Introduction to Graph Neural Networks

Disclaimer: Work in progress. Portions of these written materials are incomplete.

Motivation: Supervised learning

- This is a (supervised) machine learning problem.

- Four examples, features (f_i) and labels (y_i).
- Good enough for science. ✓

Motivation: Supervised learning

- This is a (supervised) machine learning problem.

- Four examples, features (f_i) and labels (y_i).
- Good enough for science. **Not Aperture Science!**

Motivation: Supervised learning

- This is a (supervised) machine learning problem.

- Four examples, features (f_i) and labels (y_i).

- Good enough for science. **Not Aperture Science!** X
  - I give you **graphs**. *The inputs of tomorrow!*

Introduction

- In this talk: **neural networks** for **graph-structured data**
  - *Graph Neural Networks* or **GNN**s

- Recently a very hot topic in machine learning research:
  - **Fastest-growing** area at ICLR'20;
  - One of the **top workshops** at NeurIPS'19;
  - Applied at the LHC, Pinterest, Fabula AI.
  - Used to discover **novel antibiotics.**

1 Graph data processing

Mathematical setup

- **Graph:** $G = (V, E)$
- **Node features:** $\mathbf{H} = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}; \qquad \vec{h}_i \in \mathbb{R}^F$
- **Adjacency matrix:** $\mathbf{A} \in \mathbb{R}^{N \times N}$
- **Neighbourhoods:** $\mathcal{N}_i = \{j \mid i = j \vee \mathbf{A}_{ij} \neq 0\}$
- **(Edge features):** $\vec{e}_{ij} \in \mathbb{R}^{F'}; \qquad (i, j) \in E$

- We will focus on **node classification**.
  - Can easily extend to *link prediction* and *graph classification* by reusing node features.
  - Two standard paradigms of learning in this space…

Transductive learning

Training algorithm sees *all features* (**including test nodes**)!

Inductive learning

- Here, the algorithm *does not have access to all nodes upfront*!

- This often implies that either:
  - Test nodes are (incrementally) inserted into training graphs;
  - Test graphs are **disjoint** and *completely unseen*!

- A much harder learning problem (requires generalising across *arbitrary graph structures*)!

- Many transductive methods will be inappropriate for inductive problems.

The silver bullet---a *convolutional* layer

- Graph can be seen as a strict generalisation of **images**.
    - Treat any image as a *“grid graph”*.
    - Each node corresponds to a pixel; adjacent to its four neighbours.

- CNNs leverage the *convolutional operator* to extract the spatial regularity in images

- It would be highly appropriate if we could somehow generalise it to operate on arbitrary graphs!

Convolution on images

Graph *Convolutional* Network

$$\vec{h}'_i = g(\vec{h}_a, \vec{h}_b, \vec{h}_c, \dots) \qquad (a, b, c, \dots \in \mathcal{N}_i)$$

Challenges with graph convolutions

- Desirable properties for a graph convolutional layer:
    - **Computational and storage efficiency** (~O(V + E));
    - **Fixed** number of parameters (independent of input size);
    - **Localisation** (acts on a *local neighbourhood* of a node);
    - Specifying **different importances** to different neighbours;
    - Applicability to **inductive problems**.

- Fortunately, images have a highly rigid and regular connectivity structure, making such an operator trivial to devise (small kernel matrix slid across image).

- Arbitrary graphs are a **much harder** challenge!

2 GCNs, GATs and MPNNs

Towards a simple update rule

- Let's assume our graph is *unweighted* and *undirected*.
    - That is:

$$\mathbf{A}_{ij} = \mathbf{A}_{ji} = \begin{cases} 1 & i \leftrightarrow j \\ 0 & \text{otherwise} \end{cases}$$

- We can then easily aggregate neighbourhoods through multiplying by the adjacency matrix:

$$\mathbf{H}' = \sigma(\mathbf{AHW})$$

where **W** is a learnable node-wise shared linear transformation, and σ is a nonlinearity.

- A few things need to be fixed...

Towards a simple update rule, *cont'd*

- Firstly, this update rule discards the central node.
    - Provide a simple correction:

$$\mathbf{H}' = \sigma(\tilde{\mathbf{A}}\mathbf{H}\mathbf{W})$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$.

- The update rule can now be rewritten, node-wise, as:

$$\vec{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i} \mathbf{W}\vec{h}_j\right)$$

The mean-pooling update rule

- Secondly, multiplication by **A** may increase the scale of the output features.
  - We need to normalise appropriately, e.g. by

$$\mathbf{H}' = \sigma(\mathbf{D}\tilde{}^{-1}\tilde{\mathbf{A}}\mathbf{H}\mathbf{W})$$

where **D** is the degree matrix of **A**, i.e. $\tilde{\mathbf{D}}_{ii} = \sum_j \mathbf{A}_{ij}$.

- We arrive at the *mean-pooling* update rule:

$$\vec{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i} \frac{1}{|\mathcal{N}_i|}\mathbf{W}\vec{h}_j\right)$$

which is simple but versatile (common for *inductive* problems!).

GCN (Kipf & Welling, ICLR 2017)

- If we instead use *symmetric normalisation*:

$$\mathbf{H}' = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}\mathbf{W})$$

  we obtain the **graph convolutional network** (**GCN**) update rule!

- Node-wise, this can be written as follows:

$$\vec{h}'_i = \sigma\left(\sum_{i\in\mathcal{N}_j}\frac{1}{\sqrt{|\mathcal{N}_i||\mathcal{N}_j|}}\mathbf{W}\vec{h}_j\right)$$

  And it currently represents the most popular graph convolutional layer!

- Simple and powerful, albeit not inductive.

Towards a more general update rule

- The GCN model only indirectly supports *edge features*.

- One way to correct this is to instead focus on *edge-wise* mechanisms.

- Most generally, nodes can send **messages** (arbitrary vectors) along graph edges!
    - These messages can be *conditioned* by edge features.

- A node then **aggregates** all messages sent to it (using a *permutation-invariant* function).

MPNN (Gilmer *et al.*, ICML 2017)

- Let $\vec{m}_{ij}$ be the **message** sent across edge $i \rightarrow j$, computed using a *message function*, f_e :

$$\vec{m}_{ij} = f_e(\vec{h}_i, \vec{h}_j, \vec{e}_{ij})$$

- Now, **aggregating** all messages entering a node, along with a *readout function*, f_v :

$$\vec{h}_i' = f_v\left(\vec{h}_i, \sum_{j \in \mathcal{N}_i} \vec{m}_{ji}\right)$$

  we arrive at the *message-passing neural network* (**MPNN**)!

  - f_e and f_v are usually (small) MLPs.

MPNN: Initial setup

MPNN: Computing messages

MPNN: Aggregating messages

MPNN: Next-level features

MPNN: Next-level features

Towards a "golden middle"

- The MPNN is the *most potent* GNN layer.

- However:
  - It requires storage and manipulation of *edge messages*;
  - Troublesome both memory- and representationally-wise;
  - In practice, only applicable to *small* graphs.
  - (Can think of them as MLPs of the graph domain)

- As an intermediate approach, let's consider a more general form of the GCN:

$$\vec{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}\mathbf{W}\vec{h}_j\right)$$

- GCNs define the coefficient α_{ij} **explicitly**, causing several shortcomings.

GAT (Veličković *et al.*, ICLR 2018)

- If, instead, we let α_{ij} be computed *implicitly*...

$$a_{ij} = a(\vec{h}_i, \vec{h}_j, \vec{e}_{ij})$$

$$\alpha_{ij} = \frac{\exp(a_{ij})}{\sum_{k\in\mathcal{N}_i} \exp(a_{ik})}$$

  where *a* is a learnable, shared, *self-attention mechanism* (e.g. Transformer)...

- We arrive at the *graph attention network* (**GAT**) update rule!
  - In practice, significantly stabilised through *multi-head attention*.
  - Probably not as general as MPNNs, but trivially scalable.
    - **NB** Attention computes *scalar* per edge, message function computed *vector*!

A single GAT step, visualised

3 Scaling up GNN computations

DiffPool (Ying *et al.*, NeurIPS 2018)

Differentiable graph pooling: Learn *soft cluster assignments* that gradually harden using entropy penalty

GraphSAGE (Hamilton *et al.*, NIPS 2017)

Handle large graphs by **subsampling** around each node (can even be *uniform*!)

PinSAGE (Ying *et al.*, KDD 2018)

Taking GraphSAGE to the extreme: apply to Pinterest graph (**3bn** nodes, **18bn** edges)