CS 768: Learning With Graphs **Autumn 2020-2021**

Lecture 13: Graph Convolutional Networks

Instructor: Prof. Abir De *Scriber: Shrutesh Patil, Shubhamkar Ayare*

13.1 Introduction

Consider a node u. We'd like to learn its embedding $x_u = f(\dots)$ (final *NN* below), the peculiarity being that we'd like the embedding to be such that we aren't required to change the model when a new node comes into picture. Such a model is known as *inductive model.*

To do this, with f_u as the inherent features[1] of the node, consider

$$x'_u(0) = W_0 f_u$$
$$x'_u(1) = g_1\left[x'_u(0) \odot g_2\left(\sum_{v\in N(u)} x'_v(0)\right)\right]$$
$$\vdots$$
$$x'_u(k+1) = g_1\left[x'_u(k) \odot g_2\left(\sum_{v\in N(u)} x'_v(k)\right)\right]$$

Notice the recursive formulation of the x'$_{\text{u}}$(i)'s above. This constitutes the most generic form of *Graph Convolutional Network*, as also the most rudimentary form of *Graph Neural Network.*

Here, g$_1$, g$_2$ and NN are just non-linearities. g$_1$, g$_2$ need not be the same for each "hop" - these could very well have been $g_1^{(1)}$, $g_1^{(2)}$, etc. These g$_1$, g$_2$ could also have been set from the outside. The formulation itself is independent of the number of nodes or edges of the graph. The operator '$\odot$' too may vary from formulation-to-formulation.

The *number of hops* (k) constitutes a hyper-parameter.

As an exercise[2], find the choices of g$_1$, g$_2$ and $\odot$, so that the above can be reduced to page-rank type formulation below. Consider A to be a matrix

$$x'_u(k+1) = x'_u(k) + \gamma A\left(\sum_{v\in N(u)} x'_v(k)\right)$$

And while for page-rank, one'd expect the embedding size to depend on the size of the graph, in general, we'd like the embeddings to be a compressed representative of that information with a smaller overall size.

[1] In Knowledge Graphs, inherent features could mean pre-trained embedding vectors; in Social Networks, they could be gender, location, etc

[2] Solution to the exercise: g$_1$ is the Identity operation, '$\odot$' is '+', g$_2$ is multiplication by matrix A

With the n steps/hops, this yields us n embeddings: (x'$_u$(1), x'$_u$(2), . . . , x'$_u$(n)). These can be passed into a final output layer to obtain the final embedding

$$x_u = NN(x'_u(1), ..., x'_u(n))$$

g$_1$ and g$_2$ may be chosen such that this results in a *Linear–RELU–Linear* formulation. This constitutes a *universal function approximator*. (The *linear* itself may comprise of several layers.)

To actually find NN, we note that it should be such that x$_u$ should be rendered permutation invariant.

13.2 Learning the embedding function: generative method

In order to compute x$_u$, we need to learn NN, $\{\dot{g_1}\}$, $\{\dot{g_2}\}$. We need a loss function. For this, we have several choices.

- We can use a classifier model, so that $x_u^T x_v \rightarrow +/-1$
- Shortest path distances
- Generative Model

Here we discuss the last option. Consider

$$P(e|x_u, x_v) = \sigma(x_u^T x_v)$$

So that

$$P(G|\{x_u\}, \{x_v\}) = \prod_{(u,v)\in E} \sigma(x_u^T x_v) \prod_{(u,v)\notin E} (1 - \sigma(x_u^T x_v))$$

In logarithmic form,

$$\log P(G|\{x_u\}, \{x_v\}) = \sum_{(u,v)\in E} \log \sigma(x_u^T x_v) \sum_{(u,v)\notin E} \log(1 - \sigma(x_u^T x_v))$$

However, with this, the time complexity quickly becomes impractical[3]. To reduce this complexity, we make the loss function depend only on the likelihood of edges, and not on the non-edges.

Consider a graph with 3 edges labelled 1,2,3 generated in that temporal order[4]; note that

$$P(1, 2, 3) = P(3|1, 2)P(2|1)P(1)$$

[3]The naive formulation of $P(G|\{x_u\}, \{x_v\})$ requires $O(|V|^2)$ computation time. For realistic graphs, $E = O(|V|)$, and so it might be possible to avoid $O(|V|^2)$ using approximations.

[4]So, the edge labelled 1 is the very first edge, then the edge labelled 2 appears, and then finally the edge labelled 3 appears.

This requires just $O(|E|)$ computation time.

However, this is order-dependent, and we would like to make it order-independent[5].

To do this, we consider the average over all-possible permutations $\mathrm{S_i}$ of the edges; let

- $\mathbb{P}(E)$ denote the set of all permutations of the edges in edge-set E; an example of a permutation is the sequence $\mathrm{S_1} = \{\mathrm{e_1}, \ldots, \mathrm{e}_{|E|}\}$.
- $\mathrm{P(S_i)}$ denote the probability of the graph generated according to the sequence $\mathrm{S_i}$
- $\mathrm{Prior(S_i)}$ be the probability of sequence $\mathrm{S_i}$ from $\mathbb{P}(S_i)$ sampled uniformly using BFS/DFS

so that the required average is

$$\frac{1}{|\mathbb{P}(S)|} \sum_{S_i \in \mathbb{P}(E)} P(S_i)$$

or

$$P(G) = \mathop{\mathbb{E}}_{S_i \sim Uniform(\mathbb{P}(E))} [P(S_i)]$$

However, this itself does not reduce the complexity. To reduce the complexity, we incorporate sampling, and obtain the expectation over, say, only 10 to 15 samples. In essence,

$$\begin{aligned} P(G) &= \sum_{S_i} P(G|S_i) Prior(S_i) \\ &= \sum_{S_i} P(S_i) Prior(S_i) \\ &= \mathop{\mathbb{E}}_{S_i \sim Uniform(\mathbb{P}(E))} [P(S_i)] \end{aligned}$$

To compute

$$\mathrm{P(S_i) = P(e_1, \ldots, e_n) = P(e_n — e_1, \ldots, e_{n\text{-}1})\ P(e_{n\text{-}1} — \ldots)} \ldots$$

with $e_{n-1} = (u, v)$, note that

$$P(e_{n-1}|e_1 ... e_{n-2}) = \frac{\sigma(x_u^T x_v)}{\sum_{w \in \{w|(u,w) \notin \{e_1, \ldots, e_{n-2}\}\}} \sigma(x_u^T x_w)}$$

However, this still takes O(—V—) for each edge, with $O(\eta|E||V|)$ as the overall complexity for η samples.

To actually reduce the complexity, we only consider $(u, v) \in N(u) \setminus e_1, ..., e_{n-2}$. Observe that the complexity then becomes $O(d_{\mathrm{max}})$ for each edge with $O(\eta d_{max}|E|)$ as the overall complexity. (d is the degree of the node.)

[5]Two isomorphic graphs[6]with edge sets {ab, bc, cd, da, ac} and {ab, bc, cd, da, bd} and common node set {a, b, c, d} should have the same probability[7]; however, our current model does not necessarily awards them same probabilities

[6]See `https://www.math.upenn.edu/~mlazar/math170/notes05-2.pdf` for examples of isomorphic graphs

[7]Also, P(3—1,2) = P(3—2,1); our embeddings are order-independent.

13.3 Generating a new graph from the learnt model

Next, we wish to generate a graph from the learnt parameters {g$_1$, g$_2$, NN}.

We do this incrementally starting from, say, node labelled 0 using a multinomial distribution. So, for the first edge, we use the multinomial distribution

$$\left\{ \frac{\sigma(x_0^T x_i)}{\sum_{i \neq 0} \sigma(x_0^T x_i)} \right\}_{i \neq 0}$$

Once the first edge is generated, we update the embedding x$_0$ of the node labelled 0.

For the second edge, we use the distribution

$$\left\{ \frac{\sigma(x_1^T x_i)}{\sum_{i \neq 0,1} \sigma(x_1^T x_i)} \right\}_{i \neq 0,1}$$

And so on.

To incorporate noise, consider that the actual embedding z$_u$ is close to the embedding x$_u$ output by the GCN but not exactly x$_u$: $z_u \sim \mathcal{N}(x_u, \sigma(x_u))$. And so, the edge probability weights are $\sigma(z_u^T z_v)$ rather than $\sigma(x_u^T x_v)$. In this case, see that the probability of the graph is

$$P(G) = \sum_{z|x} P(G|z) P(z|x) = \mathop{\mathbb{E}}_{z|x} [P(G|z)]$$

However, there are $|V|$ z's. To avoid this, we use sampling; and then our problem concerns maximizing

$$\log \mathop{\mathbb{E}}_{z|x} [P(G|z)] = \log \left(\frac{1}{\eta} \sum_{z_i \sim \mathcal{N}(\dots)} P(G|z) \right)$$

But, because log is concave[8], we have

$$\log \mathop{\mathbb{E}}_{z|x} [P(G|z)] \geq \mathop{\mathbb{E}}_{z|x} \log P(G|z)$$

And then, this can be simplified to

$$\mathop{\mathbb{E}}_{z|x} \log P(G|z) = \mathop{\mathbb{E}}_{z|x} \left[\sum_{i \in |E|} \log P(e_i | e_1, \dots, e_{i-1}) \right]$$

[8]Refer midsem question 2, or equivalently the Jensen's inequality