Graph Convolutional Neural Network

Mengxi Wu (mengxiwu@usc.edu)

1 Convolution

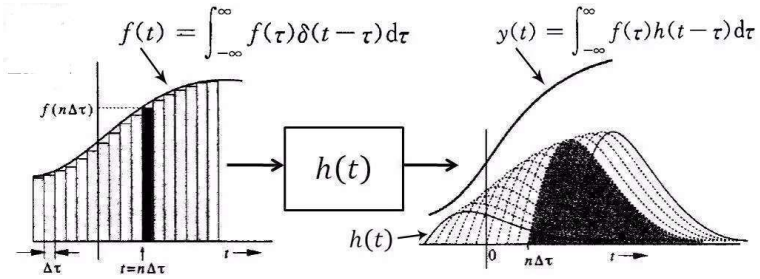

Figure 1: Spatial Convolution

Convolution decomposes a function into a series of impulse functions with different scales. For instance, for function $f(x)$, we use piecewise function to approximate. We set the length of a piece to be t_0. Let $\varepsilon(t)$ be the step function and $\delta(t)$ be the impulse function. We can present convolution in Eq.1.1,1.2,1.3,1.4.

$$f(t) \approx \sum_{n=-\infty}^{\infty} f(nt_0)(\varepsilon(t-t_0) - \varepsilon(t-(n+1)t_0)) \tag{1.1}$$

$$f(t) \approx \sum_{n=-\infty}^{\infty} f(nt_0)\frac{(\varepsilon(t-t_0)) - \varepsilon(t-(n+1)t_0)}{t_0}t_0 \tag{1.2}$$

$$f(t) = \lim_{t_0\to 0}\sum_{n=-\infty}^{\infty} f(nt_0)\frac{(\varepsilon(t-t_0)) - \varepsilon(t-(n+1)t_0)}{t_0}t_0 \tag{1.3}$$

$$f(t) = \lim_{t_0\to 0}\sum_{n=-\infty}^{\infty} f(nt_0)\delta(t-nt_0)t_0 \tag{1.4}$$

Since when $t_0 \to 0$, nt_0 is very small, we can replace nt_0 with symbol τ. Then, we have the formal representation of convolution in continuous time domain (Eq.1.5).

$$f(t) = \int_{-\infty}^{\infty} f(\tau)\delta(t-\tau)d\tau \tag{1.5}$$

$$y(t) = (f \star h)(t)\int_{-\infty}^{\infty} f(\tau)h(t-\tau)d\tau \tag{1.6}$$

Mathematically, the convolution in time domain between any two functions (Eq.1.6) first flips $h(\tau)$ and then shifts it with t to the right. Here is a reason for flipping and shifting. $h(\tau)$ can be treated as an impact factor. Old inputs will still have the power to impact the current response $y(t)$, and the power is various with time. For example, considering the response $y(4)$, the input $f(0)$ corresponds to the impact factor $h(4)$ because we are looking at its impact at four-time steps in the future. Therefore, we need to flip and shift the impact factor $h(\tau)$ first, before we multiply it with the input $f(\tau)$.

2 Fourier Transform

Convolution can be done in the spectral domain. To learn about spectral convolution, we need to first discuss Fourier series and Fourier transform. Using the Fourier series, we decompose a periodic function into a series of cosine and sine functions with different frequencies to simplify further computation and analysis. Different from the Fourier series, the Fourier transform can also handle non-periodic functions.

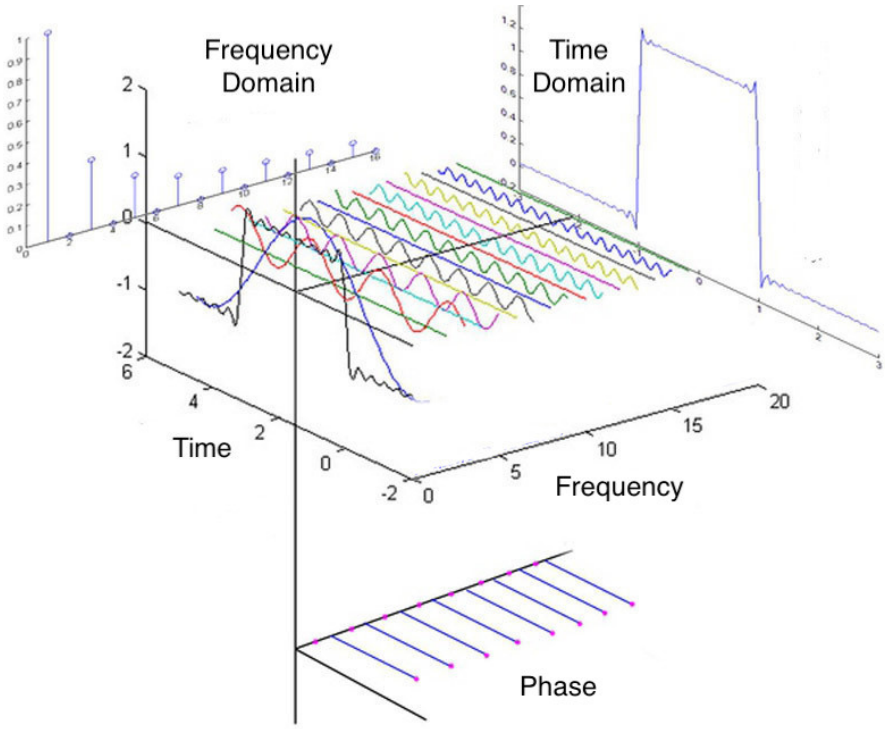

Figure 2: Fourier Series. A rectangular function in the time domain is decomposed into the sum of a series of cosine and sine functions with different frequencies.

Let $f(t)$ be a periodic function with period $T = 2L$ and augular frequency $w = \frac{2\pi}{T}$. The Fourier expansion of $f(x)$ is shown in Eq.2.1, 2.2, 2.3.

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty}[a_n cos(nwt) + b_n sin(nwt)] \tag{2.1}$$

$$a_n = \frac{2}{T}\int_{-L}^{L} f(t)cos(nwt)dt \tag{2.2}$$

$$b_n = \frac{2}{T}\int_{-L}^{L} f(t)sin(nwt)dt \tag{2.3}$$

According to Euler formula, Eq.2.4 and Eq.2.5, we can reformulate the expansion as shown in Eq.2.6. Since $a_{-n} = a_n$ and $b_{-n} = -b_n$, we can further change Eq.2.6 to Eq.2.7. Let $c_n = \frac{a_n - ib_n}{2}$. Then, we compute complex fourier series coefficient c_n by integrating the multiplication result of $f(t)$ and e^{-ikwt}. Since c_n only depend on n not t, the integration is acting on $e^{i(n-k)wt}$. When $n \neq t$, the integration is 0. When $n = k$, we have Eq.2.9, the formula of complex Fourier series coefficient.

$$cos(nwt) = \frac{1}{2}(e^{inwt} + e^{-inwt}) \tag{2.4}$$

$$sin(nwt) = \frac{1}{2i}(e^{inwt} - e^{-inwt}) \tag{2.5}$$

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \frac{a_n - ib_n}{2} e^{inwt} + \frac{a_n + ib_n}{2} e^{-inwt} \tag{2.6}$$

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \frac{a_n - ib_n}{2} e^{inwt} + \sum_{n=-\infty}^{-1} \frac{a_n - ib_n}{2} e^{inwt} = \sum_{n=-\infty}^{\infty} \frac{a_n - ib_n}{2} e^{inwt} \tag{2.7}$$

$$\int_{-L}^{L} f(t) e^{-ikwt} dt = \int_{-L}^{L} \sum_{n=-\infty}^{\infty} c_n e^{i(n-k)wt} dt \tag{2.8}$$

$$c_n = \frac{1}{T} \int_{-L}^{L} f(t) e^{-inwt} dt \tag{2.9}$$

The above derivation is for the coefficients of the complex Fourier series for the periodic function. Fourier transform is the coefficient of the expansion of a non-periodic function on a set of orthogonal basis $\{e^{inwt}\}_{n \in \mathbb{Z}}$. We can treat the non-periodic function as a periodic function whose period is infinitely large. From complex Fourier series coefficients to Fourier transform, we first replace c_n with Eq.2.9 in the complex Fourier expansion. Then, we subsitute $k = nw$ and $\Delta k = k_n - k_{n-1} = w$.

$$\begin{aligned} f(t) &= \sum_{n=-\infty}^{\infty} c_n e^{inwt} \\ &= \sum_{n=-\infty}^{\infty} (\frac{1}{T} \int_{-L}^{L} f(t) e^{-inwt} dt) e^{inwt} \\ f(t) &= \frac{1}{2\pi} \cdot \Delta k \cdot \sum_{n=-\infty}^{\infty} (\int_{-L}^{L} f(t) e^{-ikt} dt) e^{ikt} \end{aligned} \tag{2.10}$$

After taking $T = 2L$ to be infinite, the Fourier transform $\mathcal{F}(k)$ is presented in Eq.2.11.

$$\mathcal{F}(k) = \lim_{L \to \infty} \int_{-L}^{L} f(t) e^{-ikt} dt = \int_{-\infty}^{\infty} f(t) e^{-ikt} dt \tag{2.11}$$

The inverse Fourier transform can be derived from Eq.2.10 by substituting $\mathcal{F}(k)$. By Riemann sum, we reach Eq.2.12, the formula for inverse Fourier transform.

$$f(t) = \frac{1}{2\pi} \sum_{n=-\infty}^{\infty} \mathcal{F}(k) e^{ikt} \Delta k = \frac{1}{2\pi} \int_{-\infty}^{\infty} \mathcal{F}(k) e^{ikt} dk \tag{2.12}$$

For a discrete signal $f[t]$, where $t \in \mathbb{Z} \cap [0, N-1]$, we have the discrete Fourier transform (Eq.2.13). Similar to the continuous Fourier transform, $F[n]$ informs the coefficient of the complex sinusoidal component $e^{-\frac{i2\pi n}{N}}$.

$$\mathcal{F}[n] = \sum_{t=0}^{N-1} f[t] e^{-\frac{i2\pi nt}{N}} \tag{2.13}$$

The inverse discrete Fourier transform is as follows.

$$f[t] = \frac{1}{N} \sum_{n=0}^{N-1} F[n] e^{\frac{i2\pi nt}{N}} \tag{2.14}$$

3 Spectral Graph Convolution

3.1 Graph Structures

- **Directed/Undirected Graphs**. Edges in directed graphs are all directed from one node to another. Each edge in undirected graphs can be regarded as two directed edges.
- **Homogeneous/Heterogeneous Graphs**. Nodes and edges in homogeneous graphs have the same types, while nodes and edges have. different types in heterogeneous graphs.
- **Static/Dynamic Graphs**. When input feature or the topology of the graph varies with time, the graph is regarded as a dynamic graph.

3.2 From Time Signals to Graph Signals

Suppose we have a discrete time-varying signal $f[t] = [[t_0], f[t_1], ..., f[t_{N-1}]]$. We can view this signal as a chain graph (Figure 3), where each point in time t_j is represented as a node and $f[t_j]$ represents the value of the signal at that node. Let A be the adjacency matrix, and $L = D - A$ be the Laplacian of

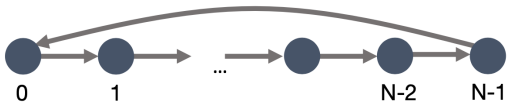

Figure 3: Time-series signal as a chain graph.

this chain graph, where D is the degree matrix. We can then represent time shifts by multiplying A,

$$Af[t] = f[(t+1) \mod N] \tag{3.1}$$

and difference operation by multiplying L.

$$Lf[t] = f[t] - f[(t+1) \mod N] \tag{3.2}$$

In summary, multiplying a signal with its adjacency matrix propagates information from node to node, and multiplying a signal with its Laplacian computes the difference between each node and its neighbors. This is the graph-based view of transforming signals through matrix multiplication. In this view, we can also make the convolution process with filter h as matrix multiplication on $f[t]$.

$$(f \star h)(t) = Q_h f[t] \tag{3.3}$$

Considering the properties of convolution, we need to require the multiplication by this matrix to satisfy translation equivariance and the equivariance of the difference operator.

$$AQ_h = Q_h A \tag{3.4}$$

$$LQ_h = Q_h L \tag{3.5}$$

When Q_h is a polynomial function of the adjacency matrix A, it can satisfy these two properties.

$$Q_h = p_N(A) = \sum_{j=0}^{N-1} \alpha_j A^j \tag{3.6}$$

According to the above observation, we can generalize these notions to more general graphs. In the special case of the chain graph, it is able to define filter matrices Q_h based on A that simultaneously commute with both A and L. However, for more general graphs, commutativity with A does not necessarily imply commutativity with L. Therefore, researchers propose symmetric normalized Laplacian $L_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$ and symmetric normalized adjacency matrix $A_{sym} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. There are two reasons why they are desirable.

- A_{sym} and L_{sym} have bounded spectrums, which gives them numerical stability.
- A_{sym} and L_{sym} are simultaneously diagonalizable, meaning they share the same eigenvectors. Defining filters based on either A_{sym} or L_{sym} implies commutativity with the other.

For any graph with symmetric normalized adjacency matrix A_{sym}, we can represent convolutional filters as matrices in the following form. Note that we can replace A_{sym} with L_{sym}.

$$Q_h = \alpha_0 I + \alpha_1 A_{sym} + \alpha_2 A_{sym}^2 + ... + \alpha_N A_{sym}^N \tag{3.7}$$

If we multiply a node feature vector x_v for $v \in V$ with Q_h, then we get

$$Q_h x_v = \alpha_0 I x_v + \alpha_1 A_{sym} x_v + \alpha_2 A_{sym}^2 x_v + ... + \alpha_N A_{sym}^N x_v \tag{3.8}$$

Eq.3.8 corresponds to the mixture of the information in the node's N-hop neighborhood, with the α_j terms controlling the strength of the information coming from different hops. If we have a matrix of node features X, then convolution becomes

$$Q_h X = \alpha_0 I X + \alpha_1 A_{sym} X + \alpha_2 A_{sym}^2 X + ... + \alpha_N A_{sym}^N X \tag{3.9}$$

3.3 The Fourier Transform and The Laplace Operator

The connection between the Fourier transform and the Laplace (difference) operator Δ enables the generalization of the Fourier transform to graphs. For an arbitrary smooth functions $f : \mathbb{R}^d \to \mathbb{R}$,

$$\Delta f(t) = \nabla^2 f(t) \tag{3.10}$$

Laplace operator Δ computes the divergence of the gradient $\nabla f(x)$. Intuitively, Laplace operator tells us the average difference between the $f(x)$ and $f(y)$ for y in the neighboring regions surrounding x. However, the most important property of Laplace operator is that its eigenfunctions correspond to the same complex exponentials that make up the Fourier basis (the complex exponentials) in the Fourier transform (Eq.3.11, where eigenvalue $\lambda = -(2nw)^2$).

$$\Delta(e^{i2nwt}) = -(2nw)^2 \cdot e^{i2nwt} \tag{3.11}$$

3.4 Graph Fourier Transform

In the discrete graphs setting, the Laplace operator corresponds to the difference operator, the Laplacian. From this perspective, The Laplacian matrix can be viewed as a discrete analog of the Laplace operator. The eigenvectors of Laplacian L act as the Fourier basis and the eigenvalues of L are frequencies of the Fourier basis.

$$(Lf)[n] = \sum_{j \in V} A[n, j](f[n] - f[j]) \tag{3.12}$$

Thus, we first apply eigendecomposition on Laplacian L to get the Fourier basis that makes up the graph signal. U is the matrix of eigenvectors, and Λ is the diagonal matrix of the eigenvalues.

$$L = U\Lambda U^T \tag{3.13}$$

As we discuss in section 2, the Fourier transform is a series of coefficients of the expansion of a signal on the Fourier basis. Therefore, we project graph signals on the Fourier basis to obtain Fourier transform/coefficients. The Fourier transform $\mathcal{F}_f$ of graph signal f is computed as follows.

$$\mathcal{F}_f = U^T f \tag{3.14}$$

and its inverse Fourier transform computed as

$$f = U\mathcal{F}_f \tag{3.15}$$

Note that there is another reason for using the eigenvectors of Laplacian as Fourier basis. This basis has the smallest Dirichlet energy. The signals defined by these eigenvectors vary in a smooth way across the graph, so they help alleviate the overfitting problem. That's why we choose the eigenvectors of the Laplacian.

3.5 Spectral Convolution for Graph

Graph convolutions in the spectral domain are defined via point-wise products. For graph signals f and h, the convolution operation will be

$$f \star h = U(\mathcal{F}_f \odot \mathcal{F}_h) = U(U^T f \odot U^T h) \tag{3.16}$$

We replace $U^T h$ with θ_h, and treat θ_h as a filter

$$f \star h = U(U^T f \odot \theta_h) = (U\mathrm{diag}(\theta_h)U^T)f \tag{3.17}$$

To ensure θ_h corresponds to a meaningful convolution on the graph, we can define θ_h as $p_N(\Lambda)$, a degree N polynomial of the eigenvalues of the Laplacian. In this way, the convolution commutes with the Laplacian. Note that we usually use L_{sym} to make the convolution commute with both L_{sym} and A_{sym}.

$$f \star h = (Up_N(\Lambda)U^T)f = p_N(L)f \tag{3.18}$$

Recursive Formulation for Fast Convolution. Defferrard et al.[2] propose to implement $p_N(L)$ using Chebyshev polynomials. Chebyshev polynomials have an efficient recursive formulation and various properties that are beneficial for polynomial approximation. Their idea is as follows.

$$p_N(\hat{L}) = \sum_{k=0}^{K} \omega_k T_k(\hat{L}) \tag{3.19}$$

$$T_k(L) = 2\hat{L}T_{k-1}(\hat{L}) - T_{k-2}(\hat{L}), T_0 = I, T_1 = \hat{L} \tag{3.20}$$

$$\hat{L} = \frac{2L}{\lambda_{max}} - I \tag{3.21}$$

where λ_{max} is the largest eigenvalue of L and ω_k is the Chebyshev coefficient. Chebyshev polynomial $T_k(\hat{L})$ is computed by the recurrence relation in Eq.3.20. If we denote $s_k = T_k(\hat{L})f$, we have

$$s_k = 2\hat{L}s_{k-1} - s_{k-2}, s_0 = f, s_1 = \hat{L}f \tag{3.22}$$

The convolution operation becomes

$$f \star h = p_N(\hat{L})f = [s_0, s_1, ...s_{K-1}]\omega \tag{3.23}$$

ω, the Chebyshev coefficients are the trainable parameters.

4 Spatial Graph Convolution

Similar to the convolution operation on an image, spatial graph convolution is based on a node's spatial relations. The spatial convolution uses the central node's representation together with its neighbors' representations to derive the updated representation for the central node. The convolution operation is done by information aggregation or message passing along the edges. For this section, the notations are as follows.

- h_v^j: The information carried by node v in j-th layer of the neural network.
- $x_(v, u)$: The information carried by the edge (v, u).
- $n[v]$: The set of neighboring nodes of v.
- W^j: Learnable parameters in j-th layer of the neural network.
- σ: A nonlinear activation function.
- U_j, M_j: Functions with learnable parameters.

4.1 Meassage Passing

Convolution is taken as a message passing process. The information can be passed from one node to another along edges.

$$h_v^j = U_j(h_v^{j-1}, \sum_{u \in n[v]} M_j(h_v^{j-1}, h_u^{j-1}, x_{(v,u)})) \tag{4.1}$$

4.2 Mean Aggregators

Mean aggregators take the elementwise mean of the vectors in $\{h_u^{j-1}, \forall u \in n[v]\}$.

$$h_v^j = \sigma(W^j \cdot \text{MEAN}(\{h_v^{j-1}\} \cup \{h_u^{j-1}, \forall u \in n[v]\})) \tag{4.2}$$

4.3 LSTM Aggregators

LSTM aggregators have the advantage of larger expressive capability compared to mean aggregators. However, they are not permutation invariant since they process their inputs in a sequential manner. For an unordered set of nodes, They just choose to operate on a random permutation of the node's neighbors.

4.4 Pooling Aggregators

Each neighbor's vector is independently multiplied with learnable parameters W^j. An elementwise max-pooling operation is applied after the multiplication to aggregate information across the neighbor set.

$$h^j_{global} = \max(\{\sigma(W^j \cdot h^j_u + b^j), \forall u \in n[v]\}) \tag{4.3}$$

where h^j_{global} is the result after pooling aggregation.

5 Graph Neural Network

5.1 Tasks

- **Node-level**. Tasks focus on operating on nodes, including node classification, node regression, and node clustering.
- **Edge-level**. Tasks focus on the models that classify edge types or predict whether there is an edge existing between two given nodes.
- **Graph-level**. Tasks include graph classification, graph regression, and graph matching.

5.2 Modules

- **Propagation Module**. The propagation module propagates information between nodes to capture useful features and topological information. The convolution operator and recurrent operator are for aggregating information from neighboring nodes, while the skip connection operation is used to gather information from historical representations of nodes.
- **Sampling Module**. Sampling module will be useful when the graph is large. It is usually combined with propagation module.
- **Pooling Module**. Pooling modules are conducted to extract information on high-level subgraphs or graphs from nodes.

5.3 Basics

The idea underlining is that nodes in a graph represent concepts, and edges represent their relationships. GNN is about taking an input graph $G = (V, E)$, together with a set of features l of the graph, and generating node embeddings z_v for $v \in V$. During learning to generate z_v, a hidden embedding h^j_v of node v at iteration j is updated according to information aggregated from the neighboring nodes of v. The final embeddings z_v for $v \in V$ are the h^m_v for $v \in V$ at the final iteration m. Let's define the function for updating be f_w. Note that f_w is shared by all nodes. The update for hidden state h^j_v is shown in Eq.5.1.

$$h^{j+1}_v = f_w(l_v, l_{ne[v]}, l_{n[v]}, \{h^j_u, j \in N(v)\}) \tag{5.1}$$

- $n[v]$: The set of neighboring nodes of node v.
- $ne[v]$: The set of edges connected to node v.
- l_v: The feature vector of node v.
- $l_{n[v]}$: Feature vectors of nodes in set $n[v]$.
- $l_{ne[v]}$: Feature vectors of edges in set $ne(v)$.

According to **Banach's Fixed Point Theorem**, as long as F_w is a **contraction map**, not matter what h^0_v is, h^j_v will converge to a fixed point. To understand contraction map, we first introduce **metric space**. A metric space (H, d) is a set H together with a distance function d which assigns a real number $d(x, y)$ to every pair x, y. A contraction map F means for any $x, y \in H$,

$$d(F(x), F(y)) \leq cd(x, y), 0 \leq c < 1 \tag{5.2}$$

A contraction map brings points closer. The new metric space obtained by applying the mapping F is smaller than the original metric space (H, d). If we repeat the contraction mapping, the original metric space will eventually be mapped to a point, which we call the fixed point. Contraction mapping is illustrated in Figure 4. Let F_w be the stacked version of $|V|$ instances of f_w. Eq.5.1 can reformulate

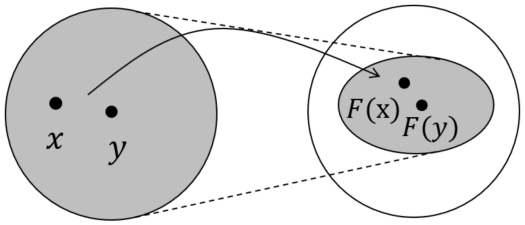

Figure 4: Contraction Mapping.

as Eq.5.3. h_j is obtained by stacking together the hidden embeddings of all nodes at iteration j. l is obtained by stacking together all the feature vectors of the graph. In practice, we use a feed-forward neural network to approximate the contraction map F_w. Before demonstrating how to make F_w be a contraction map, we briefly introduce Theorem 5.1.

$$h^{j+1} = F_w(h^j, l) \tag{5.3}$$

Theorem 5.1 *Assume the set $H \subset \mathbb{R}^n$ is convex and the function F: $H \to \mathbb{R}^n$ has continuous partial derivatives in H. If for $c < 1$, the matrix norm of the Jacobian satifies*

$$\forall x \in H, \|F'(x)\| \leq c$$

the mapping F is a contraction in H.

According to Theorem 5.1, to make F_w be contraction map we can add a penalty term to the loss to constrain the paramters update of neural network F_w. The loss term E_w is shown in Eq.5.4, where β is a tunable hyperparameter and *Loss* is determined by a downstream task such as node classification. For a specific downstream task, an output function G_w is defined to produce the outputs. *Loss* is computed from O in Eq.5.5.

$$E_w = Loss + \beta \cdot \max((\|\frac{\partial F_w}{\partial h^m}\| - c)^2, 0), 0 < c < 1 \tag{5.4}$$

$$O = G_w(h^m, l) \tag{5.5}$$

5.4 Training Graph Neural Network

The training algorithm based on a gradient-descent approach is summarized in following steps.

- The hidden embeddings $\{h_v^j, v \in V\}$ are iteratively updated by Eq.5.1 until at time they approach the fixed point solution.
- The gradients with respect to weights w, $\frac{\partial E_w}{\partial w}$, are computed.
- The weights w are updated according to gradients computed.

Noramlly, the initial hidden embeddings are set to be input features, i.e., $h_v^0 = x_v$ for $v \in V$.

6 Graph Convolutional Neural Networks

Spectral GCNN: ChebNet, FastGCN, LanczosNet, GCN, GCN2.
Spatial GCNN: MPNNs, PATCHY-SAN, LGCN, MoNet, GraphSAGE.

References

[1] Dirichlet energy. link.

[2] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

[3] dongyu. link.

[4] William L. Hamilton. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159.

[5] Justin Ko. Fourier transform - university of toronto department of mathematics. link.

[6] Qian Liu. link.

[7] Tobias Petersdorff. Fixed point iteration and contraction mapping theorem.

[8] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[9] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[10] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):1–23, 2019.