10 Graph Neural Networks

Goal

Introducing the basics of GNN and the popular variants.

Alert 10.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

Nice surveys on this topic include Bronstein et al. (2017) and Wu et al. (2020).

Bronstein, M. M., J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst (2017). "Geometric Deep Learning: Going beyond Euclidean data". *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42.

Wu, Z., S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu (2020). "A Comprehensive Survey on Graph Neural Networks". *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21.

Definition 10.2: Graph learning

Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathfrak{l})$ with nodes $\mathcal{V}$, edges $\mathcal{E}$, node feature/attribute/label $\mathfrak{l}_v \in \mathbb{R}^d$ for each node $v \in \mathcal{V}$, and edge feature/attribute/label $\mathfrak{l}_e \in \mathbb{R}^p$ for each edge $e \in \mathcal{E}$. The graph may be directed or undirected, where the direction of the edge can be easily encoded in the edge feature. We use $\mathcal{N}_v = \mathcal{N}(v) \subseteq \mathcal{V}$ to denote the neighboring nodes of v and $\mathcal{M}_v = \mathcal{M}(v) \subseteq \mathcal{E}$ for the edges that have node v as a vertex. For positional graphs, we also have an injective function $\mathfrak{p}_v : \mathcal{N}_v \to \{1, 2, \dots, |\mathcal{V}|\}$ that encodes the relative position of each neighbor of a node v. For instance, on a 2-D image, $\{1, 2, 3, 4\}$ may represent the west, north, east, and south neighbor, respectively.

Alert 10.3: All for one, and one for all

Let $(\mathcal{G}_i, \mathbf{y}_i), i = 1, \dots, n$ be a given supervised set of graphs and labels. Our goal is to learn a predictive function $\hat{\mathbf{y}}$ that maps a new test graph $\mathcal{G}$ to its corresponding label: $\hat{\mathbf{y}}(\mathcal{G}) \approx \mathbf{y}$. The labels could be at the node, edge or graph level. Do not confuse the label $\mathbf{y}$ with the feature $\mathfrak{l}$, since some authors also refer to the latter as "labeling."

Interestingly, we can piece all graphs into one large, disconnected graph, greatly simplifying our notation and without compromising generality. Note that this is more than just a reduction trick: in some cases it is actually the natural thing to do, such as in web-scale applications where the entire internet is just one giant graph. We follow this trick throughout.

Example 10.4: Some applications of graph learning

We mention some example applications of graph learning:

- Each node may represent an atom in some chemical compound while the edges model the (strength of) chemical bonds linking the atoms. We may be interested in predicting how a certain disease reacts to the chemical compound.
- All image analyses fall into graph learning with each pixel playing a node of the underlying (regular) grid and the pixel value being the node feature.
- Social network, where we may be interested in classifying the nodes or imputing missing links. For instance, each webpage is a node and hyperlinks act as edges.

Definition 10.5: Graph neural network (GNN) (Scarselli et al. 2009)

GNNs, as defined here, can be regarded as a natural extension of recurrent networks, from a chain graph to a general graph. Indeed, we define the following recursion: for all $v \in \mathcal{V}$,

$$\begin{aligned}\mathbf{h}_v &\leftarrow \mathbf{f}(\mathbf{h}_v, \mathbf{h}_{\mathcal{N}_v}, \mathfrak{l}_v, \mathfrak{l}_{\mathcal{N}_v}, \mathfrak{l}_{\mathcal{M}_v}; \mathbf{w}) \\ \mathbf{o}_v &= \mathbf{g}(\mathbf{h}_v, \mathfrak{l}_v; \mathbf{w}),\end{aligned}$$

where $\mathbf{h}_v$ is the hidden state at node v and $\mathbf{o}_v$ is its output. The two (local) update functions $\mathbf{f}, \mathbf{g}$ are parameterized by $\mathbf{w}$, which is shared among all nodes. We remark that in general it is up to us to define the neighborhoods $\mathcal{N}$ and $\mathcal{M}$, and $\mathbf{f}, \mathbf{g}$ may have slightly different forms (such as involving other inputs).

Collect all local updates into one abstract formula:

$$\mathbf{x} := \begin{bmatrix}\mathbf{h} \\ \mathbf{o}\end{bmatrix} \leftarrow \mathsf{F}(\mathbf{x}, \mathfrak{l}; \mathbf{w}). \tag{10.1}$$

Note that the input node/edge features $\mathfrak{l}$ are fixed. Thus, for a fixed weight $\mathbf{w}$, the above update defines the (enhanced) state $\mathbf{x}$ as a fixed point of the map $\mathsf{F}_{\mathfrak{l},\mathbf{w}} : \mathbf{x} \mapsto \mathsf{F}(\mathbf{x}, \mathfrak{l}; \mathbf{w})$.

To compute the state $\mathbf{x}$ with a fixed weight $\mathbf{w}$, we perform the (obvious) iteration:

$$\mathbf{x}_{t+1} = \mathsf{F}(\mathbf{x}_t, \mathfrak{l}; \mathbf{w}), \quad \mathbf{x}_0 \text{ initialized}. \tag{10.2}$$

According to Banach's fixed point theorem, (for any initialization $\mathbf{x}_0$) the above iteration converges geometrically to the unique fixed point of $\mathsf{F}_{\mathfrak{l},\mathbf{w}}$, provided that the latter is a contraction (or more generally a firm nonexpansion). For later reference, we abstract (the unique) solution of the nonlinear equation (10.1) as:

$$\mathbf{o} = \hat{\mathbf{y}}(\mathfrak{l}; \mathbf{w}),$$

where we have discarded the state $\mathbf{h}$ and only retained the output $\mathbf{o}$.

Scarselli, F., M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini (2009). "The Graph Neural Network Model". *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80.

Alert 10.6: Recursive neural network

When the underlying graph is a DAG (and the update function of a node only depends on its descendants), we may arrange the computation in (10.2) according to some topological ordering so that it stops after one (sequential) pass of all nodes. When the graph is a chain, we recover the familiar recurrent neural network.

Example 10.7: Local update function

We mention two examples of local update function:

- For positional graphs, we arrange the neighbors in $\mathbf{h}_{\mathcal{N}_v}, \mathfrak{l}_{\mathcal{N}_v}, \mathfrak{l}_{\mathcal{M}_v}$ according to their relative positions decided by $\mathfrak{p}_v$ (say in increasing order). For non-existent neighbors, we may simply pad with null values.
- For non-positional graphs, the following permutation-invariant local update is convenient:

$$\mathbf{h}_v \leftarrow \frac{1}{|\mathcal{N}_v|} \sum_{u \in \mathcal{N}_v} \mathbf{f}(\mathbf{h}_v, \mathbf{h}_u, \mathfrak{l}_v, \mathfrak{l}_u, \mathfrak{l}_{(v,u)}).$$

  More generally, we may replace the above average with any permutation-invariant function (e.g. averaged ℓ_p norm), see Xu et al. (2019) for some discussion on possible limitations of this choice.

Xu, Keyulu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka (2019). "How Powerful are Graph Neural Networks?" In: *International Conference on Learning Representations*.

Algorithm 10.8: Learning GNN

To learn the weights $\mathbf{w}$ of a GNN, we choose a loss function ℓ, and apply (stochastic) gradient descent to solve

$$\min_{\mathbf{w}} \quad \ell(\hat{\mathbf{y}}(\mathfrak{l}; \mathbf{w}), \mathbf{y}),$$

where recall that $\hat{\mathbf{y}}(\mathfrak{l}; \mathbf{w})$ is (the unique) solution of the nonlinear equation (10.1) and is practically computed by the iteration (10.2) (similar to unrolling in RNN). If $\mathsf{F}(\mathbf{x}, \mathfrak{l}; \mathbf{w})$ is differentiable in $\mathbf{w}$ and contracting in $\mathbf{x}$, then a simple application of the implicit function theorem reveals that the solution $\hat{\mathbf{y}}(\mathfrak{l}; \mathbf{w})$ is also differentiable in $\mathbf{w}$. Thus, we may apply the recurrent back-propagation algorithm. If memory is not an issue, we can also apply back-propagation through time (BPTT) by replacing $\hat{\mathbf{y}}$ with $\mathbf{o}_t$ after a fixed number of unrolling steps in (10.1).

Example 10.9: Parameterizing local update function

- Affine: Let $\mathsf{F}(\mathbf{x}, \mathfrak{l}; \mathbf{w}) = A(\mathfrak{l}; \mathbf{w})\mathbf{x} + \mathbf{b}(\mathfrak{l}; \mathbf{w})$, where the matrix A and bias vector $\mathbf{b}$ are outputs of some neural net with input $\mathfrak{l}$ and weights $\mathbf{w}$. By properly scaling A, it is easy to make F a contraction.
- More generally, we may parameterize F by a (highly) nonlinear deep network. However, care must be taken (e.g. through regularization) so that F is (close to) a contraction at the learned weights.
- We remark that in theory any parameterization of F can be used; it does not have to be a neural network.

Example 10.10: PageRank belongs to GNN

Define the normalized adjacency matrix

$$\bar{A}_{uv} = \begin{cases} \frac{1}{|\mathcal{N}_u|}, & \text{if } (u, v) \in \mathcal{E} \\ 0, & \text{otherwise} \end{cases},$$

which represents the probability of visiting a neighboring node v once we are at node u. Consider the GNN with linear state update function:

$$\mathbf{x} \leftarrow \alpha \mathbf{x}_0 + (1 - \alpha)\bar{A}^\top \mathbf{x},$$

where the parameter $\alpha \in [0, 1)$ models the probability of "telescoping" and $\mathbf{x}_0 \in \Delta$. In other words, the state of node v is an aggregation of the states of its neighbors:

$$x_v = \alpha x_{v,0} + (1 - \alpha) \sum_{(u,v) \in \mathcal{E}} \frac{1}{|\mathcal{N}_u|} x_u.$$

For any $\alpha \in (0, 1)$, the above iterate converges to a unique fixed point known as the PageRank.

Definition 10.11: Spatial convolutional networks on graphs (Bruna et al. 2014)

Given a (weighted) graph $\mathcal{G}^0 = (\mathcal{V}^0, A^0)$, where A^0 is the adjacency matrix, we define a sequence of coarsenings $\mathcal{G}^l = (\mathcal{V}^l, A^l), l = 1, \ldots, L$, where recursively each node $V \in \mathcal{V}^{l+1}$ is a subset (e.g. neighborhood) of

nodes in $\mathcal{V}^l$, i.e.

$$\mathcal{V}^{l+1} \subseteq 2^{\mathcal{V}^l}, \quad \text{and for all } U, V \in \mathcal{V}^{l+1},\ A^{l+1}_{UV} = \sum_{u \in U \subseteq \mathcal{V}^l} \sum_{v \in V \subseteq \mathcal{V}^l} A^l_{uv}.$$

Typically, the nodes in $\mathcal{V}^{l+1}$ form a partition of the nodes in $\mathcal{V}^l$, using say some graph partitioning algorithm. For instance we may cluster a node u with all "nearby" and available nodes v with $A_{uv} \leq \epsilon$, hence forming an ϵ-cover.

Let $\mathbf{x}^l = [\mathbf{x}^l_1; \dots; \mathbf{x}^l_{d_l}] \in \mathbb{R}^{|\mathcal{V}^l| d_l}$ be a d_l-channel signal on the nodes of graph $\mathcal{G}^l$. We define a layer of spatial convolution as follows:

$$\mathbf{x}^{l+1}_r = \mathscr{P}\big(\sigma(W^l_r \mathbf{x}^l)\big), \quad W^l_r \in \mathbb{R}^{|\mathcal{V}^l| \times |\mathcal{V}^l| d_l}, \quad r = 1, \dots, d_{l+1},$$

where each W^l_r is a *spatially compact* filter (with nonzero entries only when A^l_{uv} larger than some threshold), $\sigma : \mathbb{R} \to \mathbb{R}$ some (nonlinear) component-wise activation function, and $\mathscr{P}$ a pooling operator that pools the values in each neighborhood (corresponding to nodes in $\mathcal{V}^{l+1}$). The total number of parameters in the filter $\mathcal{W}^l$ is $O(|\mathcal{E}^l| d_l d_{l+1})$. Since nodes in a general graph (as opposed to regular ones such as grids) may have different neighborhoods, it is not possible to share the filter weights at different nodes (i.e. the rows in W^l_r have to be different).

Bruna, Joan, Wojciech Zaremba, Arthur Szlam, and Yann LeCun (2014). "Spectral Networks and Locally Connected Networks on Graphs". In: *International Conference on Learning Representations*.

Example 10.12: Spatial CNN (Niepert et al. 2016)

The main difficulty in extending spatial convolution to general graphs is the lack of correspondence of the nodes. Niepert et al. (2016) proposed to first label the nodes so that they are somewhat in correspondence. Consider $\mathfrak{l} : \mathcal{V} \to \mathsf{L}$ that sends a node $v \in \mathcal{V}$ to a color $\mathfrak{l}_v$ in some totally ordered set L. For instance, $\mathfrak{l}$ could simply be the node degree or computed by the WL Algorithm 10.23 below. We proceed similarly as in CNN:

- The color $\mathfrak{l}$ induces an ordering of the nodes, allowing us to select a fixed number n of nodes, starting from the "smallest" and incrementing with stride s. We pad (disconnected) trivial nodes if run out of choices.
- For each chosen node v above, we incrementally select its neighbors $\mathcal{N}_v := \bigcup_d \{u : \operatorname{dist}(u, v) \leq d\}$ using breadth first search (BFS), until exceeding the receptive field size or running out of choice.
- We recompute colors on $\mathcal{N}_v$ with the constraint $\operatorname{dist}(u, v) < \operatorname{dist}(w, v) \implies \mathfrak{l}_u < \mathfrak{l}_w$. Depending on the size of $\mathcal{N}_v$, we either select a fixed number m of (top) neighbors and recompute their colors, or pad (disconnected) trivial nodes to make the fixed number m. Lastly, we perform canoniocalization using NAUTY (McKay and Piperno 2014) while respecting the node colors.
- Finally, we collect the results into tensors with size $n \times m \times d$ for d-dim node features and $n \times m \times m \times p$ for p-dim edge features, which can be reshaped to $nm \times d$ and $nm^2 \times p$. We apply 1-d convolution with stride and receptive field size m to the first and m^2 to the second tensor.

For grid graphs, if we use the WL Algorithm 10.23 to color the nodes, then it is easy to see that the above procedure recovers the usual CNN.

Niepert, Mathias, Mohamed Ahmed, and Konstantin Kutzkov (2016). "Learning Convolutional Neural Networks for Graphs". In: *Proceedings of The 33rd International Conference on Machine Learning*, pp. 2014–2023.
McKay, Brendan D. and Adolfo Piperno (2014). "Practical graph isomorphism, II". *Journal of Symbolic Computation*, vol. 60, pp. 94–112.

Definition 10.13: Graph Laplacian

Let A be the usual adjacency matrix of an (undirected) graph and D the diagonal matrix of degrees:

$$A_{uv} = \begin{cases} 1, & \text{if } (u,v) \in \mathcal{E} \\ 0, & \text{otherwise} \end{cases}, \quad D_{uv} = \begin{cases} \sum_v A_{uv}, & \text{if } u = v \\ 0, & \text{otherwise} \end{cases}.$$

More generally, we may consider a weighted graph with (nonnegative, real-valued and symmetric) weights $A_{uv} = w_{uv}$. We define the graph Laplacian and its normalized version:

$$L = D - A, \quad \bar{L} = I - D^{-1/2} A D^{-1/2} = D^{-1/2} L D^{-1/2}.$$

Among many other nice properties, the graph Laplacian is useful because of its connection to quadratic potentials. Too see this, let $\mathbf{x}_v \in \mathbb{R}^d$ be a feature vector at each node v and we verify that

$$\begin{aligned} \frac{1}{2}\sum_{u,v} A_{uv}\|\mathbf{x}_u - \mathbf{x}_v\|_2^2 &= \frac{1}{2}\sum_{u,v} A_{uv}[\|\mathbf{x}_u\|_2^2 + \|\mathbf{x}_v\|_2^2 - 2\langle \mathbf{x}_u, \mathbf{x}_v\rangle] = \sum_u d_u \|\mathbf{x}_u\|_2^2 - \sum_{u,v} A_{uv}\langle \mathbf{x}_u, \mathbf{x}_v\rangle \\ &= \operatorname{tr}(X(D-W)X^\top) = \operatorname{tr}(XLX^\top) = \sum_{j=1}^d X_{j:} L X_{j:}^\top, \quad X = [\ldots, \mathbf{x}_v, \ldots] \in \mathbb{R}^{d \times |\mathcal{V}|}. \end{aligned}$$

Taking $d = 1$ we see that the Laplacian L is symmetric and positive semidefinite. Similarly,

$$\operatorname{tr}(X\bar{L}X^\top) = \operatorname{tr}((XD^{-1/2})L(D^{-1/2}X^\top)) = \frac{1}{2}\sum_{u,v} A_{uv} \| \tfrac{\mathbf{x}_u}{\sqrt{d_u}} - \tfrac{\mathbf{x}_v}{\sqrt{d_v}} \|_2^2.$$

Of course, the normalized graph Laplacian is also symmetric and positive semidefinite.

Exercise 10.14: Laplacian and Connectedness

Prove that the dimension of the null space of the Laplacian is exactly the number of connected components in the (weighted) graph.

Moreover, $L\mathbf{1} = 0$, so the Laplacian always has 0 as an eigenvalue and $\mathbf{1}$ as the corresponding eigenvector.

Remark 10.15: Graph Laplacian is everywhere

The graph Laplacian played significant roles in the early days of segmentation, dimensionality reduction and semi-supervised learning, see Shi and Malik (e.g. 2000), Dhillon et al. (2007), Zhu et al. (2003), Zhou et al. (2004), Coifman et al. (2005), Belkin et al. (2006), Belkin and Niyogi (2008), Hammond et al. (2011), and Shuman et al. (2013). It allows us to propagate information from one node to another through traversing the edges and to enforce global consistency through local ones. Typical ways to construct graph from sampled data include thresholding pairwise distances or comparing node features.

Shi, Jianbo and J. Malik (2000). "Normalized cuts and image segmentation". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905.

Dhillon, I. S., Y. Guan, and B. Kulis (2007). "Weighted Graph Cuts without Eigenvectors A Multilevel Approach". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 11, pp. 1944–1957.

Zhu, Xiaojin, Zoubin Ghahramani, and John Lafferty (2003). "Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions". In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, pp. 912–919.

Zhou, Dengyong, Olivier Bousquet, Thomas N. Lal, Jason Weston, and Bernhard Schölkopf (2004). "Learning with Local and Global Consistency". In: *Advances in Neural Information Processing Systems 16*, pp. 321–328.

Coifman, R. R., S. Lafon, A. B. Lee, M. Maggioni, B. Nadler, F. Warner, and S. W. Zucker (2005). "Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps". *Proceedings of the National Academy of Sciences*, vol. 102, no. 21, pp. 7426–7431.

Belkin, Mikhail, Partha Niyogi, and Vikas Sindhwani (2006). "Manifold Regularization: A Geometric Framework for Learning from Labeled and Unlabeled Examples". *Journal of Machine Learning Research*, vol. 7, pp. 2399–2434.
Belkin, Mikhail and Partha Niyogi (2008). "Towards a theoretical foundation for Laplacian-based manifold methods". *Journal of Computer and System Sciences*, vol. 74, no. 8, pp. 1289–1308.
Hammond, David K., Pierre Vandergheynst, and Rémi Gribonval (2011). "Wavelets on graphs via spectral graph theory". *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150.
Shuman, D. I., S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst (2013). "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains". *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98.

Definition 10.16: Spectral convolutional networks on graphs (Bruna et al. 2014)

Bruna et al. (2014) also defined the spectral graph convolution of two graph signals $\mathbf{x} \in \mathbb{R}^{|\mathcal{V}|}$ and $\mathbf{g} \in \mathbb{R}^{|\mathcal{V}|}$ as:

$$\mathbf{x} * \mathbf{g} := U[(U^\top \mathbf{x}) \odot (U^\top \mathbf{g})], \quad \text{where} \quad L = U\Lambda U^\top$$

is the spectral decomposition of the graph Laplacian L and $\odot$ denotes component-wise multiplication. Let $\mathbf{g}$, or equivalently $\mathbf{w} := U^\top \mathbf{g}$, represent a filter. We then define a layer of spectral graph convolution as:

$$\mathbf{x}_r^{l+1} = \sigma\big(U[W_r^l \odot (U^\top X^l)]\mathbf{1}\big), \ r = 1, \dots, d_{l+1}, \ X^l = [\mathbf{x}_1^l, \dots, \mathbf{x}_{d_l}^l], \tag{10.3}$$

where d_l is the number of channels for layer l and $\sigma : \mathbb{R} \to \mathbb{R}$ is some component-wise (nonlinear) activation function. The formula (10.3) continues to make sense if we only take say bottom s_l eigenvectors in U (corresponding to the smallest eigenvalues). Thus, the number of filter parameters in $\mathcal{W}^l$ is $O(s_l d_l d_{l+1})$, which we may reduce through interpolating a few "landmarks": $W_r^l = B\boldsymbol{\alpha}_r^l$, where B is a fixed interpolation kernel and the few knots $\boldsymbol{\alpha}_r^l$ are tunable.

When a sequence of coarsenings $\mathcal{G}^l$ is available (like the spatial convolution in Definition 10.11), we can then perform pooling on the signal X^l by pooling the values in each neighborhood (corresponding to nodes in $\mathcal{V}^{l+1}$).

Henaff et al. (2015) also considered learning the graph topology and spectral convolution alternately.

Bruna, Joan, Wojciech Zaremba, Arthur Szlam, and Yann LeCun (2014). "Spectral Networks and Locally Connected Networks on Graphs". In: *International Conference on Learning Representations*.
Henaff, Mikael, Joan Bruna, and Yann LeCun (2015). "Deep Convolutional Networks on Graph-Structured Data".

Definition 10.17: Chebyshev polynomial

Let $\mathfrak{p}_0 \equiv 1$ and $\mathfrak{p}_1(x) = x$. For $k \geq 2$ we define the k-th Chebyshev polynomial recursively:

$$\mathfrak{p}_k(x) = 2x \cdot \mathfrak{p}_{k-1}(x) - \mathfrak{p}_{k-2}(x).$$

It is known that Chebyshev polynomials form an orthogonal basis for $L^2([-1, 1], \mathrm{d}x/\sqrt{1-x^2})$.

Example 10.18: Chebyshev Net (Defferrard et al. 2016)

The spectral graph convolution in Definition 10.16 is expensive as we need to eigen-decompose the Laplacian L. However, note that

$$\mathbf{x} * \mathbf{g} := U[(U^\top \mathbf{g}) \odot (U^\top \mathbf{x})] = U[\mathrm{diag}(f(\boldsymbol{\lambda}; \mathbf{w}))(U^\top \mathbf{x})] = [U \,\mathrm{diag}(f(\boldsymbol{\lambda}; \mathbf{w}))U^\top]\mathbf{x},$$

where we assume $U^\top \mathbf{g} = f(\boldsymbol{\lambda}; \mathbf{w})$ and recall the eigen-decomposition $L = U \,\mathrm{diag}(\boldsymbol{\lambda})U^\top$. The univariate function $f : \mathbb{R} \to \mathbb{R}$ is parameterized by $\mathbf{w}$ and is applied component-wise to a vector (and component-wise

to the eigenvalues of a symmetric matrix). Then, it follows

$$\mathbf{x} * \mathbf{g} = f(L; \mathbf{w})\mathbf{x},$$

and with a polynomial function $f(\lambda; \mathbf{w}) = \sum_{j=0}^{k-1} w_j \lambda^j$ we have $\mathbf{x} * \mathbf{g} = \sum_{j=0}^{k-1} w_j L^j \mathbf{x}$, where the polynomial L^j only depends on nodes within j edges hence localized. Using the Chebyshev polynomial we may then parameterize spectral convolution:

$$\mathbf{x} * \mathbf{g} = \sum_{j=0}^{k-1} w_j \mathfrak{p}_j(\tilde{L})\mathbf{x}, \qquad \text{where} \quad \tilde{L} := 2L/\|L\| - I,$$

whose spectrum lies in $[-1, 1]$. If we define $\mathbf{x}^j = \mathfrak{p}_j(\tilde{L})\mathbf{x}$, then recursively

$$\mathbf{x}^j = 2\tilde{L}\mathbf{x}^{j-1} - \mathbf{x}^{j-2}, \quad \text{with} \quad \mathbf{x}^0 = \mathbf{x},\ \mathbf{x}^1 = \tilde{L}\mathbf{x}.$$

The above recursion indicates that Chebyshev net is similar to a k-step unrolling of GNN with linear update functions.

Thus, computing the graph convolution $\mathbf{x} * \mathbf{g}$ costs only $O(k|\mathcal{E}|)$. We easily extend to multi-channel signals $X = [\mathbf{x}_1, \dots, \mathbf{x}_s] \in \mathbb{R}^{|\mathcal{V}| \times s}$ with filters $W_r = [\mathbf{w}_0^r, \dots, \mathbf{w}_{k-1}^r] \in \mathbb{R}^{s \times k}$:

$$\mathbf{x} * \mathbf{g}_r = \sum_{i=1}^{s} \sum_{j=0}^{k-1} w_{ij}^r \mathfrak{p}_j(\tilde{L})\mathbf{x}_i = \left[\mathfrak{p}_0(\tilde{L}), \cdots, \mathfrak{p}_{k-1}(\tilde{L})\right] \mathtt{vec}(XW_r), \quad r = 1, \dots, t,$$

where s and t are the number of input and output channels, respectively. Component-wise nonlinear activation is applied afterwards, and pooling can be similarly performed as before if a sequence of coarsenings is available.

Defferrard, Michaël, Xavier Bresson, and Pierre Vandergheynst (2016). "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering". In: *Advances in Neural Information Processing Systems 29*, pp. 3844–3852.

Definition 10.19: Graph convolutional network (GCN) (Kipf and Welling 2017)

Given a weighted graph $\mathcal{G} = (\mathcal{V}, A)$, a layer of GCN is defined concisely as:

$$X^{l+1} = \sigma\big(\mathring{D}^{-1/2}\mathring{A}\mathring{D}^{-1/2}X^l W^l\big), \quad X^l = [\mathbf{x}_1^l, \dots, \mathbf{x}_s^l] \in \mathbb{R}^{|\mathcal{V}| \times s}, \quad W^l \in \mathbb{R}^{s \times t}, \tag{10.4}$$

where $\mathring{A} = A + I$ (i.e. adding self-cycle), $\mathring{D}$ is the usual diagonal degree matrix of $\mathring{A}$, and s and t are the number of input and output channels, respectively.

GCN can be motivated by setting $k = 1$ and with weight-sharing $w_{i,0}^r = -w_{i,1}^r = w_i^r$ in Chebyshev net (see Example 10.18):

$$\mathbf{x} * \mathbf{g}_r = \sum_{i=1}^{s} (w_{i,0}^r I + w_{i,1}^r \tilde{L})\mathbf{x}_i = \sum_{i=1}^{s} w_i^r (I - 2L/\|L\| + I)\mathbf{x}_i.$$

If we use the normalized Laplacian and assume $\|\bar{L}\| = 2$, then

$$\mathbf{x} * \mathbf{g}_r = \sum_{i=1}^{s} w_i^r (I + D^{-1/2}AD^{-1/2})\mathbf{x}_i = (\underbrace{I}_{\text{self-loop}} + \underbrace{D^{-1/2}AD^{-1/2}}_{\text{1-hop neighbors}})X\mathbf{w}_r.$$

Comparing to (10.4), we see that GCN first adds the self-loop to the adjacency matrix to get $\mathring{A}$ and then renormalizes to get the 1-hop neighbor term $\mathring{D}^{-1/2}\mathring{A}\mathring{D}^{-1/2}$.

Comparing to Chebyshev net, 1 layer of GCN only takes 1-hop neighbors into account while Chebyshev net takes all k-hop neighbors into account. However, this can be compensated by stacking k layers in GCN. Kipf and Welling (2017) applied GCN to semi-supervised node classification where cross-entropy on labeled nodes is minimized while the unlabeled nodes affect the Laplacian hence also learning of the weights W.

Kipf, Thomas N. and Max Welling (2017). "Semi-Supervised Classification with Graph Convolutional Networks". In: *International Conference on Learning Representations*.

Example 10.20: Simple graph convolution (SGC) (Wu et al. 2019)

As mentioned above, GCN replaces a layer of Chebyshev net with k compositions of a simple layer defined in (10.4):

$$X \to \sigma(\mathring{L}XW^1) \to \cdots \to \sigma(\mathring{L}XW^k), \quad \mathring{L} := \mathring{D}^{-1/2}\mathring{A}\mathring{D}^{-1/2}.$$

Surprisingly, Wu et al. (2019) showed that collapsing the above leads to similar performance, effectively bringing us back to Chebyshev net with a different polynomial parameterization:

$$X \to \sigma(\mathring{L}^k XW).$$

Wu et al. (2019) proved that the self-loop in $\mathring{A}$ effectively shrinks the spectrum.

Wu, Felix, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger (2019). "Simplifying Graph Convolutional Networks". In: *Proceedings of the 36th International Conference on Machine Learning*, pp. 6861–6871.

Exercise 10.21: Multiplication is indeed composition

Prove that the mapping $\mathbf{x} \mapsto L^k\mathbf{x}$ depends only on k-hop neighbors.

Alert 10.22: The deeper, the worse? (Oono and Suzuki 2020)

Both GCN and SGC seem to suggest that we do not need to build very deep graph networks. This is possibly due to the small-world phenomenon in many real-word graphs, namely that each node can be reached from any other node through very few hops. See Oono and Suzuki (2020) for an interesting result along this direction.

Oono, Kenta and Taiji Suzuki (2020). "Graph Neural Networks Exponentially Lose Expressive Power for Node Classification". In: *International Conference on Learning Representations*.

Algorithm 10.23: Iterative color refinement (Weisfeiler and Lehman 1968)

Algorithm: Weisfeiler-Lehman iterative color refinement (Weisfeiler and Lehman 1968)

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathfrak{l}^0)$
Output: $\mathfrak{l}^{|\mathcal{V}|-1}$

for $t = 0, 1, \ldots, |\mathcal{V}| - 1$ **do**
  $\mathfrak{l}^{t+1} \leftarrow \texttt{hash}\Big([\mathfrak{l}_v^t, \mathfrak{l}_{u\in\mathcal{N}_v}^t] : v \in \mathcal{V}\Big)$ `// [·] is a multiset, allowing repetitions`

Algorithm: Assuming node features $\mathfrak{l}$ from a totally ordered space L

Function $\texttt{hash}\Big([\mathfrak{l}_v, \mathfrak{l}_{u\in\mathcal{N}_v}] : v \in \mathcal{V}\Big)$**:**
  for $v \in \mathcal{V}$ **do**
    $\texttt{sort}\Big(\mathfrak{l}_{u\in\mathcal{N}_v}\Big)$ `// sort the neighbors`
    add $\mathfrak{l}_v$ as prefix to the sorted list $[\mathfrak{l}_v, \mathfrak{l}_{u\in\mathcal{N}_v}]$ `// l_v does not participate in sorting!`
    $\mathfrak{l}_v^+ \leftarrow f([\mathfrak{l}_v, \mathfrak{l}_{u\in\mathcal{N}_v}])$ `// f : L* → L strictly increasing w.r.t. lexicographic order`

We follow Shervashidze et al. (2011) to explain the Weisfeiler-Lehman (WL) iterative color refinement algorithm. Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathfrak{l})$ with node feature $\mathfrak{l}_v$ in some totally ordered space L for each node $v \in \mathcal{V}$. For instance, we may simply set $\mathfrak{l}_v \equiv 1$ and $\mathsf{L} = \{1, 2, \ldots, |\mathcal{V}|\}$ (a.k.a. colors) if no better information is available. Then, for each node (in parallel) we repeatedly aggregate information from its neighbors and reassign its node feature using a hash function (which may change from iteration to iteration).

A typical choice for the hash function is illustrated above, based on sorting the neighbors and using a strictly increasing function $f : \mathsf{L}^* \to \mathsf{L}$ that maps the smallest neighborhood $[\mathfrak{l}_v, \mathfrak{l}_{u \in \mathcal{N}_v}]$ to the smallest element in L, and so on and so forth. (Note that in this convention f may change in different iterations in WL). By construction, the node feature $\mathfrak{l}_v$ for any node will never decrease (thanks to the monotonicity of f). W.l.o.g. we may identify $\mathsf{L} = \{1, 2, \ldots, |\mathcal{V}|\}$, from which we see that the algorithm need only repeat for at most $|\mathcal{V}|$ iterations: $|\mathcal{V}|^2 \geq \sum_v \mathfrak{l}_v \geq |\mathcal{V}|$ and each non-vacuous update increases the sum by at least 1. If we maintain a histogram on the alphabet L, then we may early stop the algorithm when the histogram stops changing. WL can be implemented in almost linear time (e.g. Berkholz et al. 2017).

As mentioned in this historic comment, WL was motivated by applications in computational chemistry, where a precursor already appeared in Morgan (1965). An interesting story about Andrey Lehman is available here while an unsettling story about the disappearance of Boris Weisfeiler is available here.

Weisfeiler, Boris and Andrey Lehman (1968). "The reduction of a graph to canonical form and the algebra which appears therein". *Nauchno-Technicheskaya Informatsia*, vol. 2, no. 9, pp. 12–16.
Shervashidze, Nino, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt (2011). "Weisfeiler-Lehman Graph Kernels". *Journal of Machine Learning Research*, vol. 12, no. 77, pp. 2539–2561.
Berkholz, C., P. Bonsma, and M. Grohe (2017). "Tight Lower and Upper Bounds for the Complexity of Canonical Colour Refinement". *Theory of Computing Systems*, vol. 60, pp. 581–614.
Morgan, H. L. (1965). "The Generation of a Unique Machine Description for Chemical Structures-A Technique Developed at Chemical Abstracts Service". *Journal of Chemical Documentation*, vol. 5, no. 2, pp. 107–113.

Algorithm 10.24: Graph isomorphism test

Testing whether two graphs are isomorphic is one of the few surprising problems in NP that we do not know if it is in NPC or P. The WL Algorithm 10.23 immediately leads to an early test for graph isomorphism: we simply "glue" the two input graphs as disjoint components into one graph and start with trivial labeling $\mathfrak{l}_v \equiv 1$. Run WL Algorithm 10.23. If at some iteration the histograms on the two components/graphs differ, then we claim "non-isomorphic." Otherwise we classify as "possibly isomorphic."

The above test was mistakenly believed to be a solution to graph isomorphism (Weisfeiler and Lehman 1968) but soon counterexamples were found. Nevertheless, Babai and Kucera (1979) and Babai et al. (1980) proved that for almost all graphs, the WL test is valid. The exact power of the WL test has been characterized in Arvind et al. (2015) and Kiefer et al. (2015).

Weisfeiler, Boris and Andrey Lehman (1968). "The reduction of a graph to canonical form and the algebra which appears therein". *Nauchno-Technicheskaya Informatsia*, vol. 2, no. 9, pp. 12–16.
Babai, L. and L. Kucera (1979). "Canonical labelling of graphs in linear average time". In: *20th Annual Symposium on Foundations of Computer Science*, pp. 39–46.
Babai, László, Paul Erdös, and Stanley M. Selkow (1980). "Random Graph Isomorphism". *SIAM Journal on Computing*, vol. 9, no. 3, pp. 628–635.
Arvind, V., Johannes Köbler, Gaurav Rattan, and Oleg Verbitsky (2015). "On the Power of Color Refinement". In: *Fundamentals of Computation Theory*, pp. 339–350.
Kiefer, Sandra, Pascal Schweitzer, and Erkal Selman (2015). "Graphs Identified by Logics with Counting". In: *Mathematical Foundations of Computer Science*, pp. 319–330.

Algorithm 10.25: High dimensional WL (e.g. Grohe 2017; Weisfeiler 1976, §O)

For any $k \geq 2$, we may *lift* the WL algorithm by considering k-tuples of nodes v in $\mathcal{V}^k$. Variations on the neighborhood $\mathcal{N}_\mathsf{v}$ include:

- WL_k: $\mathcal{N}_\mathsf{v} := [\mathcal{N}_{\mathsf{v},1}, \ldots, \mathcal{N}_{\mathsf{v},k}]$, where $\mathcal{N}_{\mathsf{v},j} = [\mathsf{u} \in \mathcal{V}^k : \mathsf{u}_{\setminus j} = \mathsf{v}_{\setminus j}]$.
- fWL_k: $\mathcal{N}_\mathsf{v} := [\mathcal{N}_{\mathsf{v},u} : u \in \mathcal{V}]$, where $\mathcal{N}_{\mathsf{v},u} = [(u, \mathsf{v}_2, \ldots, \mathsf{v}_k), (\mathsf{v}_1, u, \ldots, \mathsf{v}_k), \ldots, (\mathsf{v}_1, \mathsf{v}_2, \ldots, u)]$.

- sWL$_k$ (Morris et al. 2019): $\mathcal{N}_{\mathsf{v}} := [\mathsf{u} \in \mathcal{V}^k : |\mathsf{u} \cap \mathsf{v}| = k-1]$.

We initialize k-tuples u and v with the same node feature (color) if the (ordered) subgraph they induce are isomorhpic (and with the same node features inherited from the original graph). The WL Algorithm 10.23 will be denoted as WL$_1$; see (Grohe 2017, p. 84) on how to unify the description.

It is known that WL$_{k+1}$ is as powerful as fWL$_k$ (Grohe and Otto 2015). For $k \geq 2$, WL$_{k+1}$ is strictly more powerful than WL$_k$ (Cai et al. 1992; Grohe and Otto 2015, Observation 5.13 and Theorem 5.17), while WL$_1$ is equivalent to WL$_2$ (Cai et al. 1992; Grohe and Otto 2015). Moreover, sWL$_k$ is strictly weaker than WL$_k$ (Sato 2020, page 15).

Grohe, Martin (2017). *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Cambridge University Press.
Weisfeiler, Boris (1976). *On Construction and Identification of Graphs*. Springer.
Morris, Christopher, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe (2019). "Weisfeiler and Leman Go Neural Higher-Order Graph Neural Networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*.
Grohe, Martin and Martin Otto (2015). "Pebble Games and Linear Equations". *The Journal of Symbolic Logic*, vol. 80, no. 3, pp. 797–844.
Cai, J., M. Fürer, and N. Immerman (1992). "An optimal lower bound on the number of variables for graph identification". *Combinatorica*, vol. 12, pp. 389–410.
Sato, Ryoma (2020). "A Survey on The Expressive Power of Graph Neural Networks".

Remark 10.26: The connection between WL and GCN

The similarity between WL Algorithm 10.23 and GCN is recognized in (Kipf and Welling 2017). Indeed, consider the following specialization of the `hash` function in Algorithm 10.23:

$$\mathfrak{l}_v^{l+1} = \sigma\left(\left[\tfrac{1}{d_v+1}\mathfrak{l}_v + \sum_{u\in\mathcal{N}_v} \tfrac{a_{vu}}{\sqrt{(d_v+1)(d_u+1)}}\mathfrak{l}_u^l\right]W^l\right),$$

which is exactly the GCN update in (10.4) (with the identification $X_{v:} = \mathfrak{l}_v$). From this observation we see that even with random weights W, GCN may still be able to extract useful node features, as confirmed through an example in (Kipf and Welling 2017, Appendix A.1).

More refined and exciting findings along this connection have appeared in Xu et al. (e.g. 2019), Maron et al. (2019), Morris et al. (2019), and Sato (2020) lately. See also Kersting et al. (2009) and Kersting et al. (2014) for applications to graphical models.

Kipf, Thomas N. and Max Welling (2017). "Semi-Supervised Classification with Graph Convolutional Networks". In: *International Conference on Learning Representations*.
Xu, Keyulu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka (2019). "How Powerful are Graph Neural Networks?" In: *International Conference on Learning Representations*.
Maron, Haggai, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman (2019). "Provably Powerful Graph Networks". In: *Advances in Neural Information Processing Systems 32*, pp. 2156–2167.
Morris, Christopher, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe (2019). "Weisfeiler and Leman Go Neural Higher-Order Graph Neural Networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*.
Sato, Ryoma (2020). "A Survey on The Expressive Power of Graph Neural Networks".
Kersting, Kristian, Babak Ahmadi, and Sriraam Natarajan (2009). "Counting Belief Propagation". In: *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 277–284.
Kersting, Kristian, Martin Mladenov, Roman Garnett, and Martin Grohe (2014). "Power Iterated Color Refinement". In: *The Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI)*.