inp, 5, activation=tf.nn.sigmoid)

        layer2 = tf.layers.dense(layer1, output\_dim, activation=None)

        return layer2

定义两个函数，都是由两个全连接层组成，激活函数为sigmoid，f\_w完成的是每条边兑付节点状态的贡献，如图1，加上图3的聚合过程才是。

#init input placeholder

tf.reset\_default\_graph()

comp\_inp = tf.placeholder(tf.float32, shape=(None, input\_dim), name="input")

y = tf.placeholder(tf.float32, shape=(None, output\_dim), name="target")

# state(t) & state(t-1)

state = tf.placeholder(tf.float32, shape=(None, state\_dim), name="state")

state\_old = tf.placeholder(tf.float32, shape=(None, state\_dim), name="old\_state")

# arch-node conversion matrix

ArcNode = tf.sparse\_placeholder(tf.float32, name="ArcNode")

定义tensorflow的placeholder用于数据输入。

def convergence(a, state, old\_state, k):

    with tf.variable\_scope('Convergence'):

        # assign current state to old state

        old\_state = state

        # 获取子结点上一个时刻的状态

        # grub states of neighboring node

        gat = tf.gather(old\_state, tf.cast(a[:, 0], tf.int32))

        # 去除第一列，即子结点的id

        # slice to consider only label of the node and that of it's neighbor

        # sl = tf.slice(a, [0, 1], [tf.shape(a)[0], tf.shape(a)[1] - 1])

        # equivalent code

        sl = a[:, 1:]

        # 将子结点上一个时刻的状态放到最后一列

        # concat with retrieved state

        inp = tf.concat([sl, gat], axis=1)

        # evaluate next state and multiply by the arch-node conversion matrix to obtain per-node states

        #计算子结点对父结点状态的贡献

        layer1 = f\_w(inp)

        #聚合子结点对父结点状态的贡献，得到当前时刻的父结点的状态

        state = tf.sparse\_tensor\_dense\_matmul(ArcNode, layer1)

        # update the iteration counter

        k = k + 1

    return a, state, old\_state, k

完整的实现，可以分成两个部分：构建图1中的矩阵（获取子结点 t-1时刻的状态，然后加入到最后一列，并去掉第一列，子结点id）；根据父结点相连的子结点对父结点状态的贡献，做累加，聚合得到父结点t时刻的状态，这里使用矩阵乘法计算。

def condition(a, state, old\_state, k):

    # evaluate condition on the convergence of the state

    with tf.variable\_scope('condition'):

        # 检查当前状态和上一个时刻的状态的欧式距离是否小于阈值

        # evaluate distance by state(t) and state(t-1)

        outDistance = tf.sqrt(tf.reduce\_sum(tf.square(tf.subtract(state, old\_state)), 1) + 1e-10)

        # vector showing item converged or not (given a certain threshold)

        checkDistanceVec = tf.greater(outDistance, state\_threshold)

        c1 = tf.reduce\_any(checkDistanceVec)

        # 是否达到最大迭代次数

        c2 = tf.less(k, max\_iter)

    return tf.logical\_and(c1, c2)

迭代计算是否达到阈值，满足条件就可以使用g\_w输出了。