GAT

理论

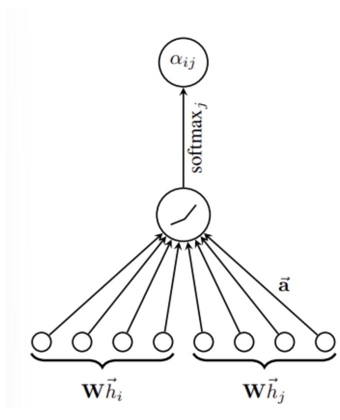
GCN缺陷：聚合邻居节点的时候认为邻居节点同等重要，但是其实节点聚合邻居节点的时候应当是存在不同程度的重要的。所以提出了GAT这种模型，也就是引入注意力机制的来给顶点根据邻居节点进行特征更新的时候根据邻居节点对顶点重要性来进行。

输入和输出

The input to our layer is a set of node features, $\mathbf{h} = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}, \vec{h}_i \in \mathbb{R}^F$, where N is the number of nodes, and F is the number of features in each node. The layer produces a new set of node features (of potentially different cardinality F'), $\mathbf{h}' = \{\vec{h}'_1, \vec{h}'_2, \ldots, \vec{h}'_N\}, \vec{h}'_i \in \mathbb{R}^{F'}$, as its output.

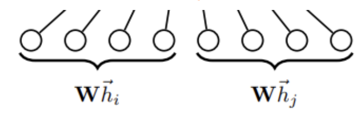
输入：h是节点特征矩阵　输出：h'是更新过后的节点特征矩阵

模型框架



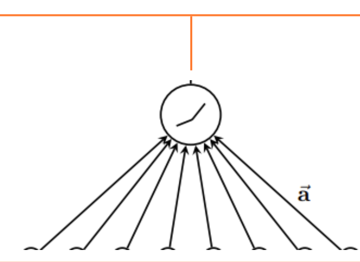
详细讲解

1. W特征权重矩阵与节点特征矩阵相乘



因为要进行节点特征维度的变化即节点数量变化，通过线性映射的方式进行变化

2. 通过注意力机制权重矩阵引入注意力机制



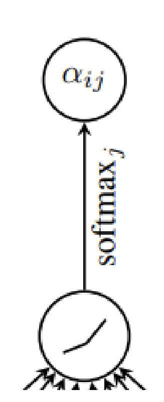
$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$$

1. 对变化后的节点特征矩阵与注意力机制权重矩阵相乘得到了注意力得分——也就是节点j特征对节点i的重要性，这部分是对全部的结构信息进行获取，也就是所有节点对所有其他节点

graph structure into the mechanism by performing *masked attention*—we only compute e_{ij} for nodes $j \in \mathcal{N}_i$, where $\mathcal{N}_i$ is some *neighborhood* of node i in the graph. In all our experiments, these will

2. 定义一个mask，由于之前是全部的节点和全部节点之间的注意力机制得分，但是我们更新节点特征信息是通过邻居节点的特征，也就是我们只考虑两个节点之间有连接（边）。

3. 利用softmax进行标准化



$$\alpha_{ij} = \mathrm{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}.$$

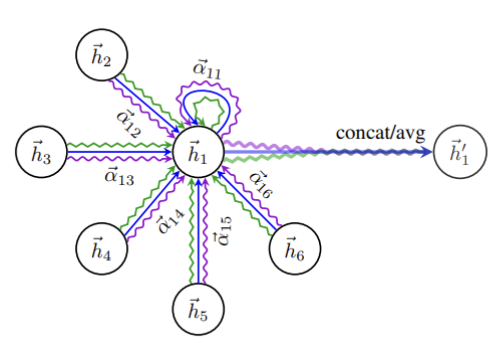
进行标准化的意义是将注意力得分转化为0-1的概率，这样更加方便顶点i从多个j节点中做出选择决策

4. 整体公式展示

$$\alpha_{ij} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \Vert \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \Vert \mathbf{W}\vec{h}_k]\right)\right)}$$

前面我们提到了a注意力机制权重其形状是（1，2*输出节点特征数量）

多头注意力机制



该图是设定了K（K=3)个头，然后我们的顶点是h1，其邻居节点是h2~h6，每个头独立的进行注意力机制，最后再每个经过注意力产生的概率分布（即h1更新节点特征的时候会有多大概率考虑h2~h6）对其平均或者拼接得到h1'

代码实现

1. 数据预处理

数据预处理部分和GCN一摸一样，用的数据集也是一样的——得到的是节点特征矩阵h和邻接矩阵adj

2. 模型设置

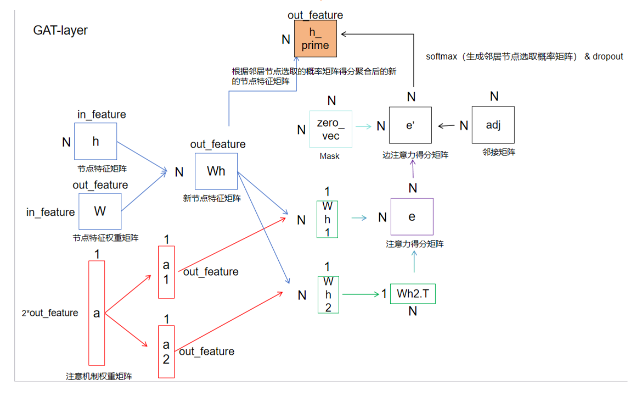
2-1 定义GraphAttentionLayer层

2-1-1初始化设定

```
## 构建GraphAttentionLayer
class GraphAttentionLayer(nn.Module):
    def __init__(self, in_features, out_features, dropout,
     alpha, concat=True):
        super(GraphAttentionLayer, self).__init__()
    ##防止过拟合
        self.dropout = dropout
    ##输入特征维度
        self.in_features = in_features
    ##输出特征维度
        self.out_features = out_features
    ##LeakyReLU激活函数的负斜率
        self.alpha = alpha
    ##是否在多个头之间进行链接
        self.concat = concat

    ##节点输入维度到输出维度的一个学习参数
        self.W = nn.Parameter(torch.empty(size=(in_features,
         out_features)))
    ##进行参数的初始化
        nn.init.xavier_uniform_(self.W.data, gain=1.414)
        ##gain 是一个可选的缩放因子，用于调整初始化的范围。在这个例子中，
         gain 被设置为根号2 （大约是1.414）。这通常用于当激活函数是
         ReLU 或者 ReLU 的变体（比如 LeakyReLU）时，以帮助保持方差。

    ##用于计算注意力机制的权重
        self.a = nn.Parameter(torch.empty(size=(2*out_features,
          1)))
    ##进行参数的初始化
        nn.init.xavier_uniform_(self.a.data, gain=1.414)
    ##进行非线性处理
        self.leakyrelu = nn.LeakyReLU(self.alpha)
```

1. 构建了节点特征变化权重矩阵W其shape为（in_features,out_features)
2. 构建了注意力机制权重矩阵a其shape为（2*features，1)

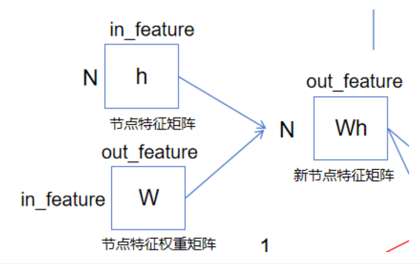
2-1-2前向传播



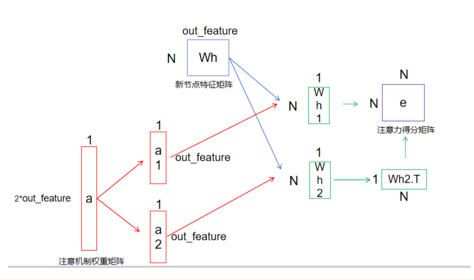
```
def forward(self, h, adj):
##对节点特征矩阵进行W的线性变换
    Wh = torch.mm(h, self.W)
##计算注意力机制的输入
    e = self._prepare_attentional_mechanism_input(Wh)
##以-9e15作为e的mask
    zero_vec = -9e15*torch.ones_like(e)
##如果如果两个节点有边链接则使其注意力得分为e，如果没有则使用-9e15的
 mask作为其得分
    attention = torch.where(adj > 0, e, zero_vec)
##使用 softmax 函数对注意力得分进行归一化
    attention = F.softmax(attention, dim=1)
##dropout，为模型的训练增加了一些正则化
    attention = F.dropout(attention, self.dropout,
     training=self.training)
##将注意力机制的矩阵和相邻的节点特征矩阵进行矩阵相乘
    h_prime = torch.matmul(attention, Wh)

##将多头注意力得分进行拼接
    if self.concat:
        return F.elu(h_prime)
    else:
        return h_prime
```

接下来是逐行代码分析

2-1-2-1 生成Wh

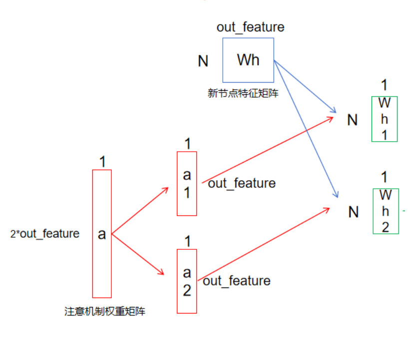


2-1-2-2 注意力机制引入



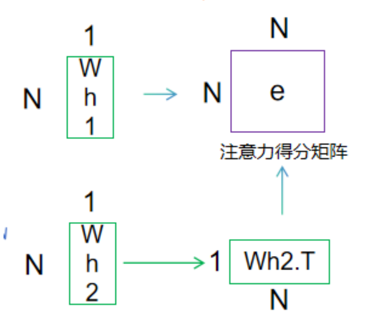
```
##节点间的原始注意力得分
    def _prepare_attentional_mechanism_input(self, Wh):
    ##Wh和self.a的前一半进行矩阵乘法
        Wh1 = torch.matmul(Wh, self.a[:self.out_features, :])
    ##Wh和self.a的后一半进行矩阵乘法
        Wh2 = torch.matmul(Wh, self.a[self.out_features:, :])
    ##Wh1和Wh2的转置进行广播加法（broadcast add）操作，得到原始注意力得
     分矩阵 e。
        e = Wh1 + Wh2.T
    ##通过LeakyReLU激活函数对注意力得分进行非线性变换。
        return self.leakyrelu(e)

    def __repr__(self):
        return self.__class__.__name__ + ' (' + str(self
         .in_features) + ' -> ' + str(self.out_features) + ')'
```

2-1-2-2-1 生成Wh1，Wh2



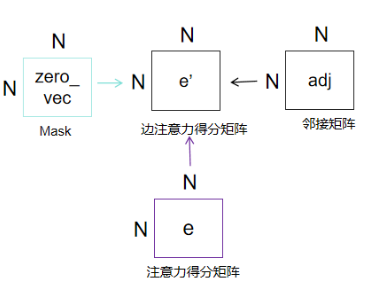
将注意力权重矩阵a进行分割，然后各自与Wh进行矩阵相乘，随后得到wh1，和wh2

2-1-2-2-2 得到注意力得分矩阵



将Wh2的转置Wh2.T与Wh1进行矩阵相加，利用广播原则，得到了注意力得分矩阵e其shape为(N,N)

2-1-2-3 转化为边注意力得分矩阵



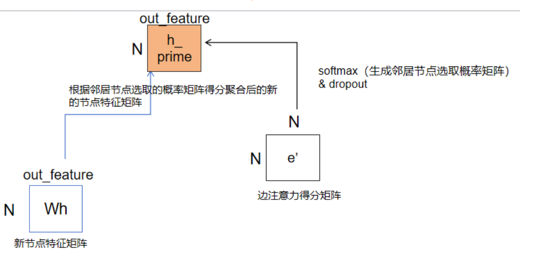
1. 先设定mask矩阵一个全为-9e15一个极小值，其目的是虽然邻接矩阵中代表节点和节点不连接的是0，但是这对于后续进行softmax的概率求解会造成影响
2. 将注意力得分矩阵更新得到边注意力得分矩阵，根据邻接矩阵将节点和节点没有连接的位置设定为-9e15的极小值

2-1-2-4 通过softmax将注意力得分转化为概率

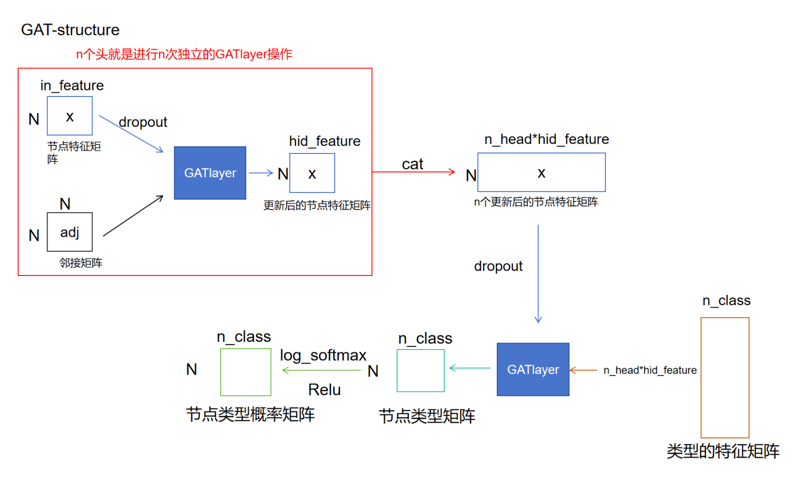
将得分转化为0-1的概率进行标准化

2-1-2-5 使用dropout防止过拟合

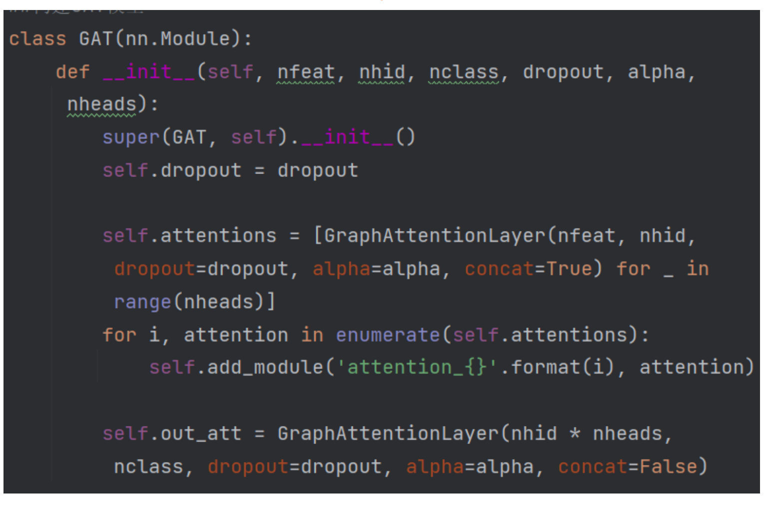
2-1-2-6 将邻居节点选取的概率矩阵和节点特征矩阵相乘得到最后的h_prime



2-2 定义GAT模型框架



2-2-1 初始化设定



```
class GAT(nn.Module):
    def __init__(self, nfeat, nhid, nclass, dropout, alpha,
     nheads):
        super(GAT, self).__init__()
        self.dropout = dropout

        self.attentions = [GraphAttentionLayer(nfeat, nhid,
         dropout=dropout, alpha=alpha, concat=True) for _ in
         range(nheads)]
        for i, attention in enumerate(self.attentions):
            self.add_module('attention_{}'.format(i), attention)

        self.out_att = GraphAttentionLayer(nhid * nheads,
         nclass, dropout=dropout, alpha=alpha, concat=False)
```

2-2-1-1 attentions

self.attentions = [GraphAttentionLayer(nfeat, nhid, dropout=dropout, alpha=alpha, concat=True) for _ in range(nheads)]
多头注意力机制的引入：创建了一个包含多个GraphAttentionLayer 层的列表，一个头就是一个GraphAttentionLayer层，每个头都具有相同的参数，并且共享输入特征。

2-2-1-2 循环遍历每个头

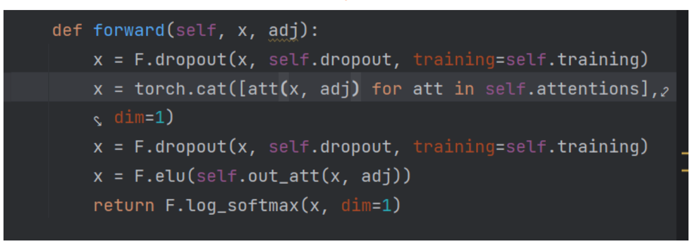
for i, attention in enumerate(self.attentions):：通过循环遍历所有的图注意力层。

self.add_module('attention_{}'.format(i), attention)：使用 add_module 方法将每个图注意力层添加为模型的子模块，这样它们就可以被PyTorch自动跟踪和管理。

2-2-1-3 输出层定义

self.out_att = GraphAttentionLayer(nhid * nheads, nclass, dropout=dropout, alpha=alpha, concat=False)：创建一个输出图注意力层，这个层将多头注意力的结果合并成一个，将输入特征的维度从 nhid * nheads 缩减到 nclass，同时不进行拼接操作。

2-2-2 前向传播



```
def forward(self, x, adj):
    x = F.dropout(x, self.dropout, training=self.training)
    x = torch.cat([att(x, adj) for att in self.attentions],
    dim=1)
    x = F.dropout(x, self.dropout, training=self.training)
    x = F.elu(self.out_att(x, adj))
    return F.log_softmax(x, dim=1)
```

2-2-2-1 dropout——节点特征

x为节点特征矩阵，x = F.dropout(x, self.dropout, training=self.training)：应用Dropout操作以防止过拟合——随机删除一定节点特征从而防止过拟合

2-2-2-2 cat——拼接每个头得到的更新后的节点特征矩阵

这一行代码使用列表推导式迭代多个注意力头，每个注意力头是self.attentions中的一个。对于每个注意力头，它调用att(x, adj)，其中att是GraphAttentionLayer的一个实例，将输入特征矩阵x和邻接矩阵adj传递给注意力层。然后，它将多个注意力头的输出在维度1上（dim=1）进行拼接，以获得最终的节点表示。

2-2-2-3 dropout——节点特征（同上）

2-2-2-4 out_att & log softmax & Relu ——得到节点类型概率矩阵