

# GNN: Review of Method and Applications

---

GNN: Review of Method and Applications

报告结构

GNN简介

GNN起源

GNN和传统NN的区别

GNN分类

GNN模型概览

图神经网络

各种GNN的变体

不同的图类型

传播类型

训练方法

GNN一般框架

Message Passing Neural Networks

Non-local Neural Networks

Graph Networks

GNN应用

开放问题

The Graph Neural Network Model

论文概要

图领域应用

GNN模型详述

学习算法

Transition和Output函数实现

实验结果

模型实现

Graph Neural Network

图卷积的演变

第一代GCN

第二代GCN

第三代GCN

论文模型

实验结果

DCNN

模型亮点

模型详述

实验结果

优缺点

Tree-LSTM

论文亮点

模型详解

Child-Sum Tree-LSTMs

N-ary Tree-LSTMs

模型训练

分类任务

语义相关性任务

实验结果

附录

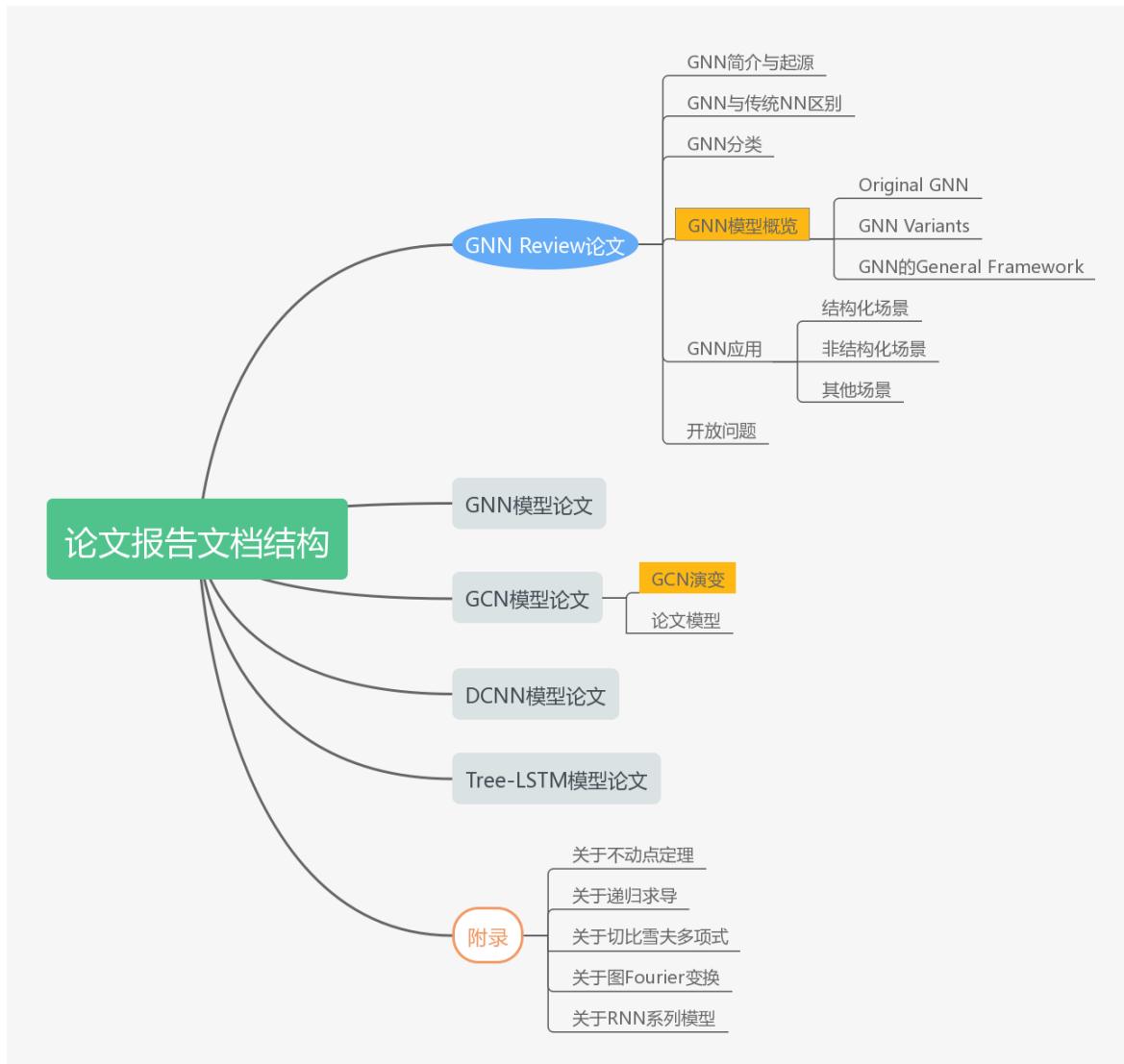
关于不动点定理

关于递归求导

关于切比雪夫多项式

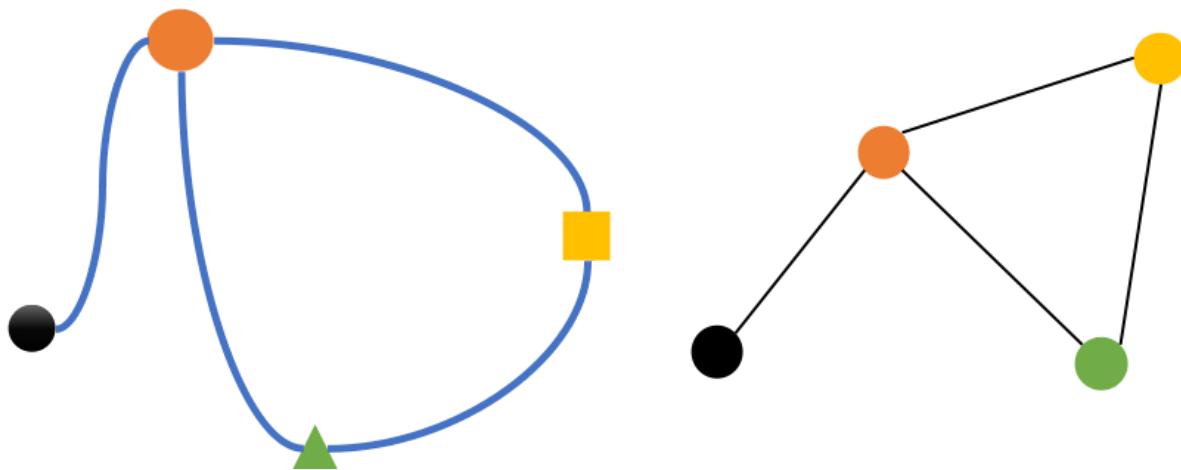
关于图Fourier变换

## 报告结构

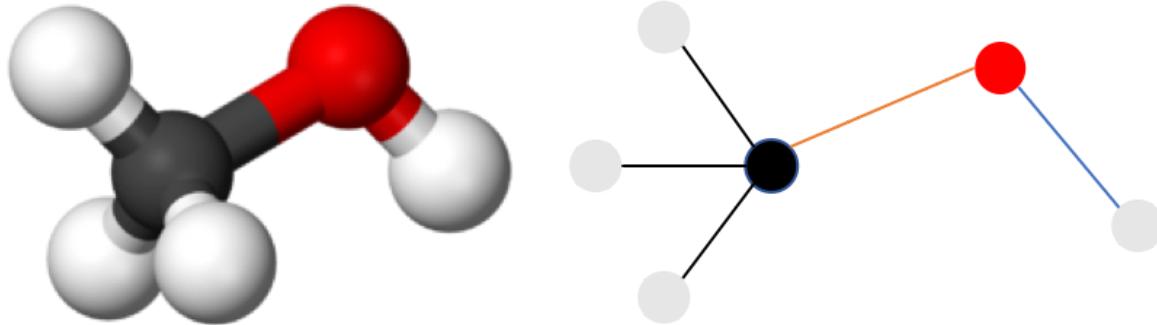


## GNN简介

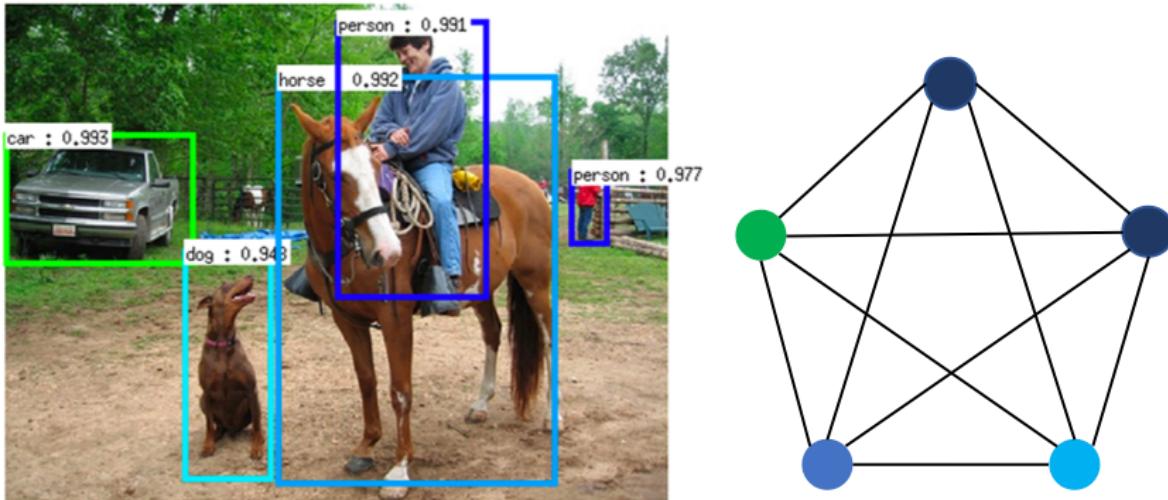
图(graph)是一种数据结构，图神经网络(GNN)是深度学习在图结构上的一个分支。常见的图结构包含节点(node)和边(edge)，其中，节点包含了实体(entity)信息，边包含实体间的关系(relation)信息。现在许多学习任务都需要**处理图结构的数据**，比如物理系统建模(physics system)、学习分子指纹(molecular fingerprints)、蛋白质接口预测(protein interface)以及疾病分类(classify diseases)，这些都需要模型能够从图结构的输入中学习相关的知识。相关的图结构例子说明如下：



上图为使用图结构来给物理系统建模，每一个节点表示物理系统中的objects，每一条边表示不同object之间的关系。



上图为分子结构，每一个节点表示原子，边表示原子之间的化学键。



上图为图像，每一个节点表示图像中的每一个物体，边表示物体之间的联系。

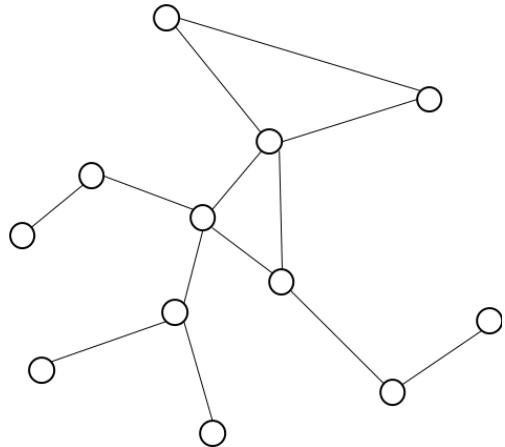
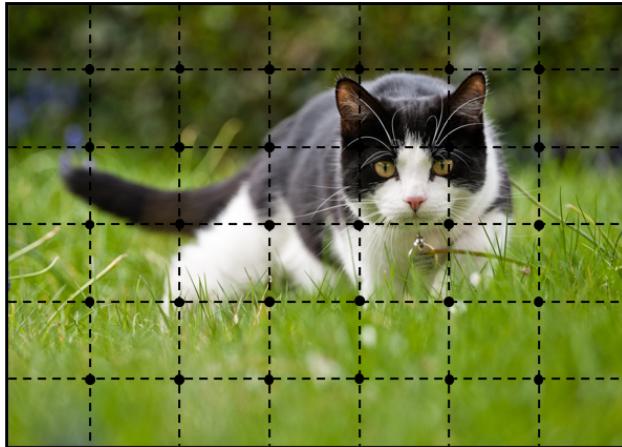
图神经网络除了能够学习结构化数据(输出数据为图结构数据)之外，还能学习到非结构化数据，比如文本(texts)和图片(images)，并能够在提取出的图结构中进行推理(reasoning)，比如句子的关系依赖树(dependency tree of sentences)和图片的情景图(scene graph of images)，这些都需要图推理模型。

**GNN是一种连接模型，通过网络中节点之间的信息传递(message passing)的方式来获取图中的依存关系(dependence of graph)**，GNN通过从节点任意深度的邻居来更新该节点状态，这个状态能够表示状态信息。

## GNN起源

GNN起源于两种动机，一种动机来自于卷积神经网络(CNN)，另一种动机来自于图嵌入(graph embedding)。

**第一种来源于CNN**, CNN能够提取出多尺度的局部空间特征，并将它们进行组合来构建更加高级的表示(expressive representations)。如果深入研究CNN和图结构的特点，可以发现CNN的核心特点在于：**局部连接(local connection)**，**权重共享(shared weights)**和**多层叠加(multi-layer)**。这些同样在图问题中非常试用，因为**图结构是最典型的局部连接结构**，其次，**共享权重可以减少计算量**，另外，**多层结构是处理分级模式(hierarchical patterns)的关键**。然而，CNN只能在欧几里得数据(Euclidean data)，比如二维图片和一维文本数据上进行处理，而这些数据只是图结构的特例而已，对于一般的图结构，可以发现很难将CNN中的卷积核(convolutional filters)和池化操作(pooling operators)迁移到图的操作上。



如上图，左图为欧几里得空间，右图为非欧几里得空间。

**另一种动机来源于图嵌入**，所谓嵌入，就是对图的节点、边或者子图(subgraph)学习得到一个低维的向量表示，传统的机器学习方法通常基于人工特征工程来构建特征，但是这种方法受限于灵活性不足、表达能力不足以工程量过大的问题，词嵌入常见的模型有**Skip-gram**, **CBOW**等，图嵌入常见模型有**DeepWalk**, **Node2Vec**等，然而，这些方法方法有两种严重的缺点，首先就是**节点编码中权重未共享**，导致**权重数量随着节点增多而线性增大**，另外就是**直接嵌入方法缺乏泛化能力**，意味着无法处理动态图以及泛化到新的图。

## GNN和传统NN的区别

首先，标准的神经网络比如CNN和RNN不能够适当地处理图结构输入，因为它们都需要节点的特征按照一定的顺序进行排列，但是，对于图结构而言，并没有天然的顺序而言，如果使用顺序来完整地表达图的话，那么就需要将图分解成所有可能的序列，然后对序列进行建模，显然，这种方式非常的冗余以及计算量非常大，与此相反，**GNN采用在每个节点上分别传播(propagate)的方式进行学习**，由此忽略了节点的顺序，相当于GNN的输出会随着输入的不同而不同。

另外，**图结构的边表示节点之间的依存关系**，然而，传统的神经网络中，依存关系是通过节点特征表达出来的，也就是说，传统的神经网络不是显式地表达这种依存关系，而是通过不同节点特征来间接地表达节点之间的关系。通常来说，**GNN通过邻居节点的加权求和来更新节点的隐藏状态**。

最后，就是对于高级的人工智能来说，**推理是一个非常重要的研究主题**，人类大脑的推理过程基本上都是基于图的方式，这个图是从日常的生活经历中学习得到的。GNN尝试从非结构化数据比如情景图片和故事文本中产生结构化的图，并通过这些图来生成更高层的AI系统。

## GNN分类

论文对GNN模型分类如下：

- 图卷积网络(Graph convolutional networks)和图注意力网络(graph attention networks)，因为涉及到传播步骤(propagation step)。
- 图的空域网络(spatial-temporal networks)，因为该模型通常用在动态图(dynamic graph)上。
- 图的自编码(auto-encoder)，因为该模型通常使用无监督学习(unsupervised)的方式。
- 图生成网络(generative networks)，因为是生成式网络。

## GNN模型概览

在该节中，首先提出原始的图神经网络问题以及处理方法，然后介绍各种GNN的变体模型，用来解决原始GNN的一些缺点，最后介绍三种通用的框架，包括MPNN、NLNN和GN。

如下表是论文通用的符号表示以及对应含义说明：

Notations	Descriptions
$\mathbb{R}^m$	$m$ -dimensional Euclidean space
$a, \mathbf{a}, \mathbf{A}$	Scalar, vector, matrix
$\mathbf{A}^T$	Matrix transpose
$\mathbf{I}_N$	Identity matrix of dimension $N$
$\mathbf{g}_\theta \star \mathbf{x}$	Convolution of $\mathbf{g}_\theta$ and $\mathbf{x}$
$N$	Number of nodes in the graph
$N^v$	Number of nodes in the graph
$N^e$	Number of edges in the graph
$\mathcal{N}_v$	Neighborhood set of node $v$
$\mathbf{a}_v^t$	Vector $\mathbf{a}$ of node $v$ at time step $t$
$\mathbf{h}_v$	Hidden state of node $v$
$\mathbf{h}_v^t$	Hidden state of node $v$ at time step $t$
$\mathbf{e}_{vw}$	Features of edge from node $v$ to $w$
$\mathbf{e}_k$	Features of edge with label $k$
$\mathbf{o}_v^t$	Output of node $v$
$\mathbf{W}^i, \mathbf{U}^i, \mathbf{W}^o, \mathbf{U}^o, \dots$	Matrices for computing $\mathbf{i}, \mathbf{o}, \dots$
$\mathbf{b}^i, \mathbf{b}^o, \dots$	Vectors for computing $\mathbf{i}, \mathbf{o}, \dots$
$\sigma$	The logistic sigmoid function
$\rho$	An alternative non-linear function
$\tanh$	The hyperbolic tangent function
$\text{LeakyReLU}$	The LeakyReLU function
$\odot$	Element-wise multiplication operation
$\parallel$	Vector concatenation

## 图神经网络

图神经网络的概念第一次在[论文](#)中提出，该论文将现存的神经网络模型扩展到处理图领域的数据。在一个图结构中，每一个节点由它自身的特征以及与其相连的节点特征来定义该节点。GNN的目标是学习得到一个状态的嵌入向量(embedding) $\mathbf{h}_v \in \mathbb{R}^s$ ，这个向量包含每个节点的邻居节点的信息，其中， $\mathbf{h}_v$ 表示节点 $v$ 的状态向量，这个向量可以用于产生输出 $\mathbf{o}_v$ ，比如输出可以是节点的标签，假设 $f$ 是带有参数的函数，叫做局部转化函数(local transition function)，这个函数在所有节点中共享，并根据邻居节点的输入来更新节点状态，假设 $g$ 为局部输出函数(local output function)，这个函数用于描述输出的产生方式。那么 $\mathbf{h}_v$ 和 $\mathbf{o}_v$ 按照如下式子产生：

$$\mathbf{o}_v = g(\mathbf{h}_v, \mathbf{x}_v) \quad (1)$$

其中， $\mathbf{x}_v$ ， $\mathbf{x}_{co[v]}$ ， $\mathbf{h}_{ne[v]}$ ， $\mathbf{x}_{ne[v]}$ 分别表示节点 $v$ 的特征向量，节点 $v$ 边的特征向量，节点 $v$ 邻居节点的状态向量和节点 $v$ 邻居节点特征向量。

假设将所有的状态向量，所有的输出向量，所有的特征向量叠加起来分别使用矩阵 $\mathbf{H}$ ,  $\mathbf{O}$ ,  $\mathbf{X}$ 和 $\mathbf{X}_N$ 来表示，那么可以得到更加紧凑的表示：

$$\mathbf{H} = F(\mathbf{H}, \mathbf{X}) \quad (2)$$

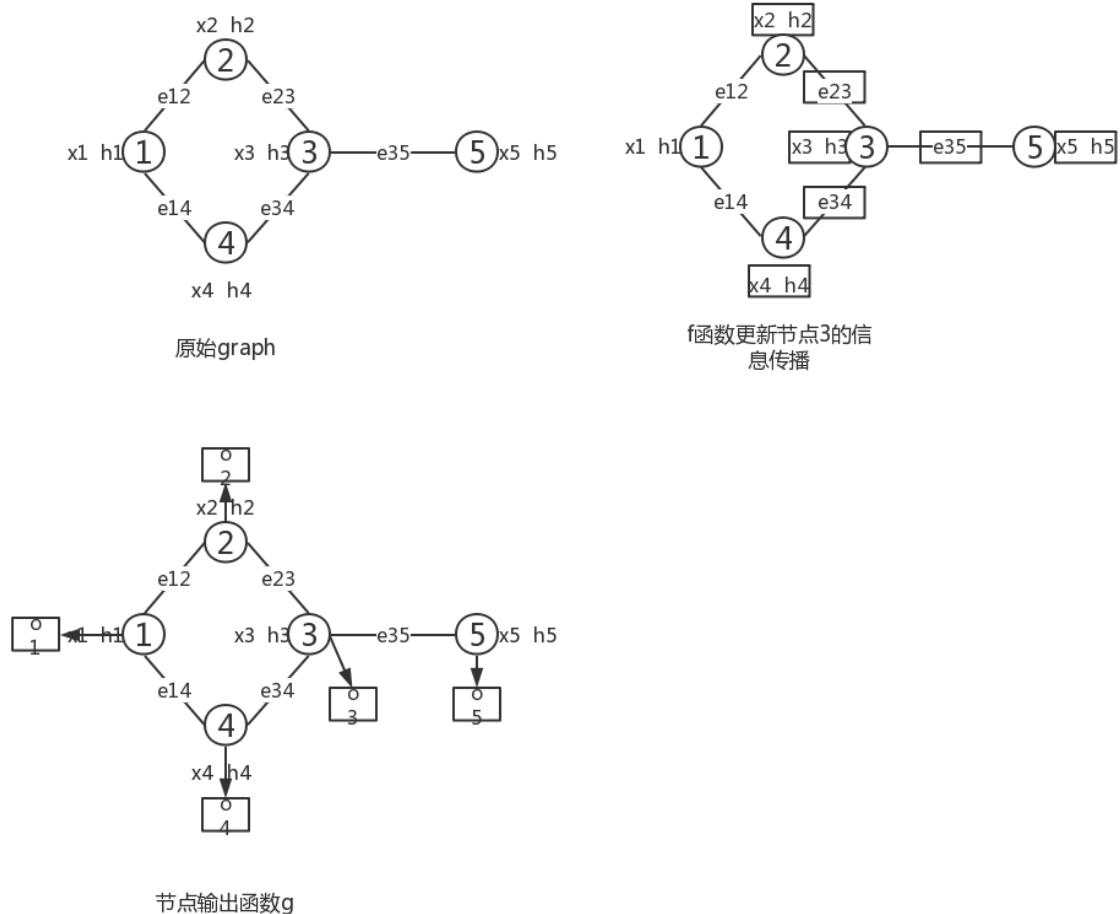
$$\mathbf{O} = G(\mathbf{H}, \mathbf{X}_N) \quad (3)$$

其中， $F$ 表示全局转化函数(global transition function)， $G$ 表示全局输出函数(global output function)，分别是所有节点 $f$ 和 $g$ 的叠加形式， $\mathbf{H}$ 是方程(2)不动点，并且在 $F$ 为收缩映射的假设下 $\mathbf{H}$ 被唯一地定义。根据Banach的不动点定理，GNN使用如下的传统迭代方法来计算状态参量：

$$\mathbf{H}^{t+1} = F(\mathbf{H}^t, \mathbf{X}) \quad (4)$$

其中， $\mathbf{H}^t$ 表示 $\mathbf{H}$ 的第 $t$ 个迭代周期的张量，按照方程(4)迭代的系统按照指数级速度收敛收敛到最终的不动点解。

整个过程示意图如下



在定义好GNN的框架之后，下一个问题是学习函数 $f$ 和 $g$ 的参数，使用目标信息(使用 $\mathbf{t}_v$ 表示特定节点的标签)来进行监督学习，loss可以定义如下：

$$loss = \sum_{i=1}^p (\mathbf{t}_i - \mathbf{o}_i) \quad (5)$$

其中， $p$ 表示监督的节点数量，学习算法使用梯度下降法，整个过程按照如下步骤进行：

- 状态 $\mathbf{h}_v^t$ 按照方程(???)的公式迭代更新 $T$ 个轮次，这时得到的 $\mathbf{H}$ 会接近不动点的解 $\mathbf{H}(T) \approx \mathbf{H}$ 。
- 权重 $\mathbf{W}$ 的梯度从loss计算得到。
- 权重 $\mathbf{W}$ 根据上一步中计算的梯度更新。

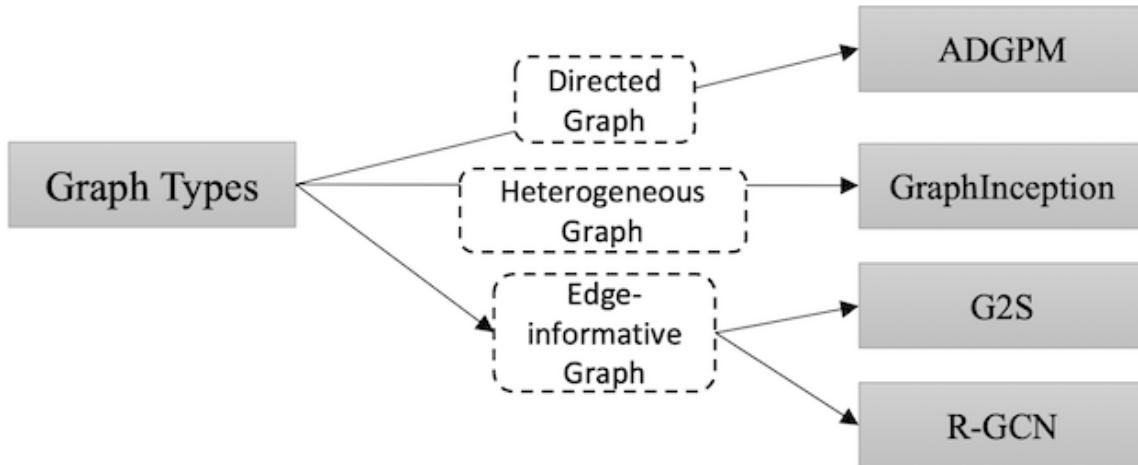
原始的GNN有如下的缺点，首先，对不动点使用迭代的方法来更新节点的隐藏状态，效率并不高，其次，在迭代过程中，原始GNN使用相同的参数，而其他比较著名的模型在不同的网络层采用不同的参数，使得模型能够学习到更加深的特征表达，而且，节点隐藏层的更新是顺序流程，可以从GRU和LSTM的cell的结构中获取灵感。另外就是一些边(edges)上可能会存在某些信息特征不能被有效地考虑进去。最后就是，如果我们要学习节点的向量表示而不是图的表示，那么使用不动点的方法是不妥当的，因为在不动点的向量表示分布在数值上会非常的平滑，这样的向量对于区分不同的节点并无太大帮助。

## 各种GNN的变体

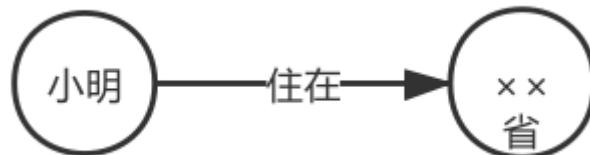
在该节中，论文首先列出各种GNN变体在不同的图类型上的操作方式，这些变体扩展了原始模型的表达能力，然后，论文列出了在传播步骤上的几种结构(包括卷积、门机制、注意力机制和跳过连接)，最后描述使用特定的训练方法来提高训练效率。

### 不同的图类型

原始的GNN输入的图结构包含带有标签信息的节点和无向边，这是最简单的图结构，其它种类的图结构主要有有向图、异质图、带有边信息图和动态图。图类型以及主要的模型如下图：



- **有向图(Directed Graphs)**: 无向边可以看成是两个有向边的结合，但是，有向边比无向边能够提供更多的信息，比如在知识图谱里面，边开始于头实体(head entity)，结束于尾实体(tail entity)，如下图



其中，这个有向边表示两个节点之间存在某种关系。

[论文DGP](#)使用两种权重矩阵 $\mathbf{W}_p$ 和 $\mathbf{W}_c$ 来结合更加精确的结构化信息，DGP的传播方式如下：

$$\mathbf{H}^t = \sigma(\mathbf{D}_p^{-1} \mathbf{A}_p \sigma(\mathbf{D}_c^{-1} \mathbf{A}_c \mathbf{H}^{t-1} \mathbf{W}_c) \mathbf{W}_p) \quad (6)$$

其中， $\mathbf{D}_p^{-1} \mathbf{A}_p$ 和 $\mathbf{D}_c^{-1} \mathbf{A}_c$ 分别是双亲(parents)和后代(children)的归一化邻接矩阵。

- 异质图(Heterogeneous Graphs): 异质图含有多种不同的节点种类, 处理这种图最简单的方法是将每种节点的类型转化为One-hot特征向量, 然后将One-hot向量和原始的节点特征进行连接, 作为该节点的特征向量。其中, [论文GraphInception](#)提出将元路径(metapath)概念用在异质图的信息传播上, 通过元路径的方式, 我们可以根据节点类型和距离来对局部范围内节点进行分组, 对于每一组, GraphInception将它作为异质图的一个子图, 然后在子图内进行传播, 并将不同异质图得到的结果进行连接得到综合的节点表示。

- 带有边信息的图(Graphs with Edge Information): 在这种图变体中, 每一个边都带有额外的信息, 比如权重和边的类型, 我们可以使用两种方法来处理这种图:

一种方式是将原始的图转化为一个二分图(bipartite graph), 处理方式为将原始的边转化为一个节点以及两条新的边, [论文G2S](#)的编码器使用如下的传播函数:

$$\mathbf{h}_v^t = \rho \left( \frac{1}{|\mathcal{N}_v|} \sum_{u \in \mathcal{N}_v} \mathbf{W}_r (\mathbf{r}_v^t \odot \mathbf{h}_u^{t-1}) + \mathbf{b}_r \right) \quad (7)$$

其中,  $\mathbf{W}_r$  和  $\mathbf{b}_r$  是不同边类型的传播参数。

另一种方式是在不同种类的边上, 使用不同的权重矩阵来进行传播的方式, 也就是说, 每一种边类型都关联一个权重矩阵, 显然, 对于边类型有很多的情况下, 这种方式的参数量会非常大, [论文r-GCN](#)采用两种方法来减小参数。

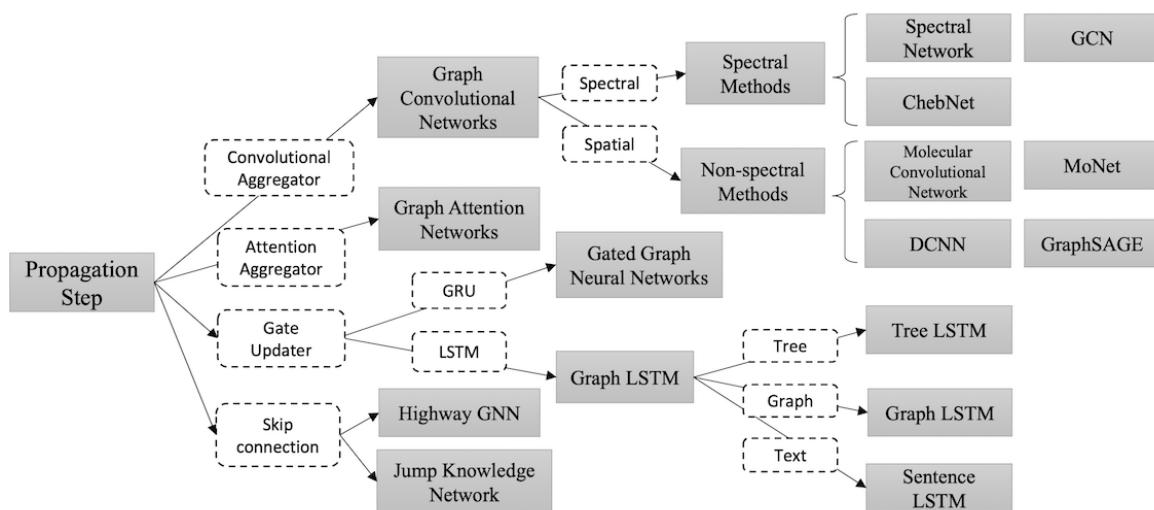
- 动态图(Dynamic Graphs): 动态图类型有静态的图结构, 并且能够处理动态的输入信号。DCRNN和STGCN首先使用GNN获取空域信息, 然后将该信息输入到一个序列模型, 比如 sequence-to-sequence模型或者CNN模型。与此相反的是, Structural-RNN和ST-GCN同时获取时间信息和空间信息。

## 传播类型

论文中的传播(propagation)指的是汇集从邻居节点和连接的边的信息, 来对节点进行更新的过程, 这个过程在模型中获取节点(或边)的隐藏状态是非常重要的。对于信息传播步骤(propagation step), 有几种主要的GNN变体, 而在输出步骤(output step)中, 研究者通常使用简单的前向传播的神经网络。

不同的GNN的信息传播变体使用下表进行列出, 这些变体采用不同的信息传播方式来从邻居节点中获取信息, 并通过设定的更新器(updater)来对节点的隐藏状态进行更新。

图的节点更新的主要信息传播(information aggregator)类型以及主要的模型如下图:



不同类别模型的Aggregator计算方法和Updater计算方法如下表

TABLE 2  
Different variants of graph neural networks.

Name	Variant	Aggregator	Updater
Spectral Methods	ChebNet	$\mathbf{N}_k = \mathbf{T}_k(\tilde{\mathbf{L}})\mathbf{X}$	$\mathbf{H} = \sum_{k=0}^K \mathbf{N}_k \Theta_k$
	1 <sup>st</sup> -order model	$\mathbf{N}_0 = \mathbf{X}$ $\mathbf{N}_1 = \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \mathbf{X}$	$\mathbf{H} = \mathbf{N}_0 \Theta_0 + \mathbf{N}_1 \Theta_1$
	Single parameter	$\mathbf{N} = (\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}})^{-1} \mathbf{X}$	$\mathbf{H} = \mathbf{N} \Theta$
	GCN	$\mathbf{N} = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X}$	$\mathbf{H} = \mathbf{N} \Theta$
Non-spectral Methods	Neural FPs	$\mathbf{h}_{\mathcal{N}_v}^t = \mathbf{h}_v^{t-1} + \sum_{k=1}^{\mathcal{N}_v} \mathbf{h}_k^{t-1}$	$\mathbf{h}_v^t = \sigma(\mathbf{h}_{\mathcal{N}_v}^t \mathbf{W}_L^{\mathcal{N}_v})$
	DCNN	Node classification: $\mathbf{N} = \mathbf{P}^* \mathbf{X}$ Graph classification: $\mathbf{N} = \mathbf{1}_N^T \mathbf{P}^* \mathbf{X} / N$	$\mathbf{H} = f(\mathbf{W}^c \odot \mathbf{N})$
	GraphSAGE	$\mathbf{h}_{\mathcal{N}_v}^t = \text{AGGREGATE}_t(\{\mathbf{h}_u^{t-1}, \forall u \in \mathcal{N}_v\})$	$\mathbf{h}_v^t = \sigma(\mathbf{W}^t \cdot [\mathbf{h}_v^{t-1} \  \mathbf{h}_{\mathcal{N}_v}^t])$
Graph Attention Networks	GAT	$\alpha_{vk} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^T [\mathbf{W}\mathbf{h}_v \  \mathbf{W}\mathbf{h}_k]))}{\sum_{j \in \mathcal{N}_v} \exp(\text{LeakyReLU}(\mathbf{a}^T [\mathbf{W}\mathbf{h}_v \  \mathbf{W}\mathbf{h}_j]))}$ $\mathbf{h}_{\mathcal{N}_v}^t = \sigma(\sum_{k \in \mathcal{N}_v} \alpha_{vk} \mathbf{W}\mathbf{h}_k)$ Multi-head concatenation: $\mathbf{h}_{\mathcal{N}_v}^t = \left\  \sum_{m=1}^M \sigma(\sum_{k \in \mathcal{N}_v} \alpha_{vk}^m \mathbf{W}^m \mathbf{h}_k) \right\ $ Multi-head average: $\mathbf{h}_{\mathcal{N}_v}^t = \sigma\left(\frac{1}{M} \sum_{m=1}^M \sum_{k \in \mathcal{N}_v} \alpha_{vk}^m \mathbf{W}^m \mathbf{h}_k\right)$	$\mathbf{h}_v^t = \mathbf{h}_{\mathcal{N}_v}^t$
Gated Graph Neural Networks	GGNN	$\mathbf{h}_{\mathcal{N}_v}^t = \sum_{k \in \mathcal{N}_v} \mathbf{h}_k^{t-1} + \mathbf{b}$	$\mathbf{z}_v^t = \sigma(\mathbf{W}^z \mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}^z \mathbf{h}_v^{t-1})$ $\mathbf{r}_v^t = \sigma(\mathbf{W}^r \mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}^r \mathbf{h}_v^{t-1})$ $\mathbf{h}_v^t = \tanh(\mathbf{W}^h \mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}(\mathbf{r}_v^t \odot \mathbf{h}_v^{t-1}))$ $\mathbf{h}_v^t = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{t-1} + \mathbf{z}_v^t \odot \mathbf{h}_v^t$
Graph LSTM	Tree LSTM (Child sum)	$\mathbf{h}_{\mathcal{N}_v}^t = \sum_{k \in \mathcal{N}_v} \mathbf{h}_k^{t-1}$	$\mathbf{i}_v^t = \sigma(\mathbf{W}^i \mathbf{x}_v^t + \mathbf{U}^i \mathbf{h}_{\mathcal{N}_v}^t + \mathbf{b}^i)$ $\mathbf{f}_{vk}^t = \sigma(\mathbf{W}^f \mathbf{x}_v^t + \mathbf{U}^f \mathbf{h}_k^{t-1} + \mathbf{b}^f)$ $\mathbf{o}_v^t = \sigma(\mathbf{W}^o \mathbf{x}_v^t + \mathbf{U}^o \mathbf{h}_{\mathcal{N}_v}^t + \mathbf{b}^o)$ $\mathbf{u}_v^t = \tanh(\mathbf{W}^u \mathbf{x}_v^t + \mathbf{U}^u \mathbf{h}_{\mathcal{N}_v}^t + \mathbf{b}^u)$ $\mathbf{c}_v^t = \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{k \in \mathcal{N}_v} \mathbf{f}_{vk}^t \odot \mathbf{c}_k^{t-1}$ $\mathbf{h}_v^t = \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)$
	Tree LSTM (N-ary)	$\mathbf{h}_{\mathcal{N}_v}^{ti} = \sum_{l=1}^K \mathbf{U}_l^i \mathbf{h}_{vl}^{t-1}$ $\mathbf{h}_{\mathcal{N}_v k}^{tf} = \sum_{l=1}^K \mathbf{U}_{kl}^f \mathbf{h}_{vl}^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{to} = \sum_{l=1}^K \mathbf{U}_l^o \mathbf{h}_{vl}^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{tu} = \sum_{l=1}^K \mathbf{U}_l^u \mathbf{h}_{vl}^{t-1}$	$\mathbf{i}_v^t = \sigma(\mathbf{W}^i \mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{ti} + \mathbf{b}^i)$ $\mathbf{f}_{vk}^t = \sigma(\mathbf{W}^f \mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v k}^{tf} + \mathbf{b}^f)$ $\mathbf{o}_v^t = \sigma(\mathbf{W}^o \mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{to} + \mathbf{b}^o)$ $\mathbf{u}_v^t = \tanh(\mathbf{W}^u \mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{tu} + \mathbf{b}^u)$ $\mathbf{c}_v^t = \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{l=1}^K \mathbf{f}_{vl}^t \odot \mathbf{c}_{vl}^{t-1}$ $\mathbf{h}_v^t = \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)$
	Graph LSTM in [44]	$\mathbf{h}_{\mathcal{N}_v}^{ti} = \sum_{k \in \mathcal{N}_v} \mathbf{U}_{m(v,k)}^i \mathbf{h}_k^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{to} = \sum_{k \in \mathcal{N}_v} \mathbf{U}_{m(v,k)}^o \mathbf{h}_k^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{tu} = \sum_{k \in \mathcal{N}_v} \mathbf{U}_{m(v,k)}^u \mathbf{h}_k^{t-1}$	$\mathbf{i}_v^t = \sigma(\mathbf{W}^i \mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{ti} + \mathbf{b}^i)$ $\mathbf{f}_{vk}^t = \sigma(\mathbf{W}^f \mathbf{x}_v^t + \mathbf{U}_{m(v,k)}^f \mathbf{h}_k^{t-1} + \mathbf{b}^f)$ $\mathbf{o}_v^t = \sigma(\mathbf{W}^o \mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{to} + \mathbf{b}^o)$ $\mathbf{u}_v^t = \tanh(\mathbf{W}^u \mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{tu} + \mathbf{b}^u)$ $\mathbf{c}_v^t = \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{k \in \mathcal{N}_v} \mathbf{f}_{vk}^t \odot \mathbf{c}_k^{t-1}$ $\mathbf{h}_v^t = \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)$

- 卷积(Convolution): 这个方向上分为频域方法和非频域(空域)方法。

频域方法使用图的频域表示，主要有以下几种模型：

- Spectral Network: [论文](#)提出了Spectral network，卷积操作定义为傅里叶频域计算图拉普拉斯(graph Laplacian)的特征值分解。这个操作可以定义为使用卷积核 $\mathbf{g}_\theta = \text{diag}(\theta)$ 对输入 $\mathbf{x} \in \mathbb{R}^N$ (每一个节点有一个标量值)的卷积操作，其中 $\theta \in \mathbb{R}^N$

$$\mathbf{g}_\theta * \mathbf{x} = \mathbf{U} \mathbf{g}_\theta(\Lambda) \mathbf{U}^T \mathbf{x} \quad (8)$$

其中， $\mathbf{U}$ 是标准化图拉普拉斯矩阵 $\mathbf{L} = \mathbf{I}_N - \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} = \mathbf{U} \Lambda \mathbf{U}^T$ 的特征向量矩阵， $\mathbf{D}$ 是度矩阵(degree matrix)， $\mathbf{A}$ 是图的邻接矩阵(adjacency matrix)， $\Lambda$ 为以特征值为对角线上的值的对角矩阵。这个操作会有导致较高的计算量。

- ChebNet: [论文](#)根据切比雪夫多项式定理，认为 $\mathbf{g}_\theta(\Lambda)$ 可以通过截取多项式的前 $K$ 项来进行估计，因此，操作为

$$\mathbf{g}_\theta \star \mathbf{x} \approx \mathbf{U} \sum_{k=0}^K \theta_k \mathbf{T}_k(\tilde{\mathbf{L}}) \mathbf{U}^T \mathbf{x} \quad (9)$$

其中,  $\tilde{\mathbf{L}} = \frac{2}{\lambda_{\max}} \mathbf{L} - \mathbf{I}_N$ ,  $\lambda_{\max}$  表示矩阵  $\mathbf{L}$  最大的特征值,  $\theta \in \mathbb{R}^K$  为切比雪夫系数向量, 切比雪夫多项式定义为  $\mathbf{T}_k(\mathbf{x}) = 2\mathbf{x}\mathbf{T}_{k-1}(\mathbf{x}) - \mathbf{T}_{k-2}(\mathbf{x})$ , 且有  $\mathbf{T}_0(\mathbf{x}) = 1$  以及  $\mathbf{T}_1(\mathbf{x}) = \mathbf{x}$ 。可以看出, 这个操作是  $K$ -localized, 因为在拉普拉斯中它是一个  $K$  阶多项式。由此避免了计算拉普拉斯的特征值。

- GCN: [论文](#) 限制了逐层的卷积操作, 并设置  $K = 1$  来减缓过拟合的问题, 它还近似了  $\lambda_{\max} \approx 2$ , 最后简化的方程如下

$$\mathbf{g}_{\theta'} \star \mathbf{x} \approx \theta'_0 \mathbf{x} + \theta'_1 (\mathbf{L} - \mathbf{I}_N) \mathbf{x} = \theta'_0 \mathbf{x} - \theta'_1 \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \mathbf{x} \quad (10)$$

使用两个无限制的参数  $\theta'_0$  和  $\theta'_1$ 。在通过设置  $\theta = \theta'_0 = -\theta'_1$  来限制参数的数量之后, 我们可以得到一下的表达式

$$\mathbf{g}_\theta \star \mathbf{x} \approx \theta \left( \mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \right) \mathbf{x} \quad (11)$$

值得一提的是, 叠加使用这个操作会导致数值不稳定性以及梯度爆炸或消失(因为不断地乘以同一个矩阵), 因此, 该论文里面使用了重规整化操作(renormalization):

$$\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \rightarrow \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \quad (12)$$

其中  $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ ,  $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$ , 最后, 论文将模型扩展为含有  $C$  个输入通道的信号  $\mathbf{X} \in \mathbb{R}^{N \times C}$  以及  $F$  个滤波器来用于提取特征

$$\mathbf{Z} = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X} \Theta \quad (13)$$

其中,  $\Theta \in \mathbb{R}^{C \times F}$  是滤波器参数矩阵,  $\mathbf{Z} \in \mathbb{R}^{N \times F}$  是卷积信号矩阵。

在所有这些频域方法中, 学习得到的滤波器都是基于拉普拉斯特征分解, 也就是取决于图的结构, 这也就意味着, 在一个特定结构上训练得到的模型, 并不能直接应用到另外一个结构不同的图上。

非频域的方法直接在图上定义卷积操作, 也就是在空域上相邻的邻居节点上进行操作。这种非频域方法的主要难点在于如何定义拥有不同邻居数量的卷积操作以及保持 CNN 的局部不变性。主要的模型有如下一些:

- Neural FPs: [论文](#) 对不同度的节点使用不同的权重矩阵

$$\mathbf{x} = \mathbf{h}_v^{t-1} + \sum_{i=1}^{|\mathcal{N}_v|} \mathbf{h}_i^{t-1} \quad (14)$$

$$\mathbf{h}_v^t = \sigma \left( \mathbf{x} \mathbf{W}_t^{|\mathcal{N}_v|} \right) \quad (15)$$

其中,  $\mathbf{W}_t^{|\mathcal{N}_v|}$  是在第  $t$  层的度为  $|\mathcal{N}_v|$  的节点的权重矩阵, 这种方法的缺点在于不能应用到节点度比较大的 graph 上。

论文的应用场景是分子指纹(Molecular Fingerprints)领域, 通过图神经网络的方式来学习分子表示, 从而提供下游任务比如相似度评估任务或分类任务需要的向量表示。

算法的实现步骤如下

**Algorithm 1** Circular fingerprints

```

1: Input: molecule, radius  $R$ , fingerprint length  $S$ 
2: Initialize: fingerprint vector  $\mathbf{f} \leftarrow \mathbf{0}_S$ 
3: for each atom  $a$  in molecule
4:    $\mathbf{r}_a \leftarrow g(a)$        $\triangleright$  lookup atom features
5: for  $L = 1$  to  $R$            $\triangleright$  for each layer
6:   for each atom  $a$  in molecule
7:      $\mathbf{r}_1 \dots \mathbf{r}_N = \text{neighbors}(a)$ 
8:      $\mathbf{v} \leftarrow [\mathbf{r}_a, \mathbf{r}_1, \dots, \mathbf{r}_N]$      $\triangleright$  concatenate
9:      $\mathbf{r}_a \leftarrow \text{hash}(\mathbf{v})$          $\triangleright$  hash function
10:     $i \leftarrow \text{mod}(r_a, S)$      $\triangleright$  convert to index
11:     $\mathbf{f}_i \leftarrow 1$             $\triangleright$  Write 1 at index
12: Return: binary vector  $\mathbf{f}$ 

```

**Algorithm 2** Neural graph fingerprints

```

1: Input: molecule, radius  $R$ , hidden weights  $H_1^1 \dots H_R^5$ , output weights  $W_1 \dots W_R$ 
2: Initialize: fingerprint vector  $\mathbf{f} \leftarrow \mathbf{0}_S$ 
3: for each atom  $a$  in molecule
4:    $\mathbf{r}_a \leftarrow g(a)$        $\triangleright$  lookup atom features
5: for  $L = 1$  to  $R$            $\triangleright$  for each layer
6:   for each atom  $a$  in molecule
7:      $\mathbf{r}_1 \dots \mathbf{r}_N = \text{neighbors}(a)$ 
8:      $\mathbf{v} \leftarrow \mathbf{r}_a + \sum_{i=1}^N \mathbf{r}_i$        $\triangleright$  sum
9:      $\mathbf{r}_a \leftarrow \sigma(\mathbf{v} H_L^N)$      $\triangleright$  smooth function
10:     $\mathbf{i} \leftarrow \text{softmax}(\mathbf{r}_a W_L)$      $\triangleright$  sparsify
11:     $\mathbf{f} \leftarrow \mathbf{f} + \mathbf{i}$             $\triangleright$  add to fingerprint
12: Return: real-valued vector  $\mathbf{f}$ 

```

模型缺点：

- **计算量较大**: 和传统的Circular fingerprints方法相比，有相近的atoms复杂度和网络深度复杂度，但是会有额外的矩阵乘法复杂度。对于网络深度为 $R$ ，fingerprint长度为 $L$ ，以及 $N$ 个原子的分子，卷积网络的特征维度为 $F$ ，计算复杂度为 $\mathcal{O}(RNFL + RNF^2)$ 。
- **每一层计算量限制**
- **信息传播效率限制**: 在图上进行信息传播的效果会受到图深度的影响。
- **无法区分立体异构**: 分子还会存在同分子式不同空间结构的情况，但是论文网络架构无法区分后者。
- DCNN: [论文](#)提出了扩散卷积神经网络，转化矩阵用来定义节点的邻居，对于节点分类任务，有

$$\mathbf{H} = f(\mathbf{W}^c \odot \mathbf{P}^* \mathbf{X}) \quad (16)$$

其中， $\mathbf{X}$ 是一个 $N \times F$ 的输入特征张量( $N$ 是节点的数量， $F$ 是特征的维度)， $\mathbf{P}^*$ 是一个 $N \times K \times N$ 的张量，包含矩阵 $\mathbf{P}$ 的power series $\{\mathbf{P}, \mathbf{P}^2, \dots, \mathbf{P}^K\}$ ， $\mathbf{P}$ 是来自于图邻接矩阵 $\mathbf{A}$ 的度标准(degree-normalized)的转化矩阵。

- DGCN: [论文](#)提出了对偶图卷积网络，同时考虑到图上的局部一致性和全局一致性，它使用两组卷积网络来获取局部/全局的一致性，并采用一个无监督的loss来组合它们，第一个卷积网络和方程(13)相同，第二个卷积网络将邻接矩阵替换为PPMI(positive pointwise mutual information)矩阵

$$\mathbf{H}' = \rho \left( \mathbf{D}_P^{-\frac{1}{2}} \mathbf{X}_P \mathbf{D}_P^{-\frac{1}{2}} \mathbf{H} \Theta \right) \quad (17)$$

其中， $\mathbf{X}_P$ 为PPMI矩阵， $\mathbf{D}_P$ 为 $\mathbf{X}_P$ 的对角度矩阵。

- GraphSAGE: [论文](#)提出了一个一般的归纳框架，这个框架通过从一个节点的局部邻居中采样和聚合特征，来产生节点的embedding

$$\mathbf{h}_{\mathcal{N}_v}^t = \text{AGGREGATE}_t \left( \{\mathbf{h}_u^{t-1}, \forall u \in \mathcal{N}_v\} \right) \quad (18)$$

$$\mathbf{h}_v^t = \sigma \left( \mathbf{W}^t \cdot \left[ \mathbf{h}_v^{t-1} \| \mathbf{h}_{\mathcal{N}_v}^t \right] \right) \quad (19)$$

但是，上述的方程并不会采用所有的邻居节点进行计算，而是使用均匀采样来得到固定大小的邻居节点集合，该论文推荐使用三种聚合函数：

- **平均值聚合(Mean aggregator)**: 使用如下方式计算

$$\mathbf{h}_v^t = \sigma \left( \mathbf{W} \cdot \text{MEAN}(\{\mathbf{h}_v^{t-1}\} \cup \{\mathbf{h}_u^{t-1}, \forall u \in \mathcal{N}_v\}) \right) \quad (20)$$

平均值聚合和其他的聚合方式不同，因为它不用进行连接操作。

- **LSTM聚合(LSTM aggregator)**: 基于LSTM的聚合器有更好的表达能力，然而，LSTM以序列的方式顺序地处理输入，因此，**它并没有排列不变性**，论文采用重排列节点邻居的方式，在无序集合中使用LSTM操作。
- **池化聚合(Pooling aggregator)**: 每一个邻居的隐藏状态输入到一个全连接层，然后使用最大池化操作应用到邻居节点集合。

$$\mathbf{h}_{\mathcal{N}_v}^t = \max \left( \{\sigma(\mathbf{W}_{\text{pool}} \mathbf{h}_u^{t-1} + \mathbf{b}), \forall u \in \mathcal{N}_v\} \right) \quad (21)$$

值得一提的是，任何对称的函数都可以用来替换这里的最大池化操作。

- **门机制(Gate)**: 目前在信息传播步骤中使用的门机制类似于GRU和LSTM模型，这种机制可以减小原始GNN模型的约束，并提升在图结构中的长期的信息传播。
  - [论文](#)提出了门控图神经网络(GGNN)，在信息传播步骤中使用GRU，将递归循环展开固定数量的步数 $T$ ，并使用按照时间序的反向传播来计算梯度。

具体来说，传播的基本递归循环是模型如下

$$\begin{aligned}\mathbf{a}_v^t &= \mathbf{A}_v^T [\mathbf{h}_1^{t-1} \dots \mathbf{h}_N^{t-1}]^T + \mathbf{b} \\ \mathbf{z}_v^t &= \sigma(\mathbf{W}^z \mathbf{a}_v^t + \mathbf{U}^z \mathbf{h}_v^{t-1}) \\ \mathbf{r}_v^t &= \sigma(\mathbf{W}^r \mathbf{a}_v^t + \mathbf{U}^r \mathbf{h}_v^{t-1}) \\ \tilde{\mathbf{h}}_v^t &= \tanh(\mathbf{W} \mathbf{a}_v^t + \mathbf{U} (\mathbf{r}_v^t \odot \mathbf{h}_v^{t-1})) \\ \mathbf{h}_v^t &= (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{t-1} + \mathbf{z}_v^t \odot \tilde{\mathbf{h}}_v^t\end{aligned}$$

按照第一个公式，节点 $v$ 首先从它的邻居节点汇集信息，其中， $\mathbf{A}_v$ 为图邻接矩阵 $\mathbf{A}$ 的子矩阵，表示节点 $v$ 和它的邻居节点的连接关系。然后，类似于GRU的节点更新函数将该节点前一个时刻的信息和与该节点相邻的其它节点的信息结合起来，以此来更新每一个节点的隐藏状态。 $\mathbf{a}$ 汇集了节点 $v$ 周围节点的信息， $\mathbf{z}$ 和 $\mathbf{r}$ 分别是更新门(update gate)和重置门(reset gate)。LSTM同样使用类似的方法来进行信息传播过程。

- [论文](#)提出了两个LSTM的扩展结构，*Child-Sum Tree-LSTM*和*N-ary Tree-LSTM*，类似于标准的LSTM单元，每一个Tree-LSTM单元(为 $v$ )包含输入门(input gate) $\mathbf{i}_v$ 和输出门(output gate) $\mathbf{o}_v$ ，记忆单元(memory cell) $\mathbf{c}_v$ 和隐藏状态(hidden state) $\mathbf{h}_v$ ，但与LSTM(LSTM单元只包含一个遗忘门(forget gate))不同的是，Tree-LSTM单元对每一个孩子节点 $k$ 都有一个遗忘门 $\mathbf{f}_{vk}$ ，这样就可以从孩子节点中选择性地汇集并组合信息。Child-Sum Tree-LSTM的转换方程如下

$$\begin{aligned}\widetilde{\mathbf{h}}_v^{t-1} &= \sum_{k \in \mathcal{N}_v} \mathbf{h}_k^{t-1} \\ \mathbf{i}_v^t &= \sigma(\mathbf{W}^i \mathbf{x}_v^t + \mathbf{U}^i \widetilde{\mathbf{h}}_v^{t-1} + \mathbf{b}^i) \\ \mathbf{f}_{vk}^t &= \sigma(\mathbf{W}^f \mathbf{x}_v^t + \mathbf{U}^f \mathbf{h}_k^{t-1} + \mathbf{b}^f) \\ \mathbf{o}_v^t &= \sigma(\mathbf{W}^o \mathbf{x}_v^t + \mathbf{U}^o \widetilde{\mathbf{h}}_v^{t-1} + \mathbf{b}^o) \\ \mathbf{u}_v^t &= \tanh(\mathbf{W}^u \mathbf{x}_v^t + \mathbf{U}^u \widetilde{\mathbf{h}}_v^{t-1} + \mathbf{b}^u) \\ \mathbf{c}_v^t &= \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{k \in \mathcal{N}_v} \mathbf{f}_{vk}^t \odot \mathbf{c}_k^{t-1} \\ \mathbf{h}_v^t &= \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)\end{aligned}$$

$\mathbf{x}_v^t$ 是标准LSTM中在时刻 $t$ 的输入。

如果一个树每一个节点最多有 $K$ 个分支，并且节点所有的孩子节点都是有序的，比如，这些孩子节点可以被从1编号到 $K$ ，那么可以使用N-ary Tree-LSTM，对于节点 $v$ ， $\mathbf{h}_{vk}^t$ 和 $\mathbf{c}_{vk}^t$ 分别表示在 $t$ 时刻，它的第 $k$ 个孩子节点的隐藏状态和记忆单元，转化方程为

$$\begin{aligned}
\mathbf{i}_v^t &= \sigma \left( \mathbf{W}^i \mathbf{x}_v^t + \sum_{l=1}^K \mathbf{U}_l^i \mathbf{h}_{vl}^{t-1} + \mathbf{b}^i \right) \\
\mathbf{f}_{vk}^t &= \sigma \left( \mathbf{W}^f \mathbf{x}_v^t + \sum_{l=1}^K \mathbf{U}_{kl}^f \mathbf{h}_{vl}^{t-1} + \mathbf{b}^f \right) \\
\mathbf{o}_v^t &= \sigma \left( \mathbf{W}^o \mathbf{x}_v^t + \sum_{l=1}^K \mathbf{U}_l^o \mathbf{h}_{vl}^{t-1} + \mathbf{b}^o \right) \\
\mathbf{u}_v^t &= \tanh \left( \mathbf{W}^u \mathbf{x}_v^t + \sum_{l=1}^K \mathbf{U}_{vl}^u \odot \mathbf{c}_{vl}^{t-1} + \mathbf{b}^u \right) \\
\mathbf{c}_v^t &= \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{l=1}^K \mathbf{f}_{vl}^t \odot \mathbf{c}_{vl}^{t-1} \\
\mathbf{h}_v^t &= \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)
\end{aligned}$$

对每个孩子节点 $k$ 都赋予一个单独的参数矩阵，使得该模型相比于Child-Sum Tree-LSTM能够学习得到更加细微的节点表示。这两种类型的Tree-LSTM都能够很容易地应用到图

- 论文提出了一个Graph LSTM的变体，用于关系抽取(relation extraction)的任务上。图和树的主要区别在于，图结构的边有它们自己的label，由此，论文采用不同的权重矩阵来表示不同的label

$$\begin{aligned}
\mathbf{i}_v^t &= \sigma \left( \mathbf{W}^i \mathbf{x}_v^t + \sum_{k \in \mathcal{N}_v} \mathbf{U}_{m(v,k)}^i \mathbf{h}_k^{t-1} + \mathbf{b}^i \right) \\
\mathbf{f}_{vk}^t &= \sigma \left( \mathbf{W}^f \mathbf{x}_v^t + \mathbf{U}_{m(v,k)}^f \mathbf{h}_k^{t-1} + \mathbf{b}^f \right) \\
\mathbf{o}_v^t &= \sigma \left( \mathbf{W}^o \mathbf{x}_v^t + \sum_{k \in \mathcal{N}_v} \mathbf{U}_{m(v,k)}^o \mathbf{h}_k^{t-1} + \mathbf{b}^o \right) \\
\mathbf{u}_v^t &= \tanh \left( \mathbf{W}^u \mathbf{x}_v^t + \sum_{k \in \mathcal{N}_v} \mathbf{U}_{m(v,k)}^{u-1} \mathbf{h}_k^{t-1} + \mathbf{b}^u \right) \\
\mathbf{c}_v^t &= \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{k \in \mathcal{N}_v} \mathbf{f}_{vk}^t \odot \mathbf{c}_k^{t-1} \\
\mathbf{h}_v^t &= \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)
\end{aligned} \tag{22}$$

其中， $m(v, k)$ 表示在节点 $v$ 和 $k$ 之间边的label。

- 注意力机制(Attention):** 注意力机制在很多基于序列任务(sequence-based tasks)比如机器翻译、机器阅读理解等等上都产生了非常好的效果。
  - 论文提出了图注意力网络(graph attention network, GAT)将注意力机制引入到信息传播步骤，这个模型通过对它的邻居节点增加注意力来计算节点的隐藏状态，和self-attention策略类似。

该论文定义了一个graph attentional layer，并通过叠加这种层来构建任意的图注意力网络，这个层计算节点对 $(i, j)$ 的注意力系数(coefficients)，计算方式如下：

$$\alpha_{ij} = \frac{\exp(\text{Leaky ReLU}(\mathbf{a}^T [\mathbf{W}\mathbf{h}_i \parallel \mathbf{W}\mathbf{h}_j]))}{\sum_{k \in \mathcal{N}_i} \exp(\text{Leaky ReLU}(\mathbf{a}^T [\mathbf{W}\mathbf{h}_i \parallel \mathbf{W}\mathbf{h}_k]))} \tag{23}$$

其中， $\alpha_{ij}$ 是节点 $j$ 对 $i$ 的注意力系数， $\mathcal{N}_i$ 表示图中节点 $i$ 的邻居节点集合，节点特征的输入集合是 $\mathbf{h} = \{\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_N\}$ ， $\mathbf{h}_i \in \mathbb{R}^F$ ，其中 $N$ 是节点的个数， $F$ 是每个节点的特征维度。这个层会产生一个新的节点特征集(可能有不同的特征维度 $F'$ )  
 $\mathbf{h}' = \{\mathbf{h}'_1, \mathbf{h}'_2, \dots, \mathbf{h}'_N\}$ ， $\mathbf{h}'_i \in \mathbb{R}^{F'}$ ，作为该层的输出， $\mathbf{W} \in \mathbb{R}^{F' \times F}$ 是共享的线性变换的权重矩阵， $\mathbf{a} \in \mathbb{R}^{2F'}$ 是单层的前向神经网络的权重向量，通过softmax函数对它进行归一化，然后使用LeakyReLU( $\alpha = 0.2$ )非线性函数。

最后每个节点的输出特征可以通过下方方程获得

$$\mathbf{h}'_i = \sigma \left( \sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W} \mathbf{h}_j \right) \tag{24}$$

另外，这个注意力层采用*multi-head attention*使得网络学习过程更加稳定，它使用 $K$ 个独立的注意力来计算隐藏状态，然后将计算出的 $K$ 个特征连接(或者求平均)，得到最终的输出表示

$$\mathbf{h}'_i = \left\| \sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \mathbf{h}_j \right\| \quad (25)$$

$$\mathbf{h}'_i = \sigma \left( \frac{1}{K} \sum_{k=1}^K \sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \mathbf{h}_j \right) \quad (26)$$

其中， $\alpha_{ij}^k$ 是归一化的注意力系数，由第 $k$ 个注意力机制得到。

这篇论文的注意力机制结构有如下几个特点：(1)node-neighbor pair的计算可以是并行化的，因此操作的效率高；(2)通过给邻居节点赋予任意的权重，可以用在有不同度的图节点上；(3)可以容易地用于归纳学习问题上。

- 跳过连接(Skip connection)：许多应用都会将图神经网络层进行叠加，以此来实现更好的结果，因为更多的层意味着每一个节点能够从更多的邻居节点中获取信息，但是，许多实验发现，更深的模型并不会表现更好，反而可能表现更坏，主要因为随着指数个数上升的相邻节点数量，更多的层可能会汇集到更多的噪声信息。

一个直接的想法是使用残差网络(residual network)，但是即使使用残差连接，有更多层的GCN在很多数据上并没有2层的GCN表现的更好。

- [论文](#)提出了Highway GCN，类似于highway network，使用了“逐层门机制”(layer-wise gates)，一个层的输出公式如下：

$$\begin{aligned} \mathbf{T}(\mathbf{h}^t) &= \sigma(\mathbf{W}^t \mathbf{h}^t + \mathbf{b}^t) \\ \mathbf{h}^{t+1} &= \mathbf{h}^{t+1} \odot \mathbf{T}(\mathbf{h}^t) + \mathbf{h}^t \odot (1 - \mathbf{T}(\mathbf{h}^t)) \end{aligned}$$

通过增加highway gates，在该论文指定的特定问题上，模型在4层的表现最好。

- 分层池化(Hierarchical Pooling)：在计算机视觉中，一个卷积层后通常会接一个池化层，来得到更加一般的特征。与这种池化层类似的是，在图结构中，一种分层池化层也能够起到类似的效果，复杂的和大规模的图通常会包含丰富的分层结构，这种结构对于节点层次(node-level)和图层次(graph-level)的分类任务非常重要。

## 训练方法

原始的图卷积神经网络在训练和优化方法上有一些缺点，比如，**GCN需要计算整个图拉普拉斯矩阵**，这个操作对于大型的图计算量非常大，另外，在第 $L$ 层的节点的embedding，是通过第 $L - 1$ 层其周围所有邻居节点的embedding递归循环得到的，因此，**单个节点的感受野会随着层数的增加而指数上升**，由此，计算单个节点的梯度会非常耗时，另外，**GCN是对固定的图结构进行训练**，缺乏归纳学习的能力。有以下几种改进的方法

- **采样(Sampling)**：GraphSAGE将full graph Laplacian替换为可学习的聚合函数(aggregation function)。在学习到聚合函数和传播函数后，GraphSAGE能够对未见过的节点产生embedding。另外，GraphSAGE使用邻居节点采样(neighbor sampling)的方法来缓和感受野扩展的扩展速度。
- **感受野控制(Receptive Field Control)**
- **数据增强(Data Augmentation)**：[论文](#)考虑到GCN需要许多额外的标签数据集用于验证，以及卷积核局部化问题，为了解决这些问题，这篇论文提出Co-Training GCN和Self-Training GCN来扩充训练数据集。
- **无监督训练(Unsupervised Training)**

## GNN一般框架

除了一些GNN的变体之外，一些一般性的框架也用于将不同的模型结合到一个单一框架中，比如MPNN(结合了各种GNN和GCN方法)，NLNN(结合几种self-attention的方法)，GN(结合MPNN和NLNN以及其他一些GNN变体)。下面对这三种框架进行仔细介绍：

## Message Passing Neural Networks

该模型提出一种general framework，用于在graph上进行监督学习。模型包含两个过程，**message passing phase**和**readout phase**。信息传递阶段就是前向传播阶段，该阶段循环运行T个steps，并通过函数 $M_t$ 获取信息，通过函数 $U_t$ 更新节点，该阶段方程如下

$$\begin{aligned}\mathbf{m}_v^{t+1} &= \sum_{w \in \mathcal{N}_v} M_t(\mathbf{h}_v^t, \mathbf{h}_w^t, \mathbf{e}_{vw}) \\ \mathbf{h}_v^{t+1} &= U_t(\mathbf{h}_v^t, \mathbf{m}_v^{t+1})\end{aligned}$$

其中， $\mathbf{e}_{vw}$ 表示从节点 $v$ 到 $w$ 的边的特征向量。

**readout**阶段计算一个特征向量用于整个图的representation，使用函数 $R$ 实现

$$\hat{\mathbf{y}} = R(\{\mathbf{h}_v^T | v \in G\}) \quad (27)$$

其中 $T$ 表示整个时间step数，其中的函数 $M_t$ ， $U_t$ 和 $R$ 可以使用不同的模型设置。

例如，考虑GGNN模型的例子，有如下函数设置

$$\begin{aligned}M_t(\mathbf{h}_v^t, \mathbf{h}_w^t, \mathbf{e}_{vw}) &= \mathbf{A}_{\mathbf{e}_{vw}} \mathbf{h}_w^t \\ U_t &= GRU(\mathbf{h}_v^t, \mathbf{m}_v^{t+1}) \\ R &= \sum_{v \in V} \sigma(i(\mathbf{h}_v^T, \mathbf{h}_v^0)) \odot (j(\mathbf{h}_v^T))\end{aligned}$$

其中， $\mathbf{A}_{\mathbf{e}_{vw}}$ 表示邻接矩阵， $i$ 和 $j$ 是函数 $R$ 的神经网络。

## Non-local Neural Networks

该NLNN模型用于**使用神经网络获取长远依赖信息**，一个位置的non-local操作使用所有positions的特征向量的加权和来作为该位置的信息。以下为通用的non-local operation定义

$$\mathbf{h}'_i = \frac{1}{\mathcal{C}(\mathbf{h})} \sum_{\forall j} f(\mathbf{h}_i, \mathbf{h}_j) g(\mathbf{h}_j) \quad (28)$$

其中， $i$ 是输出位置(节点)的索引， $j$ 是所有可能位置(节点)索引， $f(\mathbf{h}_i, \mathbf{h}_j)$ 函数得到一个scalar，用于计算节点*i*与节点*j*之间的关联度， $g(\mathbf{h}_j)$ 表示输入 $\mathbf{h}_j$ 的转换函数，因子 $\frac{1}{\mathcal{C}(\mathbf{h})}$ 用于对结果进行归一化。对于这几个函数有多种设置，各种函数配置说明如下

- **线性转换函数**，即 $g(\mathbf{h}_j) = \mathbf{W}_g \mathbf{h}_j$ ，其中， $\mathbf{W}_g$ 为可学习的权重矩阵。
- **高斯函数**，对函数 $f$ 使用高斯配置

$$f(\mathbf{h}_i, \mathbf{h}_j) = e^{\mathbf{h}_i^T \mathbf{h}_j} \quad (29)$$

其中， $\mathbf{h}_i^T \mathbf{h}_j$ 为点积相似度， $\mathcal{C}(\mathbf{h}) = \sum_{\forall j} f(\mathbf{h}_i, \mathbf{h}_j)$ 。

- **Embedded Guassian**，对Guassian函数的扩展，用于计算嵌入空间的相似度

$$f(\mathbf{h}_i, \mathbf{h}_j) = e^{\theta(\mathbf{h}_i)^T} \phi(\mathbf{h}_j) \quad (30)$$

其中， $\theta(\mathbf{h}_i) = \mathbf{W}_\theta \mathbf{h}_i$ ， $\phi(\mathbf{h}_j) = \mathbf{W}_\phi \mathbf{h}_j$ ， $\mathcal{C}(\mathbf{h}) = \sum_{\forall j} f(\mathbf{h}_i, \mathbf{h}_j)$ 。

- **Dot product**，对函数 $f$ 使用点积相似度

$$f(\mathbf{h}_i, \mathbf{h}_j) = \theta(\mathbf{h}_i)^T \phi(\mathbf{h}_j) \quad (31)$$

在这种情况下， $\mathcal{C}(\mathbf{h}) = N$ ，其中 $N$ 为位置(节点)的个数。

- **Concatenation**，即

$$f(\mathbf{h}_i, \mathbf{h}_j) = \text{ReLU} \left( \mathbf{w}_f^T [\theta(\mathbf{h}_i) \| \phi(\mathbf{h}_j)] \right) \quad (32)$$

其中， $\mathbf{w}_f$ 是一个权重向量，用于将一个vector映射成一个scalar， $\mathcal{C}(\mathbf{h}) = N$ 。

## Graph Networks

首先介绍图的定义，一个graph定义为三元组 $G = (\mathbf{u}, H, E)$ ，这里使用 $H = \{\mathbf{h}_i\}_{i=1:N^v}$ 表示节点集合， $\mathbf{h}_i$ 为节点的属性向量， $\mathbf{u}$ 是全局属性， $E = \{(\mathbf{e}_k, r_k, s_k)\}_{k=1:N^e}$ 为边的集合，其中， $\mathbf{e}_k$ 为边的属性向量， $r_k$ 为receiver node索引， $s_k$ 为sender node索引。

然后是**GN block**的结构，一个**GN block**包含三个update functions( $\phi$ )和三个aggregation functions( $\rho$ )

$$\begin{aligned}\mathbf{e}'_k &= \phi^e(\mathbf{e}_k, \mathbf{h}_{r_k}, \mathbf{h}_{s_k}, \mathbf{u}) & \bar{\mathbf{e}}'_i &= \rho^{e \rightarrow h}(E'_i) \\ \mathbf{h}'_i &= \phi^h(\bar{\mathbf{e}}'_i, \mathbf{h}_i, \mathbf{u}) & \bar{\mathbf{e}}' &= \rho^{e \rightarrow u}(E') \\ \mathbf{u}' &= \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{h}}', \mathbf{u}) & \bar{\mathbf{h}}' &= \rho^{h \rightarrow u}(H')\end{aligned}\quad (33)$$

其中， $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i, k=1:N^e}$ ， $H' = \{\mathbf{h}'_i\}_{i=1:N^v}$ ， $E' = \bigcup_i E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:N^e}$ 。函数 $\rho$ 必须对于输入的顺序有不变性，而且需要能够接受不同数量的参数。

GN block的计算步骤如下：

- 使用函数 $\phi^e$ 对每条边都使用一次，输入的参数为 $(\mathbf{e}_k, \mathbf{h}_{r_k}, \mathbf{h}_{s_k}, \mathbf{u})$ ，并返回向量 $\mathbf{e}'_k$ ，对与每个节点*i*相连的输出边的集合为 $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i, k=1:N^e}$ ，所有输出边的集合为 $E' = \bigcup_i E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:N^e}$ 。
- 对集合 $E'_i$ 使用函数 $\rho^{e \rightarrow h}$ ，得到与节点*i*相连边的信息汇集向量 $\bar{\mathbf{e}}'_i$ ，用于下一步的节点更新。
- 对每一个节点*i*使用函数 $\phi^h$ ，用于计算并更新节点的属性向量 $\mathbf{h}'_i$ ，所有节点的输出集合为 $H' = \{\mathbf{h}'_i\}_{i=1:N^v}$ 。
- 对集合 $E'$ 使用函数 $\rho^{e \rightarrow u}$ ，得到所有边的信息汇集向量 $\bar{\mathbf{e}}'$ ，用于下一步的全局更新。
- 对集合 $H'$ 使用函数 $\rho^{h \rightarrow u}$ ，得到所有节点的信息汇集向量 $\bar{\mathbf{h}}'$ ，用于下一步的全局更新。
- 对每个graph应用一次函数 $\phi^u$ ，然后计算更新全局属性向量，得到 $\mathbf{u}'$

图神经网络设计基于三个基本的原则：**flexible representations**、**configurable within-block structure**、**composable multi-block architectures**。

- GN framework支持灵活的属性表达以及不同的图结构
- GN block内函数可以有灵活的配置
- GN block可以通过堆叠的方式以及权重共享的方式来得到更深的网络

## GNN应用

论文将GNN的应用分为以下几种：

- 结构化数据场景，有明显的关系结构，比如物理系统，分子结构和知识图谱
- 非结构化场景，没有明显的关系结构，比如文本
- 其他应用场景，比如产生式模型和组合优化问题

### 1. 物理系统应用

GNN可以用于对现实世界中的物理系统建模，将物体表示为节点，将物体之间的关系表示为边，由此可以使用GNN模型对目标、关系进行推理。有如下一些论文研究相关内容：

- *Interaction Networks*：该模型对各种物理系统进行预测和推理。模型输入objects和relations，然后对它们的interaction进行推理，并预测出新的系统状态。
- *Visual Interaction Networks*：该模型实现像素级的预测。

### 2. 化学和生物学应用

GNN能够用于计算分子指纹(**molecular fingerprints**)，即使用特征向量来表示分子，用于下游任务，比如computer-aided drug design。另外，GNN也可以用于protein interface prediction，有助于药物研究。

### 3. 知识图谱应用

有论文采用GNN来解决基于out-of-knowledge-base的实体问题。也有论文采用GCN来解决跨语言的知识图谱对齐任务，论文模型将不同语言的实体嵌入到向量空间，然后使用相似度进行实体对齐。

#### 4. 图像任务

目前的图像分类任务由于大数据和GPU的强大并行计算能力获得很大的突破，但是**zero-shot and few-hot learning**在图像领域仍然非常重要。有一些模型采用GNN结合图像的结构化信息来进行图像分类，比如，知识图谱可以用于额外的信息来指导**zero-shot recognition classification**。

GNN也可以用于**visual reasoning**，计算机视觉系统通常需要结合空间信息和语义信息来实现推理，因此，很自然地想到使用Graph来实现推理任务。一个典型的任务是**visual question answering**，该任务需要分别构建图像的场景图(**scene graph**)以及问题句法图(**syntactic graph**)，然后使用GGNN来训练以及预测最终的结果。**visual reasoning**的其他应用还有**object detection, interaction detection**和**region classification**。

GNN还可以用于**semantic segmentation**，语义分割的任务在于对图像每一个像素预测出一个label，由于图像的区域往往不是网格形状的，并且需要全局信息，因此使用传统的CNN会有一些局限性，有一些论文采用图结构数据来解决这种问题。

#### 5. 文本任务

GNN能够应用到多个基于文本的任务上，既可以用于sentence-level的任务，也可以用于word-level的任务。

首先就是**文本分类任务**，GNN将一个document或者sentence表示为一个以字(或词)为节点的图结构，然后使用Text GCN来学习词汇或者文本的embedding向量，用于下游任务。

其次就是**序列标注任务**，对于图中的每一个节点都有一个隐藏状态，因此，可以使用这个隐藏状态来对每一个节点进行标注。

GNN还可以用于**机器翻译任务**，原始的机器翻译是sequence-to-sequence的任务，但是使用GNN可以将语法和语义信息编码进翻译模型中。

GNN用在**关系抽取任务**中，就是在文本中抽出不同实体的语义关系，有一些系统将这个任务看作是两个单独的任务：**命名实体识别**和**关系抽取**。

GNN用于**事件抽取任务**，就是从文本中提取出事件关键的信息，有论文通过dependency tree来实现event detection。

GNN还被用于其他应用，比如文本生成任务，关系推理任务等等。

#### 6. 产生式模型

产生式模型对于社会关系建模、新型化学结构发现以及构建知识图谱也有重要作用，有如下相关的研究论文：

- *NetGAN*: 模型使用随机漫步原理来产生图，该模型将graph generation问题转化为walk generation问题，它使用来自于特定图结构的random walks作为输入，并使用GAN的结构来训练一个产生式模型。
- *MolGAN*: 该模型一次性预测离散的图结构，并采用permutation-invariant discriminator来解决node variant问题。

#### 7. 组合优化

GNN能够应用于解决在graph上的NP-hard的优化问题，比如旅行商问题(TSP)，最小生成树问题(MSP)。

## 开放问题

GNN模型目前仍然存在一些问题

- **Shallow Structure**: 传统的神经网络可以叠加上百层来提高模型的表达能力，而实验显示，GCN叠加过多的层或导致over-smoothing问题，也就是说，最终所有的节点会收敛于相同的值。

- **Dynamic Graphs**: 静态图是稳定的，因此比较好灵活地进行建模，但是动态图是动态的结构，建模较难。
- **Non-Structure Scenarios**: 对于如何从原始的非结构化数据来产生对应的图结构并没有最优的方法。
- **Scalability**: 应用尺度问题，使用embedding的方法来处理web-scale的任务比如social networks或者recommendation systems对于使用embedding的方法来说计算量非常大。首先，由于不是欧几里得结构，所以不同的节点有不同的结构，无法使用batches；其次，计算图Laplacian矩阵对于有上百万的节点和边的图是不现实的。

## The Graph Neural Network Model

- 论文：《The Graph Neural Network Model》

### 论文概要

这篇论文是第一个提出Graph Neural Network模型的论文，它将神经网络使用在图结构数据上，并描述了神经网络模型的结构组成、计算方法、优化算法、流程实现等等。论文后面还对模型的复杂度进行了评估，以及在现实任务上进行了实验和比较(比较算法为NL、L、FNN)。该报告暂时**主要关注模型设计部分和实验结果部分，忽略复杂性评估部分**。

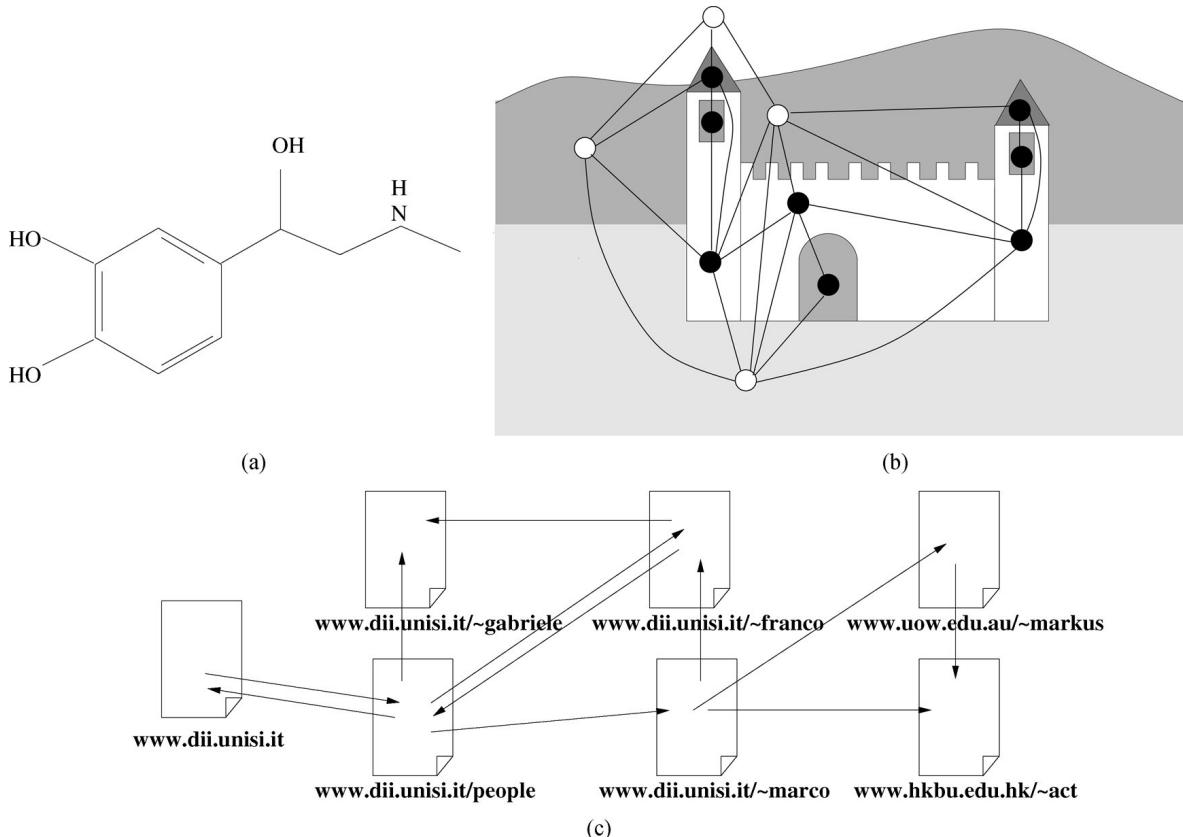
### 图领域应用

对于图领域问题，假设函数 $\tau$ 是将一个图 $G$ 和图中的一个节点 $n$ 转化为一个实值向量的函数

$$\tau(G, n) \in R^m \quad (34)$$

那么监督学习的任务就在于从已知样本中学习得到这样的函数。

图领域的应用主要可以分为两种类型：**专注于图的应用(graph-focused)**和**专注于节点的应用(node-focused)**。对于graph-focused的应用，函数 $\tau$ 和具体的节点无关，(即 $\tau(G)$ )，训练时，在一个图的数据集中进行分类或回归。对于node-focused的应用， $\tau$ 函数依赖于具体的节点 $n$ ，即 $\tau(G, n)$ ，如下例子



- 上图中(a)图是一个化学分子结构，能够使用图 $G$ 进行表示，函数 $\tau(G)$ 可能用于估计这种化学分子对人体有害的概率，因此，我们并不关注分子中具体的原子(相当于节点)，所以属于graph-focused应用。
- 上图中(b)图是一张城堡的图片，图片中的每一种结构都由节点表示，函数 $\tau(G, n)$ 可能用于预测每一个节点是否属于城堡(图中的黑点)。这种类型属于node-focused应用。

## GNN模型详述

GNN模型基于信息传播机制，每一个节点通过相互交换信息来更新自己的节点状态，直达到到某一个稳定值，GNN的输出就是在每个节点处，根据当前节点状态分别计算输出。

有如下定义：

- 一个图 $G$ 表示为一对 $(\mathcal{N}, \mathcal{E})$ ，其中， $\mathcal{N}$ 表示节点集合， $\mathcal{E}$ 表示边集。
- $ne[n]$ 表示节点 $n$ 的邻居节点集合
- $co[n]$ 表示以 $n$ 节点为顶点的所有边集合
- $\mathbf{l}_n \in \mathbb{R}^{l_N}$ 表示节点 $n$ 的特征向量
- $\mathbf{l}_{(n_1, n_2)} \in \mathbb{R}^{l_E}$ 表示边 $(n_1, n_2)$ 的特征向量
- $\mathbf{l}$ 表示所有特征向量叠在一起的向量

**注：**原论文里面 $l$ 表示label，但论文中的label指的是features of objects related to nodes and features of the relationships between the objects，也就是相关特征，所以这里一律使用特征向量翻译。

论文将图分为positional graph和nonpositional graph，对于positional graph，对于每一个节点 $n$ ，都会给该节点的邻居节点 $u$ 赋予一个position值 $\nu_n(u)$ ，该函数称为injective function， $\nu_n : ne[n] \rightarrow \{1, \dots, |\mathcal{N}|\}$ 。

假设存在一个图-节点对的集合 $\mathcal{D} = \mathcal{G} \times \mathcal{N}$ ， $\mathcal{G}$ 表示图的集合， $\mathcal{N}$ 表示节点集合，图领域问题可以表示成一个有如下数据集的监督学习框架

$$\mathcal{L} = \{(\mathcal{G}_i, n_{i,j}, \mathbf{t}_{i,j}) | \mathcal{G}_i = (\mathcal{N}_i, \mathcal{E}_i) \in \mathcal{G}; n_{i,j} \in \mathcal{N}_i; \mathbf{t}_{i,j} \in \mathbb{R}^m, 1 \leq i \leq p, 1 \leq j \leq q_i\} \quad (35)$$

其中， $n_{i,j} \in \mathcal{N}_i$ 表示集合 $\mathcal{N}_i \in \mathcal{N}$ 中的第 $j$ 个节点， $\mathbf{t}_{i,j}$ 表示节点 $n_{i,j}$ 的期望目标(即标签)。

节点 $n$ 的状态用 $\mathbf{x}_n \in \mathbb{R}^s$ 表示，该节点的输出用 $\mathbf{o}_n$ 表示， $f_w$ 为local transition function， $g_w$ 为local output function，那么 $\mathbf{x}_n$ 和 $\mathbf{o}_n$ 的更新方式如下

$$\begin{aligned} \mathbf{x}_n &= f_w(\mathbf{l}_n, \mathbf{l}_{co[n]}, \mathbf{x}_{ne[n]}, \mathbf{l}_{ne[n]}) \\ \mathbf{o}_n &= g_w(\mathbf{x}_n, \mathbf{l}_n) \end{aligned} \quad (36)$$

其中， $\mathbf{l}_n, \mathbf{l}_{co[n]}, \mathbf{x}_{ne[n]}, \mathbf{l}_{ne[n]}$ 分别表示节点 $n$ 的特征向量、与节点 $n$ 相连的边的特征向量、节点 $n$ 邻居节点的状态向量、节点 $n$ 邻居节点的特征向量。

假设 $\mathbf{x}, \mathbf{o}, \mathbf{l}, \mathbf{l}_N$ 分别为所有的状态、所有的输出、所有的特征向量、所有节点的特征向量的叠加起来的向量，那么上面函数可以写成如下形式

$$\begin{aligned} \mathbf{x} &= F_w(\mathbf{x}, \mathbf{l}) \\ \mathbf{o} &= G_w(\mathbf{x}, \mathbf{l}_N) \end{aligned} \quad (37)$$

其中， $F_w$ 为global transition function， $G_w$ 为global output function，分别是 $f_w$ 和 $g_w$ 的叠加形式。

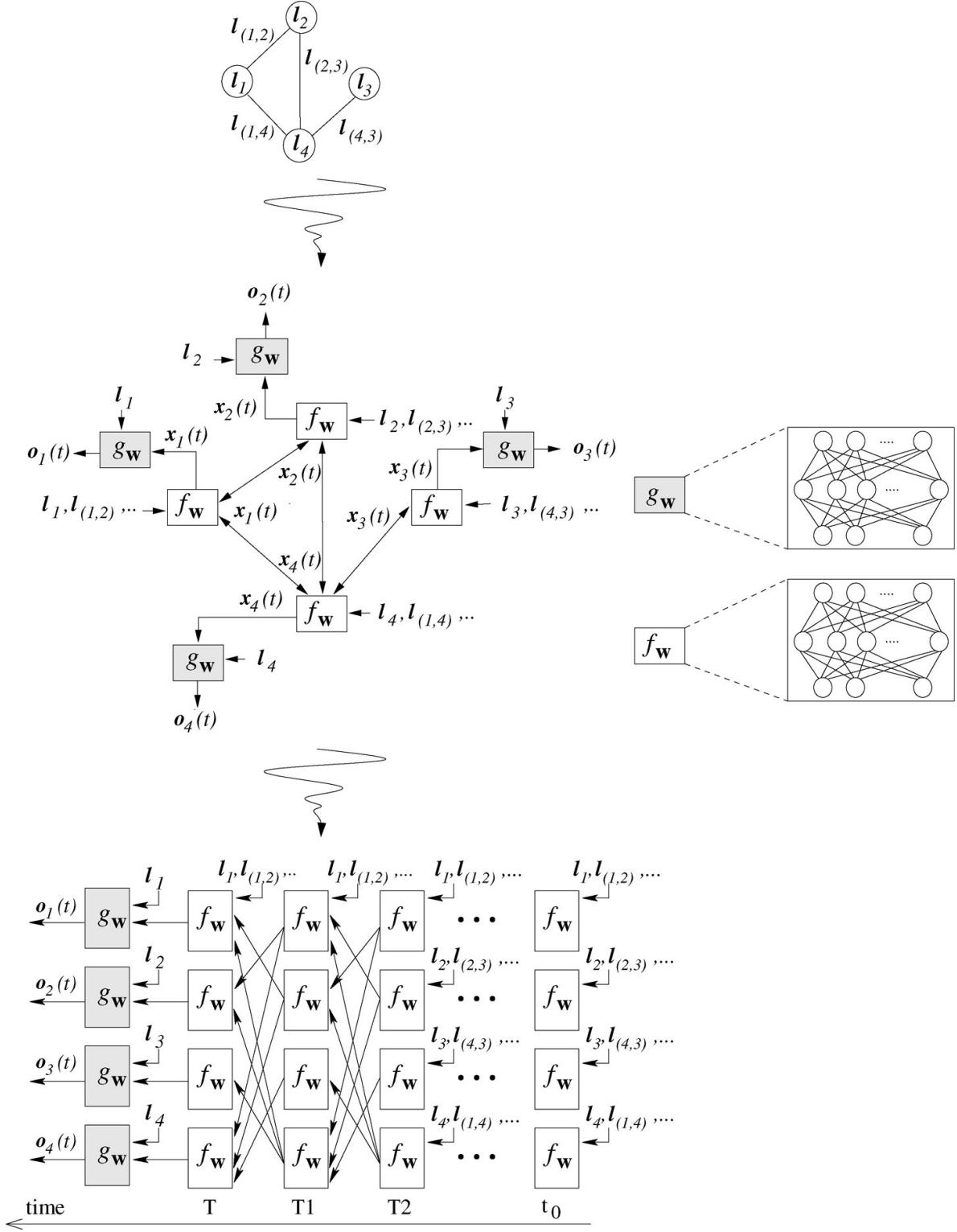
根据Banach的不动点理论，假设 $F_w$ 是一个压缩映射函数，那么式子(37)有唯一不动点解，而且可以通过迭代方式逼近该不动点

$$\mathbf{x}(t+1) = F_w(\mathbf{x}(t), \mathbf{l}) \quad (38)$$

其中， $\mathbf{x}(t)$ 表示 $\mathbf{x}$ 在第 $t$ 个迭代时刻的值，对于任意初值，迭代的误差是以指数速度减小的，使用迭代的形式写出状态和输出的更新表达式为

$$\begin{aligned} \boldsymbol{x}_n(t+1) &= f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{\text{co}[n]}, \boldsymbol{x}_{\text{ne}[n]}(t), \boldsymbol{l}_{\text{ne}[n]}) \\ \boldsymbol{o}_n(t) &= g_{\boldsymbol{w}}(\boldsymbol{x}_n(t), \boldsymbol{l}_n), \quad n \in \mathcal{N} \end{aligned}$$

GNN的信息传播流图以及等效的网络结构如下图所示



根据上图所示，顶端的图是原始的Graph，中间的图表示状态向量和输出向量的计算流图，最下面的图表示将更新流程迭代T次，并展开之后得到等效网络图。

## 学习算法

GNN的学习就是估计参数 $\boldsymbol{w}$ ，使得函数 $\varphi_{\boldsymbol{w}}$ 能够近似估计训练集

$$\mathcal{L} = \{(\mathbf{G}_i, n_{i,j}, \mathbf{t}_{i,j}) \mid \mathbf{G}_i = (\mathbf{N}_i, \mathbf{E}_i) \in \mathcal{G}; n_{i,j} \in \mathbf{N}_i; \mathbf{t}_{i,j} \in \mathbb{R}^m, 1 \leq i \leq p, 1 \leq j \leq q_i\} \quad (39)$$

其中,  $q_i$  表示在图  $G_i$  中监督学习的节点个数, 对于 graph-focused 的任务, 需要增加一个特殊的节点, 该节点用来作为目标节点, 这样, graph-focused 任务和 node-focused 任务都能统一到节点预测任务上, 学习目标可以是最小化如下二次损失函数

$$e_{\mathbf{w}} = \sum_{i=1}^p \sum_{j=1}^{q_i} (\mathbf{t}_{i,j} - \varphi_{\mathbf{w}}(\mathbf{G}_i, n_{i,j}))^2 \quad (40)$$

优化算法基于随机梯度下降的策略, 优化步骤按照如下几步进行

- 按照迭代方程迭代  $T$  次得到  $\mathbf{x}_n(t)$ , 此时接近不动点解:  $\mathbf{x}(T) \approx \mathbf{x}$
- 计算参数权重的梯度  $\partial e_{\mathbf{w}}(T)/\partial \mathbf{w}$
- 使用该梯度来更新权重  $\mathbf{w}$

这里假设函数  $F_{\mathbf{w}}$  是压缩映射函数, 保证最终能够收敛到不动点。另外, 这里的梯度的计算使用 *backpropagation-through-time algorithm*。

为了表明前面的方法是可行的, 论文接着证明了两个结论

理论1(可微性): 令  $F_{\mathbf{w}}$  和  $G_{\mathbf{w}}$  分别是 global transition function 和 global output function, 如果  $F_{\mathbf{w}}(\mathbf{x}, \mathbf{l})$  和  $G_{\mathbf{w}}(\mathbf{x}, \mathbf{l}_N)$  对于  $\mathbf{x}$  和  $\mathbf{w}$  是连续可微的, 那么  $\varphi_{\mathbf{w}}$  对  $\mathbf{w}$  也是连续可微的。

理论2(反向传播): 令  $F_{\mathbf{w}}$  和  $G_{\mathbf{w}}$  分别是 global transition function 和 global output function, 如果  $F_{\mathbf{w}}(\mathbf{x}, \mathbf{l})$  和  $G_{\mathbf{w}}(\mathbf{x}, \mathbf{l}_N)$  对于  $\mathbf{x}$  和  $\mathbf{w}$  是连续可微的。令  $\mathbf{z}(t)$  定义为

$$\mathbf{z}(t) = \mathbf{z}(t+1) \cdot \frac{\partial F_{\mathbf{w}}}{\partial \mathbf{x}}(\mathbf{x}, \mathbf{l}) + \frac{\partial e_{\mathbf{w}}}{\partial o} \cdot \frac{\partial G_{\mathbf{w}}}{\partial \mathbf{x}}(\mathbf{x}, \mathbf{l}_N) \quad (41)$$

那么, 序列  $\mathbf{z}(T), \mathbf{z}(T-1), \dots$  收敛到一个向量,  $\mathbf{z} = \lim_{t \rightarrow -\infty} \mathbf{z}(t)$ , 并且收敛速度为指数级收敛以及与初值  $\mathbf{z}(T)$  无关, 另外, 还存在

$$\frac{\partial e_{\mathbf{w}}}{\partial \mathbf{w}} = \frac{\partial e_{\mathbf{w}}}{\partial o} \cdot \frac{\partial G_{\mathbf{w}}}{\partial \mathbf{w}}(\mathbf{x}, \mathbf{l}_N) + \mathbf{z} \cdot \frac{\partial F_{\mathbf{w}}}{\partial \mathbf{w}}(\mathbf{x}, \mathbf{l}) \quad (42)$$

其中,  $\mathbf{x}$  是 GNN 的稳定状态。

算法流程如下

---

MAIN

    initialize  $w$ ;  
     $x = \text{Forward}(w)$ ;  
    **repeat**  
         $\frac{\partial e_w}{\partial w} = \text{BACKWARD}(x, w)$ ;  
         $w = w - \lambda \cdot \frac{\partial e_w}{\partial w}$ ;  
         $x = \text{FORWARD}(w)$ ;  
    **until** (a stopping criterion);  
    **return**  $w$ ;  
**end**

FORWARD( $w$ )

    initialize  $x(0)$ ,  $t = 0$ ;  
    **repeat**  
         $x(t+1) = F_w(x(t), l)$ ;  
         $t = t + 1$ ;  
    **until**  $\|x(t) - x(t-1)\| \leq \varepsilon_f$   
    **return**  $x(t)$ ;  
**end**

BACKWARD( $x, w$ )

$o = G_w(x, l_N)$ ;  
     $A = \frac{\partial F_w}{\partial x}(x, l)$ ;  
     $b = \frac{\partial e_w}{\partial o} \cdot \frac{\partial G_w}{\partial x}(x, l_N)$ ;  
    initialize  $z(0)$ ,  $t=0$ ;  
    **repeat**  
         $z(t) = z(t+1) \cdot A + b$ ;  
         $t = t + 1$ ;  
    **until**  $\|z(t-1) - z(t)\| \leq \varepsilon_b$ ;  
     $c = \frac{\partial e_w}{\partial o} \cdot \frac{\partial G_w}{\partial w}(x, l_N)$ ;  
     $d = z(t) \cdot \frac{\partial F_w}{\partial w}(x, l)$ ;  
     $\frac{\partial e_w}{\partial w} = c + d$ ;  
    **return**  $\frac{\partial e_w}{\partial w}$ ;  
**end**

---

FORWARD用于迭代计算出收敛点，BACKWARD用于计算梯度。

## Transition和Output函数实现

在GNN中，函数 $g_w$ 不需要满足特定的约束，直接使用多层前馈神经网络，对于函数 $f_w$ ，则需要着重考虑，因为 $f_w$ 需要满足压缩映射的条件，而且与不动点计算相关。下面提出两种神经网络和不同的策略来满足这些需求

### 1. Linear(nonpositional) GNN:

对于节点 $n$ 状态的计算，将方程(36)中的 $f_w$ 改成如下形式

$$\mathbf{x}_n = \sum_{u \in \text{ne}[n]} h_w(\mathbf{l}_n, \mathbf{l}_{(n,u)}, \mathbf{x}_u, \mathbf{l}_u), \quad n \in N \quad (43)$$

相当于是对节点 $n$ 的每一个邻居节点使用 $h_w$ ，并将得到的值求和来作为节点 $n$ 的状态。

由此，对上式中的函数 $h_w$ 按照如下方式实现

$$h_w(\mathbf{l}_n, \mathbf{l}_{(n,u)}, \mathbf{x}_u, \mathbf{l}_u) = \mathbf{A}_{n,u} \mathbf{x}_u + \mathbf{b}_n \quad (44)$$

其中，向量 $\mathbf{b}_n \in \mathbb{R}^s$ ，矩阵 $\mathbf{A}_{n,u} \in \mathbb{R}^{s \times s}$ 定义为两个前向神经网络的输出。更确切地说，令产生矩阵 $\mathbf{A}_{n,u}$ 的网络为*transition network*，产生向量 $\mathbf{b}_n$ 的网络为*forcing network*

*transition network*表示为 $\phi_w$

$$\phi_w : \mathbb{R}^{2l_N + l_E} \rightarrow \mathbb{R}^{s^2} \quad (45)$$

*forcing network*表示为 $\rho_w$

$$\rho_w : \mathbb{R}^{l_N} \rightarrow \mathbb{R}^s \quad (46)$$

由此，可以定义 $\mathbf{A}_{n,u}$ 和 $\mathbf{b}_n$

$$\begin{aligned} \mathbf{A}_{n,u} &= \frac{\mu}{s|\text{ne}[u]|} \cdot \Xi \\ \mathbf{b}_w &= \rho_w(\mathbf{l}_n) \end{aligned}$$

其中， $\mu \in (0, 1)$ ， $\Xi = \text{resize}(\phi_w(\mathbf{l}_n, \mathbf{l}_{(n,u)}, \mathbf{l}_u))$ ， $\text{resize}(\cdot)$ 表示将 $s^2$ 维的向量整理(reshape)成 $s \times s$ 的矩阵，也就是说，将*transition network*的输出整理成方形矩阵，然后乘以一个系数就得到 $\mathbf{A}_{n,u}$ 。 $\mathbf{b}_n$ 就是*forcing network*的输出。

在这里，假定 $\|\phi_w(\mathbf{l}_n, \mathbf{l}_{(n,u)}, \mathbf{l}_u)\|_1 \leq s$ ，这个可以通过设定*transition function*的激活函数来满足，比如设定激活函数为 $\tanh()$ 。在这种情况下， $F_w(\mathbf{x}, \mathbf{l}) = \mathbf{A}\mathbf{x} + \mathbf{b}$ ， $\mathbf{A}$ 和 $\mathbf{b}$ 分别是 $\mathbf{A}_{n,u}$ 的块矩阵形式和 $\mathbf{b}_n$ 的堆叠形式，通过简单的代数运算可得

$$\begin{aligned} \left\| \frac{\partial F_w}{\partial \mathbf{x}} \right\|_1 &= \|\mathbf{A}\|_1 \leq \max_{u \in N} \left( \sum_{n \in \text{ne}[u]} \|\mathbf{A}_{n,u}\|_1 \right) \\ &\leq \max_{u \in N} \left( \frac{\mu}{s|\text{ne}[u]|} \cdot \sum_{n \in \text{ne}[u]} \|\Xi\|_1 \right) \leq \mu \end{aligned}$$

该式表示 $F_w$ 对于任意的参数 $w$ 是一个压缩映射。

矩阵 $M$ 的1-norm定义为

$$\|M\|_1 = \max_j \sum_i |m_{i,j}| \quad (47)$$

2. **Nonlinear(nonpositional) GNN**: 在这个结构中， $h_w$ 通过多层前馈网络实现，但是，并不是所有的参数 $w$ 都会被使用，因为同样需要保证 $F_w$ 是一个压缩映射函数，这个可以通过惩罚项来实现

$$e_w = \sum_{i=1}^p \sum_{j=1}^{q_i} (\mathbf{t}_{i,j} - \varphi_w(\mathbf{G}_i, n_{i,j}))^2 + \beta L \left( \left\| \frac{\partial F_w}{\partial \mathbf{x}} \right\| \right) \quad (48)$$

其中，惩罚项 $L(y)$ 在 $y > \mu$ 时为 $(y - \mu)^2$ ，在 $y \leq \mu$ 时为0，参数 $\mu \in (0, 1)$ 定义为希望的 $F_w$ 的压缩系数。

## 实验结果

论文将GNN模型在三个任务上进行了实验：子图匹配(subgraph matching)任务，诱变(mutagenesis)任务和网页排序(web page ranking)任务。在这些任务上使用linear和nonlinear的模型测试，其中nonlinear模型中的激活函数使用sigmoid函数。

子图匹配任务为在一个大图 $G$ 上找到给定的子图 $S$ (标记出属于子图的节点), 也就是说, 函数 $\tau$ 必须学习到, 如果 $n_{i,j}$ 属于子图 $G$ , 那么 $\tau(G_i, n_{i,j}) = 1$ , 否则,  $\tau(G_i, n_{i,j}) = -1$ 。实验结果中, nonlinear模型的效果要好于linear模型的效果, 两个模型都要比FNN模型效果更好。

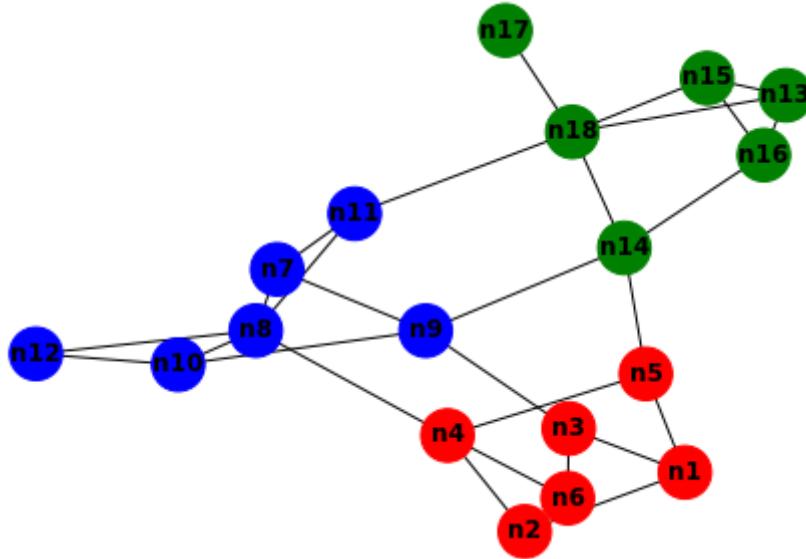
诱变问题任务是对化学分子进行分类, 识别出诱变化合物, 采用二分类方法。实验结果是nonlinear效果较好, 但不是最好。

网页排序任务是学会网页排序。实验表明虽然训练集只包含50个网页, 但是仍然没有产生过拟合的现象。

## 模型实现

在模拟的节点分类任务上实现该论文的GNN模型。

- **任务要求**为输入一个graph, 该graph的所有节点都有标签, 然后对部分节点进行训练, 在验证阶段使用另外一部分节点进行验证。输入的数据图如下图:



其中, 该graph总共有18个节点, 分别是 $\{n_1, n_2, \dots, n_{18}\}$ , 不同颜色的节点表示不同的节点类别。模拟的问题中节点类别有三类, 分别用 $\{0, 1, 2\}$ 表示, 在训练阶段, 使用节点 $\{n_1, n_2, n_3, n_7, n_8, n_9, n_{13}, n_{14}, n_{15}\}$ 进行训练, 相当于每一类取出三个节点训练, 其余的节点用于在验证阶段进行验证。

输入的数据为 `(node, label)` 列表和 `(node1, node2)` 边列表, 表示如下

```

1 # (node, label)集
2 N = [("n{}".format(i), 0) for i in range(1,7)] + \
3     [("n{}".format(i), 1) for i in range(7,13)] + \
4     [("n{}".format(i), 2) for i in range(13,19)]
5 # 边集
6 E = [( "n1", "n2"), ( "n1", "n3"), ( "n1", "n5"),
7        ( "n2", "n4"),
8        ( "n3", "n6"), ( "n3", "n9"),
9        ( "n4", "n5"), ( "n4", "n6"), ( "n4", "n8"),
10       ( "n5", "n14"),
11       ( "n7", "n8"), ( "n7", "n9"), ( "n7", "n11"),
12       ( "n8", "n10"), ( "n8", "n11"), ( "n8", "n12"),
13       ( "n9", "n10"), ( "n9", "n14"),
14       ( "n10", "n12"),
15       ( "n11", "n18"),
16       ( "n13", "n15"), ( "n13", "n16"), ( "n13", "n18"),
17       ( "n14", "n16"), ( "n14", "n18"),
18       ( "n15", "n16"), ( "n15", "n18"),

```

```
19 |     ("n17", "n18"))]
```

$N$ 为节点集合， $E$ 为边集合。

- 模型部分使用论文的linear函数来设计 $f_w$ 和 $g_w$ ，而且，这两个函数在graph所有的节点上进行共享。模型部分实现了 $\Xi$ ， $\rho$ 函数，以及完整的forward传播部分，如下简化代码：

```
1 # 实现Xi函数，输入一个batch的相邻节点特征向量对ln，返回是s*s的A矩阵
2 # ln是特征向量维度，s为状态向量维度
3 # Input : (N, 2*ln)
4 # Output : (N, S, S)
5 class Xi(nn.Module):
6     def __init__(self, ln, s):
7         ...
8     def forward(self, X):
9         ...
10
11 # 实现Rou函数
12 # Input : (N, ln)
13 # Output : (N, S)
14 class Rou(nn.Module):
15     def __init__(self, ln, s):
16         ...
17     def forward(self, X):
18         ...
19
20 # 实现Hw函数
21 # Input : (N, 2 * ln)
22 #           每一行都是一个节点特征向量和该节点的某一个邻接向量concat
23 #           得到的向量
24 # Input : (N, s)
25 #           对应中心节点的状态向量
26 # Input : (N, )
27 #           对应中心节点的度的向量
28 # Output : (N, s)
29 class Hw(nn.Module):
30     def __init__(self, ln, s, mu=0.9):
31         ...
32     def forward(self, X, H, dg_list):
33         ...
34
35 class AggrSum(nn.Module):
36     def __init__(self, node_num):
37         ...
38
39     def forward(self, H, X_node):
40         ...
41
42 # 实现GNN模型
43 class OriLinearGNN(nn.Module):
44     def __init__(self, node_num, feat_dim, stat_dim, T):
45         ...
46     # Input :
47     #   X_Node : (N, )
48     #   X_Neis : (N, )
49     #   H      : (N, s)
50     #   dg_list: (N, )
```

```

51     def forward(self, X_Node, X_Neis, dg_list):
52         ...
53         for t in range(self.T):
54             # (V, s) -> (N, s)
55             H = torch.index_select(self.node_states, 0, X_Node)
56             # (N, s) -> (N, s)
57             H = self.Hw(X, H, dg_list)
58             # (N, s) -> (V, s)
59             self.node_states = self.Aggr(H, X_Node)
60             #
61             print(H[1])
62         ...

```

可以看出，在模型训练阶段，每次forward，都会直接循环计算T次 $f_w$ 函数计算不动点，然后再计算output。

- 模型训练部分按照常规的分类模型进行训练，采用Adam优化器，学习率保持为0.01，权重衰减为0.01，使用交叉熵作为损失函数，模型训练部分代码如下

```

1  # 用于计算accuracy
2  def CalAccuracy(output, label):
3      ...
4
5  # 开始训练模型
6  def train(node_list, edge_list, label_list, T,
7  ndict_path="./node_dict.json"):
8      # 生成node-index字典
9      ...
10
11     # 现在需要生成两个向量
12     # 第一个向量类似于
13     # [0, 0, 0, 1, 1, ..., 18, 18]
14     # 其中的值表示节点的索引，连续相同索引的个数为该节点的度
15     # 第二个向量类似于
16     # [1, 2, 4, 1, 4, ..., 11, 13]
17     # 与第一个向量一一对应，表示第一个向量节点的邻居节点
18
19     # 首先统计得到节点的度
20     ...
21
22     # 然后生成两个向量
23     ...
24     # 生成度向量
25     ...
26     # 准备训练集和测试集
27     train_node_list = [0,1,2,6,7,8,12,13,14]
28     train_node_label = [0,0,0,1,1,1,2,2,2]
29     test_node_list = [3,4,5,9,10,11,15,16,17]
30     test_node_label = [0,0,0,1,1,1,2,2,2]
31
32     # 开始训练
33     model = OriLinearGNN(node_num=len(node_list),
34                           feat_dim=2,
35                           stat_dim=2,
36                           T=T)
37     optimizer = torch.optim.Adam(model.parameters(), lr=0.01,
38                                 weight_decay=0.01)
39     criterion = nn.CrossEntropyLoss(size_average=True)

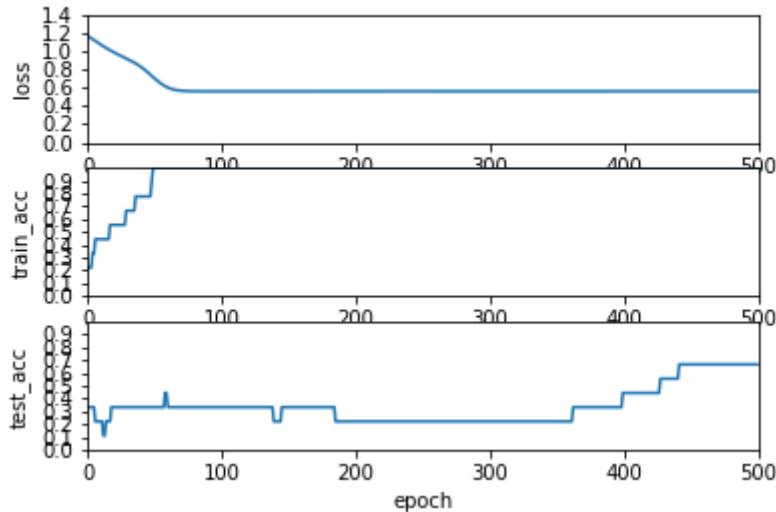
```

```

38
39     min_loss = float('inf')
40     node_inds_tensor = Variable(torch.Tensor(node_inds).long())
41     node_neis_tensor = Variable(torch.Tensor(node_neis).long())
42     train_label = Variable(torch.Tensor(train_node_label).long())
43     for ep in range(500):
44         # 运行模型得到结果
45         res = model(node_inds_tensor, node_neis_tensor, dg_list) # (V,
46         3)
46         train_res = torch.index_select(res, 0,
47             torch.Tensor(train_node_list).long())
47         test_res = torch.index_select(res, 0,
48             torch.Tensor(test_node_list).long())
49         loss = criterion(input=train_res,
50                           target=train_label)
51         loss_val = loss.item()
52         train_acc = CalAccuracy(train_res.cpu().detach().numpy(),
53             np.array(train_node_label))
54         test_acc = CalAccuracy(test_res.cpu().detach().numpy(),
55             np.array(test_node_label))
56         # 更新梯度
57         optimizer.zero_grad()
58         loss.backward(retain_graph=True)
59         optimizer.step()
60
61         if loss_val < min_loss:
62             min_loss = loss_val
63         print("==> [Epoch {}] : loss {:.4f}, min_loss {:.4f}, train_acc
64             {:.3f}, test_acc {:.3f}".format(ep, loss_val, min_loss, train_acc,
65             test_acc))

```

- 模型的训练和评估结果如下图



第一条曲线为训练loss曲线，第二条曲线为训练的acc曲线，第三条曲线为评估的acc曲线，可以看出，训练的loss很快达到了最低0.56左右，而准确率达到了1.0，基本已经过拟合，而验证集的准确率一直很低，最高在第500个epoch处上升到0.667，约为 $\frac{2}{3}$ 左右。

## Graph Neural Network

- 论文：《Semi-Supervised Classification with Graph Convolutional Networks》

## 图卷积的演变

按照图傅里叶变换的性质，可以得到如下图卷积的定义

$$(\mathbf{f} * \mathbf{h})_{\mathcal{G}} = \Phi \operatorname{diag}[\hat{h}(\lambda_1), \dots, \hat{h}(\lambda_n)] \Phi^T \mathbf{f} \quad (49)$$

其中

- 对于图 $\mathbf{f}$ 的傅里叶变换为 $\hat{\mathbf{f}} = \Phi^T \mathbf{f}$
- 对于卷积核的图傅里叶变换： $\hat{\mathbf{h}} = (\hat{h}_1, \dots, \hat{h}_n)$ , 其中

$$\hat{h}_k = \langle h, \phi_k \rangle, k = 1, 2, \dots, n \quad (50)$$

按照矩阵形式就是 $\hat{\mathbf{h}} = \Phi^T \mathbf{h}$

- 对两者的傅里叶变换向量 $\hat{\mathbf{f}} \in \mathbb{R}^{N \times 1}$ 和 $\hat{\mathbf{h}} \in \mathbb{R}^{N \times 1}$ 求element-wise乘积，等价于将 $\mathbf{h}$ 组织成对角矩阵，即 $\operatorname{diag}[\hat{h}(\lambda_k)] \in \mathbb{R}^{N \times N}$ ，然后再求 $\operatorname{diag}[\hat{h}(\lambda_k)]$ 和 $\mathbf{f}$ 矩阵乘法。
- 求上述结果的傅里叶逆变换，即左乘 $\Phi$ 。

深度学习中的卷积就是要设计**trainable**的卷积核，从公式(49)可以看出，就是要设计 $\operatorname{diag}[\hat{h}(\lambda_1), \dots, \hat{h}(\lambda_n)]$ ，由此，可以直接将其变为卷积核 $\operatorname{diag}[\theta_1, \dots, \theta_n]$ ，而不需要再将卷积核进行傅里叶变换，由此，相当于直接将变换后的参数进行学习。

## 第一代GCN

第一代GCN为

$$\mathbf{y}_{\text{output}} = \sigma(\Phi \mathbf{g}_{\theta} \Phi^T \mathbf{x}) = \sigma(\Phi \operatorname{diag}[\theta_1, \dots, \theta_n] \Phi^T \mathbf{x}) \quad (51)$$

其中， $\mathbf{x}$ 就是graph上对应每个节点的feature构成的向量， $x = (x_1, x_2, \dots, x_n)$ ，这里暂时对每个节点都使用标量，然后经过激活之后，得到输出 $\mathbf{y}_{\text{output}}$ ，之后传入下一层。

### 第一代GCN也有一些缺点

- 需要对拉普拉斯矩阵进行谱分解来求 $\Phi$ ，在graph很大的时候复杂度很高。另外，还需要计算矩阵乘积，复杂度为 $O(n^2)$ 。
- 卷积核参数为 $n$ ，当graph很大的时候， $n$ 会很大。
- 卷积核的spatial localization不好。

## 第二代GCN

图傅里叶变换是关于特征值(相当于普通傅里叶变换的频率)的函数，也就是 $F(\lambda_1), \dots, F(\lambda_n)$ ，即 $F(\Lambda)$ ，因此，将卷积核 $\mathbf{g}_{\theta}$ 写成 $\mathbf{g}_{\theta}(\Lambda)$ ，然后，将 $\mathbf{g}_{\theta}(\Lambda)$ 定义为如下**k阶多项式**

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^K \theta'_k \Lambda^k \quad (52)$$

将卷积公式带入，可以得到

$$\begin{aligned}
g_{\theta'} * \mathbf{x} &\approx \Phi \sum_{k=0}^K \theta'_k \Lambda^k \Phi^T \mathbf{x} \\
&= \sum_{k=0}^K \theta'_k (\Phi \Lambda^k \Phi^T) \mathbf{x} \\
&= \sum_{k=0}^K \theta'_k (\Phi \Lambda \Phi^T)^k \mathbf{x} \\
&= \sum_{k=0}^K \theta'_k \mathbf{L}^k \mathbf{x}
\end{aligned}$$

可以看出，这一代的GCN不需要做特征分解了，可以直接对Laplacian矩阵做变换，通过事先将Laplacian矩阵求出来，以及 $\mathbf{L}^k$ 求出来，前向传播的时候，就可以直接使用，复杂度为 $O(Kn^2)$ 。

对于每一次Laplacian矩阵 $\mathbf{L}$ 和 $\mathbf{x}$ 相乘，对于节点 $n$ ，相当于从邻居节点 $ne[n]$ 传递一次信息给节点 $n$ ，由于连续乘以了 $k$ 次Laplacian矩阵，那么相当于 $n$ 节点的 $k$ -hop之内的节点能够传递信息给 $n$ ，因此，实际上只利用了节点的K-Localized信息。

另外，可以使用切比雪夫展开式来近似 $\mathbf{L}^k$ ，任何 $k$ 次多项式都可以使用切比雪夫展开式来近似，由此，引入切比雪夫多项式的 $K$ 阶截断获得 $\mathbf{L}^k$ 近似，从而获得对 $g_{\theta}(\Lambda)$ 的近似

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^K \theta'_k T_k(\tilde{\Lambda}) \quad (53)$$

其中， $\tilde{\Lambda} = \frac{2}{\lambda_{\max}} \Lambda - \mathbf{I}_n$ ， $\theta' \in \mathbb{R}^K$ 为切比雪夫向量， $\theta'_k$ 为第 $k$ 个分量，切比雪夫多项式 $T_k(x)$ 使用递归的方式进行定义： $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ ，其中， $T_0(x) = 1, T_1(x) = x$ 。

此时，带入到卷积公式

$$\begin{aligned}
g_{\theta'} * \mathbf{x} &\approx \Phi \sum_{k=0}^K \theta'_k T_k(\tilde{\Lambda}) \Phi^T \mathbf{x} \\
&\approx \sum_{k=0}^K \theta'_k (\Phi T_k(\tilde{\Lambda}) \Phi^T) \mathbf{x} \\
&= \sum_{k=0}^K \theta'_k T_k(\tilde{\mathbf{L}}) \mathbf{x}
\end{aligned}$$

其中， $\tilde{\mathbf{L}} = \frac{2}{\lambda_{\max}} \mathbf{L} - \mathbf{I}_n$ 。

因此，可以得到输出为

$$\mathbf{y}_{\text{output}} = \sigma \left( \sum_{k=0}^K \theta'_k T_k(\tilde{\mathbf{L}}) \mathbf{x} \right) \quad (54)$$

### 第三代GCN

这一代GCN直接取切比雪夫多项式中 $K = 1$ ，此时模型是1阶近似

将 $K = 1, \lambda_{\max} = 2$ 带入可以得到

$$\begin{aligned}
g_{\theta'} * \mathbf{x} &\approx \theta'_0 \mathbf{x} + \theta'_1 (\mathbf{L} - \mathbf{I}_n) \mathbf{x} \\
&= \theta'_0 \mathbf{x} + \theta'_1 (\mathbf{L} - \mathbf{I}_n) \mathbf{x} \\
&= \theta'_0 \mathbf{x} - \theta'_1 (\mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}) \mathbf{x}
\end{aligned}$$

其中，归一化拉普拉斯矩阵 $\mathbf{L} = \mathbf{D}^{-1/2} (\mathbf{D} - \mathbf{W}) \mathbf{D}^{-1/2} = \mathbf{I}_n - \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$ 。为了进一步简化，令 $\theta'_0 = -\theta'_1$ ，此时只含有一个参数 $\theta$

$$g_{\theta'} * x = \theta \left( I_n + D^{-1/2} W D^{-1/2} \right) x \quad (55)$$

由于  $I_n + D^{-1/2} W D^{-1/2}$  的谱半径  $[0, 2]$  太大，使用归一化的 trick

$$I_n + D^{-1/2} W D^{-1/2} \rightarrow \tilde{D}^{-1/2} \tilde{W} \tilde{D}^{-1/2} \quad (56)$$

其中， $\tilde{W} = W + I_n$ ,  $\tilde{D}_{ij} = \Sigma_j \tilde{W}_{ij} \circ$

由此，带入卷积公式

$$\underbrace{g_{\theta'} * x}_{\mathbb{R}^{n \times 1}} = \theta \left( \underbrace{\tilde{D}^{-1/2} \tilde{W} \tilde{D}^{-1/2}}_{\mathbb{R}^{n \times n}} \right) \underbrace{x}_{\mathbb{R}^{n \times 1}} \quad (57)$$

如果推广到多通道，相当于每一个节点的信息是向量

$$x \in \mathbb{R}^{N \times 1} \rightarrow X \in \mathbb{R}^{N \times C} \quad (58)$$

其中， $N$  是节点数量， $C$  是通道数，或者称作表示节点的信息维度数。 $X$  是节点的特征矩阵。

相应的卷积核参数变化

$$\theta \in \mathbb{R} \rightarrow \Theta \in \mathbb{R}^{C \times F} \quad (59)$$

其中， $F$  为卷积核数量。

那么卷积结果写成矩阵形式为

$$\underbrace{Z}_{\mathbb{R}^{N \times F}} = \underbrace{\tilde{D}^{-1/2} \tilde{W} \tilde{D}^{-1/2}}_{\mathbb{R}^{N \times N}} \underbrace{X}_{\mathbb{R}^{N \times C}} \underbrace{\Theta}_{\mathbb{R}^{C \times F}} \quad (60)$$

上述操作可以叠加多层，对上述输出激活一下，就可以作为下一层节点的特征矩阵。

这一代GCN特点：

- 取  $K = 1$ ，相当于直接取邻域信息，类似于  $3 \times 3$  的卷积核。
- 由于卷积核宽度减小，可以通过增加卷积层数来扩大感受野，从而增强网络的表达能力。
- 增加了参数约束，比如  $\lambda_{\max} \approx 2$ ，引入归一化操作。

## 论文模型

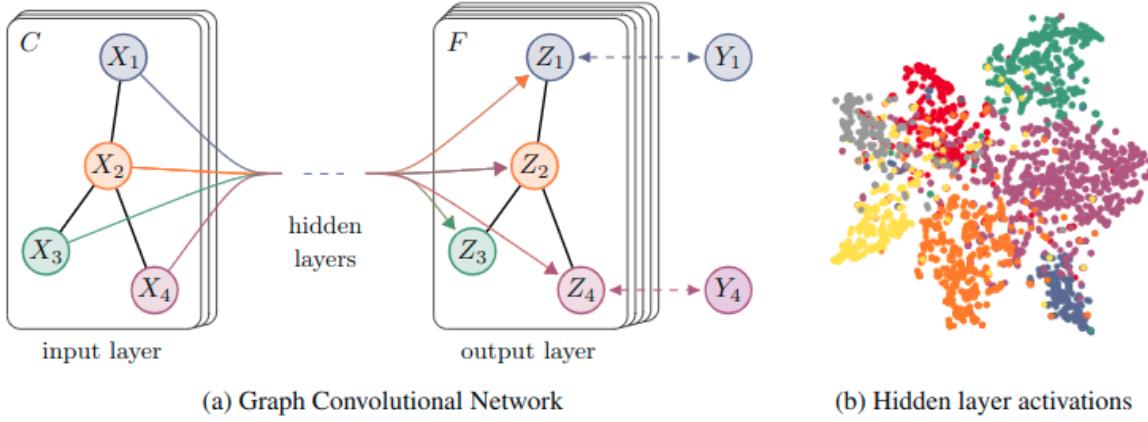
论文采用两层的GCN，用来在graph上进行半监督的节点分类任务，邻接矩阵为  $A$ ，首先计算出  $\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ ，由此，前向网络模型形式如下

$$Z = f(X, A) = \text{softmax} \left( \hat{A} \text{ReLU} \left( \hat{A} X W^{(0)} \right) W^{(1)} \right) \quad (61)$$

其中， $W^{(0)} \in \mathbb{R}^{C \times H}$  为输入层到隐藏层的权重矩阵，隐藏层的特征维度为  $H$ ， $W^{(1)} \in \mathbb{R}^{H \times F}$  为隐藏层到输出层的权重矩阵，softmax激活函数定义为  $\text{softmax}(x_i) = \frac{1}{Z} \exp(x_i)$ ， $Z = \sum_i \exp(x_i)$ ，相当于对每一列做softmax，由此，得到交叉熵损失函数为

$$\mathcal{L} = - \sum_{l \in \mathcal{Y}_L} \sum_{f=1}^F Y_{lf} \ln Z_{lf} \quad (62)$$

其中， $\mathcal{Y}_L$  为带有标签的节点集合。



上图中左图为GCN图示，输入为 $C$ 个通道，输出为 $F$ 个通道， $Y_1$ 和 $Y_2$ 为节点标签。右图为在一数据集上进行训练得到的隐藏层激活值经过t-SNE降维可视化后的结果，可以看出聚类效果较好。

## 实验结果

论文在如下几个任务中进行实验

- 在citation network中进行半监督的document classification。
- 在从knowledge graph中提取的bipartite graph中进行半监督的entity classification

实验数据说明如下

<b>Dataset</b>	<b>Type</b>	<b>Nodes</b>	<b>Edges</b>	<b>Classes</b>	<b>Features</b>	<b>Label rate</b>
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

前三个Dataset是citation network数据集，节点表示文档，边表示引用的连接，label rate表示用来有监督训练的节点数量占总节点数量比例，第四个Dataset是bipartite graph数据集。

结果如下

<b>Method</b>	<b>Citeseer</b>	<b>Cora</b>	<b>Pubmed</b>	<b>NELL</b>
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
<b>GCN (this paper)</b>	<b>70.3 (7s)</b>	<b>81.5 (4s)</b>	<b>79.0 (38s)</b>	<b>66.0 (48s)</b>
GCN (rand. splits)	$67.9 \pm 0.5$	$80.1 \pm 0.5$	$78.9 \pm 0.7$	$58.4 \pm 1.7$

可以看出，在比较的几种算法中，论文GCN的在准确率和时间上都最好。

## DCNN

- 论文：《Diffusion-Convolutional Neural Networks》

## 模型亮点

该模型对每一个节点(或边、或图)采用H个hop的矩阵进行表示，每一个hop都表示该邻近范围的邻近信息，由此，对于局部信息的获取效果比较好，得到的节点的representation的表示能力很强。

## 模型详述

假设有如下定义

- 一个graph数据集 $\mathcal{G} = \{G_t | t \in 1 \dots T\}$
- graph定义为 $G_t = (V_t, E_t)$ ，其中， $V_t$ 为节点集合， $E_t$ 为边集合
- 所有节点的特征矩阵定义为 $X_t$ ，大小为 $N_t \times F$ ，其中， $N_t$ 为图 $G_t$ 的节点个数， $F$ 为节点特征维度
- 边信息 $E_t$ 定义为 $N_t \times N_t$ 的邻接矩阵 $A_t$ ，由此可以计算出**节点度(degree)归一化**的转移概率矩阵 $P_t$ ，表示从*i*节点转移到*j*节点的概率。

对于graph来说没有任何限制，graph可以是带权重的或不带权重的，有向的或无向的。

模型的目标为预测 $Y$ ，也就是预测每一个图的节点标签，或者边的标签，或者每一个图的标签，在每一种情况中，模型输入部分带有标签的数据集合，然后预测剩下的数据的标签。

DCNN模型输入图 $\mathcal{G}$ ，返回硬分类预测值 $Y$ 或者条件分布概率 $\mathbb{P}(Y|X)$ 。该模型将每一个预测的目标对象(节点、边或图)转化为一个diffusion-convolutional representation，大小为 $H \times F$ ， $H$ 表示扩散的hops。因此，对于节点分类任务，图 $t$ 的confusion-convolutional representation为大小为 $N_t \times H \times F$ 的张量，表示为 $Z_t$ ，对于图分类任务，张量 $Z_t$ 为大小为 $H \times F$ 的矩阵，对于边分类任务，张量 $Z_t$ 为大小为 $M_t \times H \times F$ 的矩阵。示意图如下

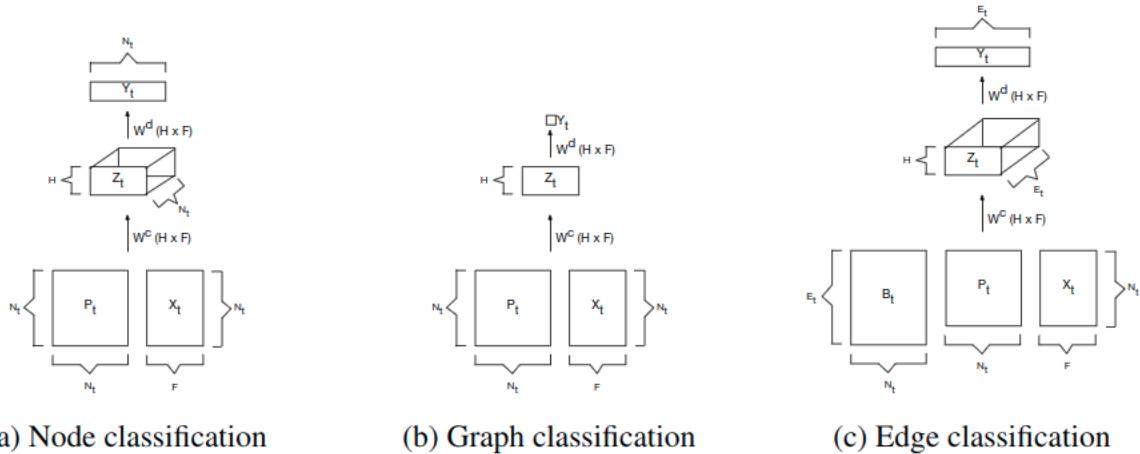


Figure 1: DCNN model definition for node, graph, and edge classification tasks.

对于**节点分类任务**，假设 $P_t^*$ 为 $P_t$ 的power series，大小为 $N_t \times H \times N_t$ ，那么对于图 $t$ 的节点*i*，第*k*个hop，第*k*维特征值 $Z_{tijk}$ 计算公式为

$$Z_{tijk} = f \left( W_{jk}^c \cdot \sum_{l=1}^{N_t} P_{tijl}^* X_{tlk} \right) \quad (63)$$

使用矩阵表示为

$$Z_t = f(W^c \odot P_t^* X_t) \quad (64)$$

其中 $\odot$ 表示element-wise multiplication，由于模型只考虑 $H$ 跳的参数，即参数量为 $O(H \times F)$ ，使得**diffusion-convolutional representation不受输入大小的限制**。

在计算出 $Z$ 之后，过一层全连接得到输出 $Y$ ，使用 $\hat{Y}$ 表示硬分类预测结果，使用 $\mathbb{P}(Y|X)$ 表示预测概率，计算方式如下

$$\hat{Y} = \arg \max (f(W^d \odot Z)) \quad (65)$$

$$\mathbb{P}(Y|X) = \text{softmax}(f(W^d \odot Z)) \quad (66)$$

对于图分类任务，直接采用所有节点表示的均值作为graph的representation

$$Z_t = f(W^c \odot \mathbf{1}_{N_t}^T P_t^* X_t / N_t) \quad (67)$$

其中， $\mathbf{1}_{N_t}$ 是全为1的 $N_t \times 1$ 的向量。

对于边分类任务，通过将每一条边转化为一个节点来进行训练和预测，这个节点与原来的边对应的首尾节点相连，转化后的图的邻接矩阵 $A'_t$ 可以直接从原来的邻接矩阵 $A_t$ 增加一个**incidence matrix**得到

$$A'_t = \begin{pmatrix} A_t & B_t^T \\ B_t & 0 \end{pmatrix} \quad (68)$$

之后，使用 $A'_t$ 来计算 $P'_t$ ，并用来替换 $P_t$ 来进行分类。

对于模型训练，使用梯度下降法，并采用early-stop方式得到最终模型。

## 实验结果

节点分类任务的实验数据集使用Cora和Pubmed数据集，包含scientific papers(相当于node)、citations(相当于edge)和subjects(相当于label)。实验评估标准使用分类准确率以及F1值。

节点分类的实验结果如下

Model	Cora			Pubmed		
	Accuracy	F (micro)	F (macro)	Accuracy	F (micro)	F (macro)
l1logistic	0.7087	0.7087	0.6829	0.8718	0.8718	0.8698
l2logistic	0.7292	0.7292	0.7013	0.8631	0.8631	0.8614
KED	0.8044	0.8044	0.7928	0.8125	0.8125	0.7978
KLED	0.8229	0.8229	0.8117	0.8228	0.8228	0.8086
CRF-LBP	0.8449	—	0.8248	—	—	—
2-hop DCNN	<b>0.8677</b>	<b>0.8677</b>	<b>0.8584</b>	<b>0.8976</b>	<b>0.8976</b>	<b>0.8943</b>

Table 1: A comparison of the performance between baseline  $\ell_1$  and  $\ell_2$ -regularized logistic regression models, exponential diffusion and Laplacian exponential diffusion kernel models, loopy belief propagation (LBP) on a partially-observed conditional random field (CRF), and a two-hop DCNN on the Cora and Pubmed datasets. The DCNN offers the best performance according to each measure, and the gain is statistically significant in each case. The CRF-LBP result is quoted from [3], which follows the same experimental protocol.

可以看出使用各种评估标准，DCNN效果都是最好的。

图分类任务的实验结果如下

Model	NCI1			NCI109		
	Accuracy	F (micro)	F (macro)	Accuracy	F (micro)	F (macro)
l1logistic	0.5728	0.5728	0.5711	0.5555	0.5555	0.5411
l2logistic	0.5688	0.5688	0.5641	0.5586	0.5568	0.5402
deepwl	0.6215	<b>0.6215</b>	0.5821	0.5801	0.5801	0.5178
2-hop DCNN	0.6250	0.5807	0.5807	0.6275	0.5884	0.5884
5-hop DCNN	<b>0.6261</b>	0.5898	<b>0.5898</b>	<b>0.6286</b>	<b>0.5950</b>	<b>0.5899</b>
Model	MUTAG			PTC		
	Accuracy	F (micro)	F (macro)	Accuracy	F (micro)	F (macro)
l1logistic	<b>0.7190</b>	0.7190	0.6405	0.5470	0.5470	0.4272
l2logistic	0.7016	0.7016	0.5795	0.5565	<b>0.5565</b>	<b>0.4460</b>
deepwl	0.6563	0.6563	0.5942	0.5113	0.5113	0.4444
2-hop DCNN	0.6635	0.7975	0.79747	<b>0.5660</b>	0.0500	0.0531
5-hop DCNN	0.6698	<b>0.8013</b>	<b>0.8013</b>	0.5530	0.0	0.0526
Model	ENZYMES					
	Accuracy	F (micro)	F (macro)			
l1logistic	0.1640	0.1640	0.0904			
l2logistic	0.2030	0.2030	0.1110			
deepwl	<b>0.2155</b>	<b>0.2155</b>	<b>0.1431</b>			
2-hop DCNN	0.1590	0.1590	0.0809			
5-hop DCNN	0.1810	0.1810	0.0991			

Table 2: A comparison of the performance between baseline methods and two and five-hop DCNNs on several graph classification datasets.

可以看出，在不同数据集上，DCNN在图分类任务上并没有明显表现出很好的效果。

## 优缺点

优点：

- 节点分类准确率很高
- 灵活性
- 快速

缺点：

- **内存占用大**：DCNN建立在密集的张量计算上，需要存储大量的张量，需要 $O(N_t^2 H)$ 的空间复杂度。
- **长距离信息传播不足**：模型对于局部的信息获取较好，但是远距离的信息传播不足。

## Tree-LSTM

### 论文亮点

将序列型的LSTM模型扩展到树型的LSTM模型，简称Tree-LSTM，并根据孩子节点是否有序，论文提出了两个模型变体，**Child-Sum Tree-LSTM模型**和**N-ary Tree-LSTM模型**。和序列型的LSTM模型的主要不同点在于，序列型的LSTM从前一时刻获取隐藏状态 $h_t$ ，而树型的LSTM从其所有的孩子节点获取隐藏状态。

### 模型详解

Tree-LSTM模型对于每一个孩子节点都会产生一个“遗忘门” $f_{jk}$ ，这个使得模型能够从所有的孩子节点选择性地获取信息和结合信息。

#### Child-Sum Tree-LSTMs

该模型的更新方程如下

$$\begin{aligned}
\tilde{h}_j &= \sum_{k \in C(j)} h_k \\
i_j &= \sigma \left( W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)} \right) \\
f_{jk} &= \sigma \left( W^{(f)} x_j + U^{(f)} h_k + b^{(f)} \right) \\
o_j &= \sigma \left( W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)} \right) \\
u_j &= \tanh \left( W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)} \right) \\
c_j &= i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k \\
h_j &= o_j \odot \tanh(c_j)
\end{aligned}$$

其中， $C(j)$ 表示 $j$ 节点的邻居节点的个数， $h_k$ 表示节点 $k$ 的隐藏状态， $i_j$ 表示节点 $j$ 的“输入门”， $f_{jk}$ 表示节点 $j$ 的邻居节点 $k$ 的“遗忘门”， $o_j$ 表示节点 $j$ 的“输出门”。

这里的关键点在于第三个公式的 $f_{jk}$ ，这个模型对节点 $j$ 的每个邻居节点 $k$ 都计算了对应的“遗忘门”向量，然后在第六行中计算 $c_j$ 时对邻居节点的信息进行“遗忘”和组合。

由于该模型是对所有的孩子节点求和，所以**这个模型对于节点顺序不敏感的**，适合于孩子节点无序的情况。

## N-ary Tree-LSTMs

假如一个树的最大分支数为 $N$ (即孩子节点最多为 $N$ 个)，而且孩子节点是有序的，对于节点 $j$ ，对于该节点的第 $k$ 个孩子节点的隐藏状态和记忆单元分别用 $h_{jk}$ 和 $c_{jk}$ 表示。模型的方程如下

$$\begin{aligned}
i_j &= \sigma \left( W^{(i)} x_j + \sum_{\ell=1}^N U_{\ell}^{(i)} h_{j\ell} + b^{(i)} \right) \\
f_{jk} &= \sigma \left( W^{(f)} x_j + \sum_{\ell=1}^N U_{k\ell}^{(f)} h_{j\ell} + b^{(f)} \right) \\
o_j &= \sigma \left( W^{(o)} x_j + \sum_{\ell=1}^N U_{\ell}^{(o)} h_{j\ell} + b^{(o)} \right) \\
u_j &= \tanh \left( W^{(u)} x_j + \sum_{\ell=1}^N U_{\ell}^{(u)} h_{j\ell} + b^{(u)} \right) \\
c_j &= i_j \odot u_j + \sum_{\ell=1}^N f_{j\ell} \odot c_{j\ell} \\
h_j &= o_j \odot \tanh(c_j)
\end{aligned}$$

值得注意的是该模型为每个孩子节点都单独地设置了参数 $U_l$ 。

## 模型训练

### 分类任务

分类任务定义为在类别集 $\mathcal{Y}$ 中预测出正确的标签 $\hat{y}$ ，对于每一个节点 $j$ ，使用一个softmax分类器来预测节点标签 $\hat{y}_j$ ，分类器取每个节点的隐藏状态 $h_j$ 作为输入

$$\begin{aligned}
\hat{p}_{\theta}(y|\{x\}_j) &= \text{softmax}\left(W^{(s)} h_j + b^{(s)}\right) \\
\hat{y}_j &= \arg \max_y \hat{p}_{\theta}(y|\{x\}_j)
\end{aligned}$$

损失函数使用negative log-likelihood

$$J(\theta) = -\frac{1}{m} \sum_{k=1}^m \log \hat{p}_\theta \left( y^{(k)} | \{x\}^{(k)} \right) + \frac{\lambda}{2} \|\theta\|_2^2 \quad (69)$$

其中， $m$ 是带有标签的节点数量， $\lambda$ 是L2正则化超参数。

## 语义相关性任务

该任务给定一个句子对(sentence pair)，模型需要预测出一个范围在 $[1, K]$ 之间的实数值，这个值越高，表示相似度越高。

论文首先对每一个句子产生一个representation，两个句子的表示分别用 $h_L$ 和 $h_R$ 表示，得到这两个representation之后，从distance和angle两个方面考虑，使用神经网络来得到 $(h_L, h_R)$ 相似度：

$$\begin{aligned} h_\times &= h_L \odot h_R \\ h_+ &= |h_L - h_R| \\ h_s &= \sigma \left( W^{(\times)} h_\times + W^{(+)} h_+ + b^{(h)} \right) \\ \hat{p}_\theta &= \text{softmax} \left( W^{(p)} h_s + b^{(p)} \right) \\ \hat{y} &= r^T \hat{p}_\theta \end{aligned}$$

其中， $r^T = [1 \ 2 \ \dots \ K]$ 。模型期望根据训练得到的参数 $\theta$ 得到的结果： $\hat{y} = r^T \hat{p}_\theta \approx y_0$ 。由此，定义一个目标分布 $p$

$$p_i = \begin{cases} y - \lfloor y \rfloor, & i = \lfloor y \rfloor + 1 \\ \lfloor y \rfloor - y + 1, & i = \lfloor y \rfloor \\ 0 & \text{otherwise} \end{cases} \quad (70)$$

其中， $1 \leq i \leq K$ ，损失函数为 $p$ 和 $\hat{p}_\theta$ 之间的KL散度：

$$J(\theta) = \frac{1}{m} \sum_{k=1}^m \text{KL} \left( p^{(k)} \| \hat{p}_\theta^{(k)} \right) + \frac{\lambda}{2} \|\theta\|_2^2 \quad (71)$$

## 实验结果

对于分类任务，实验数据使用**Stanford Sentiment Treebank**，分为五类：very negative, negative, neutral, positive和very positive。在该数据集上的测试集上的准确度结果如下：

Method	Fine-grained	Binary
RAE (Socher et al., 2013)	43.2	82.4
MV-RNN (Socher et al., 2013)	44.4	82.9
RNTN (Socher et al., 2013)	45.7	85.4
DCNN (Blunsom et al., 2014)	48.5	86.8
Paragraph-Vec (Le and Mikolov, 2014)	48.7	87.8
CNN-non-static (Kim, 2014)	48.0	87.2
CNN-multichannel (Kim, 2014)	47.4	<b>88.1</b>
DRNN (Irsoy and Cardie, 2014)	49.8	86.6
LSTM	46.4 (1.1)	84.9 (0.6)
Bidirectional LSTM	49.1 (1.0)	87.5 (0.5)
2-layer LSTM	46.0 (1.3)	86.3 (0.6)
2-layer Bidirectional LSTM	48.5 (1.0)	87.2 (1.0)
Dependency Tree-LSTM	48.4 (0.4)	85.7 (0.4)
Constituency Tree-LSTM		
– randomly initialized vectors	43.9 (0.6)	82.0 (0.5)
– Glove vectors, fixed	49.7 (0.4)	87.5 (0.8)
– Glove vectors, tuned	<b>51.0</b> (0.5)	88.0 (0.3)

**Table 2:** Test set accuracies on the Stanford Sentiment Treebank. For our experiments, we report mean accuracies over 5 runs (standard deviations in parentheses). **Fine-grained:** 5-class sentiment classification. **Binary:** positive/negative sentiment classification.

对于语义相似度度量，模型的任务是预测出两个句子语义的相似度分数。在SICK的语义相似度子任务上的测试结果如下

Method	Pearson's $r$	Spearman's $\rho$	MSE
Illinois-LH (Lai and Hockenmaier, 2014)	0.7993	0.7538	0.3692
UNAL-NLP (Jimenez et al., 2014)	0.8070	0.7489	0.3550
Meaning Factory (Bjerva et al., 2014)	0.8268	0.7721	0.3224
ECNU (Zhao et al., 2014)	0.8414	–	–
Mean vectors	0.7577 (0.0013)	0.6738 (0.0027)	0.4557 (0.0090)
DT-RNN (Socher et al., 2014)	0.7923 (0.0070)	0.7319 (0.0071)	0.3822 (0.0137)
SDT-RNN (Socher et al., 2014)	0.7900 (0.0042)	0.7304 (0.0076)	0.3848 (0.0074)
LSTM	0.8528 (0.0031)	0.7911 (0.0059)	0.2831 (0.0092)
Bidirectional LSTM	0.8567 (0.0028)	0.7966 (0.0053)	0.2736 (0.0063)
2-layer LSTM	0.8515 (0.0066)	0.7896 (0.0088)	0.2838 (0.0150)
2-layer Bidirectional LSTM	0.8558 (0.0014)	0.7965 (0.0018)	0.2762 (0.0020)
Constituency Tree-LSTM	0.8582 (0.0038)	0.7966 (0.0053)	0.2734 (0.0108)
Dependency Tree-LSTM	<b>0.8676</b> (0.0030)	<b>0.8083</b> (0.0042)	<b>0.2532</b> (0.0052)

**Table 3:** Test set results on the SICK semantic relatedness subtask. For our experiments, we report mean scores over 5 runs (standard deviations in parentheses). Results are grouped as follows: (1) SemEval 2014 submissions; (2) Our own baselines; (3) Sequential LSTMs; (4) Tree-structured LSTMs.

# 附录

## 关于不动点定理

- [参考知乎](#)

**不动点：**函数的不动点或定点指的是被这个函数映射到其自身的一个点，即 $\xi = f(\xi)$ 。

**压缩映射：**设 $f$ 在区间 $[a, b]$ 上定义， $f([a, b]) \subset [a, b]$ ，并存在一个常数 $k$ ，满足 $0 < k < 1$ ，使得对一切 $x, y \in [a, b]$ 都成立不等式 $|f(x) - f(y)| \leq k|x - y|$ ，则称 $f$ 是 $[a, b]$ 上的一个压缩映射，称常数 $k$ 为压缩常数。

**压缩映射原理：**设 $f$ 是 $[a, b]$ 上的一个压缩映射，则 $f$ 在 $[a, b]$ 中存在唯一的不动点 $\xi = f(\xi)$ ，由任何初值 $a_0 \in [a, b]$ 和递推公式 $a_{n+1} = f(a_n), n \in N_+$ ，生成的数列 $\{a_n\}$ 一定收敛于 $\xi$ 。

## 关于递归求导

前面的关于GNN的内容里面提到梯度的求解效率并不高，这里解释说明如下：

假设存在如下递归的式子，和式子(4)相同

$$\mathbf{H}^{t+1} = F(\mathbf{H}^t, \mathbf{X}) \quad (72)$$

函数 $F$ 的参数使用 $W$ 表示，现在输入 $\mathbf{H}^0$ 和 $\mathbf{X}$ ，那么按照时序递归往前计算，有

$$\begin{aligned}\mathbf{H}^1 &= F(\mathbf{H}^0, \mathbf{X}) \\ \mathbf{H}^2 &= F(\mathbf{H}^1, \mathbf{X}) \\ &\dots \\ \mathbf{H}^T &= F(\mathbf{H}^{T-1}, \mathbf{X})\end{aligned}$$

现在，需要求 $\mathbf{H}^T$ 关于 $W$ 的偏导数，按照链式法则

$$\begin{aligned}\frac{\partial \mathbf{H}^T(W)}{\partial W} &= \frac{\partial F(\mathbf{H}^{T-1}(W), W; X)}{\partial W} \\ &= \frac{\partial F(\mathbf{H}^{T-1}, W; X)}{\partial W} + \frac{\partial F(\mathbf{H}^{T-1}(W), W; X)}{\partial \mathbf{H}^{T-1}(W)} \cdot \frac{\partial \mathbf{H}^{T-1}(W)}{\partial W}\end{aligned}$$

在此说明一下，公式第一行等号右边将函数写成 $F(\mathbf{H}^{T-1}(W), W; X)$ 的原因是，写在分号前面的量为与 $W$ 相关的“函数”，之后需要使用多元微分来求偏微分，分号后面的 $X$ 由于一直保持不变，所以可以暂时当做常量看待。公式第二行第一项中的 $\mathbf{H}^{T-1}$ 是前向计算的第 $T-1$ 时刻的 $\mathbf{H}$ 的值，不是关于 $W$ 的函数，第二项中的 $W$ 是 $W$ 在 $T-1$ 时刻的值，而不是变量。

可以看出，公式最后出现了 $\frac{\partial \mathbf{H}^{T-1}(W)}{\partial W}$ ，和公式左边形式差别在于 $T-1$ ，因此，按照这样的法则可以一直递归求下去得到最终的 $\frac{\partial \mathbf{H}^T(W)}{\partial W}$ ，但是在求的过程中，必须要先按照顺序依次计算出 $\mathbf{H}^1 \dots \mathbf{H}^{T-1}$ ，然后再逆序地递归求 $\frac{\partial \mathbf{H}^{T-1}(W)}{\partial W} \dots \frac{\partial \mathbf{H}^1(W)}{\partial W}$ ，显然，这个求梯度的过程必须要在完成 $T$ 次函数 $F$ 运算之后才能进行，而且还需要递归求解，显然效率比较低。

## 关于切比雪夫多项式

- [参考知乎](#)

## 关于图Fourier变换

- [参考知乎](#)
- [参考博客](#)

根据卷积原理，卷积公式可以写成

$$f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\} \quad (73)$$

正、逆Fourier变换

$$\mathcal{F}(v) = \int_{\mathbb{R}} f(x) e^{-2\pi i x \cdot v} dx \quad (74)$$

$$f(x) = \int_{\mathbb{R}} \mathcal{F}(v) e^{2\pi i x \cdot v} dv \quad (75)$$

一阶导数定义

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \quad (76)$$

拉普拉斯相当于二阶导数

$$\Delta f(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \quad (77)$$

在graph上，定义一阶导数为

$$f'_{*g}(x) = f(x) - f(y) \quad (78)$$

对应的拉普拉斯算子定义为

$$\Delta_{*g} f'(x) = \sum_{y \sim x} (f(x) - f(y)) \quad (79)$$

假设 $D$ 为 $N \times N$ 的度矩阵(degree matrix)

$$D(i, j) = \begin{cases} d_i & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \quad (80)$$

$A$ 为 $N \times N$ 的邻接矩阵(adjacency matrix)

$$A(i, j) = \begin{cases} 1 & \text{if } x_i \sim x_j \\ 0 & \text{otherwise} \end{cases} \quad (81)$$

那么图上的Laplacian算子可以写成

$$L = D - A \quad (82)$$

标准化后得到

$$L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \quad (83)$$

定义Laplacian算子的目的是为了找到Fourier变换的基。

传统Fourier变换的基就是Laplacian算子的一组特征向量

$$\Delta e^{2\pi i x \cdot v} = \lambda e^{2\pi i x \cdot v} \quad (84)$$

类似的，在graph上，有

$$\Delta f = (D - A)f = Lf \quad (85)$$

图拉普拉斯算子作用在由图节点信息构成的向量 $f$ 上得到的结果等于图拉普拉斯矩阵和向量 $f$ 的点积。

那么graph上的Fourier基就是 $L$ 矩阵的 $n$ 个特征向量 $U = [u_1 \dots u_n]$ ， $L$ 可以分解成 $L = U \Lambda U^T$ ，其中， $\Lambda$ 是特征值组成的对角矩阵。

传统的Fourier变换与graph的Fourier变换区别

	传统Fourier变换	Graph Fourier变换
Fourier变换基	$e^{-2\pi i xv}$	$U^T$
逆Fourier变换基	$e^{2\pi i xv}$	$U$
维度	$\infty$	点的个数 $n$

将  $f(i)$  看成是第  $i$  个点上的 signal，用向量  $x = (f(1) \dots f(n)) \in \mathbb{R}^n$  来表示。矩阵形式的 graph 的 Fourier 变换为

$$\mathcal{G}\mathcal{F}\{x\} = U^T x \quad (86)$$

类似的 graph 上的 Fourier 逆变换为

$$\mathcal{I}\mathcal{G}\mathcal{F}\{x\} = Ux \quad (87)$$

## 关于RNN系列模型

- [参考知乎](#)
- [参考简书](#)

**RNN的处理对象：**RNN通常处理的是**序列输入(sequence)**，比如一个句子，或者一段视频。

句子中的前后相邻的词汇是相互联系的，而且是有序的，在理解一段句子时，需要按照顺序排列整个序列，与此类似，视频中每一帧之间也是相互联系的，而且是有序的，理解一段视频需要有序的播放。

### RNN相关任务

- **文本分类：**输入一段句子，模型预测该句子所属的类别

比如有三类文本，类别为 {军事类, 医疗类, 科技类}，模型输入一段句子，并将该句子归类到这三类中的一类，比如 某某公司新出了一种治疗感染的特效药 属于 医疗类。

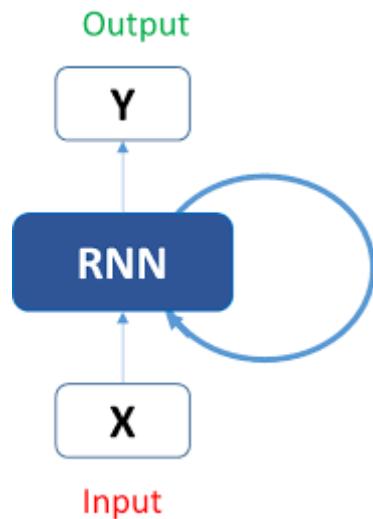
- **词性预测：**输入一段句子，模型预测该句子每个词汇的词性

比如输入一句话 我吃苹果，那么模型的输出为 我(nn) 吃(v) 苹果(nn)。

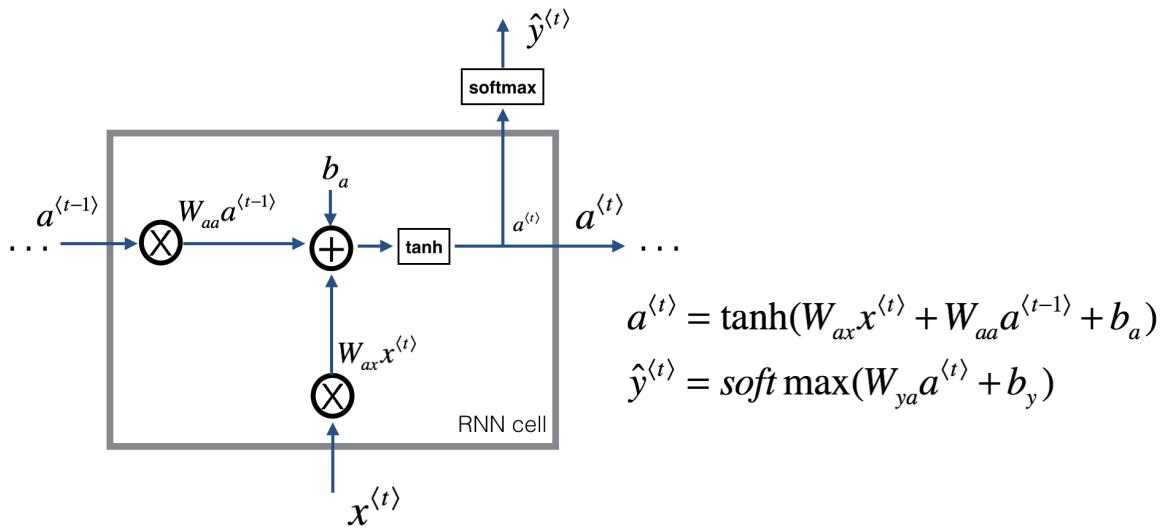
**RNN和普通前馈神经网络比较：**以上述任务为例，如果使用普通神经网络，可以将输入的句子作为特征向量直接进行分类，但是，这样做忽略了**句子中词汇之间的关联性**，比如在 吃 这个动词后，很有可能接着是一个名词，而不太可能是一个动词，这种有序关系的建模使用普通的网络较难处理。

## 基础RNN模型

基础的RNN模型示意图如下

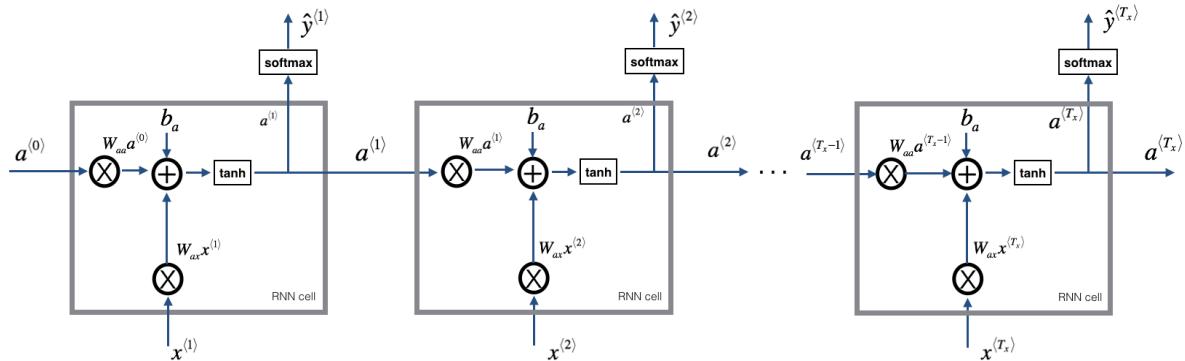


其中的蓝色框**RNN**表示一个RNN的cell，在这个cell里面进行向量运算的操作，**X**表示输入的向量值，**Y**表示输出的向量值，自连接的弧线表示cell输出的值会作为下一时刻的输入。将上图中RNN的cell操作进行展开，得到如下图

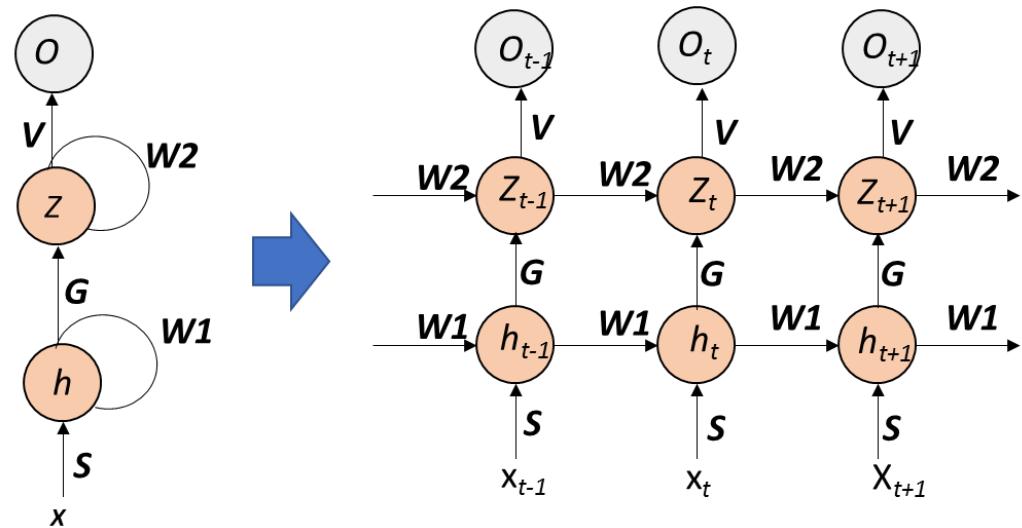


其中的 $a$ 就是RNN的cell的自连接弧的输出或输入。

如果将RNN的cell按照时序进行展开，得到如下图



可以看出，在每一个时刻，输入为该时刻的输入 $x^{<t>}$ 以及上一时刻的cell的输出 $a^{<t-1>}$ ，输出为 $y^{<t>}$ 和 $a^{<t>}$ 。展开后实际上相当于为一层网络层，可以通过叠加两个RNN的cell，来增强模型的表达能力，即让第一个RNN的cell的输出作为第二个RNN的cell的输入，示意图如下



a) 2-layer Recurrent Neural Network (RNN)

b) Unfolded 2-layer Recurrent Neural Network (RNN)

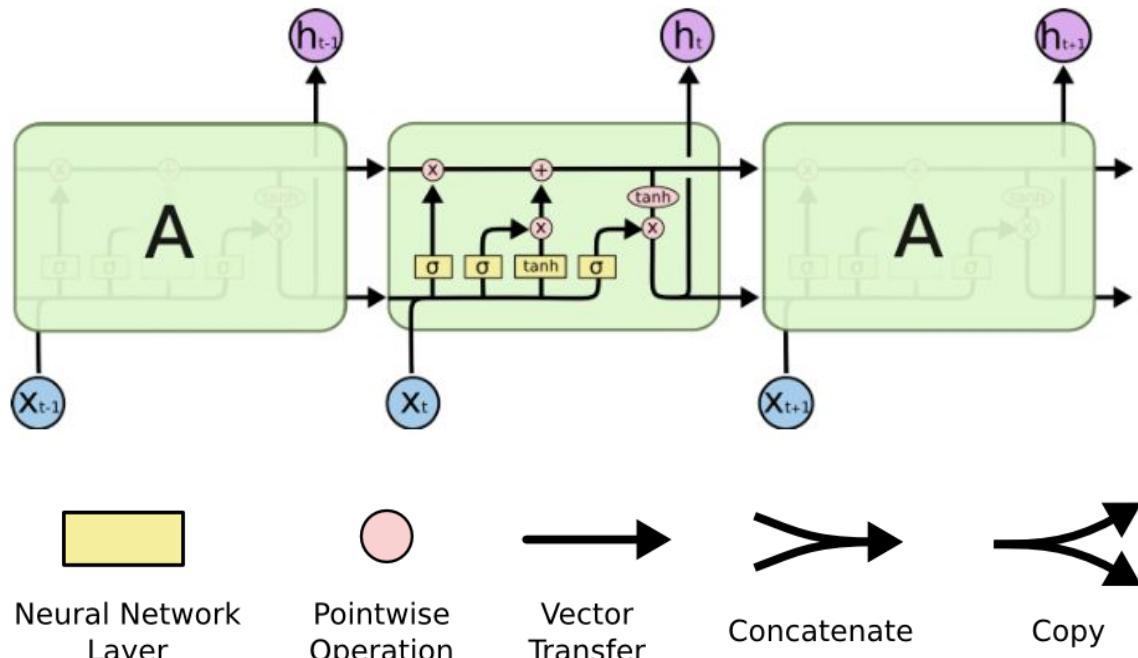
根据上图可以看出，第一层的输出 $G$ 作为第二层的输入，网络的输出为第二层的输出 $V$ 。

**RNN缺点：**原始的RNN在训练过程中可能会产生梯度爆炸或消失([可以参见](#))，因此，需要对训练过程采取一些措施，比如梯度裁剪，将梯度强制限制在饱和梯度范围内。

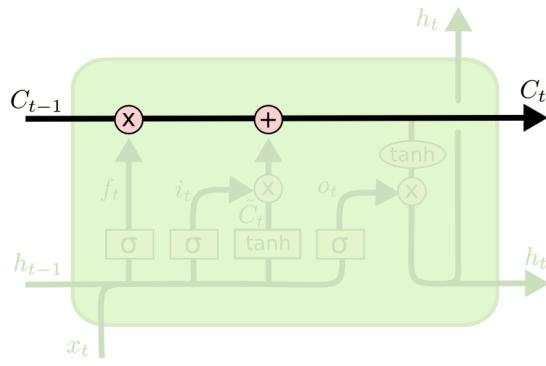
## LSTM模型

根据前面RNN的模型可以看出，RNN的关键点之一是可以将先前的信息使用到后面的时刻，对于短句子，比比如 `the cloud in the [sky]`，预测的关键词 `sky` 和前文的距离较短，RNN可以学会使用先前的信息，但是，对于长句子，比如 `I grew up in France...I speak fluent [French]`，如果我们需要模型预测出应该是哪一种语言，由于 `French` 和前文的距离很长，那么模型需要考虑位置很远的相关信息。然而，对于较长的句子，RNN会丧失连接如此远信息的能力。

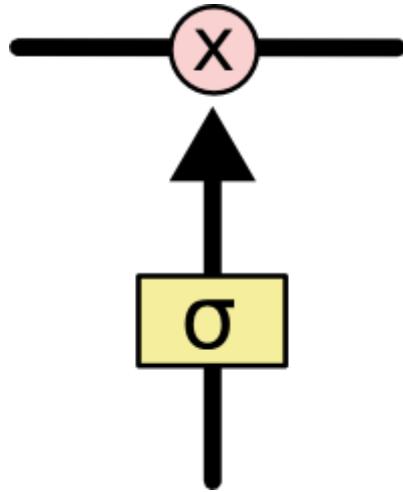
LSTM(Long Short Term Memory)网络，是一种RNN的变体，可以学习到长期的依赖信息，LSTM通过刻意的设计来避免长期依赖问题，使得LSTM结构本身就具有记住长期信息的能力。LSTM的结构示意图如下



**LSTM关键在于cell的状态**，这个状态类似于传送带，在图上方的水平线上贯穿运行，如下图

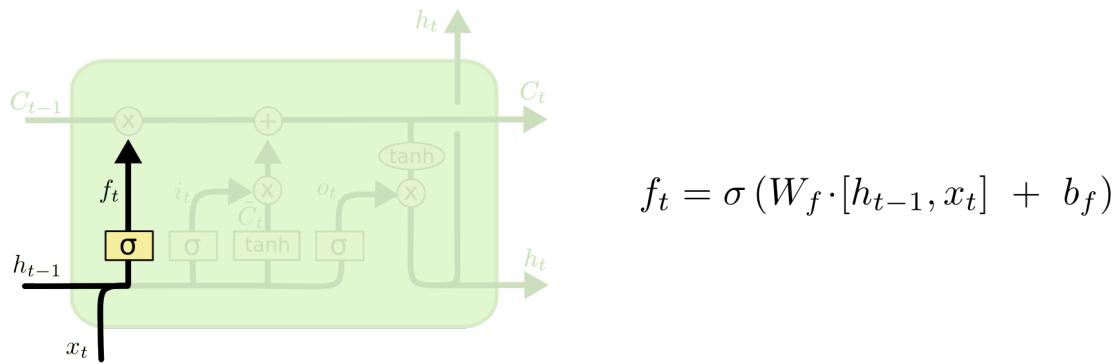


LSTM通过精心设计的gate(门)的结构来实现去除或者增加信息到细胞状态，门就是一种让信息选择式通过的方法，包含一个sigmoid层和一个按位的乘法操作，sigmoid层输出0到1之间的值，描述每个部分有多少量可以通过。门机制结构如下示意图

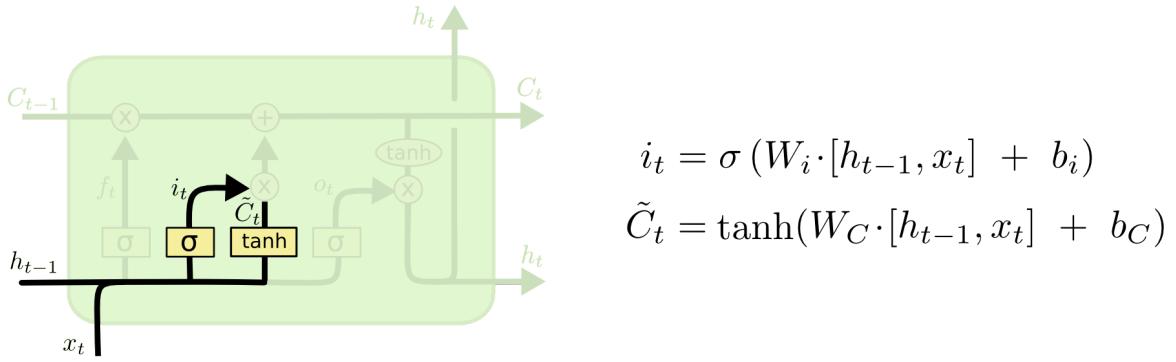


LSTM有三个门，用于控制信息的流动。

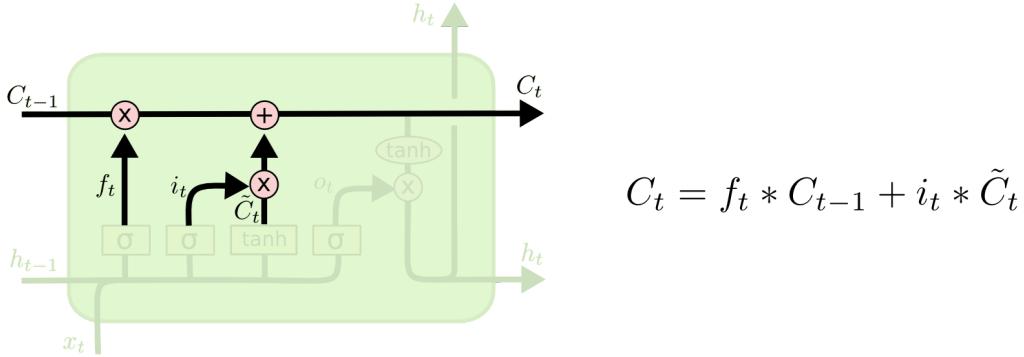
LSTM第一步是决定我们会从细胞状态中丢弃或保留哪些信息。这个决定通过遗忘门完成，这个门输入 $\mathbf{h}_{t-1}$ 和 $\mathbf{x}_t$ ，输出一个在0到1之间的数值(向量)，然后与上一时刻的细胞状态 $\mathbf{C}_{t-1}$ 的按元素相乘。如下图



下一步是确定什么样的新信息被存放在细胞状态中，包含两个部分，一个部分是sigmoid层为输入门，决定需要输入多少信息，另一部分是tanh层，用于创建该时刻的输入信息向量 $\mathbf{\tilde{C}}_t$ ，这个向量用于之后加入到细胞状态中。示意图如下

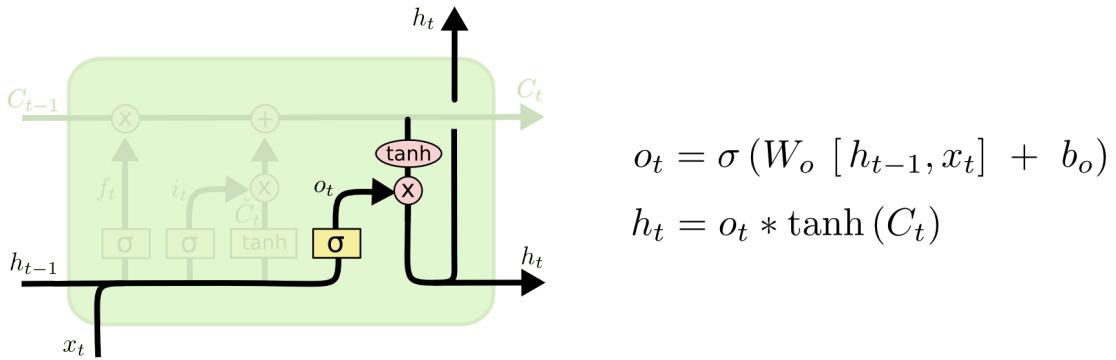


现在，需要更新旧的细胞状态 $C_{t-1}$ ，首先将旧状态与 $f_t$ 相乘，表示需要过滤哪些信息，然后和这一时刻输入的信息相加，表示将该时刻的信息加入到细胞状态，示意图如下



最后，我们需要确定输出什么值，这个输出基于我们当前时刻的细胞状态，但也是一个过滤后的信息，首先通过一个  $\text{tanh}$  层进行处理，得到输出信息的形式，然后通过一个  $\text{sigmoid}$  层来确定输出哪些信息。

由此，就可以得到最后的输出 $h_t$ ，如下示意图



通过不断进行以上过程，就可以将一个序列(比如句子)编码成最终的输出向量，然后通过这个输出向量经过全连接层进行分类或回归，另外，类似于RNN，通过叠加两个LSTM的cell来得到两层的LSTM，以得到更高级的句子representation。

因此，最终将所有的公式汇集在一起

$$\begin{aligned}
 i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
 f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
 o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
 u_t &= \tanh(W_u \cdot [h_{t-1}, x_t] + b_u) \\
 c_t &= i_t \odot u_t + f_t \odot c_{t-1} \\
 h_t &= o_t \odot \tanh(c_t)
 \end{aligned}$$

## RNN应用

RNN适合应用于**序列型输入**，每个时刻的输入的token需要是一个向量，通过取**每一时刻的输出或者最后一个时刻的输出**来得到句子的representation，用于文本相关的下游任务，比如文本分类、词性标注、机器翻译等等。

其中较为关键的是如何将序列的每一个**token**转化为向量输入，一般情况下有如下几种方法：

- **随机初始化**：在模型初始化阶段给每一个token随机初始化赋予一个向量，然后**在loss优化过程中使用随机梯度下降法进行学习**。
- **使用统计表征**：最简单的就是one-hot向量，假设所有token的总数为 $N$ ，一个token对应的向量为

$$E_t = [0, 0, \dots, 1, \dots, 0], E_t \in R^N \quad (88)$$

- **使用预训练词向量**：通过word2vec对训练预料进行预训练得到预训练词向量，相当于将每一个token转化成一个vector，使用这个vector作为RNN模型每一时刻的输入向量即可。