图学习背后的数学笔记

伍元凯 (https://kaimaoge.github.io)

发布时间：2023 年 9 月

目录

插图

第一章　傅里叶变化和卷积定理

傅里叶变换有时间再写

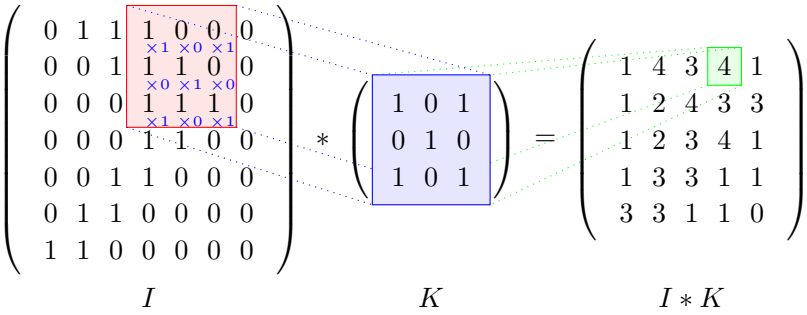

图 1.1: 2 维卷积可视化，卷积神经网络的基础构成

定义 1. 函数 f 和 g 是定义在 $\mathbb{R}^n$ 上的可测函数（*measurable function*），f 与 g 的卷积记作 $f*g$，它是其中一个函数反转（*reverse*），并平移后，与另一个函数的乘积的积分，是一个对平移量的函数，也就是：

$$(f*g)(t)=\int_{\mathbb{R}^n} f(\tau)g(t-\tau)\,d\tau.$$

卷积在神经网络中有着广泛的应用，图 1.1可视化了卷积神经网络中卷积的基本操作。卷积定理是傅立叶变换满足的一个重要性质。卷积定理指出，函数卷积的傅立叶变换是函数傅立叶变换的乘积。这里主要描述下离散信号傅里叶变换下的卷积定理及其证明。

现在用 $\mathcal{F}$ 表示离散时间傅立叶变换（DTFT）算子，存在一个类似的定理。考虑两个序列 $g[n]$ 和 $h[n]$，它们的变换分别为 G 和 H：

$$G(s)\triangleq\mathcal{F}\{g\}(s)=\sum_{n=-\infty}^{\infty} g[n]\cdot e^{-i2\pi sn},\quad s\in\mathbb{R}, \tag{1.1}$$

$$H(s)\triangleq\mathcal{F}\{h\}(s)=\sum_{n=-\infty}^{\infty} h[n]\cdot e^{-i2\pi sn},\quad s\in\mathbb{R}. \tag{1.2}$$

离散序列 g 和 h 的卷积定义为：

$$r[n]\triangleq(g*h)[n]=\sum_{m=-\infty}^{\infty} g[m]\cdot h[n-m]=\sum_{m=-\infty}^{\infty} g[n-m]\cdot h[m]. \tag{1.3}$$

离散序列的卷积定理为：

$$R(s)=\mathcal{F}\{g*h\}(s)=G(s)H(s). \tag{1.4}$$

证明. 首先，我们将卷积的定义应用到 $R(s)$ 中：

$$R(s) = \sum_{n=-\infty}^{\infty} r[n] \cdot e^{-i2\pi sn} = \sum_{n=-\infty}^{\infty} \left(\sum_{m=-\infty}^{\infty} g[n-m] \cdot h[m] \right) \cdot e^{-i2\pi sn}.$$

接下来，我们将求和符号交换顺序，将内部的两个求和分开：

$$R(s) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} g[n-m] \cdot h[m] \cdot e^{-i2\pi sn}.$$

然后，我们注意到 $g[n-m] \cdot e^{-i2\pi sn}$ 是 n 的函数，因此可以看作 n 的傅立叶变换 $G(s)$。因此，我们可以将上式写为：

$$R(s) = \sum_{m=-\infty}^{\infty} G(s) \cdot h[m].$$

最后，我们可以将 $G(s)$ 移出求和符号，得到：

$$R(s) = G(s) \cdot \left(\sum_{m=-\infty}^{\infty} h[m] \right) = G(s) \cdot H(s).$$

□

第二章　图拉普拉斯的理解，为什么它如此重要？

定义 2. 邻接矩阵 **A** 是一个 $N \times N$ 的矩阵，其中第 i 行和第 j 列的元素 a_{ij} 等于从 v_j 到 v_i 的边的权重，对于任意 i, j。如果对于所有的 i，$a_{ii} = 0$，则图中没有自环；而 $a_{ij} = a_{ji}$ 表示图是无向的。

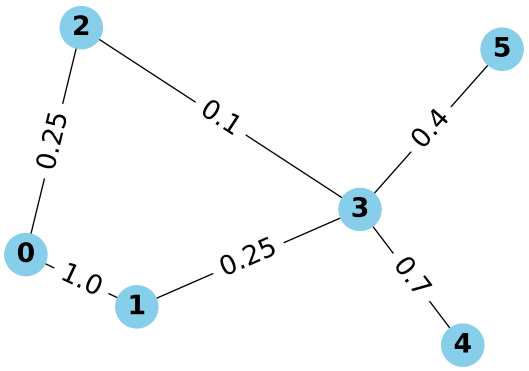

图 2.1: 图示例

图 2.1的邻接矩阵:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0.25 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0.25 & 0 & 0 \\ 0.25 & 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0.25 & 0.1 & 0 & 0.7 & 0.4 \\ 0 & 0 & 0 & 0.7 & 0 & 0 \\ 0 & 0 & 0 & 0.4 & 0 & 0 \end{bmatrix} \tag{2.1}$$

定义 3. 无向图的度矩阵 **D** 是一个 $N \times N$ 的对角矩阵，其中每个对角元素表示相应节点的度数（边的数量或边权值之和），即 $d_{ii} = \sum_j a_{ij}$，其中 a_{ij} 是矩阵 **A** 的第 i 行第 j 列的元素。如果图是有向的，则我们可以定义入度矩阵和出度矩阵，分别通过只考虑入边和出边的权值之和。

公式 2.1的度矩阵为

$$\mathbf{D} = \begin{bmatrix} 1.25 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.25 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.35 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.45 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.4 \end{bmatrix} \tag{2.2}$$

在介绍图拉普拉斯之前，先简要介绍下拉普拉斯的概念：在数学以及物理中，拉普拉斯算子或是拉普拉斯算符（英语：Laplace operator, Laplacian）是由欧几里得空间中的一个函数的梯度的散度给出的微分算子，通常写成 Δ、∇^2 或 $\nabla \cdot \nabla$。

这名字是为了纪念法国数学家皮耶-西蒙 · 拉普拉斯（1749–1827）而命名的。他在研究天体力学在数学中首次应用算子。

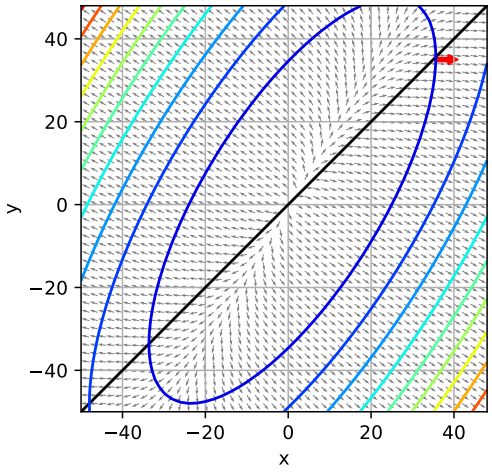

图 2.2: 函数及其梯度场

在图 2.3中，梯度场表示函数变化的方向，如果想象某个流体分子在梯度场上移动，它将沿着梯度场指向的方向移动（图 2.3中的箭头），而在某些位置，流体分子都倾向于离开该点，所有的分子都倾向于散开，我们定义此时散度（divergence）为正，如图中中间的那些点，反之散度为负。

定义 4. *拉普拉斯算子是 n 维欧几里得空间中的一个二阶微分算子，其定义为对函数 f 先作梯度运算（∇f）后，再作散度运算（$\nabla \cdot \nabla f$）的结果。因此如果 f 是二阶可微的实函数，则 f 的拉普拉斯算子定义为：$\Delta f = \nabla^2 f = \nabla \cdot \nabla f$.*

拉普拉斯算子是 n 维欧几里得空间中的一个二阶微分算子，其定义为对函数 f 先作梯度运算（∇f）后，再作散度运算（$\nabla \cdot \nabla f$）的结果。因此如果 f 是二阶可微的实函数，则 f 的拉普拉斯算子定义为：

$$\Delta f = \nabla^2 f = \nabla \cdot \nabla f. \tag{2.3}$$

然后，计算该函数的梯度 ∇f 和拉普拉斯算子 Δf：

梯度 ∇f 是一个向量，其中包含了函数 $f(x, y)$ 在每个方向上的偏导数。给出例子 $f(x, y) = x^2 + y^2$，在这个例子中：

$$\nabla f = (2x, 2y) \tag{2.4}$$

接下来，计算拉普拉斯算子 Δf，它是梯度的散度 $\nabla \cdot \nabla f$。在这个例子中：

$$\Delta f = \nabla \cdot \nabla f = \frac{\partial}{\partial x}(2x) + \frac{\partial}{\partial y}(2y) = 2 + 2 = 4 \tag{2.5}$$

这意味着，在 $f(x, y) = x^2 + y^2$ 形成的梯度场中，粒子总是倾向于离开。更为普遍的拉普拉斯是和 x 和 y 的值有关，如 $f(x, y) = 3 + \cos(\frac{x}{2}) \sin(\frac{y}{2})$，其散度为

$$\Delta f = -\frac{1}{2} \cos\left(\frac{x}{2}\right) \sin\left(\frac{y}{2}\right) \tag{2.6}$$

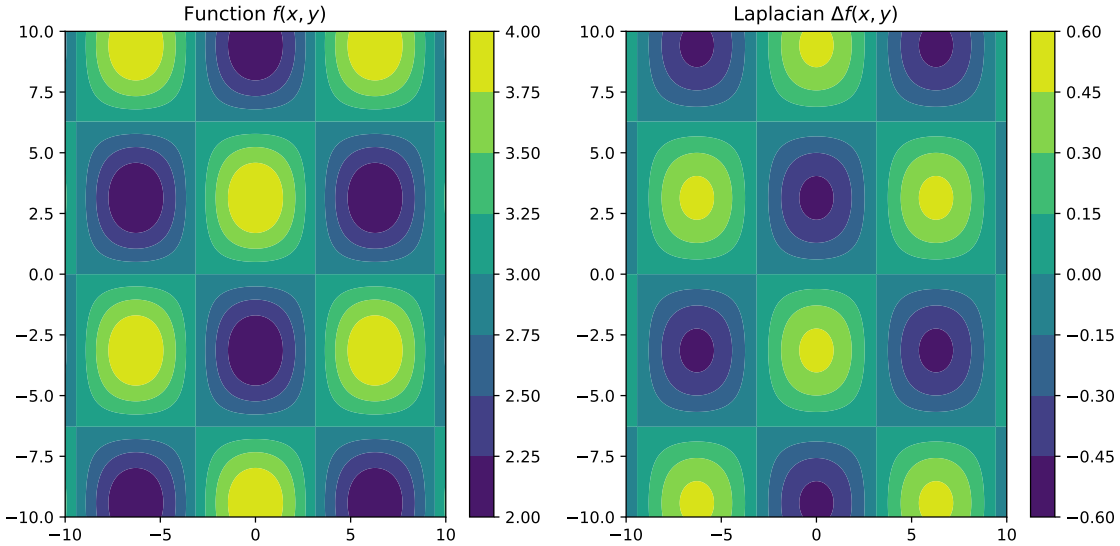

图 2.3: $f(x,y) = 3 + \cos(\frac{x}{2})\sin(\frac{y}{2})$ 的拉普拉斯算子

下面我们利用泰勒级数展开，导出离散拉普拉斯算子，然后再导出图拉普拉斯，首先假设离散函数空间中最小的步长单位是 h，即

$$x_{i+1} - x_i = h, x_i - x_{i-1} = h, \cdots, \tag{2.7}$$

然后分别利用泰勒级数将 $f(x_{i+1})$ 和 $f(x_{i-1})$ 的函数值在 x_i 点展开，可得:

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \frac{f'''(x_i)}{3!}h^3 + \ldots \tag{2.8}$$

$$f(x_{i-1}) = f(x_i) - f'(x_i)h + \frac{f''(x_i)}{2!}h^2 - \frac{f'''(x_i)}{3!}h^3 + \ldots \tag{2.9}$$

将公式加上公式可得到离散二阶导数的估计:

$$f''(x_i) = \frac{f(x_{i+1}) + f(x_{i-1}) - 2f(x_i)}{h^2} - O(h^2), \tag{2.10}$$

推广到 2 维空间，我们得到离散拉普拉斯算子:

$$\Delta f = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y), \tag{2.11}$$

另一个角度，拉普拉斯算子计算了周围点与中心点的梯度差。当 $f(x,y)$ 受到扰动之后，其可能变为相邻的 $f(x\pm1,y)$ 和 $f(x,y\pm1)$ 之一，拉普拉斯算子得到的是对该点进行微小扰动后可能获得的总增益（或者说是总变化）。

通过图 2.5，我们可以发现离散拉普拉斯算子就是在欧式空间上的表示，欧式空间可以看做一种所有离散点 $f(x,y)$ 和 $f(x\pm1,y)$ 以及 $f(x,y\pm1)$ 都相连，且距离都相等的图。我们假设点 (x,y) 为节点 v，$(x+1,y),(x-1,y),(x,y+1),(x,y-1)$ 构成 v 的邻居节点 $\mathcal{N}(v)$。有:

$$\Delta f = \sum_{u\in\mathcal{N}(v)} (f_u - f_v), \tag{2.12}$$

考虑图中边的权重，边权重 A_{uv} 较大的假设会对 v 的影响较大（在公式 2.12中邻居之间的边权重为 1，非邻居为 0），此时有

$$\Delta f = \sum_{u\in\mathcal{N}(v)} A_{uv}(f_u - f_v), \tag{2.13}$$

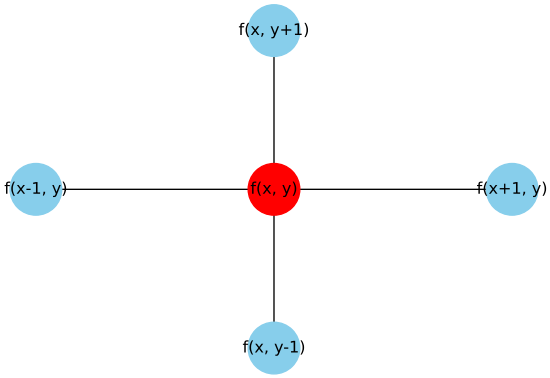

图 2.4: 欧式空间离散拉普拉斯和图拉普拉斯关系

继续推导

$$\begin{aligned}\Delta f &= \sum_{u\in\mathcal{N}(v)} A_{uv}(f_u - f_v) \\ &= \sum_{u\in\mathcal{N}(v)} A_{uv} f_u - \sum_{u\in\mathcal{N}(v)} A_{uv} f_v \\ &= D_u f_u - A_{v:} f\end{aligned} \tag{2.14}$$

对所有节点，有 $\Delta f = (\mathbf{D} - \mathbf{A})f$，$\mathbf{L} = \mathbf{D} - \mathbf{A}$ 即为图上的拉普拉斯算子。

定义 5. 无向图的组合图拉普拉斯矩阵 $\mathbf{L}$，其具有邻接矩阵和度矩阵 $\mathbf{A}$ 和 $\mathbf{D}$，是一个 $N \times N$ 矩阵，定义如下：

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

。

我们可得到公式 (2.1)的拉普拉斯矩阵为

$$\mathbf{L} = \begin{bmatrix} 1.25 & -1 & -0.25 & 0 & 0 & 0 \\ -1 & 1.25 & 0 & -0.25 & 0 & 0 \\ -0.25 & 0 & 0.35 & -0.1 & 0 & 0 \\ 0 & -0.25 & -0.1 & 1.5 & -0.7 & -0.4 \\ 0 & 0 & 0 & -0.7 & 0.7 & 0 \\ 0 & 0 & 0 & -0.4 & 0 & 0.4 \end{bmatrix} \tag{2.15}$$

得到了图拉普拉斯矩阵之后，可以开始深入的去理解最早一个图卷积工作 Bruna et al. [2014]。

首先来看图拉普拉斯特征值和特征向量的特点，其可以用来衡量信号在图上的平滑程度，且构成了互相正交的基向量。如果 $\mathbf{x}$ 是一个 N 维向量，节点 i 上的平滑度函数的自然定义如下：

$$\|\nabla \mathbf{x}\|_{\mathbf{A}}^2(i) = \sum_j A_{ij}[x(i) - x(j)]^2 \tag{2.16}$$

以及整体平滑度函数为：

$$\|\nabla \mathbf{x}\|_{\mathbf{A}}^2 = \sum_i \sum_j A_{ij}[x(i) - x(j)]^2 = \mathbf{x}\mathbf{L}\mathbf{x}^T, \tag{2.17}$$

根据这个定义，最平滑的向量是一个常数向量：

$$\mathbf{v}_0 = \arg\min_{\mathbf{x}\in\mathbb{R}^N, \|\mathbf{x}\|=1} \|\nabla \mathbf{x}\|_{\mathbf{A}}^2 = \frac{1}{\sqrt{N}}\mathbf{1}_N, \tag{2.18}$$

每个连续的 $\mathbf{v}_i = \arg\min_{\mathbf{x}\in\mathbb{R}^N, \|\mathbf{x}\|=1, x\perp\{\mathbf{v}_0,\ldots,\mathbf{v}_{i-1}\}} \|\nabla\mathbf{x}\|_{\mathbf{\Lambda}}^2$ 是图拉普拉斯矩阵 $\mathbf{L}$ 的特征向量，而特征值 λ_i 允许从向量 $\mathbf{x}$ 在 $[\mathbf{v}_0,\ldots,\mathbf{v}_{N-1}]$ 中的系数中读取向量 $\mathbf{x}$ 的平滑性。

解说 1. $\mathbf{Lv} = \lambda\mathbf{v}$，则称 λ 为矩阵 $\mathbf{L}$ 的一个特征值（标量），$\mathbf{v}$ 为矩阵 $\mathbf{L}$ 对应于特征值的一个特征向量（$\lambda \in \mathbb{R}$）。可根据特征向量和特征值的以下特性，用 $\mathbf{V}$ 表示特征向量的矩阵组合 $[\mathbf{v}_0,\ldots,\mathbf{v}_{N-1}]$，$\mathbf{\Lambda} = \begin{bmatrix} \lambda_0 & 0 & 0 & \ldots & 0 \\ 0 & \lambda_1 & 0 & \ldots & 0 \\ 0 & 0 & \lambda_2 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & \lambda_{N-1} \end{bmatrix}$

推导出图拉普拉斯矩阵特征向量的以上性质：

- $\mathbf{L}^k\mathbf{x} = \lambda^k\mathbf{x}$,
- $\mathbf{L}^{-1}\mathbf{x} = \frac{1}{\lambda}\mathbf{x}$,
- $\mathbf{L} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$,
- $\mathbf{V}\mathbf{V}^T = \mathbf{I}$

我们把公式 (2.1)定义的图的图拉普拉斯特征向量在图上做了可视化，可直观的发现特征向量越来越不平滑，**图拉普拉斯的特征向量提供了一组互相正交在图上较为平滑的基，而傅里叶变换则是提供了不同周期下的正余弦函数作为实数域上的基**：

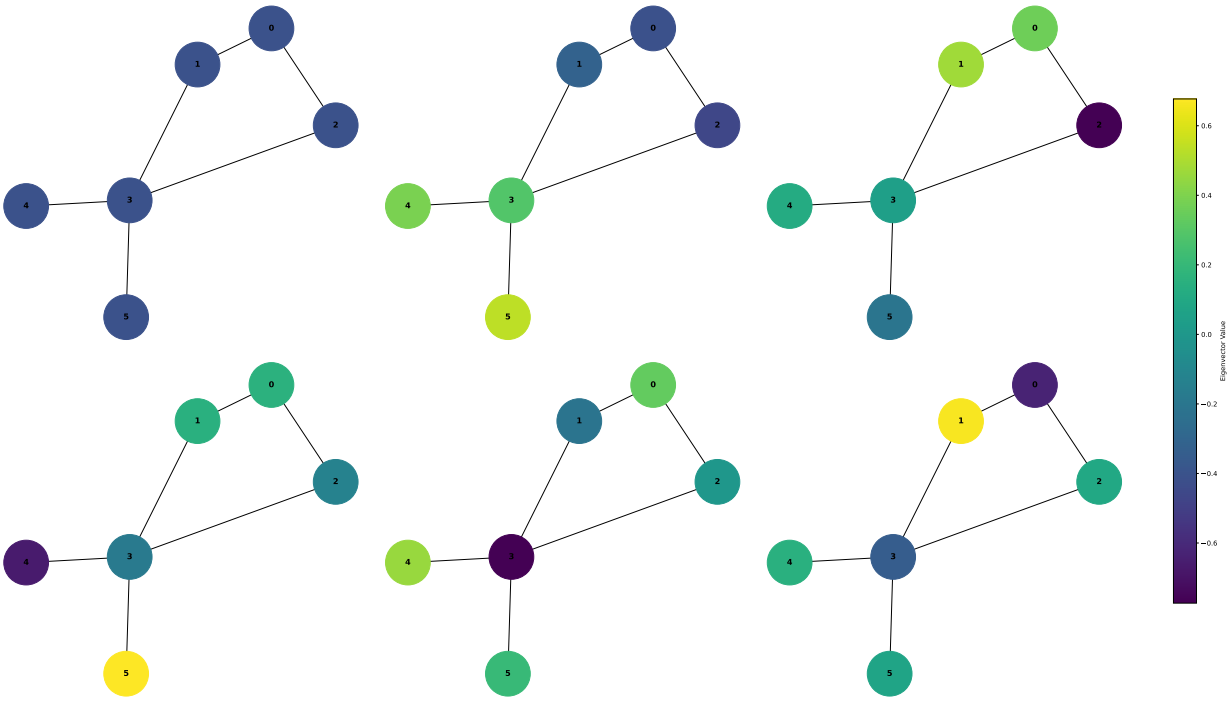

图 2.5: 公式 (2.1)定义的图拉普拉斯的 6 个特征向量在图上的可视化

对于向量 $\mathbf{x}$，定义图频域变换为 $\mathcal{F}(\mathbf{x}) = \mathbf{V}^T\mathbf{x}$，而频域逆变换为 $\mathcal{F}^{-1}(\mathbf{x}) = \mathbf{V}\mathbf{x}$，想要模仿传统欧式空间上的卷积操作，我们可以利用卷积定理，在频域上操作，卷积定理有：

$$\mathbf{g} * \mathbf{x} = \mathcal{F}^{-1}\left(\mathcal{F}\left(\mathbf{g}\right) \cdot \mathcal{F}(\mathbf{x})\right), \tag{2.19}$$

那么图的卷积公式可以表示为

$$\mathbf{g} * \mathbf{x} = \mathbf{V}\left(\mathbf{V}^T\mathbf{g} \cdot \mathbf{V}^T\mathbf{x}\right), \tag{2.20}$$

将 $\mathbf{V}^T\mathbf{g}$ 视为神经网络的参数 $\mathbf{G}$，其 $\mathbf{G}$ 为对角矩阵，基础卷积操作可看做：

$$\mathbf{g} * \mathbf{x} = \sigma\left(\mathbf{V}\mathbf{G}\mathbf{V}^T\mathbf{x}\right), \tag{2.21}$$

模仿普通卷积中，多个卷积核的结果相加的操作，得到第一代图神经网络：

$$\mathbf{x}_j^{l+1} = \sigma\left(\mathbf{V}\sum_{i=1}^{f_{k-1}}\mathbf{G}_{i,j}^l\mathbf{V}^T\mathbf{x}_i^l\right), (j = 1, \cdots, f_l) \tag{2.22}$$

其中 $\mathbf{G}_{i,j}^l$ 是对角矩阵即神经网络的参数，而且与之前一样，σ 是非线性映射函数。$\mathbf{x}_i^l$ 是图信号第 i 维的特征。在实践中，可以只取特征向量 $\mathbf{V}$ 中特征值最小的部分对应的部分特征向量，减少计算复杂度。

```python
class SpecNet(nn.Module):
    """
    :param num_nodes: Number of graph node.
    :param num_features: Input features number.
    :param num_out: Output features number
    :param num_eigen: The number of eigenvector used.
    """

    def __init__(self, num_nodes, num_features, num_out, num_eigen,
                 use_bias = False):
        super(SpecNet, self).__init__()

        self.num_nodes = num_nodes
        self.num_features = num_features
        self.num_out = num_out
        self.num_eigen = num_eigen

        self.G = nn.ParameterList()
        for i in range(num_features):
            for j in range(num_out):
                self.G.append(nn.Parameter(torch.diag(torch.randn(num_eigen))))

        if use_bias:
            self.bias = nn.Parameter(torch.FloatTensor(torch.zeros(size=(num_out,))))
        else:
            self.register_parameter('bias', None)

        self.initialize_weights()

    def initialize_weights(self):
        for weight in self.G:
            nn.init.xavier_uniform_(weight)
        if self.bias is not None:
            nn.init.zeros_(self.bias)

    def forward(self, x, V):
        result = []
        for j in range(self.num_out):
            temp = torch.zeros((x.shape[0], self.num_nodes))
            for i in range(self.num_features):
                temp += torch.mm(V, torch.mm(self.G[j*self.num_features + i],
                        torch.mm(V.T, x[:, :, i].T))).T
            result.append(temp)

        # Stack the results along a new dimension (last dimension)
        result = torch.stack(result, dim=-1)

```

```
48         return result
```

Code Listing 2.1: 第一代 GCN

参考文献

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and deep locally connected networks on graphs. In *2nd International Conference on Learning Representations, ICLR 2014*, 2014.