L6: Non-linear models, deep learning architectures for networked datasets

Caterina De Bacco

May 12, 2025

1. Introduction

In the previous lecture we learned how to model networks with probabilistic models and latent variables.

> *Question*: How about deep learning?

One can ask what do we need all these models for now that we can have all the powerful predictive tools from deep learning? In this lecture we address this question.

2. Graph neural networks

The first step to answering that question is to understand how do deep learning methods operate on networked data.
Graph neural networks (GNN) are the deep learning tools that take in input an adjacency matrix $A \in \mathbb{R}^{N \times N}$, in addition to features $X \in \mathbb{R}^{N \times K}$ on nodes.

> *Question*: How is A being used?

GNNs use the information about edges to *aggregate* features based on network neighborhoods.
This is in contrast to standard neural networks (NN) where aggregation happens, but is not driven by A.
We denote a generic aggregation function of vectors with the symbol $\oplus$. Typically one considers permutation invariant aggregations, such as sum, mean, max, etc ....
Beside this, all of the other ingredients of a GNNs are similar to those of NNs, i.e. feature transformations and update functions.
Here is the overall list of ingredients for GNNs:

- neighborhood $\partial_i := \left\{ j : A_{ij} > 0 \right\}$ (GNN-specific)
- set of features of the neighbors $X_{\partial_i} = \left\{ x_j | j \in \partial_i \right\}$(GNN-specific)
- transformation $\psi(x_i)$, transforms the feature vector x_i of node i

- update $\phi(x_i, X_{\partial i})$, updates the features of node i, given that of its neighbors (GNN-specific)

Example cases are:

$$\begin{aligned} \psi(x) &= Wx + b \quad \text{affine transformation} \qquad (1) \\ \phi(x, z) &= \sigma(\psi(x) + Uz) \quad \text{activation function} \quad , \qquad (2) \end{aligned}$$

where W, b, U are learnable parameters and σ is an activation function, e.g. rectified linear unit (ReLU).

2.1. Focus on prediction tasks

An important point to make about GNNs is the following remark.

REMARK 1. *A is usually not the central variable under focus, it is rather an auxiliary dataset to be used to improve performance in prediction tasks that focus on nodes.*

For instance, one may want to learn labels on nodes, as in node classification, aided by the knowledge of A in addition to features X, as opposed to standard (without a graph) node classification that only relies on X.
Sometime link prediction on A is also considered as a downstream task, but one relies on the availability of X also in this case, where other methods (e.g. Probabilistic Graphical Models, PGM) do not need that type of input to solve a link prediction task. In the absence of a genuine feature matrix, one has to build a dummy matrix, e.g. the identity or using the rows of A as features.
An example application that appears in various works is the analysis of citation networks, where nodes are papers and edges are citations between them. In addition to this network, one has access to features extracted from bags of words representations, and have some different (usually categorical) attributes that can be interpreted as labels, e.g. topics, to be learned in a node classification task.

2.2. Types of GNNs

Given the general structure above, there exists different types of GNNs based on the particular choices of ingredients. Here we describe the 3 main ones. We define the updated representations at any given layer ℓ of the GNN as $h_i^{(\ell)} \in \mathbb{R}^F$, where F is the representation dimension at that layer (we may omit sometime the layer index to keep it simple). At layer 0, we have the exact correspondence between representation and input feature: $h_i^{(0)} = x_0$.

Convolutional. Aggregation is done with fixed weights c_{ij} as:

$$h_i^{(\ell+1)} = \phi\left(h_i^{(\ell)}, \bigoplus_{j \in \partial_i} c_{ij}\, \psi(h_j^{(\ell)})\right) \quad . \qquad (3)$$

The weights c_{ij} specify the importance of node j in determining the representation of node i. It can be chosen based on A, e.g. proportional to the degree of node j.

Attentional. Aggregation is done with implicit interaction weights $a(x_i, x_j)$ as:

$$h_i^{(\ell+1)} = \phi\left(h_i^{(\ell)}, \bigoplus_{j \in \partial_i} a(h_i^{(\ell)}, h_j^{(\ell)})\, \psi(h_j^{(\ell)})\right) \quad . \tag{4}$$

The weights $a(h_i^{(\ell)}, h_j^{(\ell)})$ are learnable "attention" mechanisms that compute the importance coefficients $\alpha_{ij} := a(h_i^{(\ell)}, h_j^{(\ell)})$ implicitly. Note that these are feature-dependent, as they depend on the $h_i^{(\ell)}, h_j^{(\ell)}$.
For instance, in the work Veličković *et al.* (2017) they propose:

$$\begin{aligned} \epsilon_{ij} &= LeakyReLU\left(\bar{a}[Wh_i^{(\ell)} || Wh_j^{(\ell)}]\right) \in \mathbb{R} & (5) \\ a(h_i^{(\ell)}, h_j^{(\ell)}) &= \text{softmax}_j\left(\epsilon_{ij}\right) \quad , & (6) \end{aligned}$$

where $\bar{a} \in \mathbb{R}^{2F'}$ is a learnable vector of parameters, $W \in \mathbb{R}^{F' \times F}$ is a weight matrix that transforms an F-dimensional representation $h_i^{(\ell)}$ into a (usually) higher-dimensional representation of dimension F'; $||$ denotes concatenation operation.

Message-passing. Aggregation is done via messages $\nu(h_i^{(\ell)}, h_j^{(\ell)})$ as in a message-passing routines:

$$h_i = \phi\left(h_i^{(\ell)}, \bigoplus_{j \in \partial_i} \nu(h_i^{(\ell)}, h_j^{(\ell)})\right) \quad . \tag{7}$$

The weights $\nu(h_i^{(\ell)}, h_j^{(\ell)})$ are learnable "message" sent through the network.
From the 3 types, we can conclude that there is an hierarchy, where convolutional is a particular case of attention, and attention is a particular case of message-passing routine.

Variational graph autoencoder. Here the autoencoder part is used to encapsulate another GNNs (e.g. a convolutional one, GCN) into the framework of variational autoencoders (VAE). This means that there are latent variables $Z \in \mathbb{R}^{N \times K}$ that one wants to learn as in a VAE, i.e. using a variational distribution as estimate of the posterior on Z:

$$\begin{aligned} q(Z|A, X) &= \prod_i^N q(z_i|X, A) \quad \text{with} \quad q(z_i|X, A) = \mathcal{N}\left(z_i|\mu_i, \text{diag}(\sigma_i^2)\right) & (8) \\ \mu &= GCN_\mu(X, A) & (9) \\ \log \sigma &= GCN_\sigma(X, A) \quad . & (10) \end{aligned}$$

Hence, the key aspect is that the parameters of the variational distributions are parametrized using a graph neural network.
In Kipf and Welling (2016) they then specify a likelihood function for the network as:

$$\begin{aligned} P(A|Z) &= \prod_{i,j} P(A_{ij}|z_i, z_j) & (11) \\ P(A_{ij} = 1|z_i, z_j) &= \sigma(z_i \cdot z_j) \quad , & (12) \end{aligned}$$

where $\sigma(\cdot)$ is the logistic function. This resembles what seen in previous lectures with mixed-membership models and dot product factorization. The main difference is the non-linearity introduced by the GCN architectures to learn the parameters of the variational distributions.
Learning is then performed as usual in VI by maximizing the ELBO:

$$ELBO(q) = \mathbb{E}_q \left[\log P(A|Z)\right] - KL\left(q(Z|X,A)||P(Z)\right) \quad , \tag{13}$$

where $P(Z)$ is the prior.
One can similarly specify a non-probabilistic autoencoder.

> *Question*: How?

3. Benefits and weaknesses: non-linearity and oversmoothing

Now that we an idea of how these models use the networks data A, we can ask:

> *Question*: What is the advantage of using GNNs?

- **Non-linearity**. Certain datasets may benefits from non-linearities, as those brought in by a deep learning architecture. In contrast, PGMs are usually linear.
- **Easy incorporation of node features**. A matrix X can be simply incorporated as usual in a neural network, no need for additional and custom analytic calculations as often done in PGMs. As they are black-boxes, one does not have to worry about making ad-hoc derivations depending on the choices of the model architecture. This allows to solve node classification tasks or to improve link prediction when features are informative.

> *Question*: Are there any drawbacks in using them, as opposed to linear models like PGMs?

Indeed the two main benefits above can backfire, and become weaknesses, depending on the application.

- **Overfitting**. To allow for non-linearities we need to account for a possibly large number of parameters, hence the risk of overfitting. Typically, graph data are sparse and many dataset are small, as the number of node can be order of hundreds and only one network sample is observed. In contrast, image datasets can have thousands of images stored.
- **Relying on features.** Many network datasets do not come with node features. However, GNNs need to be given an X in input. Hence, one has to build a dummy feature. On the contrary, if a feature is available, it is not clear how GNNs can select whether to use it or not, in case this is uninformative and may even hurt link prediction. Instead, PGMs that can incorporate node features are usually developed with mechanisms to learn the feature importance and are capable of discarding it if necessary.
- **Interpretability**. As in all black-box methods, it is not clear how to interpret the output parameters, and one needs a posteriori ad-hoc methods. E.g. K-means on the final layer embeddings.

- **Oversmoothing**. Node features in GNNs tend to become more similar with the increase of the architecture depth. This leads to the so called *oversmoothing* issue, the exponential convergence of all node features towards the same constant value as the number of layer increases. To prevent this, GNNs architectures are often set with a small number of layers, e.g. 2. In contrast, NNs used in computer vision rely on tens or hundreds of layers.
- **Heterophily or disassortativity**. Because of the way they aggregate over node neighbors, GNNs may perform poorly in datasets with a disassortative community structure. This is further exacerbated by the oversmoothing, as long-range correlations need to be learned but cannot if we only set 2 layers.
  For an extensive discussion about oversmoothing you can read Rusch *et al.* (2023).

4. Deep learning architectures for networked datasets: summary

- GNNs use networked data in input to aggregate representations using neighborhoods
- The other ingredients remain the same as those of NNs
- They require node features in input, also when only doing link predictions
- They have a list of benefits and weaknesses (like any other model), need to keep this in mind when using them

References

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, arXiv preprint arXiv:1710.10903 (2017).

T. N. Kipf and M. Welling, arXiv preprint arXiv:1611.07308 (2016).

T. K. Rusch, M. M. Bronstein, and S. Mishra, arXiv preprint arXiv:2303.10993 (2023).