Deep Learning and Applied Artificial Intelligence 2020
Virality Prediction

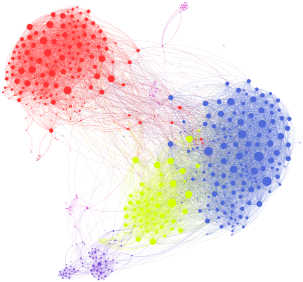
Andrea Caciolai, Donato Crisostomi

Master Degree in
Computer Science
Sapienza, University of Rome

A.Y. 2019 - 2020

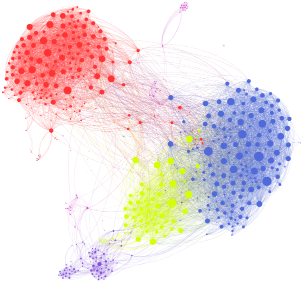
Motivation

- What is virality? Many phenomena exhibit **spreading behaviour**, like diseases or **news**, or in general **information** in a community.



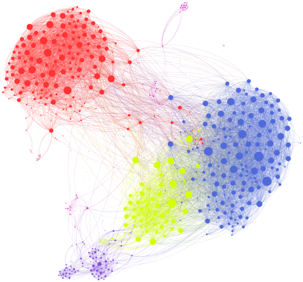
Motivation

- What is virality? Many phenomena exhibit **spreading behaviour**, like diseases or **news**, or in general **information** in a community.
- The ability to predict the spreading potential is valuable.



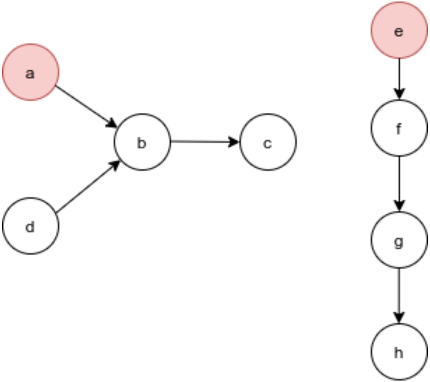
Motivation

- What is virality? Many phenomena exhibit **spreading behaviour**, like diseases or **news**, or in general **information** in a community.
- The ability to predict the spreading potential is valuable.
- **Graphs** serve as an useful abstraction to model real world situations, and are well suited to represent spreading patterns:
  - **nodes** represent components of interest (e.g. users in a social network);
  - **edges** define existing relations among these components;
  - **node signal** represents the information.



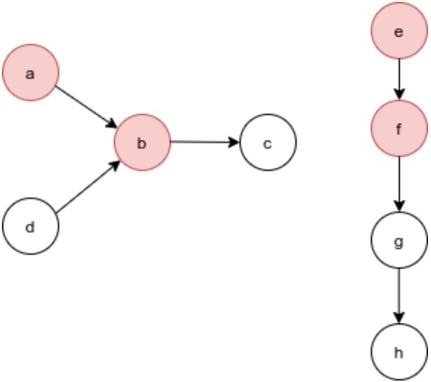
Cascades

The spread of a piece of information m originates a set of **cascades** in the network.



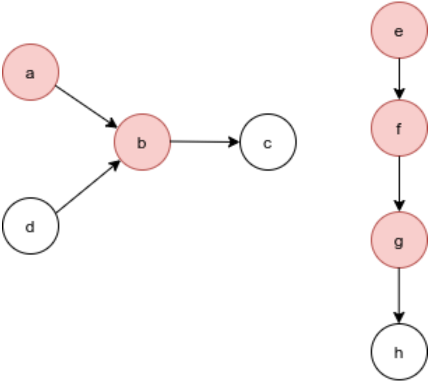
Cascades

The spread of a piece of information *m* originates a set of **cascades** in the network.



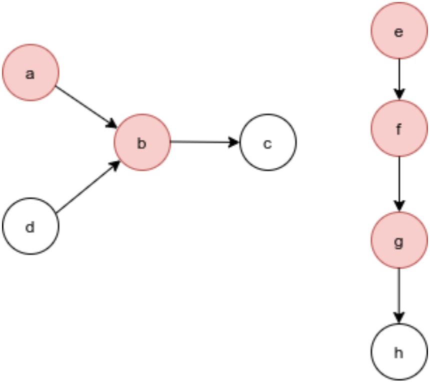
Cascades

The spread of a piece of information *m* originates a set of **cascades** in the network.



Cascades

The spread of a piece of information m originates a set of **cascades** in the network.



Early adopters $= \{a, e\}$ Final adopters $= \{a, b, e, f, g\}$

Formalization

- For our task, we distinguish two sets of nodes:
  1. **early adopters**, nodes producing the information;
  2. **final adopters**, nodes who receive the information from other nodes.

Formalization

- For our task, we distinguish two sets of nodes:
  1. **early adopters**, nodes producing the information;
  2. **final adopters**, nodes who receive the information from other nodes.
- Accordingly, each node v will be characterized by the following two features
  - whether it is an early adopter:

$$s_v^{(0)} = \text{initial activation state of node } v$$

  - and whether it is a final adopter, which is the label we want to predict:

$$s_v^{(K)} = \text{final activation state of node } v$$

Formalization

- For our task, we distinguish two sets of nodes:
  1. **early adopters**, nodes producing the information;
  2. **final adopters**, nodes who receive the information from other nodes.
- Accordingly, each node v will be characterized by the following two features
  - whether it is an early adopter:
    $$s_v^{(0)} = \text{initial activation state of node } v$$
  - and whether it is a final adopter, which is the label we want to predict:
    $$s_v^{(K)} = \text{final activation state of node } v$$
- The final virality coefficient for the piece of information m is obtained by counting the final adopters

$$n_\infty^m = \cdots = n_{K+1}^m = n_K^m = \sum_{v \in \mathcal{V}} s_v^{(K)}$$

Approaches

- **feature-based** methods and **representation learning** methods;
- hand-crafted features needed for the former;
- embedding the graphs into a vector space allows to use conventional ML techniques;
- in **geometric deep learning** deep models are generalized to non-euclidean domains.

Synthetic data

- Deep learning models need a significant amount of data to achieve good performances;
- Privacy rules and limitations over the number of possible requests make it hard to obtain real data from social networks;
- To address these issues we artificially generated synthetic graphs as a playground for our models;

Synthetic data

- Deep learning models need a significant amount of data to achieve good performances;
- Privacy rules and limitations over the number of possible requests make it hard to obtain real data from social networks;
- To address these issues we artificially generated synthetic graphs as a playground for our models;

The synthetic data generation involves two steps:

1. generating the **social structure** of interest;
2. generating a certain number of **information cascades**;

Social structure

- To artificially generate a social network structure which resembles a real one, **random graph models** are usually used.

Social structure

- To artificially generate a social network structure which resembles a real one, **random graph models** are usually used.
- A good model should allow creating graphs for which the degree distribution follows a **power-law**, as happens in real social networks.

Social structure

- To artificially generate a social network structure which resembles a real one, **random graph models** are usually used.
- A good model should allow creating graphs for which the degree distribution follows a **power-law**, as happens in real social networks.

A power law is a functional relationship

$$y = ax^{-c}$$

between two quantities, where one quantity varies as a power of the other.

Social structure

- To artificially generate a social network structure which resembles a real one, **random graph models** are usually used.
- A good model should allow creating graphs for which the degree distribution follows a **power-law**, as happens in real social networks.

A power law is a functional relationship

$$y = ax^{-c}$$

between two quantities, where one quantity varies as a power of the other.
By applying the logarithm to both parts we have that

$$log(y) = log(ax^{-c}) \tag{1}$$

$$log(y) = log(a) - c \cdot log(x) \tag{2}$$

Social structure

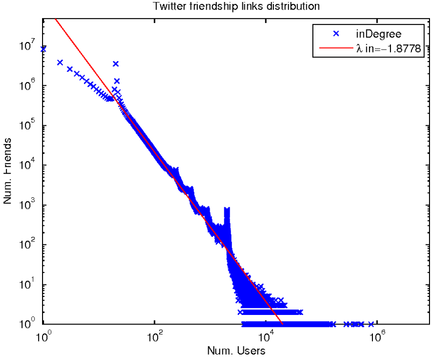


Figure: Twitter degree distribution.

Social structure

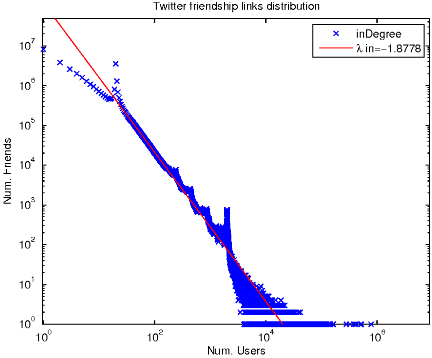


Figure: Twitter degree distribution.

- Exponentially more likely to pick "normal people" with few followers rather than popular profiles.

Preferential attachment

The **preferential attachment** model is a simple random graph model producing power-law graphs.

Preferential attachment

The **preferential attachment** model is a simple random graph model producing power-law graphs.

1. begin with a single node with a self loop;

Preferential attachment

The **preferential attachment** model is a simple random graph model producing power-law graphs.

1. begin with a single node with a self loop;
2. when you have built a graph with $N-1$ nodes, you add the N-th node with an edge that goes from N to a node i chosen accordingly with a probability proportional to the degree of i

$$Pr\left\{\text{neighbor of } N \text{ is } i\right\} = \frac{deg(i)}{\sum_{k=1}^{N} deg(k)}$$

Cascades

The cascades are generated with the **Independent Cascades** model.

Cascades

The cascades are generated with the **Independent Cascades** model.

The following assumptions hold:

- at start, k nodes hold some piece of information (the seed set);
- the time is discrete;
- information spreads over time.

Cascades

The cascades are generated with the **Independent Cascades** model.

The following assumptions hold:

- at start, k nodes hold some piece of information (the seed set);
- the time is discrete;
- information spreads over time.

The model then works as follows

Cascades

The cascades are generated with the **Independent Cascades** model.

The following assumptions hold:

- at start, k nodes hold some piece of information (the seed set);
- the time is discrete;
- information spreads over time.

The model then works as follows

- at time t_0 the only persons having the information will be the ones in the seed set;

Cascades

The cascades are generated with the **Independent Cascades** model.

The following assumptions hold:

- at start, k nodes hold some piece of information (the seed set);
- the time is discrete;
- information spreads over time.

The model then works as follows

- at time t_0 the only persons having the information will be the ones in the seed set;
- at time t_i for each of the edges incident on the nodes having the information we will be flipping a coin:

Cascades

The cascades are generated with the **Independent Cascades** model.

The following assumptions hold:

- at start, k nodes hold some piece of information (the seed set);
- the time is discrete;
- information spreads over time.

The model then works as follows

- at time t_0 the only persons having the information will be the ones in the seed set;
- at time t_i for each of the edges incident on the nodes having the information we will be flipping a coin:
  - with prob p the information will spread on that edge;

Cascades

The cascades are generated with the **Independent Cascades** model.

The following assumptions hold:

- at start, k nodes hold some piece of information (the seed set);
- the time is discrete;
- information spreads over time.

The model then works as follows

- at time t_0 the only persons having the information will be the ones in the seed set;
- at time t_i for each of the edges incident on the nodes having the information we will be flipping a coin:
    - with prob p the information will spread on that edge;
    - else the edge is lost forever.

Real data

- To see if the model generalized to a real world network we collected users and their tweets on **Twitter**;

Real data

- To see if the model generalized to a real world network we collected users and their tweets on **Twitter**;
- The process to obtain real data from Twitter involved two steps:
  1. retrieving the social network relative to a subgraph of Twitter;
  2. obtaining the cascades from the tweets of the users in the subgraph.

Social structure

To obtain a subgraph of Twitter we scraped the social network in a **Breadth First**-fashion

Social structure

To obtain a subgraph of Twitter we scraped the social network in a **Breadth First**-fashion

1. start with a queue containing a random english speaking user;

Social structure

To obtain a subgraph of Twitter we scraped the social network in a **Breadth First**-fashion

1. start with a queue containing a random english speaking user;
2. collect all his followers and followees and add them to the queue;

Social structure

To obtain a subgraph of Twitter we scraped the social network in a **Breadth First**-fashion

1. start with a queue containing a random english speaking user;
2. collect all his followers and followees and add them to the queue;
3. pop the next user from the queue and repeat step 2 until the desired number of users is reached;

Cascades

- Given the set of users U collected in the previous step and fixed a time window $[T_s, T_e]$ we obtained all the tweets from U falling in this time window;

Cascades

- Given the set of users U collected in the previous step and fixed a time window $[T_s, T_e]$ we obtained all the tweets from U falling in this time window;
- Obtained the hashtag for every tweet, which is our piece of information;

Cascades

- Given the set of users U collected in the previous step and fixed a time window $[T_s, T_e]$ we obtained all the tweets from U falling in this time window;
- Obtained the hashtag for every tweet, which is our piece of information;

Recreated for each distinct hashtag a propagation cascade:

Cascades

- Given the set of users U collected in the previous step and fixed a time window $[T_s, T_e]$ we obtained all the tweets from U falling in this time window;
- Obtained the hashtag for every tweet, which is our piece of information;

Recreated for each distinct hashtag a propagation cascade:

1. order the tweets containing the hashtags by timestamp;

Cascades

- Given the set of users U collected in the previous step and fixed a time window $[T_s, T_e]$ we obtained all the tweets from U falling in this time window;
- Obtained the hashtag for every tweet, which is our piece of information;

Recreated for each distinct hashtag a propagation cascade:

1. order the tweets containing the hashtags by timestamp;
2. create the first cascade with the first tweet author as root node;

Cascades

- Given the set of users U collected in the previous step and fixed a time window $[T_s, T_e]$ we obtained all the tweets from U falling in this time window;
- Obtained the hashtag for every tweet, which is our piece of information;

Recreated for each distinct hashtag a propagation cascade:

1. order the tweets containing the hashtags by timestamp;
2. create the first cascade with the first tweet author as root node;
3. for each remaining tweet t:

Cascades

- Given the set of users U collected in the previous step and fixed a time window $[T_s, T_e]$ we obtained all the tweets from U falling in this time window;
- Obtained the hashtag for every tweet, which is our piece of information;

Recreated for each distinct hashtag a propagation cascade:

1. order the tweets containing the hashtags by timestamp;
2. create the first cascade with the first tweet author as root node;
3. for each remaining tweet t:
   3.1 let u be the node relative to the author of t;

Cascades

- Given the set of users U collected in the previous step and fixed a time window $[T_s, T_e]$ we obtained all the tweets from U falling in this time window;
- Obtained the hashtag for every tweet, which is our piece of information;

Recreated for each distinct hashtag a propagation cascade:

1. order the tweets containing the hashtags by timestamp;
2. create the first cascade with the first tweet author as root node;
3. for each remaining tweet t:
   3.1 let u be the node relative to the author of t;
   3.2 if u has an incoming edge from an existing cascade tree c, then add it to c;

Cascades

- Given the set of users U collected in the previous step and fixed a time window $[T_s, T_e]$ we obtained all the tweets from U falling in this time window;
- Obtained the hashtag for every tweet, which is our piece of information;

Recreated for each distinct hashtag a propagation cascade:

1. order the tweets containing the hashtags by timestamp;
2. create the first cascade with the first tweet author as root node;
3. for each remaining tweet t:
    3.1 let u be the node relative to the author of t;
    3.2 if u has an incoming edge from an existing cascade tree c, then add it to c;
    3.3 else create a new cascade tree with u as root;

Cascades

- Given the set of users U collected in the previous step and fixed a time window $[T_s, T_e]$ we obtained all the tweets from U falling in this time window;
- Obtained the hashtag for every tweet, which is our piece of information;

Recreated for each distinct hashtag a propagation cascade:

1. order the tweets containing the hashtags by timestamp;
2. create the first cascade with the first tweet author as root node;
3. for each remaining tweet t:
   3.1 let u be the node relative to the author of t;
   3.2 if u has an incoming edge from an existing cascade tree c, then add it to c;
   3.3 else create a new cascade tree with u as root;

The roots of the cascade trees were used as early adopters, the remaining nodes as final.

Dataset

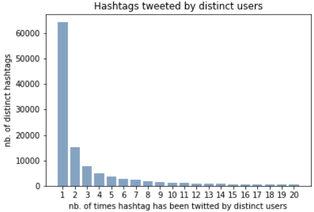
- The scraping process resulted in a dataset containing
  - $\approx 30k$ users;
  - $\approx 500k$ edges;
  - $> 1m$ tweets;
    - $\approx 400k$ hashtags;
    - $\approx 65k$ distinct;

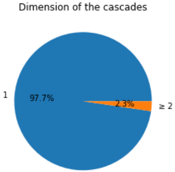
Dataset

- The scraping process resulted in a dataset containing
  - $\approx 30k$ users;
  - $\approx 500k$ edges;
  - $> 1m$ tweets;
    - $\approx 400k$ hashtags;
    - $\approx 65k$ distinct;
- The synthetic dataset has similar numbers for what concerns the static structure, but is much less sparse;

Sparsity

- The collected dataset suffers from severe **sparsity**;
- Even ignoring lone hashtags, most of the cascades are **shallow**
- This is mainly due to two reasons:
  1. Virality is intrinsecally **rare**;
  2. We are observing an **incomplete subnetwork** of the real social network, possibly disconnecting deep cascades;





Node features

- Representation learning techniques may fail to capture some local node properties;
- Interesting features can be used to enrich the nodes;

Node features

- Representation learning techniques may fail to capture some local node properties;
- Interesting features can be used to enrich the nodes;

For each node, we computed the following features:

Node features

- Representation learning techniques may fail to capture some local node properties;
- Interesting features can be used to enrich the nodes;

For each node, we computed the following features:

- **local clustering coefficient**, quantifies how close its neighbours are to being a clique;

$$C_i = \frac{\#\text{ of existing edges in } N(v_i)}{\#\text{ of all possible edges in } N(v_i)} \tag{3}$$

Node features

- Representation learning techniques may fail to capture some local node properties;
- Interesting features can be used to enrich the nodes;

For each node, we computed the following features:

- **local clustering coefficient**, quantifies how close its neighbours are to being a clique;

$$C_i = \frac{\#\text{ of existing edges in } N(v_i)}{\#\text{ of all possible edges in } N(v_i)} \tag{3}$$

- **eigenvector centrality**, measure of the node influence in the network based on the concept that connections to high-scoring nodes contribute more to the score of the node than connections to low-scoring nodes;

Node features

- Representation learning techniques may fail to capture some local node properties;
- Interesting features can be used to enrich the nodes;

For each node, we computed the following features:

- **local clustering coefficient**, quantifies how close its neighbours are to being a clique;

$$C_i = \frac{\#\text{ of existing edges in } N(v_i)}{\#\text{ of all possible edges in } N(v_i)} \tag{3}$$

- **eigenvector centrality**, measure of the node influence in the network based on the concept that connections to high-scoring nodes contribute more to the score of the node than connections to low-scoring nodes;
- **PageRank** coefficient, kind of eigenvector centrality originally used by Google to represent the likelihood that a person randomly clicking on links will arrive at any particular webpage;

Node features

- Representation learning techniques may fail to capture some local node properties;
- Interesting features can be used to enrich the nodes;

For each node, we computed the following features:

- **local clustering coefficient**, quantifies how close its neighbours are to being a clique;

$$C_i = \frac{\#\text{ of existing edges in } N(v_i)}{\#\text{ of all possible edges in } N(v_i)} \tag{3}$$

- **eigenvector centrality**, measure of the node influence in the network based on the concept that connections to high-scoring nodes contribute more to the score of the node than connections to low-scoring nodes;
- **PageRank** coefficient, kind of eigenvector centrality originally used by Google to represent the likelihood that a person randomly clicking on links will arrive at any particular webpage;
- **Authority** and **Hubs** coefficients, a good hub represents a node that points to many other nodes, while a good authority represents a node that is linked by many different hubs.

Graph Convolution

- Graphs are **non-Euclidean domains**, that do not share the flat, grid-like structure of Euclidean space.
- We want to capture the **structure** of the domain, which is as important as the data on the domain.
- Convolution enforces by construction useful **priors** (self-similarity, locality), but convolution relies on the structured nature of Euclidean space.
- Graph convolution can be defined in different ways, and in recent years a great number of models, relying on different convolutional layers, have been designed.
- The model we used is based on the Graph Attention (GAT) layer.
- We also tried the layer of Graph Convolutional Network (GCN) architecture, to draw a comparison.
- The latter is considered in the GDL literature as a **spectral** approach, while the former is considered as a **spatial** approach.

Graph Convolution in the spectral domain (1)

- **Spectral** approaches define the convolution operation on graphs' nodes in the spectral domain, as the multiplication of a node signal $\mathbf{x} \in \mathbb{R}^n$ with a filter $\mathbf{g}_\theta = diag(g_\theta^{(1)}, \ldots, g_\theta^{(n)})$ in the Fourier domain.

$$\mathbf{g}_\theta \star \mathbf{x} = \mathbf{U}\mathbf{g}_\theta\mathbf{U}^\top\mathbf{x} \tag{4}$$

- This definition exploits several properties:
  1. One of the convolution **defining** properties is that it is **diagonalized** by the Fourier transform, meaning

$$\mathcal{F}\{(\mathbf{g} \star \mathbf{x})\} = \underbrace{\mathcal{F}\{\mathbf{g}\}\mathcal{F}\{\mathbf{x}\}}_{\text{simple product}} \tag{5}$$

  2. Although the Fourier transform of a node signal on a graph is not clearly defined, the Laplacian (differential operator) has its graph counterpart

$$\Delta\mathbf{f} = \underbrace{\left(\mathbf{I}_n - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\right)}_{\text{normalized graph Laplacian}}\mathbf{f}. \tag{6}$$

Graph Convolution in the spectral domain (2)

3. The Fourier basis is a set of **eigenfunctions** of the Laplacian

$$\mathcal{F}\{f(x)\} = \hat{f}(x) = \int f(x) \overbrace{e^{-2\pi i x\xi}}^{\text{plane waves are Fourier basis}} dx \tag{7}$$

$$\Delta \underbrace{\left(e^{-2\pi i x\xi}\right)}_{\text{plane wave}} = 4\pi^2 |\xi|^2 \underbrace{e^{-2\pi i x\xi}}_{\text{Laplacian eigenfunction}} \tag{8}$$

and we can generalize this to non-Euclidean (graph) domain by taking as Fourier basis the **eigenvectors** of the graph Laplacian

$$\Delta = \mathbf{U\Lambda U}^\top \tag{9}$$

$$\widehat{\mathbf{x}} = \mathbf{U}^\top \mathbf{x}, \qquad \mathbf{x} = \mathbf{U}\widehat{\mathbf{x}} \tag{10}$$

Exploiting these properties, it is

$$\mathbf{g}_\theta \star \mathbf{x} = \underbrace{\mathbf{U}}_{\text{back to spatial domain}} \overbrace{\mathbf{g}_\theta}^{\text{conv. in Fourier domain}} \underbrace{\mathbf{U}^\top \mathbf{x}}_{\text{to Fourier domain}} \tag{11}$$

with $\mathbf{g}_\theta = \mathbf{g}_\theta(\mathbf{\Lambda}) =$ learnable **spectral kernel functions**.

GCN Layer

- **Simplification:** $\mathbf{g}_\theta(\boldsymbol{\Lambda})$ is computationally expensive, so we compute a **truncated expansion** in terms of **Chebyshev polynomials**

$$\mathbf{g}_\theta(\boldsymbol{\Lambda}) \approx \sum_{k=0}^{K} \theta'_k T_k \underbrace{(\tilde{\boldsymbol{\Lambda}})}_{\text{renormalized}} \tag{12}$$

$$\mathbf{g}'_\theta \star \mathbf{x} \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{\boldsymbol{L}})\mathbf{x} \tag{13}$$

in which the Laplacian enters up to its K-th power, hence it depends on node signals from a K-th order neighborhood.

- **Simplification:** Each layer only computes one hop $(K = 1)$

$$\mathbf{g}'_\theta \star \mathbf{x} \approx \theta'_0\mathbf{x} - \theta'_1\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{x} \tag{14}$$

- **Simplification:** $\theta = \theta'_0 = -\theta'_1$, so the layer actually computes

$$\mathbf{g}'_\theta \star \mathbf{x} \approx \overbrace{\theta}^{\text{learnable}} \underbrace{\left(\mathbf{I}_n + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\right)}_{\text{fixed}} \mathbf{x} \tag{15}$$

GAT Layer (1)

- Problems of GCN:
  - Preprocessing computation of $\tilde{\boldsymbol{A}} = \mathbf{I}_n + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ means model cannot be transfered on unseen graphs
  - Parameters θ are shared across the nodes in a neighborhood, all have **same importance**
- GAT addresses these problems by defining convolution directly in the **spatial domain**

1. **Input**: set of node features

$$\mathbf{H} = \{\ \mathbf{h}_1, \dots, \mathbf{h}_n\},\ \ \mathbf{h}_i \in \mathbb{R}^F \tag{16}$$

2. Shared linear transformation applied to every node

$$\mathbf{h}_i \mapsto \mathbf{W}\mathbf{h}_i = \tilde{\boldsymbol{h}}_i \tag{17}$$

3. Given the i-th node, **masked attention** is performed to compute **attention coefficients** for each node j in its neighborhood

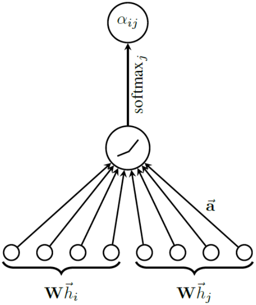
$$\alpha_{ij} = \mathrm{softmax}_j(e_{ij}) \quad e_{ij} = a(\tilde{\boldsymbol{h}}_i, \tilde{\boldsymbol{h}}_j) = \sigma(\mathbf{a}^\top[\tilde{\boldsymbol{h}}_i; \tilde{\boldsymbol{h}}_j]) \tag{18}$$

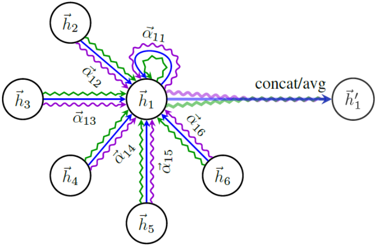
where $a(\cdot,\cdot)$ is an **attention mechanism** implemented as a single layer MLP.

GAT Layer (2)

4. **Output**: The attention coefficients are used to perform a linear combination of the features of the corresponding nodes in the neighborhood of each node i, plus a nonlinearity

$$\mathbf{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \tilde{\boldsymbol{h}}_j\right). \tag{19}$$





Our model

- The model we have used consists in several layers of **graph convolution** (both GCN and GAT can be used) to extract a meaningful representation $\mathbf{r}_v$ for each node v, given its **features** (the node **signal**) $\mathbf{x}_v$ and the features of the other nodes in the graph.

$$\mathbf{r}_v^{(\ell)} = GC_\ell \circ GC_{\ell-1} \circ \cdots \circ GC_1(\mathbf{x}_v), \qquad \ell = 1, \dots, K-1 \tag{20}$$

- The final layer produces a representation

$$s_v = \underbrace{\sigma}_{\text{sigmoid}} \left(GraphConv_K \left(\mathbf{r}_v^{(K-1)} \right) \right) \tag{21}$$

  that is the node **final activation state** $s_v \in [0, 1]$, a predictor of the information spreading to the node: an activation close to 1 means the node has **adopted** the information.

- To predict the global **virality** of the piece of information m in the network (number of final adopters), we aggregate the node activation states by **graph sum pooling**, to obtain

$$n_\infty^m = \hat{y}_m = \sum_{v \in \mathcal{V}} s_v. \tag{22}$$

Training

- **Loss function**.
  We first tried a loss function defined accordingly to the task objective: predict the virality of a piece of information. Therefore we tried the MRSE loss, defined as

$$\mathcal{L}_{MRSE} = \frac{1}{M} \sum_{m=1}^{M} \left(\frac{\hat{y}_m - y_m}{y_m} \right)^2 . \tag{23}$$

  However, this yielded poor results, so we switched to a per-node binary cross-entropy loss:

$$\mathcal{L}_{BCE} = -\frac{1}{M} \sum_{m=1}^{M} \sum_{v \in \mathcal{V}_m} y_v \log \hat{y}_v + (1 - y_v) \log(1 - \hat{y}_v). \tag{24}$$

- **Regularization**.
  - To regularize learning we used **dropout**, that act as a regularizer by randomly removing edges in the network hence penalizing **coadaptation**.
  - Beside regular dropout, that randomly drops edges between units in consecutive layers, we also utilize **edge dropout**, that randomly drops edges in the graph, introducing noise in the node signal propagation and hence enforcing robustness to this noise, preventing overfitting.

Results

We evaluated our model with both the convolutional layers presented before, and also with both the **real** and **synthetic** data, to draft a comparison.

- We evaluated the models in terms of F1 score since we trained them with binary cross entropy.
- Nevertheless, they showed significantly better performance on the virality prediction task as defined in principle, i.e. as a "regression" over the graph.

F1 Score		
	Real data	Synthetic data
GCN	0.7271	0.7448
GAT	0.7841	0.8297

Conclusions

- In this project we proposed a **Geometric Deep Learning** approach to the problem of **virality prediction** on social networks (Twitter).
- The main difficulty we faced has been on **data**.
    - **Difficult to obtain**: with the GDPR policies Twitter strictly regulates the access to data.
    - **Sparse**: albeit counterintuitive, wide spread of information on social networks is **rare**, so a learning model has to learn spreading patterns with very few informative samples.
- The point above is a general, unsolved problem, and other works in this area "solved" it by carefully selecting informative samples among huge collection of data. This in our opinion induces a **bias**, since the data that the model is shown does **not** correspond to how data in the real world is distributed.
- A possibility for future work on the project might be on how to apply signal processing techniques for reconstructing sparse signals (e.g. **compressed sensing**) on non-Euclidean domains (graphs).

Thank you for your attention