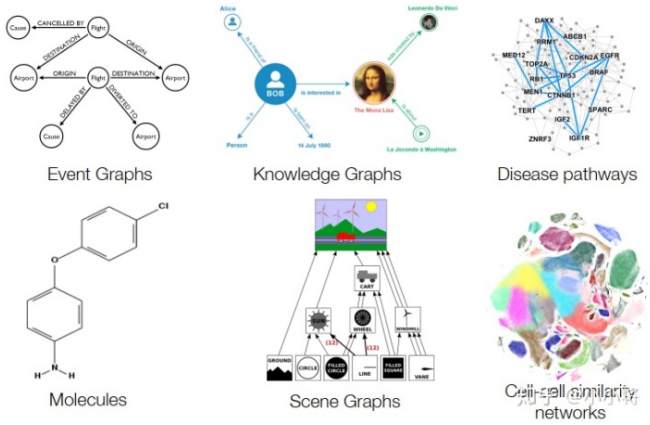
2020年，我终于决定入门GCN

知 https://zhuanlan.zhihu.com/p/112277874

None Sun Nov, 22 01:28

我们面对的很多数据其实是图（graph），图在生活中无处不在，如社交网络，知识图谱，蛋白质结构等。在2020年这个寒冬，窝在家里的小编终于打算入门GNN（Graph Neural Networks）中的分支：GCN（Graph Convolutional Networks）。



图的概念

对于图，我们习惯上用

$$G = (V, E)$$

表示。这里

$$V$$

是图中节点的集合，而

$$E$$

为边的集合，这里记图的节点数为

$$N$$

。一个

$$G$$

中有3个比较重要的矩阵：

数学表示是比较抽象的，下面是一个实例：

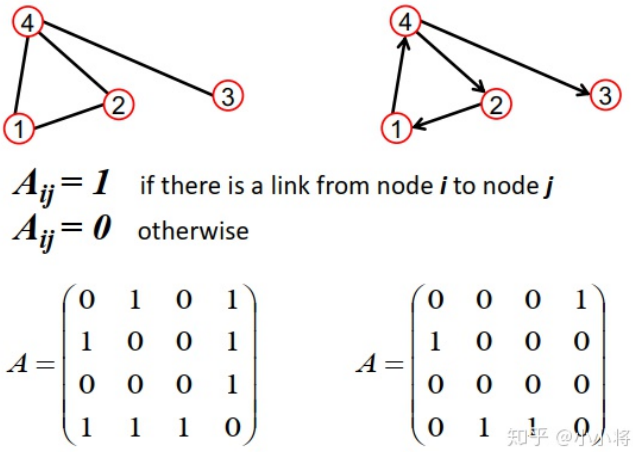


图1 图以及邻接矩阵（来源：stanford cs224w）

注意左图是无向图，而右图是有向图，前者的邻接矩阵是对称的，而后者是不对称的。

相比图像和文本，图这种拓扑结构是较复杂的：任意的节点数以及节点间的复杂关系：

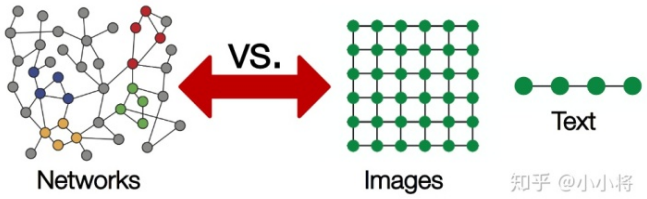


图2 图与图像和文本的结构对比（来源：stanford cs224w）

这种复杂性给神经网络在图上的应用带来了一定困难，但是我们依然有解决办法。

学习新特征

深度学习中最重要的是学习特征：随着网络层数的增加，特征越来越抽象，然后用于最终的任务。对于图任务来说，这点同样适用，我们希望深度模型从图的最初始特征

$$X$$

出发学习到更抽象的特征，比如学习到了某个节点的高级特征，这个特征根据图结构融合了图中其他节点的特征，我们就可以用这个特征用于节点分类或者属性预测。那么图网络就是要学习新特征，用公式表达就是：

$$H^{(k+1)} = f(H^{(k)}, A)$$

这里k指的是网络层数，

$$H^{(k)}$$

就是网络第k层的特征，其中

$$H^{(0)} = X$$

。那么现在的问题是如何学习，我们可以从CNN中得到启发：

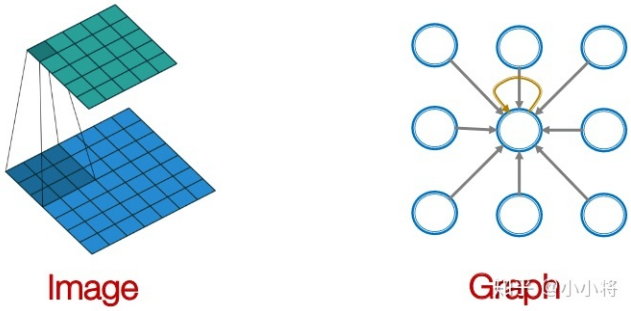


图3 CNN与图学习类比（来源：stanford cs224w）

这是一个简单的3x3卷积层，每个新特征的学习是这样的：对其领域（3x3局部空间）的特征进行变换（

$$w_i x_i$$

），然后求和（

$$\sum_i w_i x_i$$

）。类比到图学习上，每个节点的新特征可以类似得到：对该节点的邻域节点特征进行变换，然后求和。用公式表达就是：

$$H^{(k+1)} = f(H^{(k)}, A) = \sigma(AH^{(k)}W^{(k)})$$

这里的

$$W^k$$

是学习权重，维度为

$$F^{k-1} \times F^k$$

，而

$$\sigma(\cdot)$$

是激活函数，比如是ReLU，这是神经网路的基本单元。上述公式其实就是对领域内节点特征求和，这里：

$$\begin{aligned}(AH)_i &= A_i H \\ &= \sum_j A_{ij} H_j\end{aligned}$$

其中邻接矩阵

$$A$$

是0-1矩阵，当节点j与节点i连接时，

$$A_{ij} = 1$$

，所以节点i的新特征就是其邻接节点的特征和。

其实我们可以将上述学习分成三个部分：

- 变换（transform）：对当前的节点特征进行变换学习，这里就是乘法规则（Wx）；
- 聚合（aggregate）：聚合领域节点的特征，得到该节点的新特征，这里是简单的加法规则；
- 激活（activate）：采用激活函数，增加非线性。

其实这就算是图卷积（graph convolution）了，首先这里的权重是所有节点共享的，类比于CNN中的参数共享；另外可以将节点的邻居节点看成感受野，随着网络层数的增加，感受野越来越大，即节点的特征融合了更多节点的信息。直观的图卷积示意图如下：

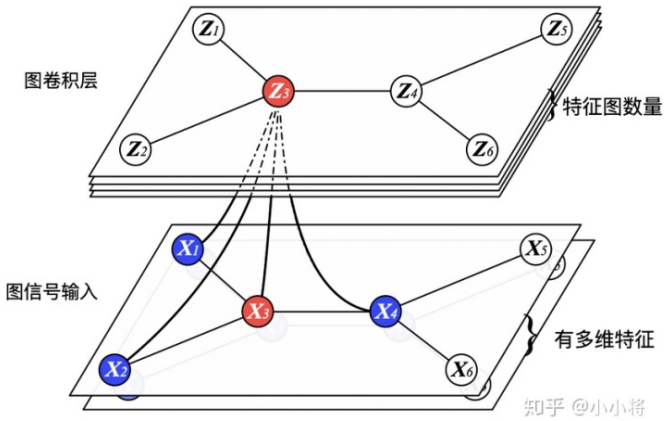


图4 图卷积的示意图 （来源：https://www.jianshu.com/p/2fd5a2454781）

图卷积

上述的加法规则只是一个简单实现，其存在两个问题：首先在计算新特征时没有考虑自己的特征，这肯定是个重大缺陷；另外采用加法规则时，对于度大的节点特征越来越大，而对于度小的节点却相反，这可能导致网络训练过程中梯度爆炸或者消失的问题。

针对第一个问题，我们可以给图中每个节点增加自连接，实现上可以直接改变邻接矩阵：

$$\tilde{A} = A + I_N$$

针对第二个问题，我们可以对邻接矩阵进行归一化，使得

$$A$$

的每行和值为1，在实现上我们可以乘以度矩阵的逆矩阵：

$$\tilde{D}^{-1}\tilde{A}$$

，这里的度矩阵是更新

$$A$$

后重新计算的。这样我们就得到：

$$H^{(k+1)} = f(H^{(k)}, A) = \sigma(\tilde{D}^{-1}\tilde{A}H^{(k)}W^{(k)})$$

相比加法规则，这种聚合方式其实是对领域节点特征求平均，这里：

$$\begin{aligned}(\tilde{D}^{-1}\tilde{A}H)_i &= (\tilde{D}^{-1}\tilde{A})_i H \\ &= (\sum_k \tilde{D}_{ik}^{-1}\tilde{A}_i)H \\ &= (\tilde{D}_{ii}^{-1}\tilde{A}_i)H \\ &= \tilde{D}_{ii}^{-1}\sum_j \tilde{A}_{ij}H_j \\ &= \sum_j \frac{1}{\tilde{D}_{ii}}\tilde{A}_{ij}H_j\end{aligned}$$

由于

$$\tilde{D} = \sum_j \tilde{A}_{ij}$$

，所以这种聚合方式其实就是求平均，对领域节点的特征是求平均值，这样就进行了归一化，避免求和方式所造成的问题。

更进一步地，我们可以采用对称归一化来进行聚合操作，这就是论文1中所提出的图卷积方法：

$$H^{(k+1)} = f(H^{(k)}, A) = \sigma(\tilde{D}^{-0.5}\tilde{A}\tilde{D}^{-0.5}H^{(k)}W^{(k)})$$

这种新的聚合方法不再是单单地对邻域节点特征进行平均，这里：

$$\begin{aligned}(\tilde{D}^{-0.5}\tilde{A}\tilde{D}^{-0.5}H)_i &= (\tilde{D}^{-0.5}\tilde{A})_i\tilde{D}^{-0.5}H \\ &= (\sum_k \tilde{D}_{ik}^{-0.5}\tilde{A}_i)\tilde{D}^{-0.5}H \\ &= \tilde{D}_{ii}^{-0.5}\sum_j \tilde{A}_{ij}\sum_k \tilde{D}_{jk}^{-0.5}H_j \\ &= \tilde{D}_{ii}^{-0.5}\sum_j \tilde{A}_{ij}\tilde{D}_{jj}^{-0.5}H_j \\ &= \sum_j \frac{1}{\sqrt{\tilde{D}_{ii}\tilde{D}_{jj}}}\tilde{A}_{ij}H_j\end{aligned}$$

可以看到这种聚合方式不仅考虑了节点i的度，而且也考虑了邻居节点j的度，当邻居节点j的度较大时，而特征反而会受到抑制。

这种图卷积方法其实**谱图卷积**的一阶近似（first-order approximation of spectral graph convolutions），关于更多的数学证明比较难理解，这里不做展开，详情可见论文。

定义了图卷积，我们只需要将图卷积层堆积起来就构成了图卷积网络GCN：

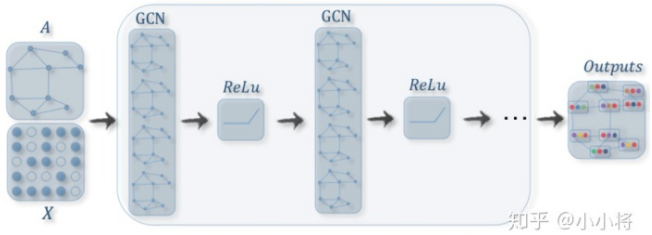


图5 GCN示意图

其实图神经网路（GNN，Graph Neural Network）是一个庞大的家族，如果按照

$$f$$

分类，其可以分成以下类型：

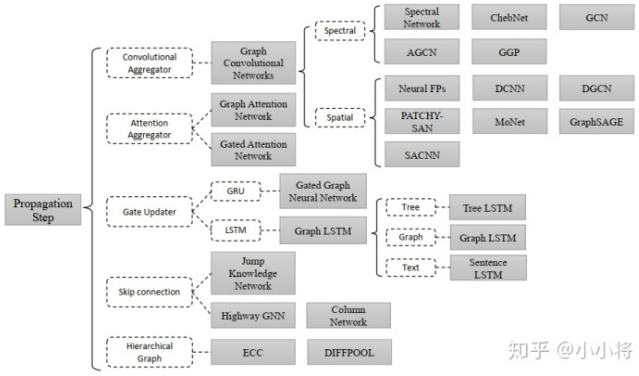


图6 GNN分类

可以看到GCN只是其中的一个很小的分支，我们上面所述的GCN其实是属于谱图卷积。更多关于GNN的学习，可以阅读这三篇综述文章：

- Graph Neural Networks: A Review of Methods and Application
- A Comprehensive Survey on Graph Neural Networks
- Deep Learning on Graphs: A Survey

GCN的PyTorch实现

虽然GCN从数学上较难理解，但是它的实现是非常简单的，值得注意的一点是一般情况下邻接矩阵

$$A$$

是稀疏矩阵，所以我们在实现矩阵乘法时，采用稀疏运算会更高效。这里我们参考论文作者的[官方实现](#)。首先是图卷积层的实现：

```
import torch
    import torch.nn as nn


    class GraphConvolution(nn.Module):
        '''GCN layer'''

        def __init__(self, in_features, out_features, bias=True):
            super(GraphConvolution, self).__init__()
            self.in_features = in_features
            self.out_features = out_features
            self.weight = nn.Parameter(torch.Tensor(in_features, out_features))
            if bias:
                self.bias = nn.Parameter(torch.Tensor(out_features))
            else:
                self.register_parameter('bias', None)

            self.reset_parameters()

        def reset_parameters(self):
            nn.init.kaiming_uniform_(self.weight)
            if self.bias is not None:
                nn.init.zeros_(self.bias)

        def forward(self, input, adj):
            support = torch.mm(input, self.weight)
            output = torch.spmm(adj, support)
            if self.bias is not None:
                return output + self.bias
            else:
                return output

        def extra_repr(self):
            return 'in_features={}, out_features={}, bias={}'.format(
                self.in_features, self.out_features, self.bias is not None
            )
```

对于GCN，只需要将图卷积层堆积起来就可以，这里我们实现一个两层的GCN：

```
class GCN(nn.Module):
    '''a simple two layer GCN'''
    def __init__(self, nfeat, nhid, nclass):
        super(GCN, self).__init__()
        self.gc1 = GraphConvolution(nfeat, nhid)
        self.gc2 = GraphConvolution(nhid, nclass)

    def forward(self, input, adj):
        h1 = F.relu(self.gc1(input, adj))
        logits = self.gc2(h1, adj)
        return logits
```

这里的激活函数采用ReLU，后面我们将用这个网络实现一个图中节点的半监督分类任务。

半监督分类实例

这里给出的是GCN论文中的一个半监督分类任务，官方代码也给出这个任务。我们要处理的数据集是[cora数据集]，该数据集是一个论文图，共2708个节点，每个节点都是一篇论文，所有样本点被分为7类别：

Case_Based, Genetic_Algorithms, Neural_Networks, Probabilistic_Methods, Reinforcement_Learning, Rule_Learning, Theory

每篇论文都由一个1433维的词向量表示，即节点特征维度为1433。词向量的每个特征都对应一个词，取0表示该特征对应的词不在论文中，取1则表示在论文中。每篇论文都至少引用了一篇其他论文，或者被其他论文引用，这是一个连通图，不存在孤立点。

这里的任务是给定图中某些节点的类别，然后训练一个网络能够预测其它节点标签，所以这里一个半监督学习任务。我们建立一个两层GCN来解决这个问题：

$$Z = f(X; A) = \mathrm{softmax}(\hat{A}(\mathrm{ReLU}(\hat{A}XW^{(0)})W^{(1)}), \quad \hat{A} = \tilde{D}^{-0.5}\tilde{A}\tilde{D}^{-0.5}$$

从结构上看，中间层用于提出特征，而最后一层的节点特征用于分类任务（送入softmax，计算交叉熵）：

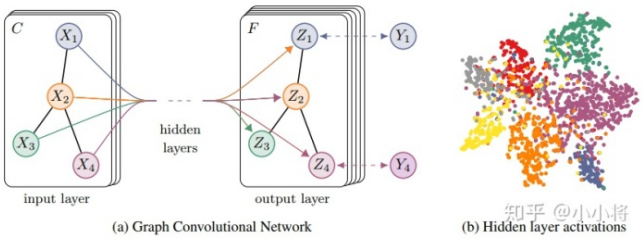


图7 两层GCN用于分类任务

数据的提取，论文官方实现已经给出，我们只需要load就可以：

```
# https://github.com/tkipf/pygcn/blob/master/pygcn/utils.py
adj, features, labels, idx_train, idx_val, idx_test = load_data(path='./data/cora/')
```

值得注意的有两点，一是论文引用应该是单向图，但是在网络时我们要先将其转成无向图，或者说建立双向引用，我发现这个对模型训练结果影响较大：

```
# build symmetric adjacency matrix
adj = adj + adj.T.multiply(adj.T > adj) - adj.multiply(adj.T > adj)
```

另外官方实现中对邻接矩阵采用的是普通均值归一化，当然我们也可以采用对称归一化方式：

```
def normalize_adj(adj):
    '''compute L=D^-0.5 * (A+I) * D^-0.5'''
    adj += sp.eye(adj.shape[0])
    degree = np.array(adj.sum(1))
    d_hat = sp.diags(np.power(degree, -0.5).flatten())
    norm_adj = d_hat.dot(adj).dot(d_hat)
    return norm_adj
```

这里我们只采用图中140个有标签样本对GCN进行训练，每个epoch计算出这些节点特征，然后计算loss：

```
loss_history = []
    val_acc_history = []
    for epoch in range(epochs):
        model.train()
        logits = model(features, adj)
        loss = criterion(logits[idx_train], labels[idx_train])

        train_acc = accuracy(logits[idx_train], labels[idx_train])

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        val_acc = test(idx_val)
        loss_history.append(loss.item())
        val_acc_history.append(val_acc.item())
        print('Epoch {:03d}: Loss {:.4f}, TrainAcc {:.4}, ValAcc {:.4f}'.format(
            epoch, loss.item(), train_acc.item(), val_acc.item()))
```

只需要训练200个epoch，我们就可以在测试集上达到80%左右的分类准确，GCN的强大可想而知：

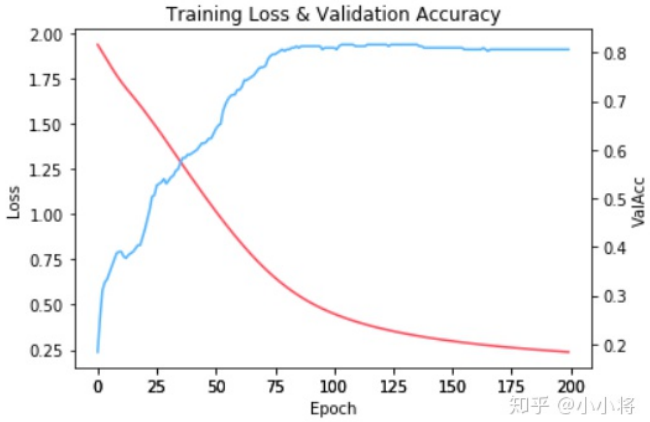


图8 训练收敛曲线

结语

GCN只是GNN中的冰山一角，这可能连入门都不算，但是千里之行始于足下。

参考

1. Semi-Supervised Classification with Graph Convolutional Networks
2. How to do Deep Learning on Graphs with Graph Convolutional Networks
3. Graph Convolutional Networks
4. Graph Convolutional Networks in PyTorch
5. 回顾频谱图卷积的经典工作：从ChebNet到GCN
6. 图数据集之cora数据集介绍- 用pyton处理 - 可用于GCN任务