Some Papers of PyG

陈乐偲

PyG的核心感觉是消息传递类，将GNN、LAP（标签传播）、PointNet（3d点云网络）、node2vec等方法统一为消息的传递和聚合等操作。并且基于此实现了相关方法，在PyG的框架下都可以简单实现。本文阅读PyG实现的部分论文，对常见的相关方法做了简单的了解。

GNNs

GCN

针对半监督结点分类任务的图卷积神经网络。基于一阶局部近似提出了一种新的图谱卷积运算。该方法的复杂度只与图的边数及图的隐藏层表示的数目成线性关系，且该方法在不少学术引用网络和知识图谱上取得了比对比方法很好的效果。

之前的结点分类方法使用如下的损失函数

$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{\text{reg}}, \quad \text{with} \quad \mathcal{L}_{\text{reg}} = \sum_{i,j} A_{ij} \|f(X_i) - f(X_j)\|^2 = f(X)^\top \Delta f(X). \tag{1}$$

该损失函数的第一项为结点分类的直接损失，第二项称为拉普拉斯惩罚，要求图中相邻结点的嵌入向量表示尽可能地相似。但这些方法显式地要求网络去学习第二项任务，学习相邻结点的相似性，但这有时候并不是我们想要的，而且这样子也不是端到端的学习。

而GCN采用如下的公式计算隐层，从而将第二项拉普拉斯惩罚融入了网络的学习中。

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right). \tag{2}$$

该公式基于图谱卷积，这是一种基于傅里叶变换的图信号处理方式。其中卷积由如下公式定义，

$$g_\theta \star x = U g_\theta U^\top x, \tag{3}$$

2.1 SPECTRAL GRAPH CONVOLUTIONS

We consider spectral convolutions on graphs defined as the multiplication of a signal $x \in \mathbb{R}^N$ (a scalar for every node) with a filter $g_\theta = \mathrm{diag}(\theta)$ parameterized by $\theta \in \mathbb{R}^N$ in the Fourier domain, i.e.:

$$g_\theta \star x = U g_\theta U^\top x \,, \tag{3}$$

where U is the matrix of eigenvectors of the normalized graph Laplacian $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^\top$, with a diagonal matrix of its eigenvalues Λ and $U^\top x$ being the graph Fourier transform of x. We can understand g_θ as a function of the eigenvalues of L, i.e. $g_\theta(\Lambda)$. Evaluating Eq. 3 is computationally expensive, as multiplication with the eigenvector matrix U is $\mathcal{O}(N^2)$. Furthermore, computing the eigendecomposition of L in the first place might be prohibitively expensive for large graphs. To circumvent this problem, it was suggested in Hammond et al. (2011) that $g_\theta(\Lambda)$ can be well-approximated by a truncated expansion in terms of Chebyshev polynomials $T_k(x)$ up to K^{th} order:

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda}) \,, \tag{4}$$

with a rescaled $\tilde{\Lambda} = \frac{2}{\lambda_{\text{max}}} \Lambda - I_N$. λ_{max} denotes the largest eigenvalue of L. $\theta' \in \mathbb{R}^K$ is now a vector of Chebyshev coefficients. The Chebyshev polynomials are recursively defined as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$. The reader is referred to Hammond et al. (2011) for an in-depth discussion of this approximation.

Going back to our definition of a convolution of a signal x with a filter $g_{\theta'}$, we now have:

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x \,, \tag{5}$$

with $\tilde{L} = \frac{2}{\lambda_{\text{max}}} L - I_N$; as can easily be verified by noticing that $(U \Lambda U^\top)^k = U \Lambda^k U^\top$. Note that this expression is now K-localized since it is a K^{th}-order polynomial in the Laplacian, i.e. it depends only on nodes that are at maximum K steps away from the central node (K^{th}-order neighborhood). The complexity of evaluating Eq. 5 is $\mathcal{O}(|\mathcal{E}|)$, i.e. linear in the number of edges. Defferrard et al. (2016) use this K-localized convolution to define a convolutional neural network on graphs.

但直接计算拉普拉斯矩阵的特征向量复杂度太高，用切比雪夫多项式可以得到卷积的近似

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x \,, \tag{5}$$

In this linear formulation of a GCN we further approximate $\lambda_{\max} \approx 2$, as we can expect that neural network parameters will adapt to this change in scale during training. Under these approximations Eq. 5 simplifies to:

$$g_{\theta'} \star x \approx \theta'_0 x + \theta'_1 \left(L - I_N\right) x = \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x \,, \tag{6}$$

with two free parameters θ'_0 and θ'_1. The filter parameters can be shared over the whole graph. Successive application of filters of this form then effectively convolve the k^{th}-order neighborhood of a node, where k is the number of successive filtering operations or convolutional layers in the neural network model.

In practice, it can be beneficial to constrain the number of parameters further to address overfitting and to minimize the number of operations (such as matrix multiplications) per layer. This leaves us with the following expression:

$$g_\theta \star x \approx \theta \left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}\right) x \,, \tag{7}$$

with a single parameter $\theta = \theta'_0 = -\theta'_1$. Note that $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ now has eigenvalues in the range $[0, 2]$. Repeated application of this operator can therefore lead to numerical instabilities and exploding/vanishing gradients when used in a deep neural network model. To alleviate this problem, we introduce the following *renormalization trick*: $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$, with $\tilde{A} = A + I_N$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.

We can generalize this definition to a signal $X \in \mathbb{R}^{N \times C}$ with C input channels (i.e. a C-dimensional feature vector for every node) and F filters or feature maps as follows:

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta \,, \tag{8}$$

where $\Theta \in \mathbb{R}^{C \times F}$ is now a matrix of filter parameters and $Z \in \mathbb{R}^{N \times F}$ is the convolved signal matrix. This filtering operation has complexity $\mathcal{O}(|\mathcal{E}|FC)$, as $\tilde{A}X$ can be efficiently implemented as a product of a sparse matrix with a dense matrix.

在最大特征值满足假设的情况下，可以得到如下的局部近似

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta \,, \tag{8}$$

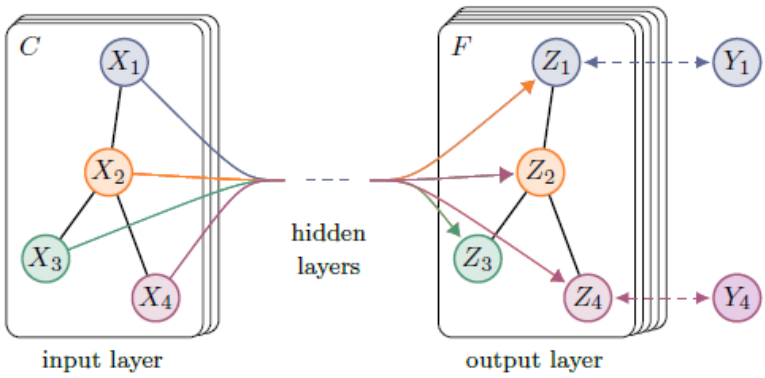
采用双层GCN，

$$Z = f(X, A) = \operatorname{softmax}\Big(\hat{A}\ \operatorname{ReLU}\Big(\hat{A} X W^{(0)}\Big)\, W^{(1)}\Big) \,. \tag{9}$$

其中损失函数由交叉熵损失定义，

$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf} \,, \tag{10}$$

下图阐释了GCN，并且可视化了隐层的激活情况，



(a) Graph Convolutional Network

(b) Hidden layer activations

基于Tesonflow实现了相关框架。

对比方法主要如下

- 基于标签传播的方法(LP)
- 基于随机游走和向量嵌入的方法(DeepWalk)
- etc.

主要公式总结如下，

公式1为整张图的整体表示

公式2为针对具体结点的表示

CLASS `GCNConv` (**in_channels**: int, **out_channels**: int, **improved**: bool = `False`, **cached**: bool = `False`, **add_self_loops**: bool = `True`, **normalize**: bool = `True`, **bias**: bool = `True`, ****kwargs**) [source]

The graph convolutional operator from the "Semi-supervised Classification with Graph Convolutional Networks" paper

$$\mathbf{X}' = \hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\mathbf{X}\mathbf{\Theta},$$

where $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ denotes the adjacency matrix with inserted self-loops and $\hat{D}_{ii} = \sum_{j=0} \hat{A}_{ij}$ its diagonal degree matrix. The adjacency matrix can include other values than `1` representing edge weights via the optional `edge_weight` tensor.

Its node-wise formulation is given by:

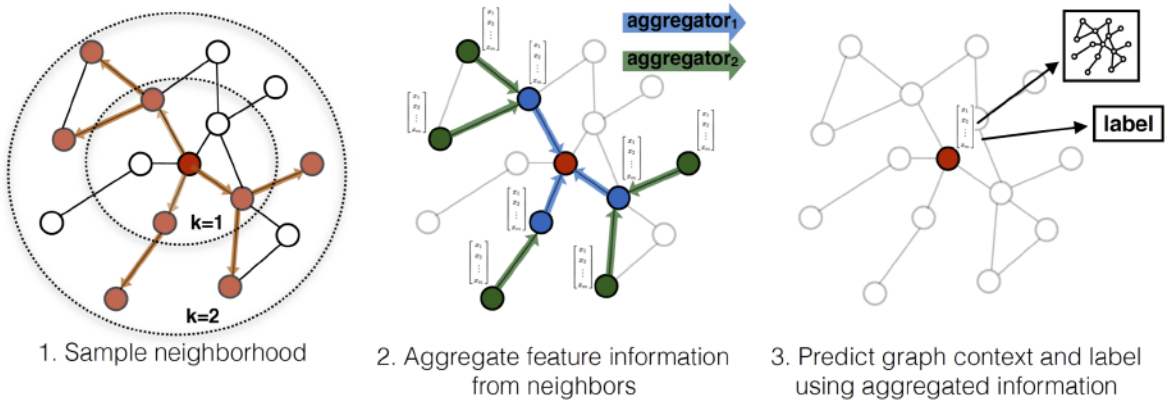
$$\mathbf{x}'_i = \mathbf{\Theta} \sum_{j \in \mathcal{N}(v) \cup \{i\}} \frac{e_{j,i}}{\sqrt{\hat{d}_j \hat{d}_i}} \mathbf{x}_j$$

with $\hat{d}_i = 1 + \sum_{j \in \mathcal{N}(i)} e_{j,i}$, where $e_{j,i}$ denotes the edge weight from source node `j` to target node `i` (default: `1.0`)

SAGE

基于图的向量嵌入(graph embedding)的方法已经证明了其强大性，但很多方法基于转导学习(transductive learning) 而非归纳学习(inductive learning) , 这些模型需要使用到整张图的每一个结点。SAGE为基于采样sample和聚合aggregate的方法，基于采样使得模型不需要获得整张图的结点，而聚合的定义使得模型更具有通用性，可以在模型的框架下定义不同的聚合函数实现不同的目的。该模型再很多归纳学习的数据集上取得了很好的成果，模型对于未曾见到过的结点和图，具有很强的学习能力。

模型分为采样和聚合两步，通过聚合预测图的上下文和标签。



1. Sample neighborhood　2. Aggregate feature information from neighbors　3. Predict graph context and label using aggregated information

相关工作有

- 基于矩阵分解等表示图结点的方法，如PageRank
- 有监督的图结构学习方法
- GCN

该模型实际上是图同构的WL测试的一种近似。本质都是学习图的拓扑结构，区别是SAGE采样聚合函数取代了WL测试中的哈希操作。

算法前向传播流程如下，不断迭代地聚合邻居信息和结点本身的隐层信息。

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

损失函数如下定义，第一项为使得两个相邻结点的表示尽可能相似，第二项为用P随机采样不连边的两个点，使得这两个点之间尽可能不相似，Q表示采样的格式。

$$J_{\mathcal{G}}(\mathbf{z}_u) = -\log\left(\sigma(\mathbf{z}_u^{\top}\mathbf{z}_v)\right) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log\left(\sigma(-\mathbf{z}_u^{\top}\mathbf{z}_{v_n})\right), \tag{1}$$

聚合函数有如下几种架构

- 均值聚合
- LSTM聚合
- 池化聚合

均值聚合即将所有邻居的特征取均值然后加上结点本身的特征。

$$\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W} \cdot \text{MEAN}(\{\mathbf{h}_v^{k-1}\} \cup \{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}). \tag{2}$$

LSTM聚合

但LSTM针对的是有序的数据，所以聚合时需要使用随机排列打乱邻居的顺序使得模型可以学习到无序的邻居特征。

池化聚合

$$\text{AGGREGATE}_k^{\text{pool}} = \max(\{\sigma\left(\mathbf{W}_{\text{pool}}\mathbf{h}_{u_i}^k + \mathbf{b}\right), \forall u_i \in \mathcal{N}(v)\},$$

在很多工作中采用了最大值池化，但该文章发现，实际上采用最大值池化和均值池化对模型的表现并没有显著差别，甚至推广到任意具有排列不变性的函数都可以作为聚合函数。

文章主要与用DeepWalk方法学习图的特征表示进行了对比。

SAGE中的均值聚合，可以如下定义。

CLASS **SAGEConv** (**in_channels**: Union[int, Tuple[int, int]], **out_channels**: int, **normalize**: bool = `False`, **root_weight**: bool = `True`, **bias**: bool = `True`, ****kwargs**) [source]

The GraphSAGE operator from the "Inductive Representation Learning on Large Graphs" paper

$$\mathbf{x}'_i = \mathbf{W}_1 \mathbf{x}_i + \mathbf{W}_2 \cdot \mathrm{mean}_{j \in \mathcal{N}(i)} \mathbf{x}_j$$

PARAMETERS:

- **in_channels** (*int or tuple*) – Size of each input sample. A tuple corresponds to the sizes of source and target dimensionalities.
- **out_channels** (*int*) – Size of each output sample.
- **normalize** (*bool, optional*) – If set to `True`, output features will be ℓ_2-normalized, *i.e.*, $\frac{\mathbf{x}'_i}{\|\mathbf{x}'_i\|_2}$. (default: `False`)
- **root_weight** (*bool, optional*) – If set to `False`, the layer will not add transformed root node features to the output. (default: `True`)
- **bias** (*bool, optional*) – If set to `False`, the layer will not learn an additive bias. (default: `True`)
- ****kwargs** (*optional*) – Additional arguments of `torch_geometric.nn.conv.MessagePassing`.

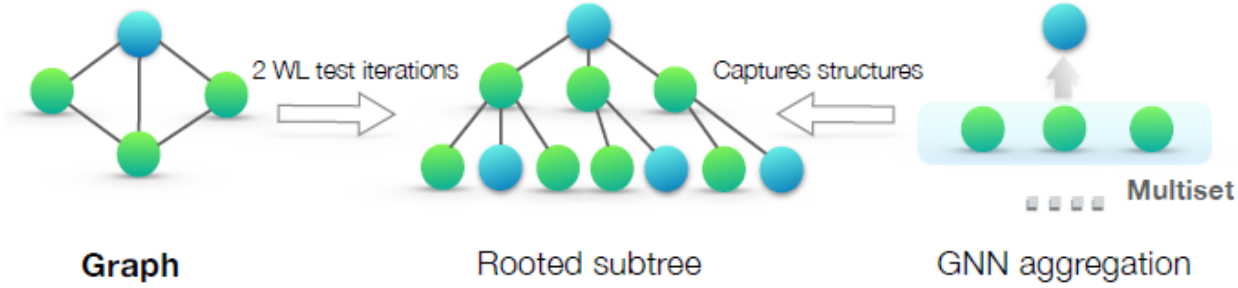
GIN

图神经网络(GNN)已被证明具有很强的学习能力。文章基于WL测试为衡量标准，提出了一种衡量GNN能力的表达。并经过理论推导得出，GNN需要获得和WL测试一样的能力的条件，并且证明了很多流行的GNN的变种，如GCN,SAGE等都不满足该条件。文章并提出了一种满足该条件的GNN，称为Graph Isomorphism Network (GIN).

图神经网络的架构可以统一表示为，AGGREGATE聚合邻居的信息，并将其COMBINE整合到自己的信息上。最终用READOUT读取结点的嵌入表示等。

$$a_v^{(k)} = \mathrm{AGGREGATE}^{(k)}\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right), \quad h_v^{(k)} = \mathrm{COMBINE}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right), \tag{2.1}$$

2层GNN其实类似于2次WL测试的迭代步骤。



GCN和SAGE的公式具体如下，

GNNs is crucial. A number of architectures for AGGREGATE have been proposed. In the pooling variant of GraphSAGE (Hamilton et al., 2017a), AGGREGATE has been formulated as

$$a_v^{(k)} = \text{MAX}\left(\left\{\text{ReLU}\left(W \cdot h_u^{(k-1)}\right),\ \forall u \in \mathcal{N}(v)\right\}\right), \tag{2.2}$$

where W is a learnable matrix, and MAX represents an element-wise max-pooling. The COMBINE step could be a concatenation followed by a linear mapping $W \cdot \left[h_v^{(k-1)}, a_v^{(k)}\right]$ as in GraphSAGE. In Graph Convolutional Networks (GCN) (Kipf & Welling, 2017), the element-wise *mean* pooling is used instead, and the AGGREGATE and COMBINE steps are integrated as follows:

$$h_v^{(k)} = \text{ReLU}\left(W \cdot \text{MEAN}\left\{h_u^{(k-1)},\ \forall u \in \mathcal{N}(v) \cup \{v\}\right\}\right). \tag{2.3}$$

Many other GNNs can be represented similarly to Eq. 2.1 (Xu et al., 2018; Gilmer et al., 2017).

For node classification, the node representation $h_v^{(K)}$ of the final iteration is used for prediction. For graph classification, the READOUT function aggregates node features from the final iteration to obtain the entire graph's representation h_G:

$$h_G = \text{READOUT}\left(\left\{h_v^{(K)} \mid v \in G\right\}\right). \tag{2.4}$$

READOUT can be a simple permutation invariant function such as summation or a more sophisticated graph-level pooling function (Ying et al., 2018; Zhang et al., 2018).

一个具备和WL测试一样的能力的网络在聚合函数上，必须能够将不同的多重集（邻居特征向量集合）映射到不同的向量。而GCN和SAGE都采用MEAN，MAX等方式聚合信息，但该聚合方式并不满足该要求。

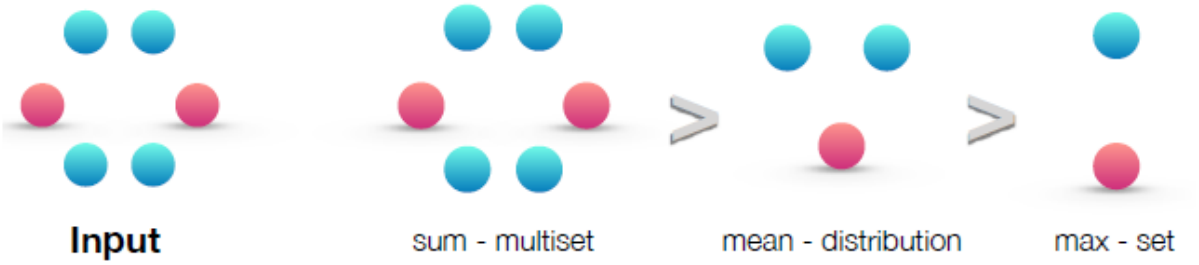
经过一番推导，文章得出以下定理

Corollary 6. *Assume $\mathcal{X}$ is countable. There exists a function $f : \mathcal{X} \to \mathbb{R}^n$ so that for infinitely many choices of ϵ, including all irrational numbers, $h(c, X) = (1+\epsilon) \cdot f(c) + \sum_{x \in X} f(x)$ is unique for each pair (c, X), where $c \in \mathcal{X}$ and $X \subset \mathcal{X}$ is a multiset of bounded size. Moreover, any function g over such pairs can be decomposed as $g(c, X) = \varphi\left((1+\epsilon) \cdot f(c) + \sum_{x \in X} f(x)\right)$ for some function φ.*

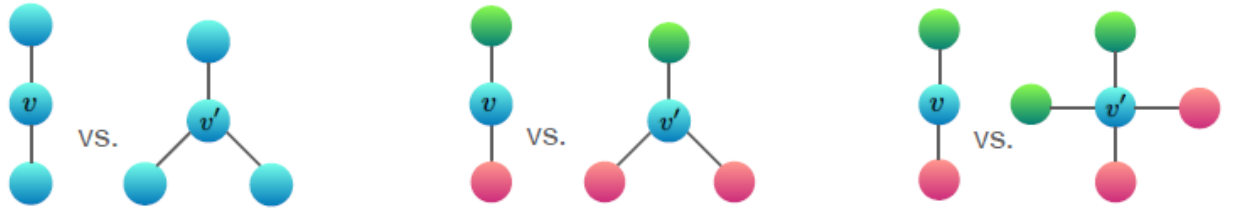
也即如下定义的GIN可以满足要求，

$$h_v^{(k)} = \text{MLP}^{(k)}\left(\left(1+\epsilon^{(k)}\right) \cdot h_v^{(k-1)} + \sum\nolimits_{u \in \mathcal{N}(v)} h_u^{(k-1)}\right). \tag{4.1}$$

下图形象表达了不同聚合函数(SUM,MEAN,MAX)学习到的信息，并且将其排序。



并且举了MAX函数不能区分但MEAN函数可以区分的例子。



(a) Mean and Max both fail (b) Max fails (c) Mean and Max both fail

在表示图结构的时候，用到了所有迭代步骤的结点向量表示，而非仅有最后一层的表示。这也与WL测试有关。

Theorem 3. *Let $\mathcal{A} : \mathcal{G} \to \mathbb{R}^d$ be a GNN. With a sufficient number of GNN layers, $\mathcal{A}$ maps any graphs G_1 and G_2 that the Weisfeiler-Lehman test of isomorphism decides as non-isomorphic, to different embeddings if the following conditions hold:*

a) $\mathcal{A}$ aggregates and updates node features iteratively with

$$h_v^{(k)} = \phi\left(h_v^{(k-1)}, f\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right)\right),$$

where the functions f, which operates on multisets, and ϕ are injective.

b) $\mathcal{A}$'s graph-level readout, which operates on the multiset of node features $\left\{h_v^{(k)}\right\}$, is injective.

由定理3可以得出，用如下的READOUT函数可以使得图表示是单射。

$$h_G = \text{CONCAT}\left(\text{READOUT}\left(\left\{h_v^{(k)} | v \in G\right\}\right) \mid k = 0, 1, \ldots, K\right). \tag{4.2}$$

By Theorem 3 and Corollary 6, if GIN replaces READOUT in Eq. 4.2 with summing all node features from the same iterations (we do not need an extra MLP before summation for the same reason as in Eq. 4.1), it provably generalizes the WL test and the WL subtree kernel.

文章接着分析了GCN和SAGE，主要有以下发现

- 之前的方法的单隐层结构表示是不足够的，并用定理证明并举了反例
- MEAN和MAX聚合都存在不能学习的情况，证明并举了反例
- MEAN聚合学习到了邻居结点的分布，并证明
- MAX聚合学习到了不重复集合的分布，并证明

实验与其证明不够强大的GNN其他变种进行对比，并且与WL测试进行对比。

代码总结

GIN公式如下，可以整体表示也可以具体到每一个结点

CLASS **GINConv** (**nn**: Callable, **eps**: float = `0.0`, **train_eps**: bool = `False`, ****kwargs**) [source]

The graph isomorphism operator from the "How Powerful are Graph Neural Networks?" paper

$$\mathbf{x}_i' = h_{\mathbf{\Theta}}\left((1+\epsilon)\cdot\mathbf{x}_i + \sum_{j\in\mathcal{N}(i)}\mathbf{x}_j\right)$$

or

$$\mathbf{X}' = h_{\mathbf{\Theta}}\left((\mathbf{A}+(1+\epsilon)\cdot\mathbf{I})\cdot\mathbf{X}\right),$$

here $h_{\mathbf{\Theta}}$ denotes a neural network, *.i.e.* an MLP.

PARAMETERS:

- **nn** (*torch.nn.Module*) – A neural network $h_{\mathbf{\Theta}}$ that maps node features `x` of shape `[-1, in_channels]` to shape `[-1, out_channels]`, *e.g.*, defined by `torch.nn.Sequential`.
- **eps** (*float*, *optional*) – (Initial) ϵ-value. (default: `0.`)
- **train_eps** (*bool*, *optional*) – If set to `True`, ϵ will be a trainable parameter. (default: `False`)
- ****kwargs** (*optional*) – Additional arguments of `torch_geometric.nn.conv.MessagePassing`.

GAT

图自注意力卷积，启发于NLP中的attention机制。

允许每个结点自己学习到不同邻居对其的重要性，并且可以在不借助任何复杂运算（如矩阵求逆等）的前提下实现。

文章将之前的方法主要分为两类，

一是基于图谱结构的方法，从图信号分解中基于拉普拉斯矩阵的方法，到用切比雪夫卷积的方法ChebConv降低了时间复杂度，到GCN用一跳邻居结构简化了ChebConv。但这些方法本质上都基于图的拉普拉斯矩阵，所以本质上利用率图的结构信息，使得方法本身对于转导学习具有较明显的弊端，在图结构改变的条件下方法的弊端暴露出来。

二是不基于图谱结构的方法，比如直接将CNN的方法用于图结构上，但CNN基于权重共享，并不能对不同的结点有不同的参数表示。再到SAGE，基于采样和聚合的手段，每次采样一定数目的邻居，使得方法对于转导学习更为鲁棒。

文章提出GAT的方法，不需要利用图谱结构，同时允许不同的结点有不同的参数，并且可以自己学习到邻居的重要性，从而不需要像SAGE一样显式采样。

方法有以下几个优点，

- 计算高效，有很好的并行性

- 可以对不同的邻居分配不同的权重
- 可以直接运用于转导学习
- 在很多数据集上，如学术引用网络、蛋白质交互网络上取得了很好的表现

GAT核心公式如下

$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j) \tag{1}$$

that indicate the *importance* of node j's features to node i. In its most general formulation, the model allows every node to attend on every other node, *dropping all structural information*. We inject the 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 be exactly the first-order neighbors of i (including i). To make coefficients easily comparable across different nodes, we normalize them across all choices of j using the softmax function:

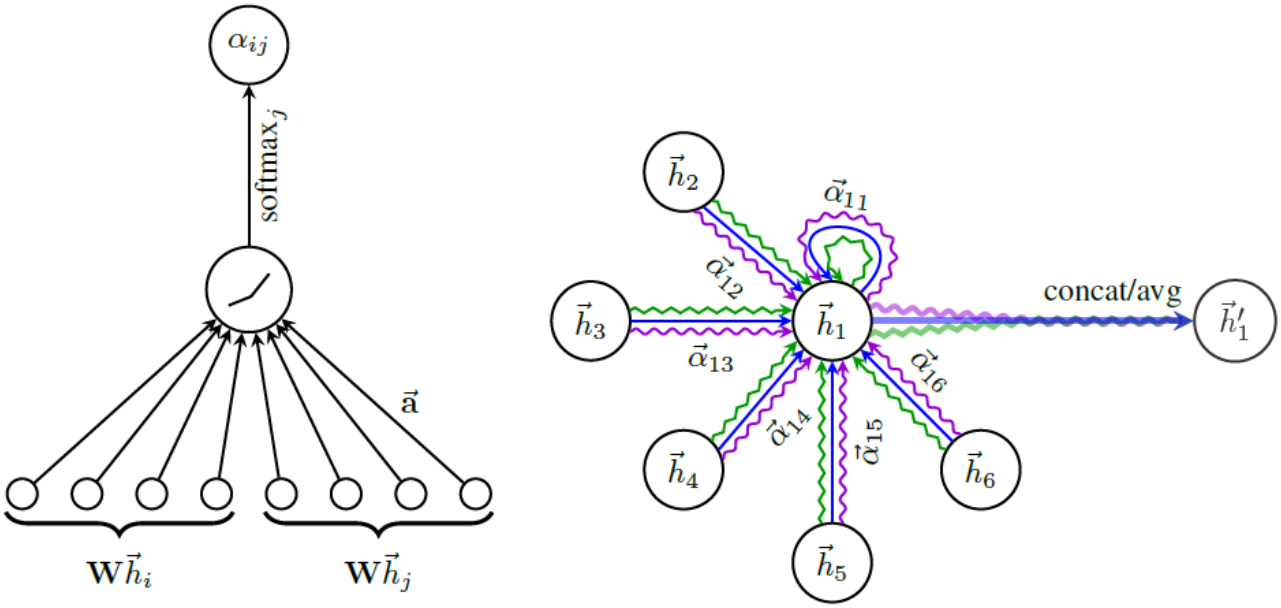
$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}_i}\exp(e_{ik})}. \tag{2}$$

In our experiments, the attention mechanism a is a single-layer feedforward neural network, parametrized by a weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2F'}$, and applying the LeakyReLU nonlinearity (with negative input slope $\alpha = 0.2$). Fully expanded out, the coefficients computed by the attention mechanism (illustrated by Figure 1 (left)) may then be expressed as:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)} \tag{3}$$

where $\cdot^T$ represents transposition and $\|$ is the concatenation operation.

下图展示了GAT的计算图，



GAT模型对于多头注意力机制同样支持，只需要将不同头的K个注意力连接起来，但特别的，在最后一层(预测层)，用均值操作取代连接操作。

$$\vec{h}_i' = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right) \tag{5}$$

where $\|$ represents concatenation, α_{ij}^k are normalized attention coefficients computed by the k-th attention mechanism (a^k), and $\mathbf{W}^k$ is the corresponding input linear transformation's weight matrix. Note that, in this setting, the final returned output, $\mathbf{h}'$, will consist of KF' features (rather than F') for each node.

Specially, if we perform multi-head attention on the final (prediction) layer of the network, concatenation is no longer sensible—instead, we employ *averaging*, and delay applying the final nonlinearity (usually a softmax or logistic sigmoid for classification problems) until then:

$$\vec{h}_i' = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right) \tag{6}$$

The aggregation process of a multi-head graph attentional layer is illustrated by Figure 1 (right).

文章详细分析了模型的其他优点;

- 模型为一个结点的邻居分配了不同的重要度排序，也具有很好的可解释性，可以通过一些可视化手段展示。
- 模型并不要求图是无向图，由于注意力只通过有向边传递。
- 注意力只利用了图的局部结构，使得模型直接适用于转导学习。
- SAGE基于采样的手段，使得一个结点不能访问到其所有的邻居，而GAT没有此弊端。
- 基于LSTM聚合的SAGE方法，需要利用随机打乱使得LSTM学习不到序列顺序，而注意力机制天然是无序的。

回顾核心公式，其中采用了LeakyRelu使得注意力不会为0.

CLASS **GATConv** (**in_channels**: Union[int, Tuple[int, int]], **out_channels**: int, **heads**: int = `1`, **concat**: bool = `True`, **negative_slope**: float = `0.2`, **dropout**: float = `0.0`, **add_self_loops**: bool = `True`, **bias**: bool = `True`, ****kwargs**) [source]

The graph attentional operator from the "Graph Attention Networks" paper

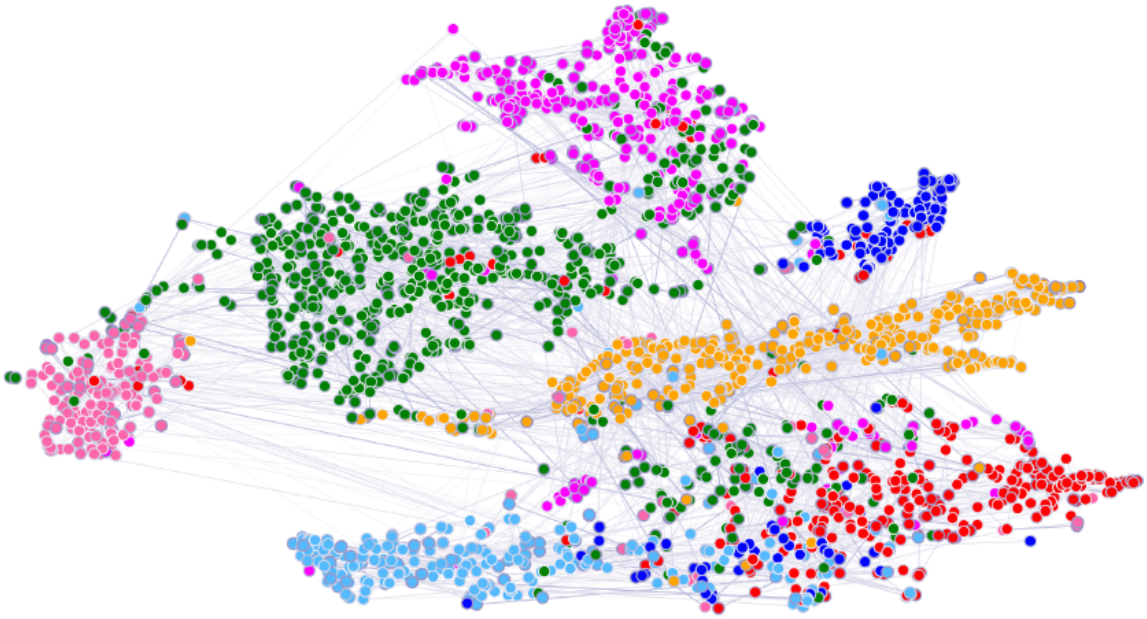
$$\mathbf{x}_i' = \alpha_{i,i}\mathbf{\Theta}\mathbf{x}_i + \sum_{j\in\mathcal{N}(i)} \alpha_{i,j}\mathbf{\Theta}\mathbf{x}_j,$$

where the attention coefficients $\alpha_{i,j}$ are computed as

$$\alpha_{i,j} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}^\top[\mathbf{\Theta}\mathbf{x}_i \,\|\, \mathbf{\Theta}\mathbf{x}_j]\right)\right)}{\sum_{k\in\mathcal{N}(i)\cup\{i\}}\exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}^\top[\mathbf{\Theta}\mathbf{x}_i \,\|\, \mathbf{\Theta}\mathbf{x}_k]\right)\right)}.$$

实验部分，主要于SAGE-MEAN/MAX/LSTM进行了对比。

下图为模型学习到的特征用TSNE方法降维后的可视化，



VGAE

VGAE，图变分自编码器。假设Z的分布在给定X、A的条件下满足高维正态分布。实际上，GCN模型只需要学习正态分布的均值和方差两个参数即可。损失由变分性质推得。

Inference model We take a simple inference model parameterized by a two-layer GCN:

$$q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A}) = \prod_{i=1}^{N} q(\mathbf{z}_i \mid \mathbf{X}, \mathbf{A}), \quad \text{with} \quad q(\mathbf{z}_i \mid \mathbf{X}, \mathbf{A}) = \mathcal{N}(\mathbf{z}_i \mid \boldsymbol{\mu}_i, \mathrm{diag}(\boldsymbol{\sigma}_i^2)). \tag{1}$$

Here, $\boldsymbol{\mu} = \mathrm{GCN}_{\boldsymbol{\mu}}(\mathbf{X}, \mathbf{A})$ is the matrix of mean vectors $\boldsymbol{\mu}_i$; similarly $\log \boldsymbol{\sigma} = \mathrm{GCN}_{\boldsymbol{\sigma}}(\mathbf{X}, \mathbf{A})$. The two-layer GCN is defined as $\mathrm{GCN}(\mathbf{X}, \mathbf{A}) = \tilde{\mathbf{A}}\,\mathrm{ReLU}\big(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W}_0\big)\mathbf{W}_1$, with weight matrices $\mathbf{W}_i$. $\mathrm{GCN}_{\boldsymbol{\mu}}(\mathbf{X}, \mathbf{A})$ and $\mathrm{GCN}_{\boldsymbol{\sigma}}(\mathbf{X}, \mathbf{A})$ share first-layer parameters $\mathbf{W}_0$. $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$ and $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ is the symmetrically normalized adjacency matrix.

Generative model Our generative model is given by an inner product between latent variables:

$$p(\mathbf{A} \mid \mathbf{Z}) = \prod_{i=1}^{N} \prod_{j=1}^{N} p(A_{ij} \mid \mathbf{z}_i, \mathbf{z}_j), \quad \text{with} \quad p(A_{ij} = 1 \mid \mathbf{z}_i, \mathbf{z}_j) = \sigma(\mathbf{z}_i^{\top} \mathbf{z}_j), \tag{2}$$

where A_{ij} are the elements of $\mathbf{A}$ and $\sigma(\cdot)$ is the logistic sigmoid function.

Learning We optimize the variational lower bound $\mathcal{L}$ w.r.t. the variational parameters $\mathbf{W}_i$:

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A})}\big[\log p(\mathbf{A} \mid \mathbf{Z})\big] - \mathrm{KL}\big[q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A}) \,||\, p(\mathbf{Z})\big], \tag{3}$$

VGAE的特例是GAE，图自编码器，如下定义。编码器由GCN定义，学习结点的嵌入Z，尽可能重构邻接矩阵A。GAE实际上是VGAE的无概率特例。

Non-probabilistic *graph auto-encoder* (GAE) model For a non-probabilistic variant of the VGAE model, we calculate embeddings $\mathbf{Z}$ and the reconstructed adjacency matrix $\hat{\mathbf{A}}$ as follows:

$$\hat{\mathbf{A}} = \sigma\big(\mathbf{Z}\mathbf{Z}^{\top}\big), \quad \text{with} \quad \mathbf{Z} = \mathrm{GCN}(\mathbf{X}, \mathbf{A}). \tag{4}$$

VGAE在链接预测上对比了谱聚类Spetial Cluster、DeepWalk等传统方法，取得很好的效果。

可视化学习得到的隐变量如下图，

ARGAE

对抗正则图自编码器，将GAN，AAE的方法引入GNN。

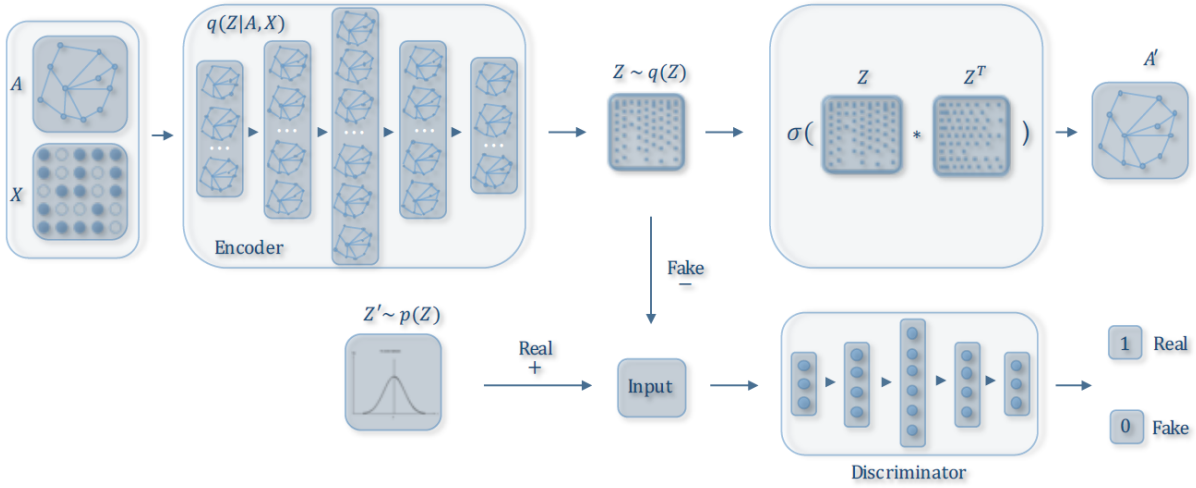
很多图神经网络致力于保持图的拓扑结构（GCN使得相邻结点的表示向量尽可能相似）或减小图重构损失（GAE），但没有考虑分布规律。ARGAE利用了对抗模型的方式，同时考虑了结点的分布规律。

之前的图表示学习方法大致可以分为三类，

- 基于概率的方法，node2vec,deepwalk等
- 基于矩阵分解的方法，pagerank，谱聚类等
- 基于深度学习的方法

但上述方法都没有考虑隐变量的分布规律。

模型结构如下，



算法流程如下，同时训练生成模型和判别模型。

Algorithm 1 Adversarially Regularized Graph Embedding

Require:
$\mathbf{G} = \{\mathbf{V}, \mathbf{E}, \mathbf{X}\}$: a Graph with links and features;
T: the number of iterations;
K: the number of steps for iterating discriminator;
d: the dimension of the latent variable
Ensure: $\mathbf{Z} \in \mathbb{R}^{n \times d}$

1: **for** iterator = 1,2,3, $\cdots\cdots$, T **do**
2: Generate latent variables matrix $\mathbf{Z}$ through Eq.(4);
3: **for** k = 1,2, $\cdots\cdots$, K **do**
4: Sample m entities $\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\}$ from latent matrix $\mathbf{Z}$
5: Sample m entities $\{\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(m)}\}$ from the prior distribution p_z
6: Update the discriminator with its stochastic gradient:

$$\nabla \frac{1}{m} \sum_{i=1}^{m} [\log \mathcal{D}(\mathbf{a}^i) + \log (1 - \mathcal{D}(\mathbf{z}^{(i)}))]$$

end for
7: Update the graph autoencoder with its stochastic gradient by Eq. (10) for ARGA or Eq. (11) for ARVGA;
end for
8: **return** $\mathbf{Z} \in \mathbb{R}^{n \times d}$

任务是链接预测和结点聚类，对比方法有DeepWalk，谱聚类、KNN聚类等。

DropEdge

深层CNN已经被证明具有很强的学习能力，但相较之下深层GCN却没有很好的效果。文章指出深层GCN具有过拟合和过平滑的问题，提出DropEdge，即在训练过程中随机删去一些边，并证明该方法可以减轻过拟合和过平滑的问题，并可以很好地应用在浅层和深层GCN上提高其表现。DropEdge可以理解为一种数据扩增的形式，也可以理解为一种消息传递的减速器。文章同时对DropEdge为什么能减缓过平滑问题做了理论证明分析。

DropEdge可以和以下方法对比，

- Dropout，DropEdge可以堪称Dropout在GCN中的情形，故DropEdge具有Dropout的优点
- DropNode，如SAGE基于随机采样的方法，但该方法在训练过程中丧失了部分结点的特征
- Graph-Sparsification，一种去掉图中冗余边的方法，但DropEdge并不需要显式进行此任务

Points

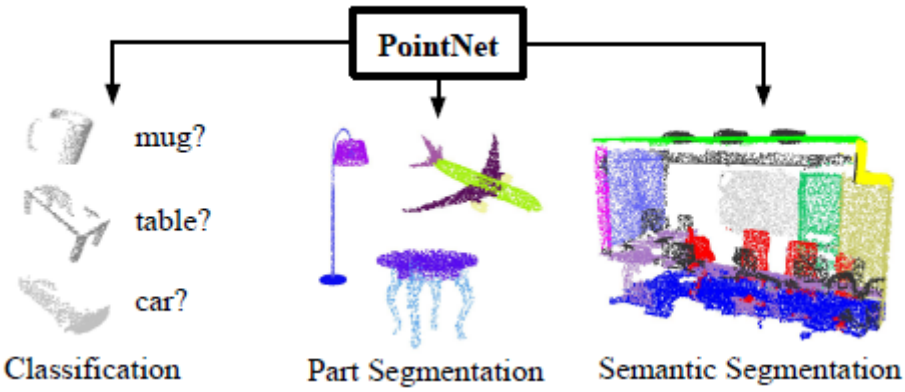
PointNet

3d点云是重要的3d数据表示形式，先前将点云数据转化为体素表示加3d卷积的方法非常消耗时间和空间。PointNet直接学习点云数据，并且学习到点云数据的排列不变性质。

之前基于深度学习的方式处理3d物体有几种方式，

- 基于体素，但只适用于稀疏情况
- 基于多视角，将3d问题转化为2d问题，但对于场景理解等问题不直接
- 基于流行学习，但流形应用于家具等物体上存在困难
- 将3d特征转化为向量表示，但模型能力取决于向量表示本身

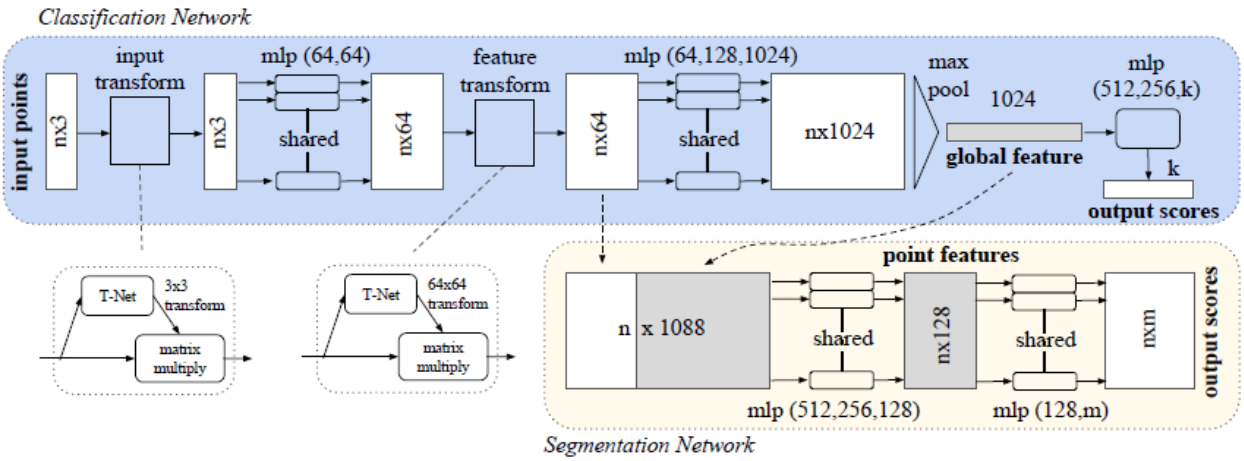
文章提出PointNet可以胜任分类分割等任务，



模型结构如下，其中T-Net为变换网络,结合多个共享权重的MLP提取到排列不变特征，再使用最大值池化提取全局特征。

对于分类任务，将全局特征经过最后一层mlp分类。

对于分割任务，将全局特征和局部特征结合，使用卷积得到每个点的分割结果。



点云数据具有以下特征，

- 无序性
- 结点交互性
- 平移旋转不变性

PoinyNet采用对称函数，最大值池化学习数据的无序性。

文章对比了其他几种无序性学习的方式，如排序，乱序RNN等，均指出其弊端，对比说明最大值池化的好处。

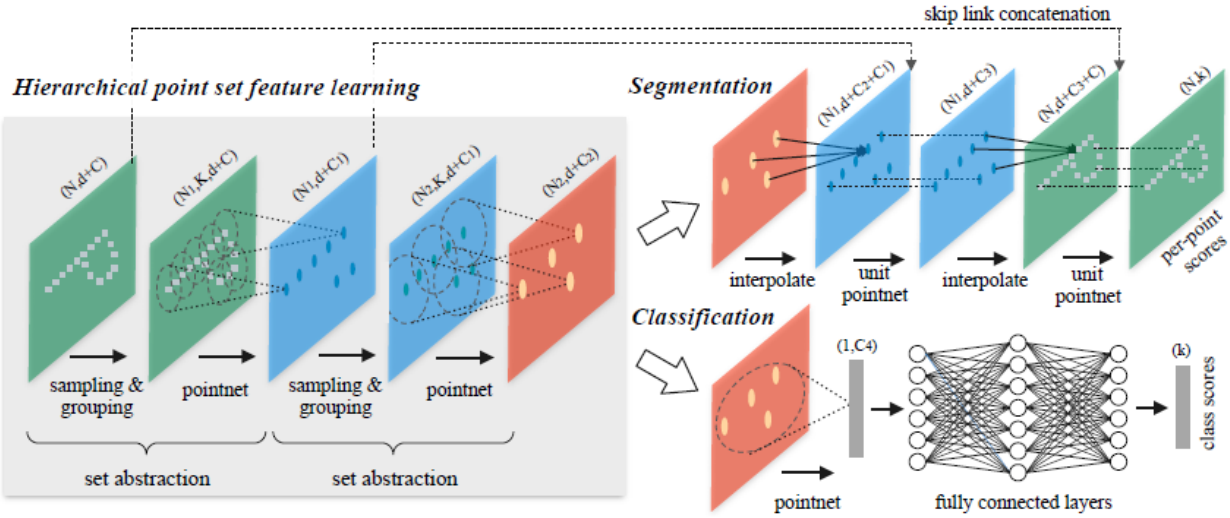
文章使用共享权重的MLP学习局部特征，从而学习结点交互性。对于旋转不变性，比起其他类型的数据，点云数据的平移旋转实现简单，只需要使用T-Net对每个点进行仿射变换。

实验与3dCNN进行对比。在准确率和时间上均大部分超过。

PointNet++

改进PointNet，主要解决PointNet并没有很好地学习到点云在度量空间中的局部特性的问题。并且针对点云数据非均匀分布的特性，提出更为鲁棒的策略。

模型结构如下，下采样部分使用采样和卷积交替进行，对于分类任务将下采样提取的特征经过三层全连接层得到最后每一类的分数，对于分割任务使用上采样得到每个点的分数。



层级特征提取，每一层由采样、分组、MLP层三个步骤组成

采样采用最远点采样（FPS），该方法相对于随机采样更能从样本中得到点与点之间的依存关系。

Sampling layer. Given input points $\{x_1, x_2, ..., x_n\}$, we use iterative farthest point sampling (FPS) to choose a subset of points $\{x_{i_1}, x_{i_2}, ..., x_{i_m}\}$, such that x_{i_j} is the most distant point (in metric distance) from the set $\{x_{i_1}, x_{i_2}, ..., x_{i_{j-1}}\}$ with regard to the rest points. Compared with random sampling, it has better coverage of the entire point set given the same number of centroids. In contrast to CNNs that scan the vector space agnostic of data distribution, our sampling strategy generates receptive fields in a data dependent manner.

根据采样点作为中心点分组，分组有两种方式，基于球半径和K近邻，也即寻找位于以中心点为球心的球内的点或者寻找k近邻点。文章建议用球半径的方式以更好地保持局部空间结构。

Grouping layer. The input to this layer is a point set of size $N \times (d + C)$ and the coordinates of a set of centroids of size $N' \times d$. The output are groups of point sets of size $N' \times K \times (d + C)$, where each group corresponds to a local region and K is the number of points in the neighborhood of centroid points. Note that K varies across groups but the succeeding *PointNet layer* is able to convert flexible number of points into a fixed length local region feature vector.

MLP层，相比于PointNet直接将每个点的坐标作为MLP的2输入，PointNet++中的MLP每个点相对于中心点的相对坐标作为输入，以此捕获点之间的局部相对关系。

PointNet layer. In this layer, the input are N' local regions of points with data size $N' \times K \times (d+C)$. Each local region in the output is abstracted by its centroid and local feature that encodes the centroid's neighborhood. Output data size is $N' \times (d + C')$.

The coordinates of points in a local region are firstly translated into a local frame relative to the centroid point: $x_i^{(j)} = x_i^{(j)} - \hat{x}^{(j)}$ for $i = 1, 2, ..., K$ and $j = 1, 2, ..., d$ where $\hat{x}$ is the coordinate of the centroid. We use PointNet [20] as described in Sec. 3.1 as the basic building block for local pattern learning. By using relative coordinates together with point features we can capture point-to-point relations in the local region.

对于点云数据非均匀分布的特点，采用多尺度采样MSG和多分辨率采样MRG。两种方法的示意图见下，相当于用不同尺度/不同分辨率采样，然后将多次采样结果拼接。文章同时对比了两种方法分别的利弊。

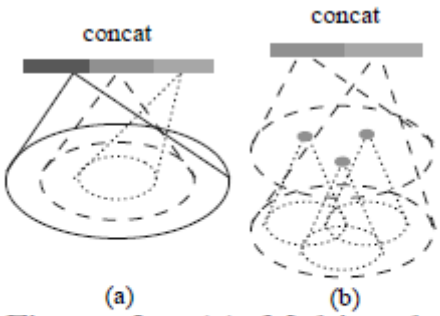
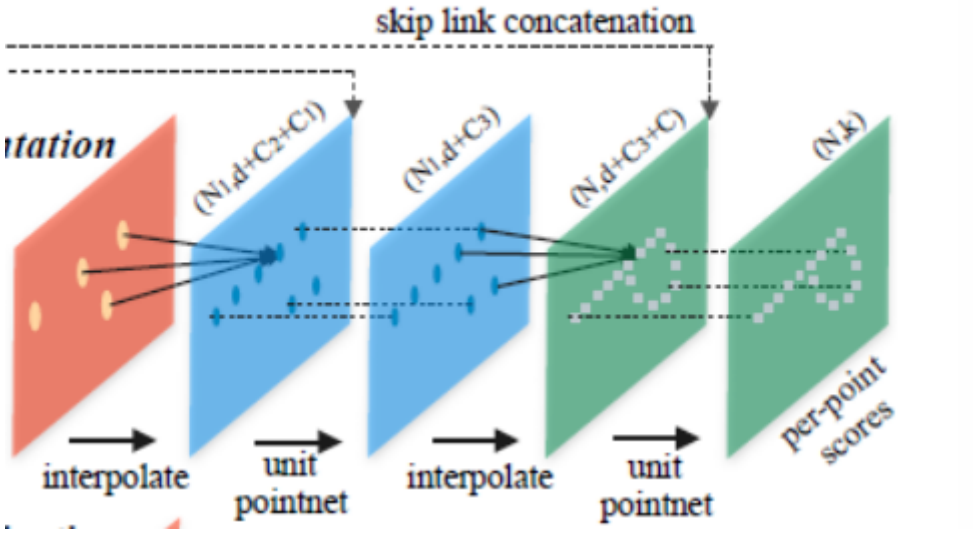


Figure 3: (a) Multi-scale grouping (MSG); (b) Multi-resolution grouping (MRG).

对于分割任务，需要进行上采样，即从采样的部分点的特征推广到每个点的特征，文章采用了特征传播的方法，传播时进行类似1x1卷积的网络层unit pointnet进行卷积操作，然后用如下公式进行插值。



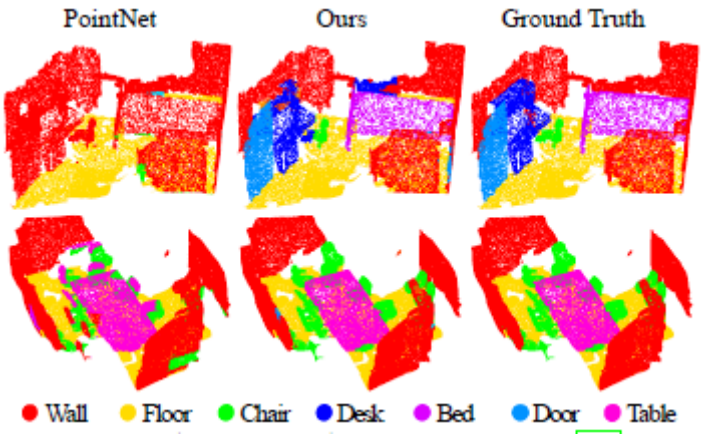
插值公式如下，

$$f^{(j)}(x) = \frac{\sum_{i=1}^{k} w_i(x) f_i^{(j)}}{\sum_{i=1}^{k} w_i(x)} \quad \text{where} \quad w_i(x) = \frac{1}{d(x, x_i)^p}, \; j = 1, ..., C \tag{2}$$

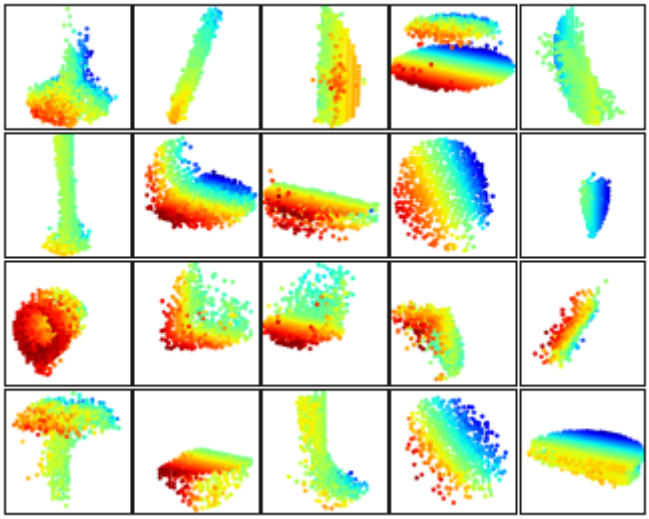
同时采用了跳层连接，以同时获取不同尺度的特征。

该方法可用于体素表示、图片（如MNIST手写数据集)、点云数据（ModelNet40），文章对此与简单的MLP、卷积神经网络、PointNet进行对比。并探究了MSG、MRG等不同技术的作用。

下图展示了PointNet++相对于PointNet不能很好分割的例子的效果。



可视化隐藏层的激活情况，

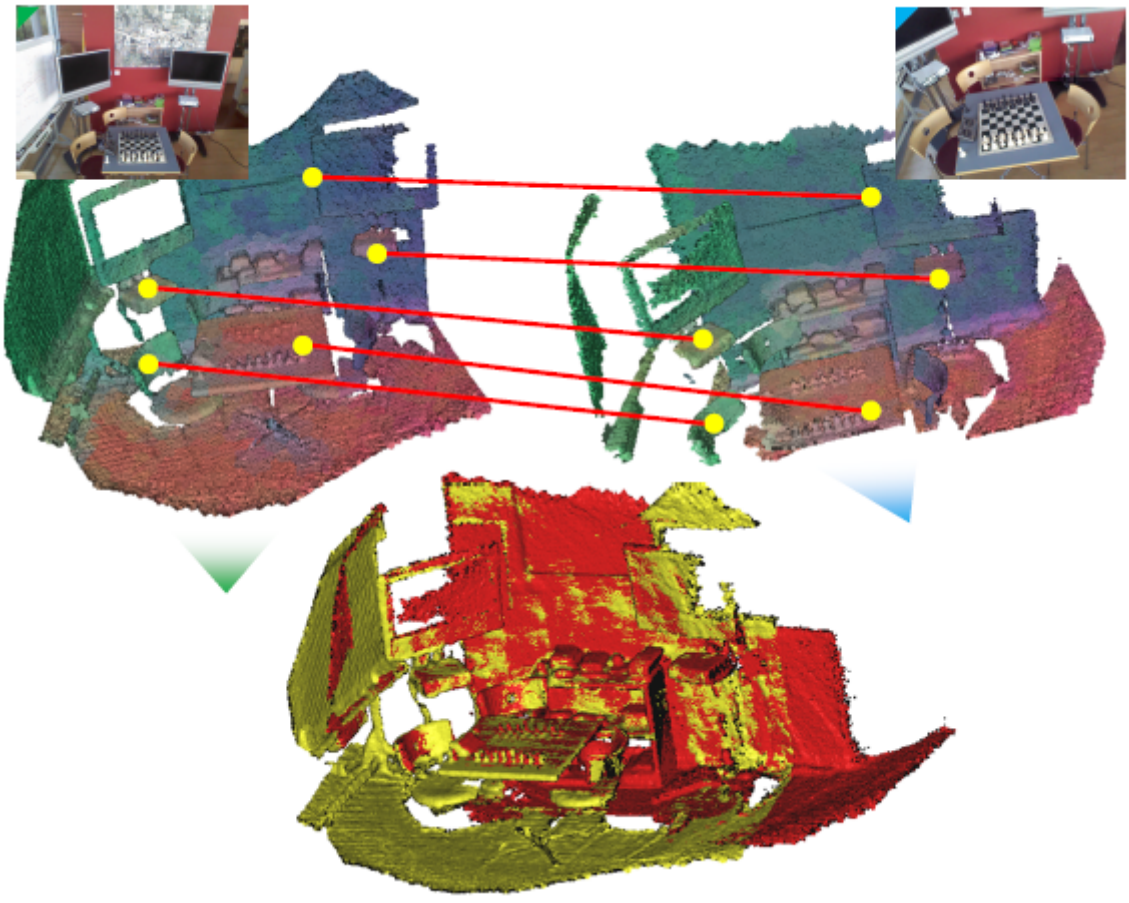


文章同时提及该方法也可以推广到非欧空间上，对流形等结构数据也同样适用。

PPF

PPF(Point Pair Feature) ,尝试获取点对之间的特征对点云数据进行学习。文章指出，PointNet很好地学习到了点云数据的排列不变性、PointNet++学习到了点云密度分布的特征。但PPF学到了点云数据的更多结构性特征。

文章动机可以用下图阐释，学习到了点与点之间的相互关系特征更有利于点云学习。



点云数据具有在刚体变换（平移、旋转）不变的性质，

PPF特征如下定义，其中n为每个点相对于物体表面地法向量，可以证明如下方式定义得到的PPF具有刚体变换下的不变性。

Point Pair Features (PPF) Point pair features are anti-symmetric 4D descriptors, describing the surface of a pair of oriented 3D points $\mathbf{x}_1$ and $\mathbf{x}_2$, constructed as:

$$\psi_{12} = (\|\mathbf{d}\|_2, \angle(\mathbf{n}_1, \mathbf{d}), \angle(\mathbf{n}_2, \mathbf{d}), \angle(\mathbf{n}_1, \mathbf{n}_2)) \quad (4)$$

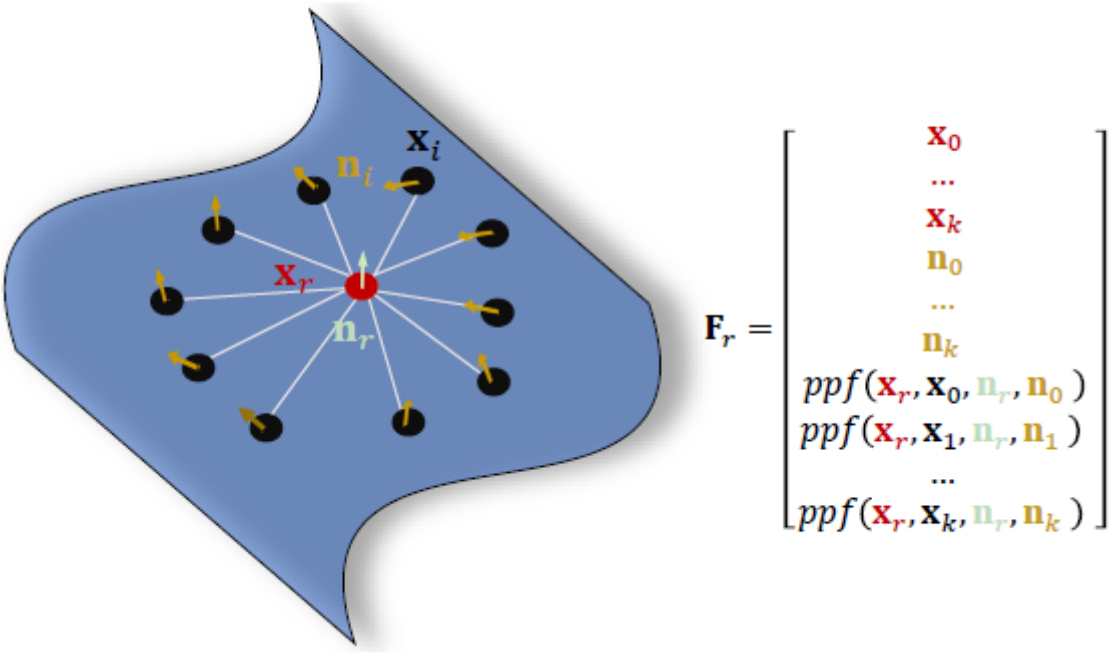
where $\mathbf{d}$ denotes the difference vector between points, $\mathbf{n}_1$ and $\mathbf{n}_2$ are the surface normals at $\mathbf{x}_1$ and $\mathbf{x}_2$. $\|\cdot\|$ is the Euclidean distance and $\angle$ is the angle operator computed in a numerically robust manner as in [2]:

$$\angle(\mathbf{v}_1, \mathbf{v}_2) = \text{atan2}\left(\|\mathbf{v}_1 \times \mathbf{v}_2\|, \mathbf{v}_1 \cdot \mathbf{v}_2\right) \quad (5)$$

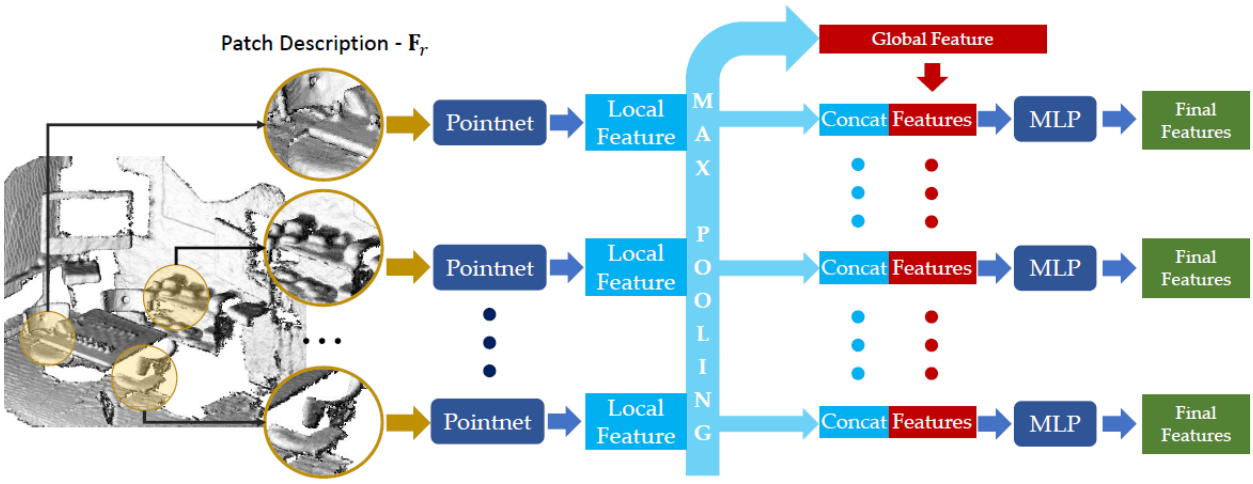
网络将每个结点的位置信息、法向量信息、点对之间的PPF信息作为MLP的输入。

$$\mathbf{F}_r = \{\mathbf{x}_r, \mathbf{n}_r, \mathbf{x}_i, \cdots, \mathbf{n}_i, \cdots, \psi_{ri}, \cdots\} \quad (6)$$

用下图表示，



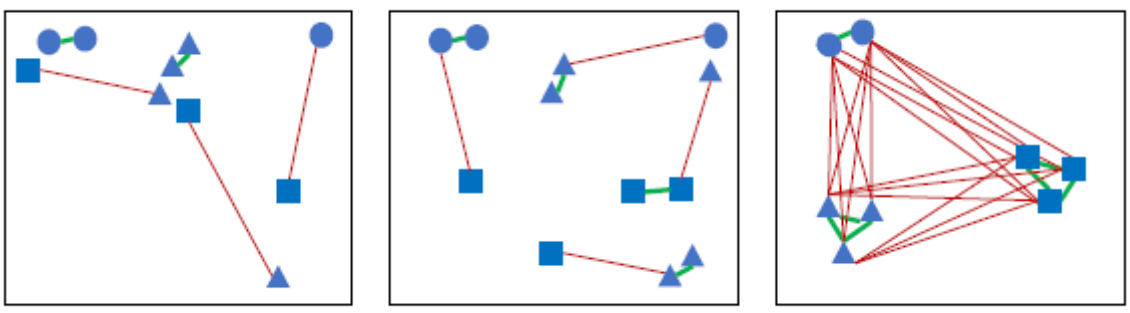
结构如下，与PointNet相同地采用了先提取局部特征，再提取全局特征的方式，区别在于输入的为每个局部的Patch Description，每个Patch Description定义为上述F_R，



网络应当学习到Patch的局部特征，因此希望邻近的patch具有相似的特征，而较远的patch具有差别较大的特征，

常用的有2元组损失、3元组损失等，但文章指出这都可以归结为N元组损失，

NOTE:下图中绿色边表示相近的patch，应该鼓励特征的相似性，而红色相反，



借助哈达玛乘积等数学符号，N元组损失可以简单表达，也即在计算中，将实际的距离通过阈值t二进制化为M，与预测的距离进行计算，

Generalizing these losses to N-patches, we propose N-tuple loss, an N-to-N contrastive loss, to correctly learn to solve this combinatorial problem by catering for the many-to-many relations as depicted in Fig. 4. Given the ground truth transformation $\mathbf{T}$, N-tuple loss operates by constructing a correspondence matrix $\mathbf{M} \in \mathbb{R}^{N \times N}$ on the points of the aligned fragments. $\mathbf{M} = (m_{ij})$ where:

$$m_{ij} = \mathbb{1}(\|\mathbf{x}_i - \mathbf{T}\mathbf{y}_j\|_2 < \tau) \tag{7}$$

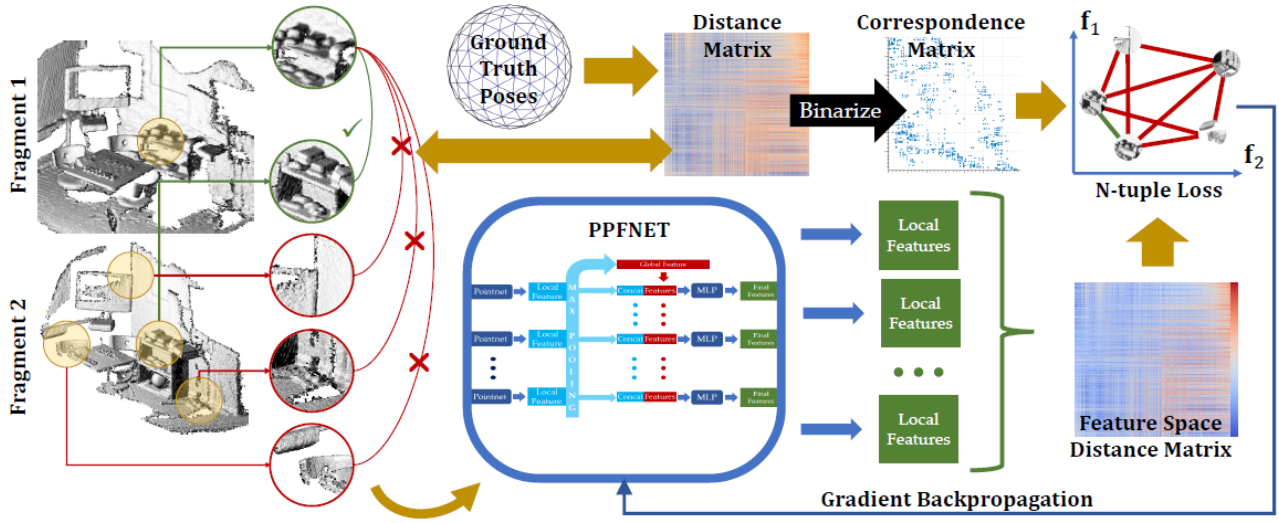
$\mathbb{1}$ is an indicator function. Likewise, we compute a feature-space distance matrix $\mathbf{D} \in \mathbb{R}^{N \times N}$ and $\mathbf{D} = (d_{ij})$ where

$$d_{ij} = \|f(\mathbf{x}_i) - f(\mathbf{y}_j)\|_2 \tag{8}$$

The N-tuple loss then functions on the two distance matrices solving the correspondence problem. For simplisity of expression, we define an operation $\sum^*(\cdot)$ to sum up all the elements in a matrix. N-tuple loss can be written as:

$$L = \sum{}^* \left(\frac{\mathbf{M} \circ \mathbf{D}}{\|\mathbf{M}\|_2^2} + \alpha \frac{max(\theta - (1 - \mathbf{M}) \circ \mathbf{D}, 0)}{N^2 - \|\mathbf{M}\|_2^2} \right) \tag{9}$$

整体结构表示如下，

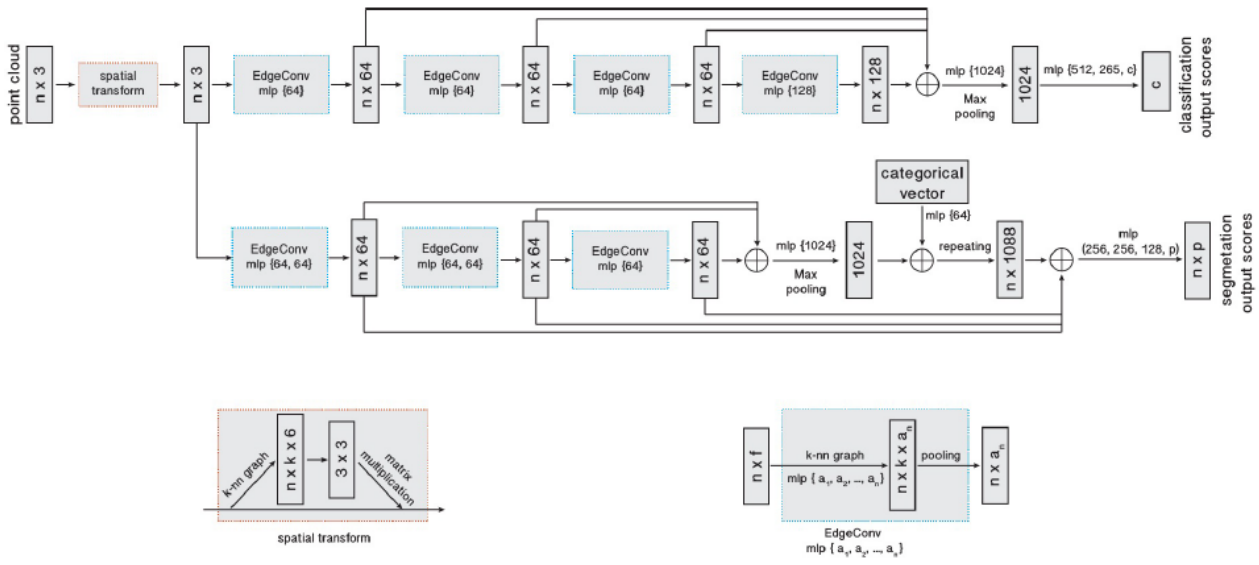


总结，PFF利用PPF特征获取局部信息，通过N元组损失获取全局信息，并用实验证明，将两者运用于简单的PointNet上均能提升模型的效果，而最好的模型是结合使用所有技术的模型。

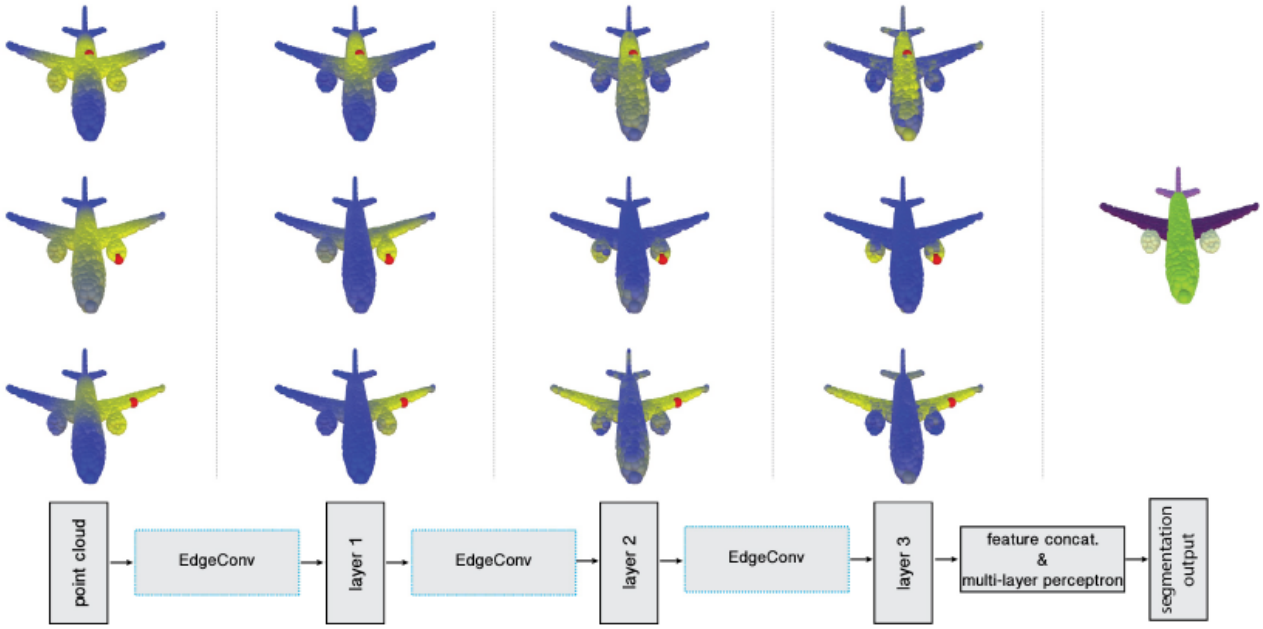
DGCNN

动态图卷积模型，总结前人的方法提出通用性的EdgeConv，指出该方法可以很好地学习局部特征，同时堆叠起来学习全局特征。

模型结构和PointNet基本一致，区别是用EdgeConv取代了PointNet中的MLP。与PointNet使用全部节点、PointNet++使用采样的部分结点的静态图方法不同，DGCNN基于每一层的特征动态构建k近邻图，进行空间变换和边卷积操作。



下图形象展示了动态近邻图，红色结点为中心点，黄色结点表示在特征空间上与中心点更接近的结点。



该图展示了EdgeConv，

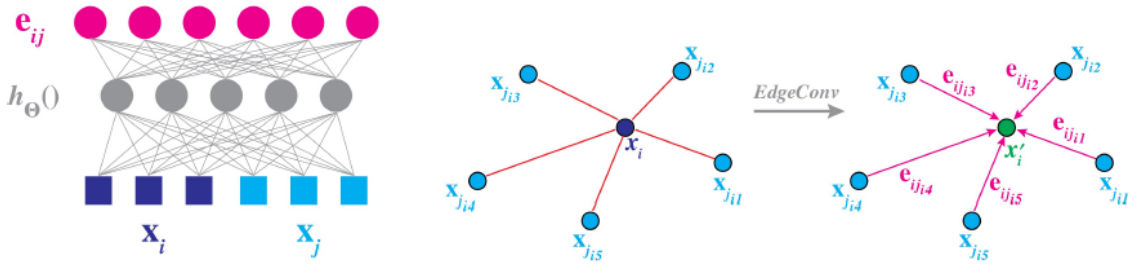


Fig. 2. **Left**: Computing an edge feature, $\mathbf{e}_{ij}$ (top), from a point pair, $\mathbf{x}_i$ and $\mathbf{x}_j$ (bottom). In this example, $h_\Theta()$ is instantiated using a fully connected layer, and the learnable parameters are its associated weights. **Right**: The EdgeConv operation. The output of EdgeConv is calculated by aggregating the edge features associated with all the edges emanating from each connected vertex.

EdgeConve可以写为如下公式，方框可为Sum或Max操作，

$$\mathbf{x}'_i = \square_{j:(i,j)\in\mathcal{E}} h_\Theta(\mathbf{x}_i, \mathbf{x}_j). \qquad (1)$$

而PointNet，PointNet++中使用的方法其实均为EdgeConv的特例，

DGCNN is related to two classes of approaches, PointNet and graph CNNs, which we show to be particular settings of our method. We summarize different methods in Table 1.

PointNet is a special case of our method with $k = 1$, yielding a graph with an empty edge set $\mathcal{E} = \varnothing$. The edge function used in PointNet is $h_{\Theta}(\mathrm{x}_i, \mathrm{x}_j) = h_{\Theta}(\mathrm{x}_i)$, which considers global but not local geometry. PointNet++ tries to account for local structure by applying PointNet in a local manner. In our parlance, PointNet++ first constructs the graph according to the Euclidean distances between the points, and in each layer applies a graph coarsening operation. For each layer, some points are selected using farthest point sampling (FPS); only the selected points are preserved while others are directly discarded after this layer. In this way, the graph becomes smaller after the operation applied on each layer. In contrast to DGCNN, PointNet++ computes pairwise distances using point input coordinates, and hence their graphs are fixed during training. The edge function used by PointNet++ is $h_{\Theta}(\mathrm{x}_i, \mathrm{x}_j) = h_{\Theta}(\mathrm{x}_j)$, and the aggregation operation is also a max.

其他的一些方法也可覆盖在EdgeConv的框架下，文章建议使用如下地EdgeConv，同时获取结点绝对特征和相对特征，

$$h_{\Theta}(\mathrm{x}_i, \mathrm{x}_j) = \bar{h}_{\Theta}(\mathrm{x}_i, \mathrm{x}_j - \mathrm{x}_i). \tag{7}$$

对于分割任务，与PointNet进行了对比，部分结果可视化，

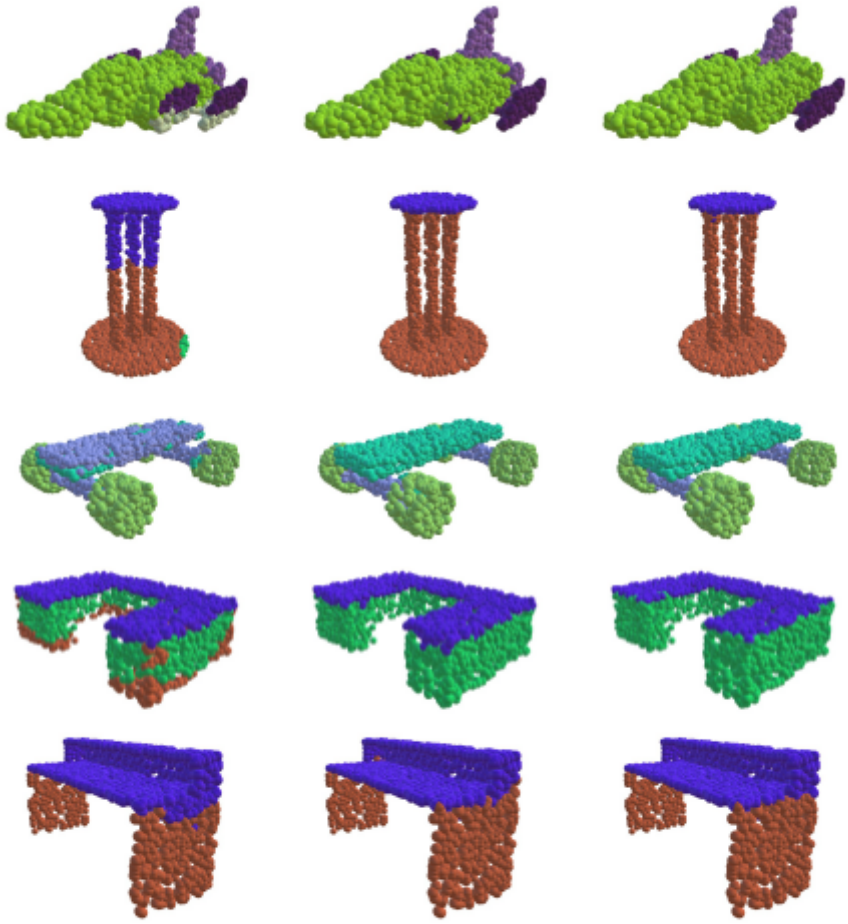
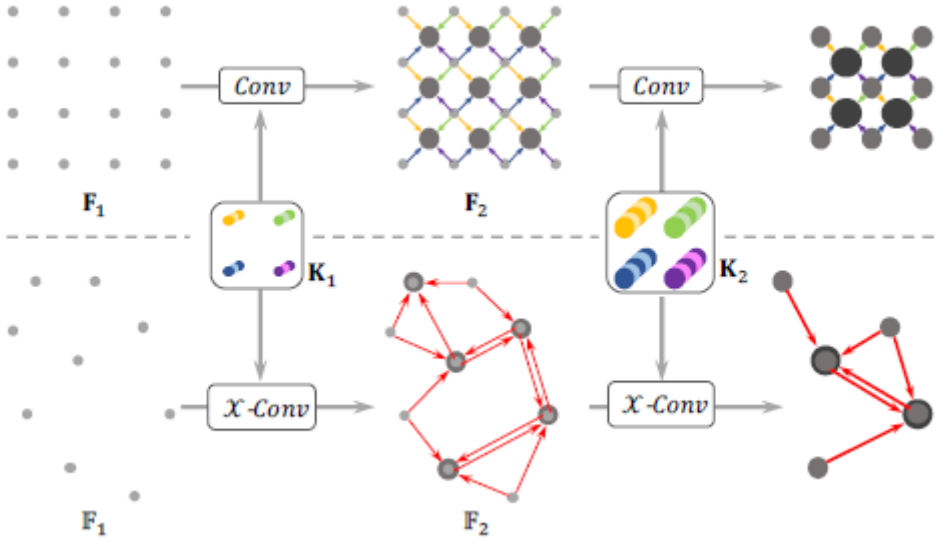


Fig. 7. Compare part segmentation results. For each set, from left to right: PointNet, ours and ground truth.

PointCNN

提出X-Conv，基于X变换的卷积操作，用于处理点云数据。证明了X-变换可以起到空间转换网络的作用，同时可以同时起到分配权重和排列的作用。

将图像卷积推广到图结构卷积，



X-Conv考虑了结点的特征和相对位置，

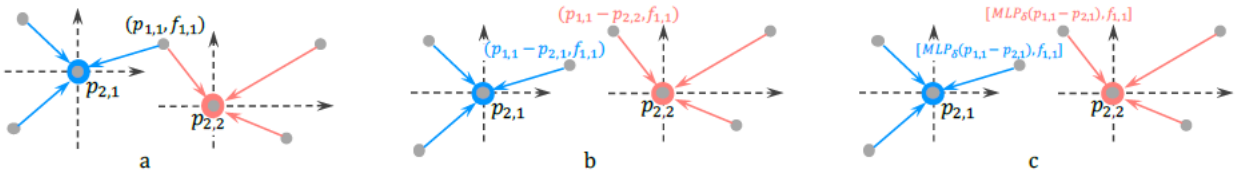
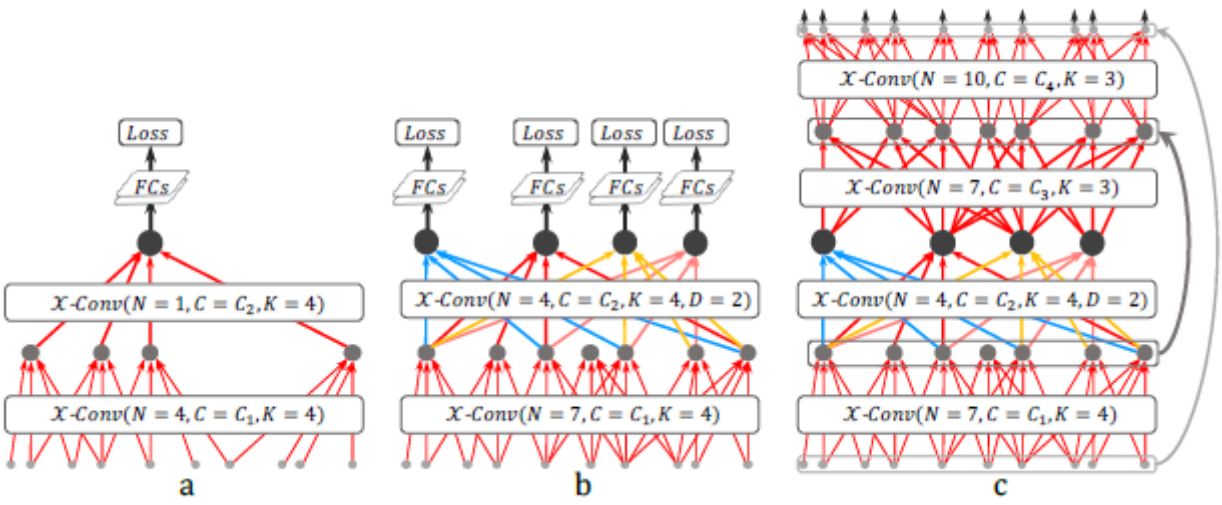


Figure 3: The process for converting point coordinates to features. Neighboring points are transformed to the local coordinate systems of the representative points (a and b). The local coordinates of each point are then individually lifted and combined with the associated features (c).

原文公式，K为卷积核、MLP为多层感知机、P为点位置、p为卷积的结点、F为结点特征。

$$\mathbf{F}_p = \mathcal{X}\text{-}Conv(\mathbf{K}, p, \mathbf{P}, \mathbf{F}) = Conv(\mathbf{K}, MLP(\mathbf{P}-p) \times [MLP_\delta(\mathbf{P}-p), \mathbf{F}]), \qquad (2)$$

网络整体框架图，



文章针对ModelNet40做了实验，证明方法优于PointNet等方法。

PyG代码中类定义为，

CLASS XConv (in_channels: int, out_channels: int, dim: int, kernel_size: int, hidden_channels: Optional[int] = None, dilation: int = 1, bias: bool = True, num_workers: int = 1) [source]

The convolutional operator on $\mathcal{X}$-transformed points from the "PointCNN: Convolution On X-Transformed Points" paper

$$\mathbf{x}_i' = \mathrm{Conv}\left(\mathbf{K}, \gamma_{\mathbf{\Theta}}(\mathbf{P}_i - \mathbf{p}_i) \times \left(h_{\mathbf{\Theta}}(\mathbf{P}_i - \mathbf{p}_i) \,\|\, \mathbf{x}_i\right)\right),$$

where $\mathbf{K}$ and $\mathbf{P}_i$ denote the trainable filter and neighboring point positions of $\mathbf{x}_i$, respectively. $\gamma_{\mathbf{\Theta}}$ and $h_{\mathbf{\Theta}}$ describe neural networks, *i.e.* MLPs, where $h_{\mathbf{\Theta}}$ individually lifts each point into a higher-dimensional space, and $\gamma_{\mathbf{\Theta}}$ computes the $\mathcal{X}$- transformation matrix based on *all* points in a neighborhood.

Related

GNN等相关的内容，引用的文献等，基于GNN的其他应用、变种（感觉不是很有意思）等，标签传播等其他非GNN的方法等。

ChebConv

GCN文章中提到的ChebConv，用切比雪夫多项式来近似图信号处理中的卷积操作。

文章由CNN和图信号处理的理论启发，尝试用于解决图结点表示等问题。

主要有以下贡献

- 基于图信号处理建立了将CNN用于图结构的理论基础
- 提出一种快速的卷积方式
- 提出一种高效的池化方式

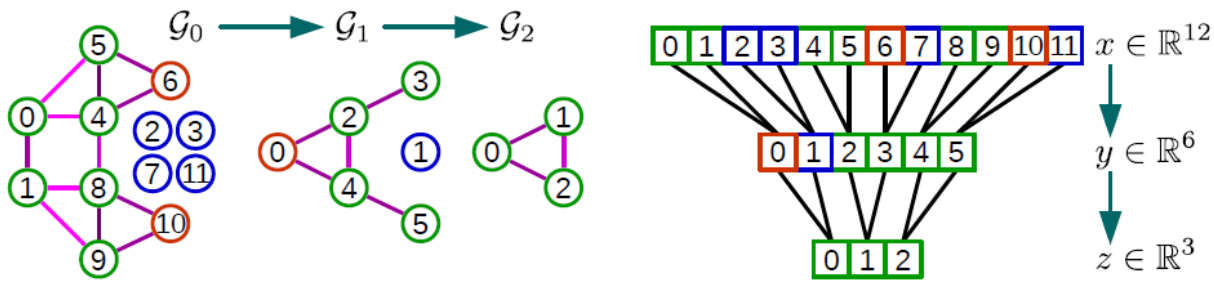
通过递归地计算切比雪夫多项式，可以达到比直接计算拉普拉斯矩阵的谱分解更快的复杂度。

Recall that the Chebyshev polynomial $T_k(x)$ of order k may be computed by the stable recurrence relation $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_0 = 1$ and $T_1 = x$. These polynomials form an orthogonal basis for $L^2([-1,1], dy/\sqrt{1-y^2})$, the Hilbert space of square integrable functions with respect to the measure $dy/\sqrt{1-y^2}$. A filter can thus be parametrized as the truncated expansion

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda}), \tag{4}$$

of order $K-1$, where the parameter $\theta \in \mathbb{R}^K$ is a vector of Chebyshev coefficients and $T_k(\tilde{\Lambda}) \in \mathbb{R}^{n\times n}$ is the Chebyshev polynomial of order k evaluated at $\tilde{\Lambda} = 2\Lambda/\lambda_{max} - I_n$, a diagonal matrix of scaled eigenvalues that lie in $[-1,1]$. The filtering operation can then be written as $y = g_\theta(L)x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x$, where $T_k(\tilde{L}) \in \mathbb{R}^{n\times n}$ is the Chebyshev polynomial of order k evaluated at the scaled Laplacian $\tilde{L} = 2L/\lambda_{max} - I_n$. Denoting $\bar{x}_k = T_k(\tilde{L})x \in \mathbb{R}^n$, we can use the recurrence relation to compute $\bar{x}_k = 2\tilde{L}\bar{x}_{k-1} - \bar{x}_{k-2}$ with $\bar{x}_0 = x$ and $\bar{x}_1 = \tilde{L}x$. The entire filtering operation $y = g_\theta(L)x = [\bar{x}_0, \ldots, \bar{x}_{K-1}]\theta$ then costs $\mathcal{O}(K|\mathcal{E}|)$ operations.

用池化操作获得图的粗粒度特征，下图形象展示了池化的过程。实现了将12维的向量，减少为3维向量的功能。



文章提出使用二叉平衡树高效实现该池化操作。

文章在1d和2d数据集上进行了实验。

在MNIST手写数据集上与传统的CNN如Lenet5进行了对比试验，

首先将2d的图片同股票k近邻图的方式转换为图结构表示，其中权重用高斯权重定义。

$$W_{ij} = \exp\left(-\frac{\|z_i - z_j\|_2^2}{\sigma^2}\right), \tag{8}$$

代码如下

CLASS **ChebConv** (in_channels, out_channels, K, normalization='sym', bias=True, **kwargs) [source]

The chebyshev spectral graph convolutional operator from the "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering" paper

$$\mathbf{X}' = \sum_{k=1}^{K} \mathbf{Z}^{(k)} \cdot \mathbf{\Theta}^{(k)}$$

where $\mathbf{Z}^{(k)}$ is computed recursively by

$$\begin{aligned}\mathbf{Z}^{(1)} &= \mathbf{X} \\ \mathbf{Z}^{(2)} &= \hat{\mathbf{L}} \cdot \mathbf{X} \\ \mathbf{Z}^{(k)} &= 2 \cdot \hat{\mathbf{L}} \cdot \mathbf{Z}^{(k-1)} - \mathbf{Z}^{(k-2)}\end{aligned}$$

and $\hat{\mathbf{L}}$ denotes the scaled and normalized Laplacian $\frac{2\mathbf{L}}{\lambda_{\max}} - \mathbf{I}$.

k-GNN

文章在理论上分析了GNN和1阶WL测试之间的联系，并将之前的方法通称为1-GNN。文章证明了1-GNN方法在判断图同构问题上并没有1-WL测试强大，但也证明了在某种合适的参数初始化情况下，1-GNN可以达到和1-WL同样的效果。但1-GNN以及1-WL都存在一定的弊端，对于某些图并不能很好的判别。而k-WL具有很好的判别效果，基于k-WL，文章提出k-GNN，并且在显存有限等条件下提出了简化版的模型。

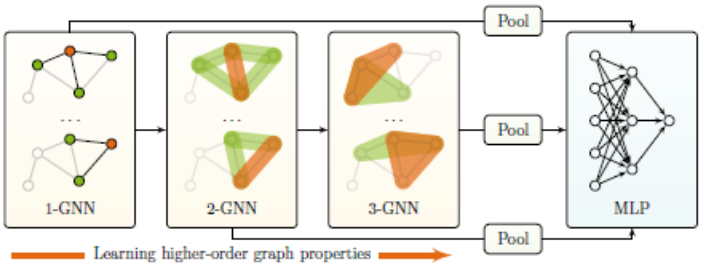
1-WL和1-GNN仅将每个结点当成独立单元，但k-WL和k-GNN将k个结点的组合当成一个独立的单元。使用k-GNN，可以更好地获得图的层级结构。将每k个结点的集合当成一个整体，并再次基础上可以定义其邻居结构，在其邻居结构上类似GNN做聚合操作。

k-dimensional Graph Neural Networks

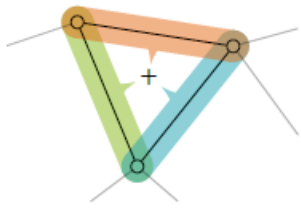
In the following, we propose a generalization of 1-GNNs, so-called k-GNNs, which are based on the k-WL. Due to scalability and limited GPU memory, we consider a set-based version of the k-WL. For a given k, we consider all k-element subsets $[V(G)]^k$ over $V(G)$. Let $s = \{s_1, \ldots, s_k\}$ be a k-set in $[V(G)]^k$, then we define the *neighborhood* of s as

$$N(s) = \{t \in [V(G)]^k \mid |s \cap t| = k - 1\}.$$

k-GNN的图示如下所示，左图展示了k-GNN的邻居情乱搞，红色单元表示结点，绿色单元表示红色单元的邻居。右图展示了如何利用池化操作从2-GNN转化为3-GNN的示意。



(a) Hierarchical 1-2-3-GNN network architecture



(b) Pooling from 2- to 3-GNN.

精确的k-GNN公式不仅需要运用到局部邻居N_L，还需要利用到全局邻居N_G。

k-GNN的局部邻居是k-GNN中的直接邻居，而其全局邻居定义属于1-GNN的邻居但不属于k-GNN直接邻居的结点集合。

局部邻居和全局邻居是独立的部分，可以分开计算。

$$f_k^{(t)}(s) = \sigma\left(f_k^{(t-1)}(s) \cdot W_1^{(t)} + \sum_{u \in N_L(s) \cup N_G(s)} f_k^{(t-1)}(u) \cdot W_2^{(t)}\right).$$

Moreover, one could split the sum into two sums ranging over $N_L(s)$ and $N_G(s)$ respectively, using distinct parameter matrices to enable the model to learn the importance of local and global neighborhoods. To scale k-GNNs to larger datasets and to prevent overfitting, we propose *local* k-GNNs, where we omit the global neighborhood of s, i.e.,

$$f_{k,\mathrm{L}}^{(t)}(s) = \sigma\left(f_{k,\mathrm{L}}^{(t-1)}(s) \cdot W_1^{(t)} + \sum_{u \in N_L(s)} f_{k,\mathrm{L}}^{(t-1)}(u) \cdot W_2^{(t)}\right).$$

但在显存有限的情况下，对公式进行简化，仅考虑局部邻居。

该方法也可以称为图卷积GraphConv，公式和传统1-GNN基本一致，但推广到了k-GNN。

CLASS `GraphConv` (**in_channels**: Union[int, Tuple[int, int]], **out_channels**: int, **aggr**: str = `'add'`, **bias**: bool = `True`, ****kwargs**) [source]

The graph neural network operator from the "Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks" paper

$$\mathbf{x}'_i = \mathbf{\Theta}_1 \mathbf{x}_i + \mathbf{\Theta}_2 \sum_{j \in \mathcal{N}(i)} e_{j,i} \cdot \mathbf{x}_j$$

where $e_{j,i}$ denotes the edge weight from source node `j` to target node `i` (default: `1`)

PARAMETERS:

- **in_channels** (*int or tuple*) – Size of each input sample. A tuple corresponds to the sizes of source and target dimensionalities.
- **out_channels** (*int*) – Size of each output sample.
- **aggr** (*string, optional*) – The aggregation scheme to use (`"add"` , `"mean"` , `"max"`). (default: `"add"`)
- **bias** (*bool, optional*) – If set to `False` , the layer will not learn an additive bias. (default: `True`)
- ****kwargs** (*optional*) – Additional arguments of `torch_geometric.nn.conv.MessagePassing` .

UniMP

模型将基于GNN和基于消息传递机制(LAP）的图表示学习方法融合在一个统一的结构下。并启发于transformer，类似tranformer的自注意力机制定义了类似于transformer的GraphTransformer操作，在消息传播的过程中同时进行标签传播和特征传播。并且采用了transformer中的掩码机制辅助训练。

模型结构图如下，左边为模型输入，X表示结点特征，Y表示结点标签，A为邻接矩阵。中间为和transformer类似的GraphTransformer。右边为在随机掩码的基础上定义的预测的任务。

主要分三部分

- 统一LAP和GNN
- GraphTransformer
- 基于随机掩码的预测任务

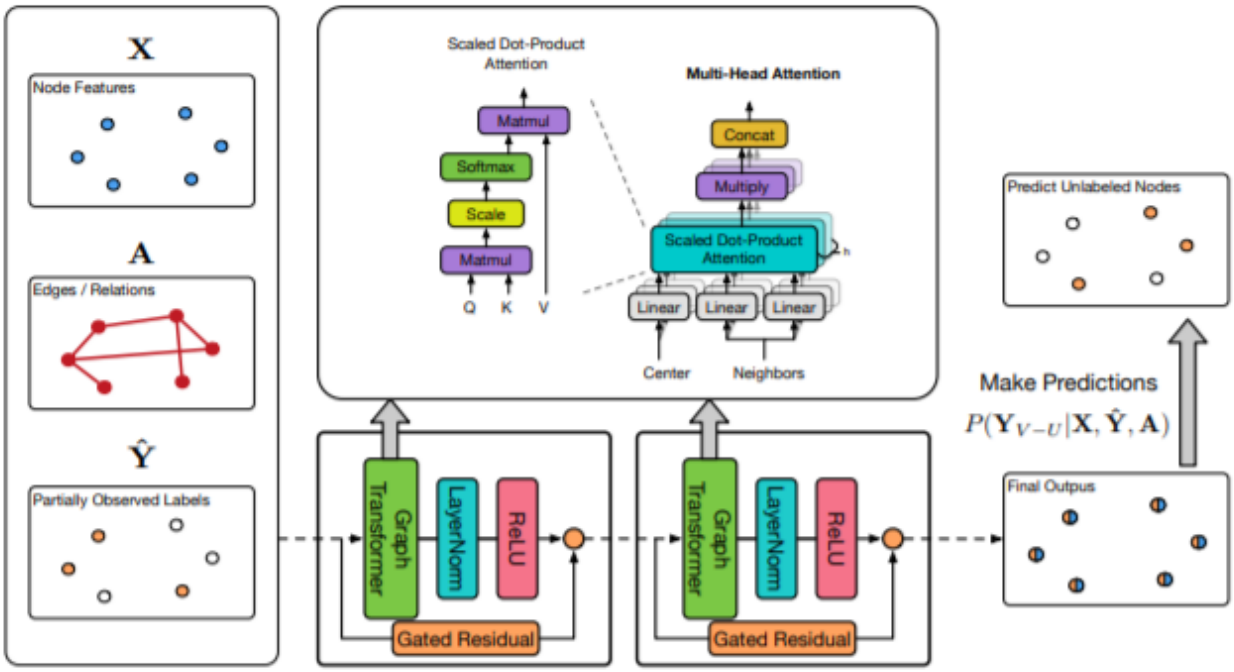


Figure 1: The architecture of our UniMP.

模型统一了GNN和LAP

In semi-supervised node classification, based on the Laplacian smoothing assumption, the GNN transforms and propagates nodes features X across the graph by several layers, including linear layers and nonlinear activation to build the approximation of the mapping: $X \rightarrow Y$. The feature propagation scheme of GNN in layer l is:

$$\begin{aligned} H^{(l+1)} &= \sigma(D^{-1}AH^{(l)}W^{(l)}) \\ Y &= f_{out}(H^{(L)}) \end{aligned} \tag{1}$$

where the σ is an activation function, $W^{(l)}$ is the trainable weight in the l-th layer, and the $H^{(l)}$ is the l-th layer representations of nodes. $H^{(0)}$ is equal to node input features X. Finally, a f_{out} output layer is applied on the final representation to make prediction for Y.

As for LPA, it also assumes the labels between connected nodes are smoothing and propagates the labels iteratively across the graph. Given an initial label matrix $\hat{Y}^{(0)}$, which consists of one-hot label indicator vectors $\hat{y}_i^0$ for the labeled nodes or zeros vectors for the unlabeled. A simple iteration equation of LPA is formulated as following:

$$\hat{Y}^{(l+1)} = D^{-1}A\hat{Y}^{(l)} \tag{2}$$

Labels are propagated from each other nodes through a normalized adjacency matrix $D^{-1}A$.

公式与transformer相同，

$$
\begin{aligned}
q_{c,i}^{(l)} &= W_{c,q}^{(l)} h_i^{(l)} + b_{c,q}^{(l)} \\
k_{c,j}^{(l)} &= W_{c,k}^{(l)} h_j^{(l)} + b_{c,k}^{(l)} \\
e_{c,ij} &= W_{c,e} e_{ij} + b_{c,e} \\
\alpha_{c,ij}^{(l)} &= \frac{\langle q_{c,i}^{(l)}, k_{c,j}^{(l)} + e_{c,ij} \rangle}{\sum_{u \in \mathcal{N}(i)} \langle q_{c,i}^{(l)}, k_{c,u}^{(l)} + e_{c,iu} \rangle}
\end{aligned} \tag{3}
$$

同样可以采用多头注意力机制，

After getting the graph multi-head attention, we make a message aggregation from the distant j to the source i:

$$
\begin{aligned}
v_{c,j}^{(l)} &= W_{c,v}^{(l)} h_j^{(l)} + b_{c,v}^{(l)} \\
\hat{h}_i^{(l)} &= \Big\|_{c=1}^{C} \Big[\sum_{j \in \mathcal{N}(i)} \alpha_{c,ij}^{(l)} (v_{c,j}^{(l)} + e_{c,ij}) \Big] \\
r_i^{(l)} &= W_r^{(l)} h_i^{(l)} + b_r^{(l)} \\
\beta_i^{(l)} &= \text{sigmoid}(W_g^{(l)} [\hat{h}_i^{(l)}; r_i^{(l)}; \hat{h}_i^{(l)} - r_i^{(l)}]) \\
h_i^{(l+1)} &= \text{ReLU}(\text{LayerNorm}((1 - \beta_i^{(l)}) \hat{h}_i^{(l)} + \beta_i^{(l)} r_i^{(l)}))
\end{aligned} \tag{4}
$$

where the $\|$ is the concatenation operation for C head attention. Comparing with the Equation 1, multi-head attention matrix replaces the original normalized adjacency matrix as transition matrix for message passing. The distant feature h_j is transformed to $v_{c,j} \in \mathbb{R}^d$ for weighted sum. In addition, inspired by (Li et al., 2019; Chen et al., 2020) to prevent oversmoothing, we propose a gated residual connections between layers by $r_i \in \mathbb{R}^d$ and $\beta_i^{(l)} \in \mathbb{R}^1$.

Specially, similar to GAT, if we apply the Graph Transformer on the output layer, we will employ averaging for multi-head output as following:

$$
\begin{aligned}
\hat{h}_i^{(l)} &= \frac{1}{C} \sum_{c=1}^{C} \Big[\sum_{j \in \mathcal{N}(i)} \alpha_{c,ij}^{(l)} (v_{c,j}^{(l)} + e_{c,ij}^{(l)}) \Big] \\
h_i^{(l+1)} &= (1 - \beta_i^{(l)}) \hat{h}_i^{(l)} + \beta_i^{(l)} r_i^{(l)}
\end{aligned} \tag{5}
$$

与BERT相似的预测任务，

Previous works on GNNs seldom consider using the partially observed labels $\hat{Y}$ in both training and inference stages. They only take those labels information as ground truth target to supervised train their model's parameters θ with given X and A:

$$\arg\max_{\theta} \quad \log p_\theta(\hat{Y}|X,A) = \sum_{i=1}^{\hat{V}} \log p_\theta(\hat{y}_i|X,A) \tag{8}$$

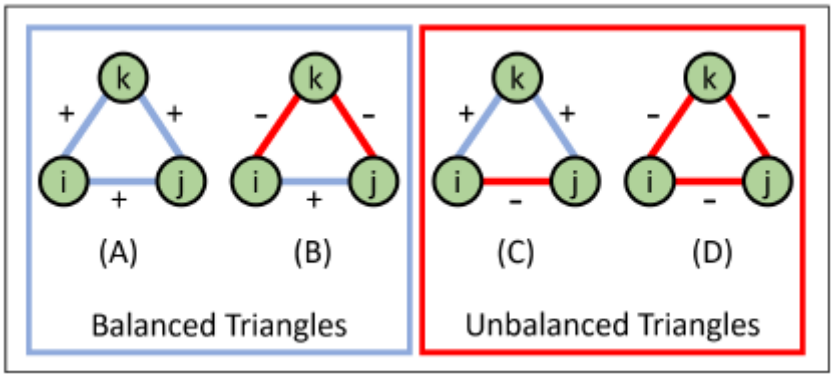
where $\hat{V}$ represents the partial nodes with labels. However, our UniMP model propagates nodes features and labels to make prediction: $p(y|X,\hat{Y},A)$. Simply using above objective for our model will make the label leakage in the training stage, causing poor performance in inference. Learning from BERT, which masks input words and makes prediction for them to pretrain their model (masked word prediction), we propose a masked label prediction strategy to train our model. During training, at each step, we corrupt the $\hat{Y}$ into $\tilde{Y}$ by randomly masking a portion of node labels to zeros and keep the others remain, which is controlled by a hyper-parameter called label_rate. Let those masked labels be $\bar{Y}$, our objective function is to predict $\bar{Y}$ with given X, $\tilde{Y}$ and A:

$$\arg\max_{\theta} \quad \log p_\theta(\bar{Y}|X,\tilde{Y},A) = \sum_{i=1}^{\bar{V}} \log p_\theta(\bar{y}_i|X,\tilde{Y},A) \tag{9}$$

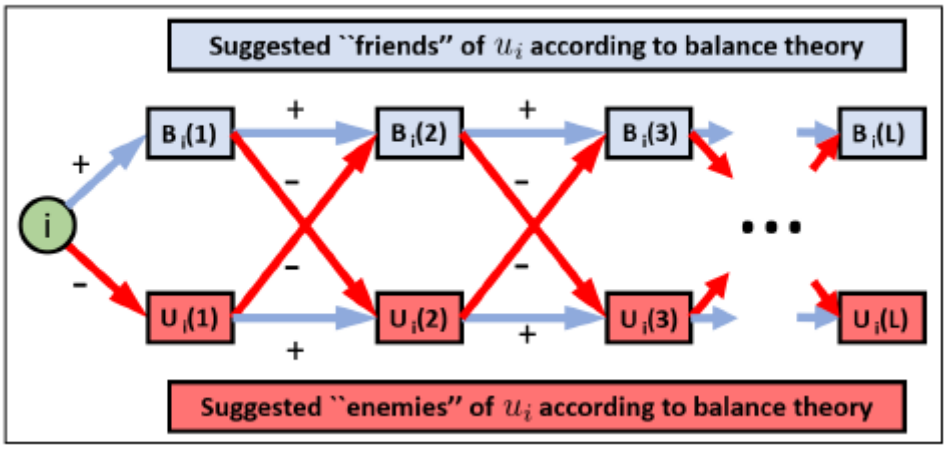
SignedGCN

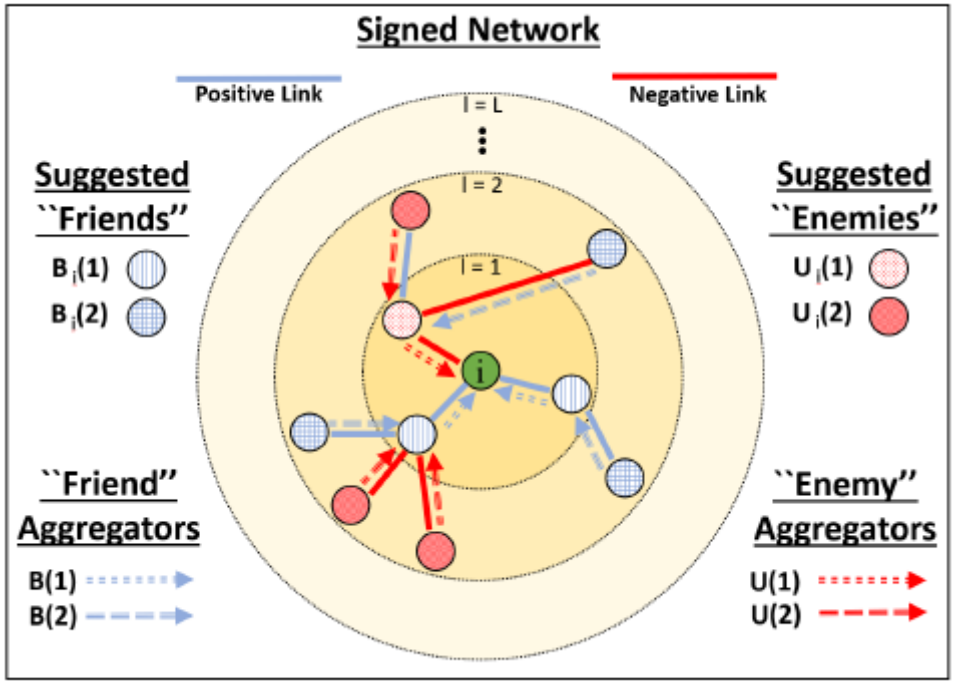
传统GCN并不适用于带符号的图，比如图中边有正负边的区分的情况（如敌我关系图）。传统GCN中仅区分存在边、不存在边，而带符号图中点对有三种关系，正边、负边、无边。SigendGCN定义了适用于带符号图的GCN，并且基于三元闭包传递理论定义了消息传递机制。

三元闭包理论如下，



由三元闭包理论可以定义'建议'朋友和'建议'敌人的概念，即基于朋友的朋友、敌人的朋友等关系，如下图，





由此可以定义SignedGCN的消息传递公式，

Algorithm 2: Signed GCN Embedding Generation.

Input: $\mathcal{G} = (\mathcal{U}, \mathcal{E}^+, \mathcal{E}^-)$; an initial seed node representation $\{\mathbf{x}_i, \forall u_i \in \mathcal{U}\}$; number of aggregation layers L; weight matrices $\mathbf{W}^{B(l)}$ and $\mathbf{W}^{U(l)}, \forall l \in \{1, \ldots, L\}$; non-linear function σ

Output: Low-dimensional representations $\mathbf{z}_i, \forall u_i \in \mathcal{U}$

$\mathbf{h}_i^{(0)} \leftarrow \mathbf{x}_i, \forall u_i \in \mathcal{U}$

for $u_i \in \mathcal{U}$ **do**

$\quad \mathbf{h}_i^{B(1)} \leftarrow \sigma\left(\mathbf{W}^{B(1)}\left[\sum_{j \in \mathcal{N}_i^+} \frac{\mathbf{h}_j^{(0)}}{|\mathcal{N}_i^+|}, \mathbf{h}_i^{(0)}\right]\right)$

$\quad \mathbf{h}_i^{U(1)} \leftarrow \sigma\left(\mathbf{W}^{U(1)}\left[\sum_{k \in \mathcal{N}_i^-} \frac{\mathbf{h}_k^{(0)}}{|\mathcal{N}_i^-|}, \mathbf{h}_i^{(0)}\right]\right)$

end

if $L > 1$ **then**

$\quad$ **for** $l = 2 \ldots L$ **do**

$\quad\quad$ **for** $u_i \in \mathcal{U}$ **do**

$$\mathbf{h}_i^{B(l)} = \sigma\left(\mathbf{W}^{B(l)}\left[\sum_{j \in \mathcal{N}_i^+} \frac{\mathbf{h}_j^{B(l-1)}}{|\mathcal{N}_i^+|}, \sum_{k \in \mathcal{N}_i^-} \frac{\mathbf{h}_k^{U(l-1)}}{|\mathcal{N}_i^-|}, \mathbf{h}_i^{B(l-1)}\right]\right)$$

$$\mathbf{h}_i^{U(l)} = \sigma\left(\mathbf{W}^{U(l)}\left[\sum_{j \in \mathcal{N}_i^+} \frac{\mathbf{h}_j^{U(l-1)}}{|\mathcal{N}_i^+|}, \sum_{k \in \mathcal{N}_i^-} \frac{\mathbf{h}_k^{B(l-1)}}{|\mathcal{N}_i^-|}, \mathbf{h}_i^{U(l-1)}\right]\right)$$

$\quad\quad$ **end**

$\quad$ **end**

end

$\mathbf{z}_i \leftarrow [\mathbf{h}_i^{B(L)}, \mathbf{h}_i^{U(L)}], \forall u_i \in \mathcal{U}$

LAP

基于带标签的无向图的标签传播算法，基于简单的迭代策略，但在给定条件下可以证明收敛性，且文章证明该算法与其他算法的联系。

定义如下的概率转移矩阵T，

$$T_{ij} = P(j \to i) = \frac{w_{ij}}{\sum_{k=1}^{l+u} w_{kj}} \tag{2}$$

算法如下执行，不断迭代直至收敛，

The label propagation algorithm is as follows:

1. Propagate $Y \leftarrow TY$
2. Row-normalize Y.
3. Clamp the labeled data. Repeat from step 1 until Y converges.

Correct&Smooth

提出一种简单的基于迭代法的图表示学习方法。基于GNN的方法虽然强大，但可解释性较低，同时很难在大数据集上加速。

Correct and Smooth的方法首先，首先采用迭代法纠正模型的预测误差，然后使用平滑化的方法使得相邻结点具有相似的特征。

对比方法是MLP和GCN，在某些结点分类的转导学习数据集上取得了更好的表现。

Correct步，

2.2 CORRECTING FOR ERROR IN BASE PREDICTIONS WITH RESIDUAL PROPAGATION

Next, we improve the accuracy of the base prediction Z by incorporating labels to correlate errors. The key idea is that we expect *errors* in the base prediction to be positively correlated along edges in the graph. In other words, an error at node i increases the chance of a similar error at neighboring nodes of i. We should "spread" such uncertainty over the graph. Our approach here is inspired in part by residual propagation (Jia & Benson, 2020), where a similar concept is used for node regression tasks, as well as generalized least squares and correlated error models more broadly (Shalizi, 2013).

To this end, we first define an error matrix $E \in \mathbb{R}^{n \times c}$, where error is the *residual* on the training data and zero elsewhere:

$$E_{L_t} = Z_{L_t} - Y_{L_t}, \quad E_{L_v} = 0, \quad E_U = 0. \tag{1}$$

The residuals in rows of E corresponding to training nodes are zero only when the base predictor makes a perfect predictions. We smooth the error using the label spreading technique of Zhou et al. (2004), optimizing the objective

$$\hat{E} = \underset{W \in \mathbb{R}^{n \times c}}{\arg\min}\, \mathrm{trace}(W^T(I - S)W) + \mu \|W - E\|_F^2. \tag{2}$$

The first term encourages smoothness of the error estimation over the graph, and is equal to $\sum_{j=1}^{c} w_j^T(I - S)w_j$, where w_j is the jth column of W. The second term keeps the solution close to the initial guess E of the error. As in Zhou et al. (2004), the solution can be obtained via the iteration $E^{(t+1)} = (1 - \alpha)E + \alpha S E^{(t)}$, where $\alpha = 1/(1 + \mu)$ and $E^{(0)} = E$, which converges rapidly to $\hat{E}$. This iteration is a diffusion, propagation, or spreading of the error, and we add the smoothed errors to the base prediction to get corrected predictions $Z^{(r)} = Z + \hat{E}$. We emphasize that this is a post-processing technique and there is no coupled training with the base predictions.

This type of propagation is provably the right approach under a Gaussian assumption in regression problems (Jia & Benson, 2020); however, for the classification problems we consider, the smoothed errors $\hat{E}$ might not be at the right scale. We know that in general,

$$\|E^{(t+1)}\|_2 \leq (1 - \alpha)\|E\| + \alpha \|S\|_2 \|E^{(t)}\|_2 = (1 - \alpha)\|E\|_2 + \alpha \|E^{(t)}\|_2. \tag{3}$$

When $E^{(0)} = E$, we then have that $\|E^{(t)}\|_2 \leq \|E\|_2$. Thus, the propagation cannot completely correct the errors on all nodes in the graph, as it does not have enough "total mass," and we find that adjusting the scale of the residual can help substantially in practice. To do this, we propose two variations of scaling the residual.

Smooth步，

2.3 SMOOTHING FINAL PREDICTIONS WITH PREDICTION CORRELATION

At this point, we have a score vector $Z^{(r)}$, obtained from correcting the base predictor Z with a model for the correlated error $\hat{E}$. To make a final prediction, we further smooth the corrected predictions. The motivation is that adjacent nodes in the graph are likely to have similar labels, which is expected given homophily or assortative properties of a network. Thus, we can encourage smoothness over the distribution over labels by another label propagation. First, we start with our best guess $G \in \mathbb{R}^{n \times c}$ of the labels:

$$G_{L_t} = Y_{L_t}, \quad G_{L_v,U} = Z^{(r)}_{L_v,U}. \tag{4}$$

Here, we set the training nodes back to their true labels and use the corrected predictions for the validation and unlabeled nodes (we can also use the true validation labels, which we discuss later in the experiments). We then iterate $G^{(t+1)} = (1-\alpha)G + \alpha S G^{(t)}$ with $G^{(0)} = G$ until convergence to give the final prediction $\hat{Y}$. The classification for a node $i \in U$ is $\arg\max_{j \in \{1,\dots,c\}} \hat{Y}_{ij}$.

As with error correlation, the smoothing here is a post-processing step, decoupled from the other steps. This type of prediction smoothing is similar in spirit to APPNP (Klicpera et al., 2018), which we compare against later. However, APPNP is trained end-to-end, propagates on final-layer representations instead of softmaxes, does not use labels, and is motivated differently.

Node2Vec

基于随机游走的概率方法获得节点的向量嵌入表示，传统的基于矩阵分解的方法具有很高的复杂度，对于现实中巨大的网络很难适用。

与skip-gram类似，node2vec基于随机游走获取上下文，最大化以下的条件似然概率，

$$Pr(n_i|f(u)) = \frac{\exp(f(n_i) \cdot f(u))}{\sum_{v \in V} \exp(f(v) \cdot f(u))}.$$

With the above assumptions, the objective in Eq. 1 simplifies to:

$$\max_f \sum_{u \in V} \left[-\log Z_u + \sum_{n_i \in N_S(u)} f(n_i) \cdot f(u) \right]. \tag{2}$$

对于每个节点的随游走可以分为BFS、DFS两种，

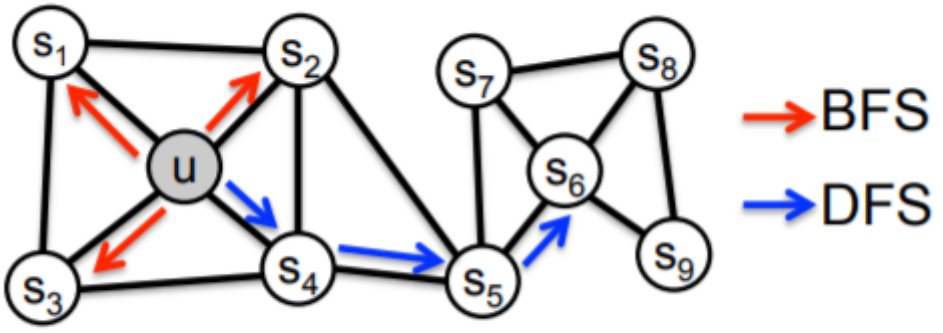
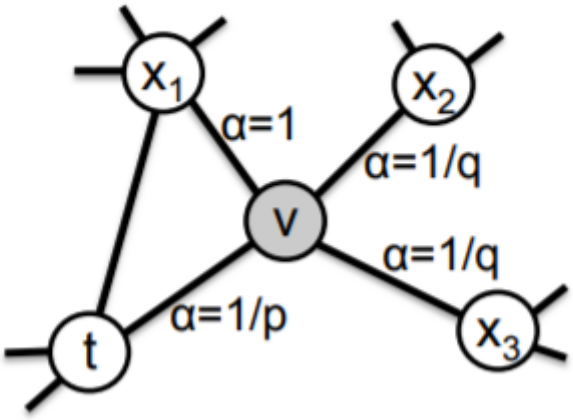


Figure 1: BFS and DFS search strategies from node u ($k = 3$).

Node2vec采用参数p、q控制BFS和DFS的比例，



文章对比了基于矩阵的谱聚类方法，和其他两种基于随机游走的方法DeepWalk和LINE，并指出其他两种方法都可以类似地表示为node2vec在特定p、q参数取值下的变种。

Meta2Vec

可以看作在异质图上的node2vec，定义了异质图（图结点有不同种类，如学术网络中结点有作者、会议、期刊等）上的随机游走，并提出metapath等概念，基于此提出算法metapath2vec以及其优化版本metapath2vec++。

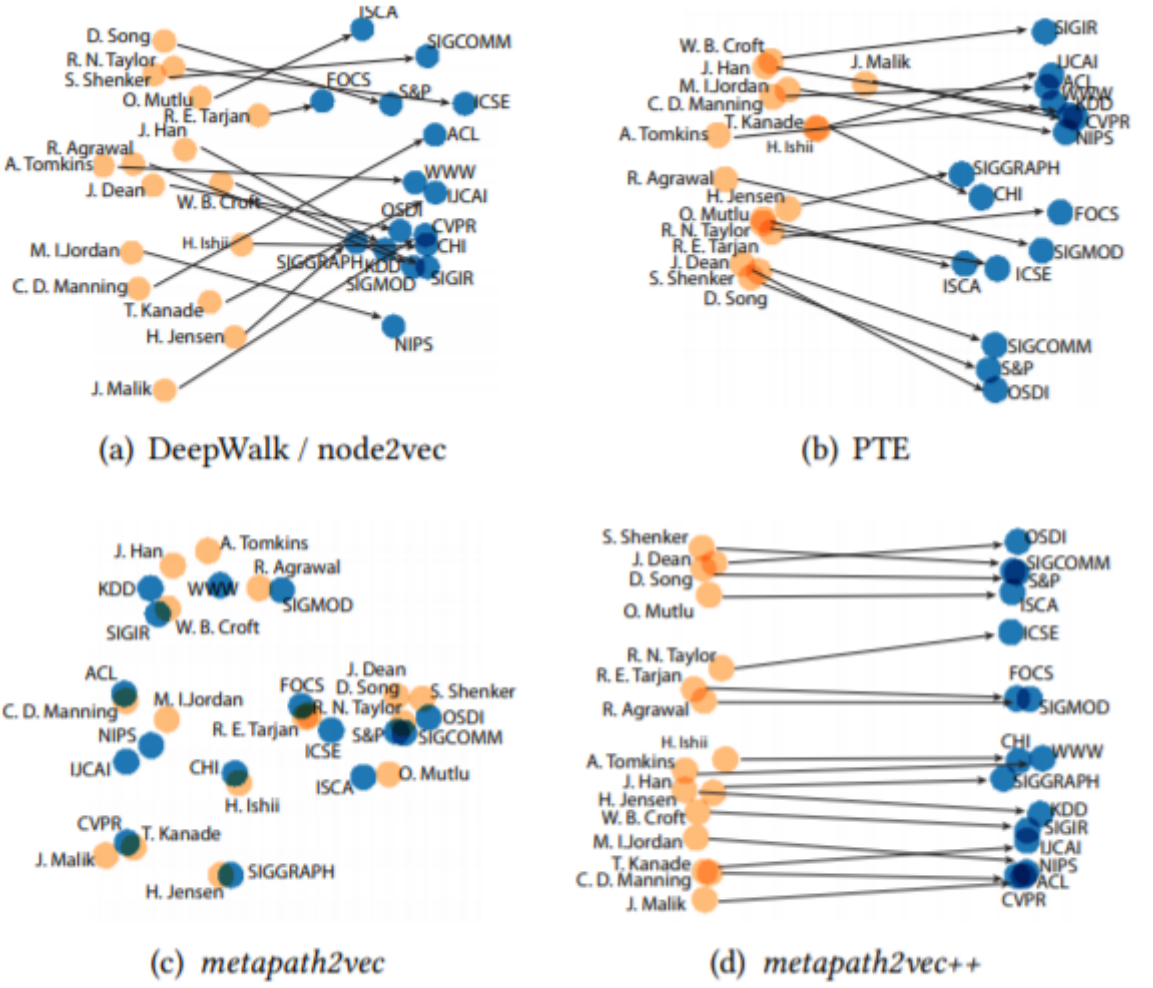


Figure 1: 2D PCA projections of the 128D embeddings of 16 top CS conferences and corresponding high-profile authors.

Seq2Seq

分析了如何将有序的LSTM运用在无序集合上面。在SAGE等被引用，但也有论文证明该方法不能很好地适应无序的结构。

WL Test

与GCN关联较大的算法，也被GIN等用作评价GCN的理论上界，对k-GNN等也具有启发价值。

用于判断两张图是否同构的一种算法。

本质为不断迭代地将邻居信息进行哈希操作，也可定义为消息传递类。

CLASS `WLConv` [source]

The Weisfeiler Lehman operator from the "A Reduction of a Graph to a Canonical Form and an Algebra Arising During this Reduction" paper, which iteratively refines node colorings:

$$\mathbf{x}_i' = \text{hash}\left(\mathbf{x}_i, \{\mathbf{x}_j : j \in \mathcal{N}(i)\}\right)$$

forward (**x**: torch.Tensor, **edge_index**: Union[torch.Tensor, torch_sparse.tensor.SparseTensor]) → torch.Tensor [source]

histogram (**x**: torch.Tensor, **batch**: Optional[torch.Tensor] = `None`, **norm**: bool = `False`) → torch.Tensor [source]

Given a node coloring `x`, computes the color histograms of the respective graphs (separated by `batch`).

reset_parameters ( ) [source]

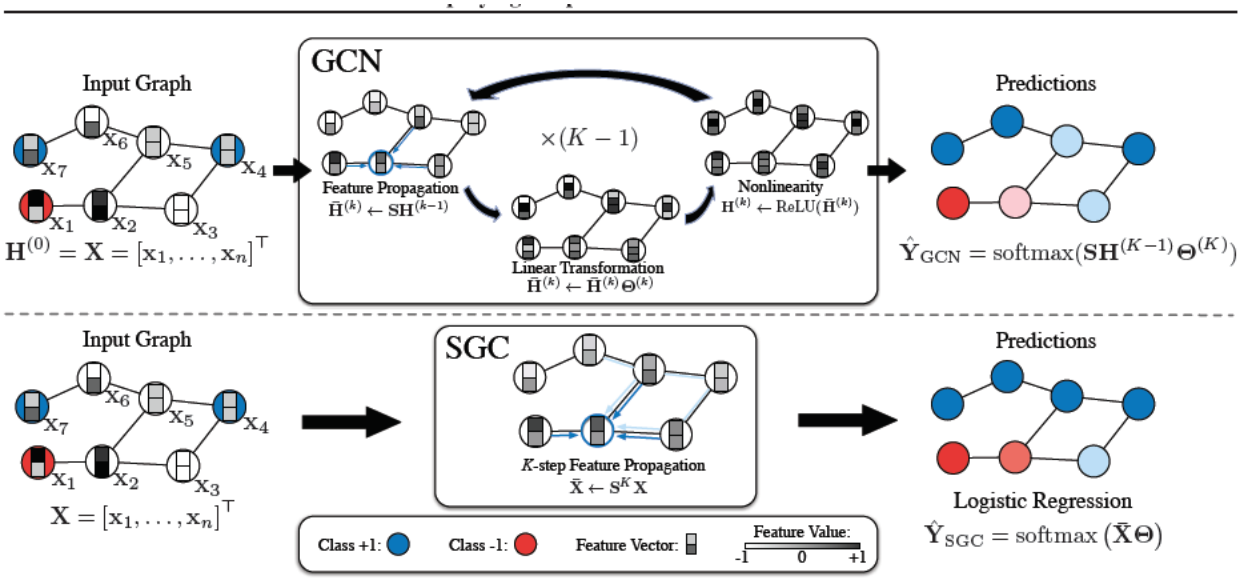
SimpleGCN

旨在简化GCN框架，证明GCN框架并不需要那么复杂，提供了直觉的解释和数学证明。

SimpleGCN移除了GCN中每一层之间的非线性激活函数，并且证明了该简化操作并不会很降低准确率，同时可以在大数据集上加速。

直观的解释是，用于视觉等的层与层之间的非线性激活函数，起到了从低维特征学习到高维特征的作用，但在GCN中层与层之间其实起到了联系的作用，所以非线性层的重要性并不那么明显。

GCN和SimpleGCN的对比如下：



SimpleGCN去掉了隐层与隐层之间的非线性激活函数，

$$\bar{\mathbf{h}}_i^{(k)} \leftarrow \frac{1}{d_i+1}\mathbf{h}_i^{(k-1)} + \sum_{j=1}^{n} \frac{a_{ij}}{\sqrt{(d_i+1)(d_j+1)}}\mathbf{h}_j^{(k-1)}. \tag{2}$$

More compactly, we can express this update over the entire graph as a simple matrix operation. Let $\mathbf{S}$ denote the "normalized" adjacency matrix with added self-loops,

$$\mathbf{S} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}, \tag{3}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\tilde{\mathbf{D}}$ is the degree matrix of $\tilde{\mathbf{A}}$. The simultaneous update in Equation 2 for all nodes becomes a simple sparse matrix multiplication

$$\bar{\mathbf{H}}^{(k)} \leftarrow \mathbf{S}\mathbf{H}^{(k-1)}. \tag{4}$$

最终公式如下，

$$\hat{\mathbf{Y}} = \text{softmax}\left(\mathbf{S}\ldots\mathbf{S}\mathbf{S}\mathbf{X}\mathbf{\Theta}^{(1)}\mathbf{\Theta}^{(2)}\ldots\mathbf{\Theta}^{(K)}\right). \tag{7}$$

To simplify notation we can collapse the repeated multiplication with the normalized adjacency matrix $\mathbf{S}$ into a single matrix by raising $\mathbf{S}$ to the K-th power, $\mathbf{S}^K$. Further, we can reparameterize our weights into a single matrix $\mathbf{\Theta} = \mathbf{\Theta}^{(1)}\mathbf{\Theta}^{(2)}\ldots\mathbf{\Theta}^{(K)}$. The resulting classifier becomes

$$\hat{\mathbf{Y}}_{\text{SGC}} = \text{softmax}\left(\mathbf{S}^K\mathbf{X}\mathbf{\Theta}\right), \tag{8}$$

which we refer to as Simple Graph Convolution (SGC).

文章同时给出了理论证明，

GCN来源于图信号的傅里叶变换，而SimpleGCN则起到了低通滤波的作用。

公式也很简单

CLASS **SGConv** (**in_channels**: int, **out_channels**: int, **K**: int = `1`, **cached**: bool = `False`, **add_self_loops**: bool = `True`, **bias**: bool = `True`, ****kwargs**) [source]

The simple graph convolutional operator from the "Simplifying Graph Convolutional Networks" paper

$$\mathbf{X}' = \left(\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\right)^K\mathbf{X}\mathbf{\Theta},$$

where $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ denotes the adjacency matrix with inserted self-loops and $\hat{D}_{ii} = \sum_{j=0} \hat{A}_{ij}$ its diagonal degree matrix. The adjacency matrix can include other values than `1` representing edge weights via the optional `edge_weight` tensor.