Deep learning of contagion dynamics on complex networks

Murphy, Laurence and Allard

3 February 2021

Key Contributions

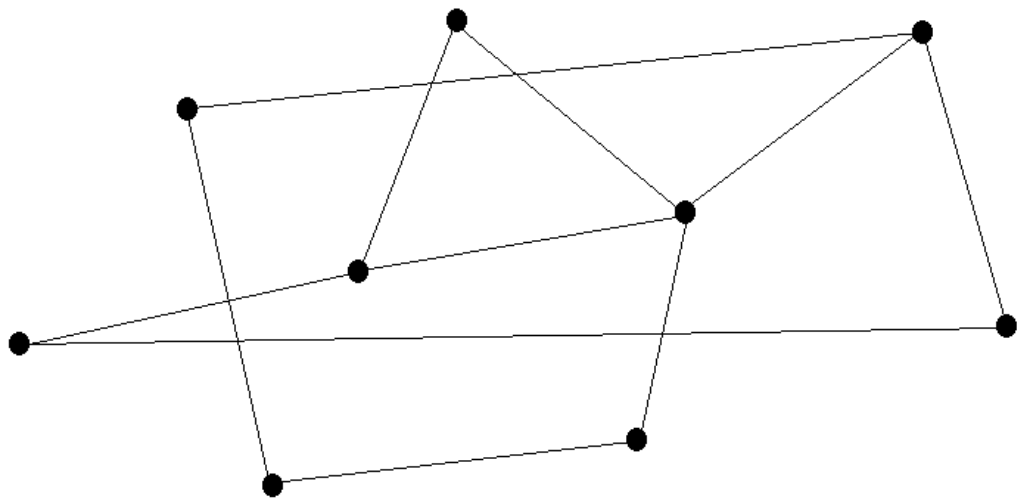
- A training procedure and appropriate GNN architecture.
- Assessing validity of the approach.
- Providing predictions for unseen network structures.

Setup

Have graph G.

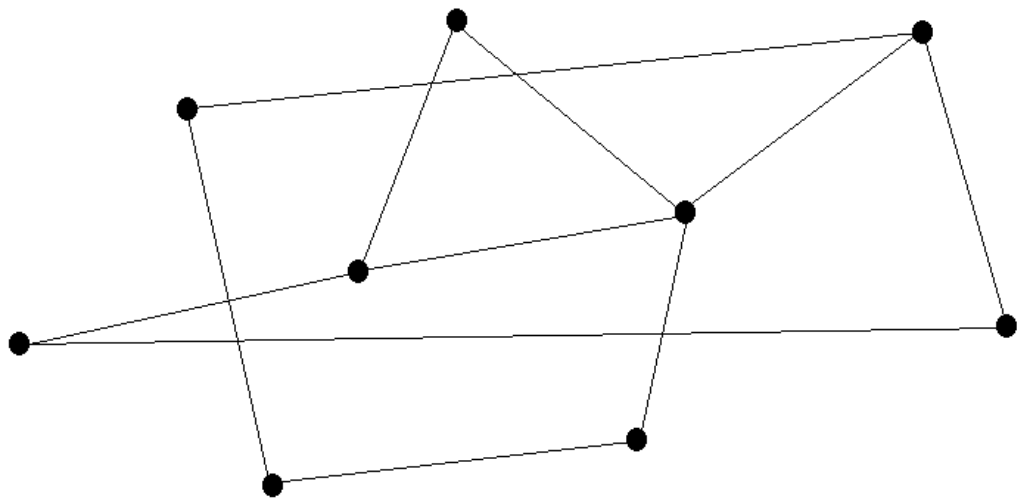
Setup

Have graph G.



Setup

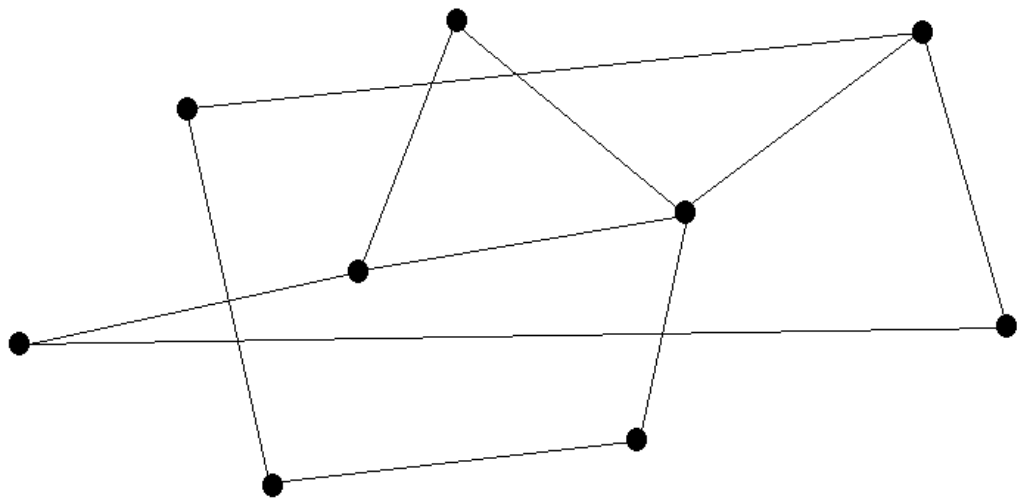
Have graph G.



$\mathcal{V} = \{v_1, \ldots, v_N\}$ is the node set.

Setup

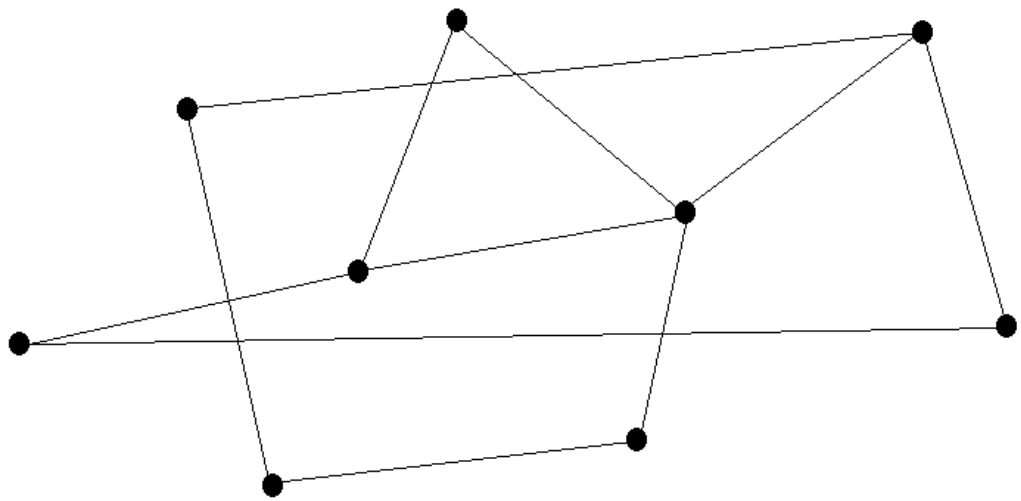
Have graph G.



$\mathcal{V} = \{v_1, \ldots, v_N\}$ is the node set.
$\mathcal{E}$ is the edge set.

Setup

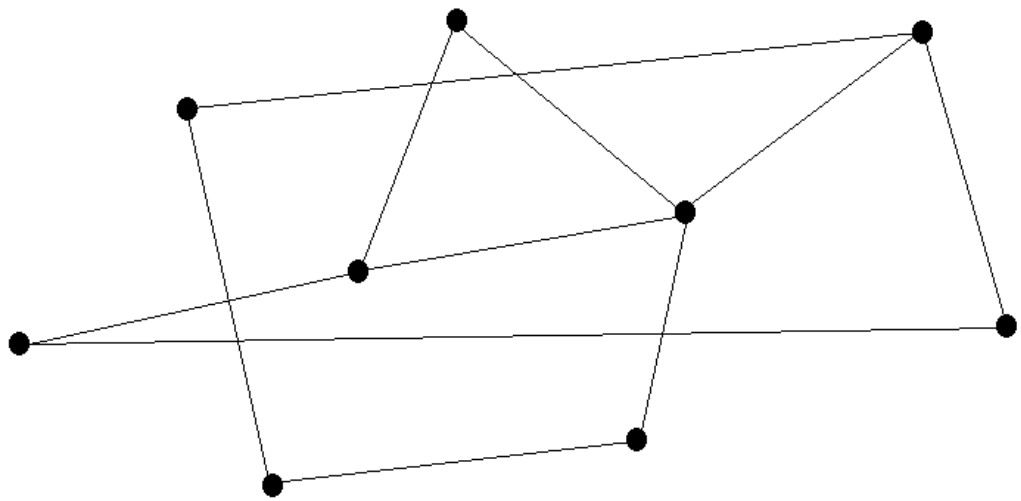
Have graph G.



$\mathcal{V} = \{v_1, \ldots, v_N\}$ is the node set.
$\mathcal{E}$ is the edge set.
$\Phi_i = (\phi_1(v_i), \ldots, \phi_Q(v_i))$ are the attributes of node v_i.

Setup

Have graph G.



$\mathcal{V} = \{v_1, \ldots, v_N\}$ is the node set.
$\mathcal{E}$ is the edge set.
$\Phi_i = (\phi_1(v_i), \ldots, \phi_Q(v_i))$ are the attributes of node v_i.
$\Omega_{ij} = (\omega_1(e_{ij}), \ldots, \omega_P(e_{ij}))$ are the attributes of edge e_{ij}.

Setup

Have some unknown dynamical process $\mathcal{M}$, which generates a time series D on G.

Setup

Have some unknown dynamical process $\mathcal{M}$, which generates a time series D on G.

$D = (\mathbf{X}, \mathbf{Y})$, where $\mathbf{X} = (X_1, \ldots, X_T)$ and $\mathbf{Y} = (Y_1, \ldots, Y_T)$.

Setup

Have some unknown dynamical process $\mathcal{M}$, which generates a time series D on G.

$D = (\mathbf{X}, \mathbf{Y})$, where $\mathbf{X} = (X_1, \ldots, X_T)$ and $\mathbf{Y} = (Y_1, \ldots, Y_T)$.

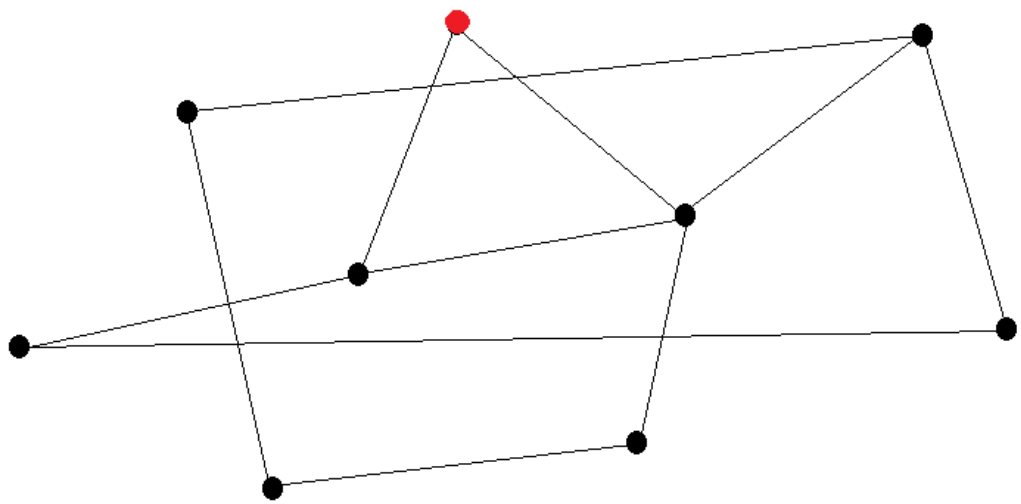
$X_t \in \mathcal{S}^{|\mathcal{V}|}$ is the state of the nodes at time t, and $Y_t = \mathcal{M}(X_t, G)$.

Setup

Have some unknown dynamical process $\mathcal{M}$, which generates a time series D on G.

$D = (\mathbf{X}, \mathbf{Y})$, where $\mathbf{X} = (X_1, \ldots, X_T)$ and $\mathbf{Y} = (Y_1, \ldots, Y_T)$.

$X_t \in \mathcal{S}^{|\mathcal{V}|}$ is the state of the nodes at time t, and $Y_t = \mathcal{M}(X_t, G)$.

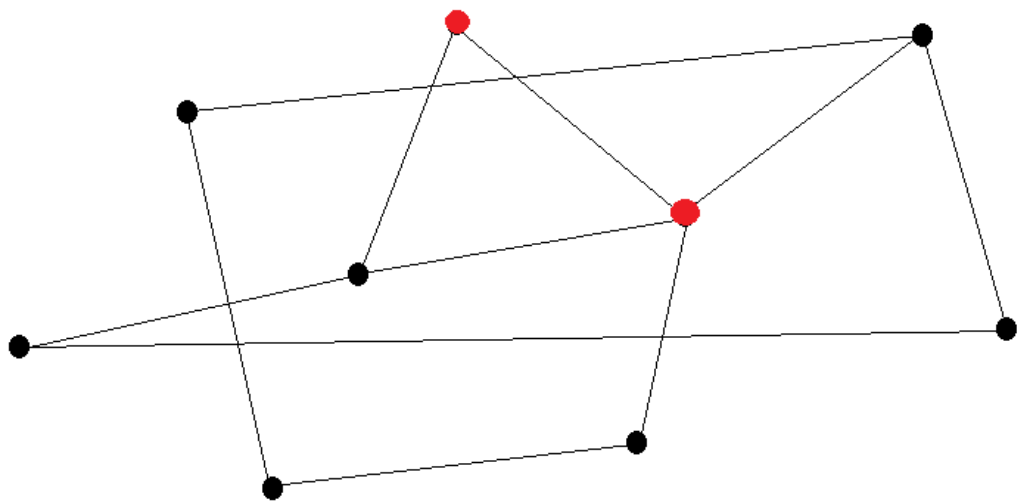


Setup

Have some unknown dynamical process $\mathcal{M}$, which generates a time series D on G.

$D = (\mathbf{X}, \mathbf{Y})$, where $\mathbf{X} = (X_1, \ldots, X_T)$ and $\mathbf{Y} = (Y_1, \ldots, Y_T)$.

$X_t \in \mathcal{S}^{|\mathcal{V}|}$ is the state of the nodes at time t, and $Y_t = \mathcal{M}(X_t, G)$.

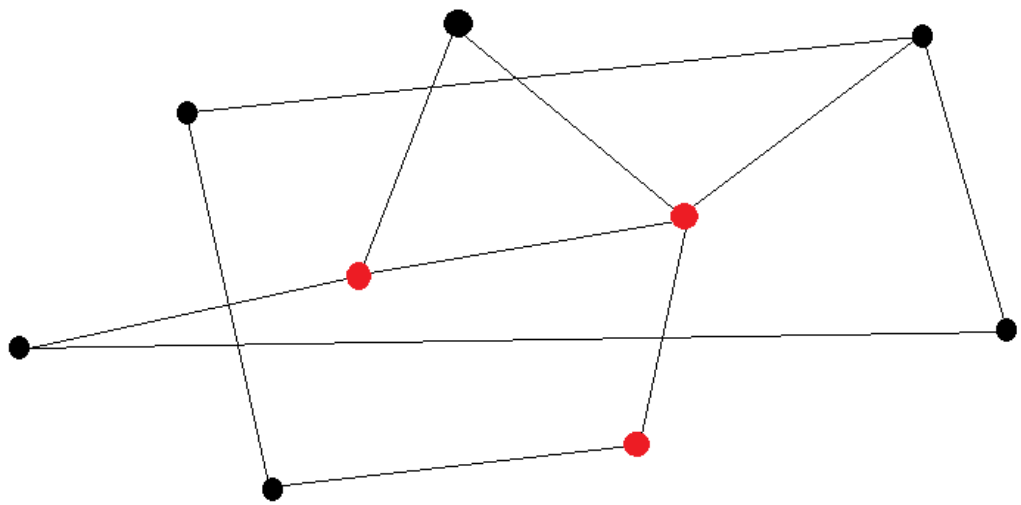


Setup

Have some unknown dynamical process $\mathcal{M}$, which generates a time series D on G.

$D = (\mathbf{X}, \mathbf{Y})$, where $\mathbf{X} = (X_1, \ldots, X_T)$ and $\mathbf{Y} = (Y_1, \ldots, Y_T)$.

$X_t \in \mathcal{S}^{|\mathcal{V}|}$ is the state of the nodes at time t, and $Y_t = \mathcal{M}(X_t, G)$.



Setup

In particular, we define the observed outcome as

$$\left(\tilde{y}_i(t)\right)_m = \delta\left(x_i(t+\Delta t), m\right), \forall m \in \mathcal{S}.$$

Setup

In particular, we define the observed outcome as

$$\left(\tilde{y}_i(t)\right)_m = \delta\left(x_i(t+\Delta t), m\right), \forall m \in \mathcal{S}.$$

We assume that $\mathcal{M}$ acts on X_t locally and identically at all times, according to the structure of G, such that for node v_i

Setup

In particular, we define the observed outcome as

$$\left(\tilde{y}_i(t)\right)_m = \delta\left(x_i(t+\Delta t), m\right), \forall m \in \mathcal{S}.$$

We assume that $\mathcal{M}$ acts on X_t locally and identically at all times, according to the structure of G, such that for node v_i

$$y_i = f\left(x_i, \Phi_i, x_{\mathcal{N}_i}, x_{\mathcal{N}_i}, \Omega_{i\mathcal{N}_i}\right).$$

Setup

In particular, we define the observed outcome as

$$\left(\tilde{y}_i(t)\right)_m = \delta\left(x_i(t+\Delta t), m\right), \forall m \in \mathcal{S}.$$

We assume that $\mathcal{M}$ acts on X_t locally and identically at all times, according to the structure of G, such that for node v_i

$$y_i = f\left(x_i, \Phi_i, x_{\mathcal{N}_i}, x_{\mathcal{N}_i}, \Omega_{i\mathcal{N}_i}\right).$$

Goal: Build a model $\hat{\mathcal{M}}$, parameterised by tunable Θ, such that

$$\hat{\mathcal{M}}\left(X_t', G'; \Theta\right) \approx \mathcal{M}\left(X_t', G'\right).$$

Setup

In particular, we define the observed outcome as

$$\left(\tilde{y}_i(t)\right)_m = \delta\left(x_i(t+\Delta t), m\right), \forall m \in \mathcal{S}.$$

We assume that $\mathcal{M}$ acts on X_t locally and identically at all times, according to the structure of G, such that for node v_i

$$y_i = f\left(x_i, \Phi_i, x_{\mathcal{N}_i}, x_{\mathcal{N}_i}, \Omega_{i\mathcal{N}_i}\right).$$

Goal: Build a model $\hat{\mathcal{M}}$, parameterised by tunable Θ, such that

$$\hat{\mathcal{M}}\left(X_t', G'; \Theta\right) \approx \mathcal{M}\left(X_t', G'\right).$$

Therefore, the outcomes from the GNN will be

$$\hat{y}_i = \hat{f}\left(x_i, \Phi_i, x_{\mathcal{N}_i}, \Phi_{\mathcal{N}_i}, \Omega_{i\mathcal{N}_i}; \Theta\right)..$$

Loss Function

We need some global loss function, say $\mathcal{L}(\Theta)$, that can be decomposed locally.

Loss Function

We need some global loss function, say $\mathcal{L}(\Theta)$, that can be decomposed locally. The following is used

$$\mathcal{L}(\Theta) = \sum_{t \in \mathcal{T}'} \sum_{v_i \in \mathcal{V}'(t)} \frac{w_i(t)}{Z'} L(y_i(t), \hat{y}_i(t)).$$

Here

Loss Function

We need some global loss function, say $\mathcal{L}(\Theta)$, that can be decomposed locally. The following is used

$$\mathcal{L}(\Theta) = \sum_{t \in \mathcal{T}'} \sum_{v_i \in \mathcal{V}'(t)} \frac{w_i(t)}{Z'} L(y_i(t), \hat{y}_i(t)).$$

Here

- $w_i(t)$ - the weight assigned to node v_i at time t;

Loss Function

We need some global loss function, say $\mathcal{L}(\Theta)$, that can be decomposed locally. The following is used

$$\mathcal{L}(\Theta) = \sum_{t \in \mathcal{T}'} \sum_{v_i \in \mathcal{V}'(t)} \frac{w_i(t)}{Z'} L(y_i(t), \hat{y}_i(t)).$$

Here

- $w_i(t)$ - the weight assigned to node v_i at time t;
- $Z' = \sum_{t \in \mathcal{T}'} \sum_{v_i \in \mathcal{V}'(t)} w_i(t)$ - normalising factor;

Loss Function

We need some global loss function, say $\mathcal{L}(\Theta)$, that can be decomposed locally. The following is used

$$\mathcal{L}(\Theta) = \sum_{t \in \mathcal{T}'} \sum_{v_i \in \mathcal{V}'(t)} \frac{w_i(t)}{Z'} L(y_i(t), \hat{y}_i(t)).$$

Here

- $w_i(t)$ - the weight assigned to node v_i at time t;
- $Z' = \sum_{t \in \mathcal{T}'} \sum_{v_i \in \mathcal{V}'(t)} w_i(t)$ - normalising factor;
- $\mathcal{V}'(t)$ and $\mathcal{T}'$ - training node set and training time set.

We take the choice of weights to be

Loss Function

We need some global loss function, say $\mathcal{L}(\Theta)$, that can be decomposed locally. The following is used

$$\mathcal{L}(\Theta) = \sum_{t \in \mathcal{T}'} \sum_{v_i \in \mathcal{V}'(t)} \frac{w_i(t)}{Z'} L(y_i(t), \hat{y}_i(t)).$$

Here

- $w_i(t)$ - the weight assigned to node v_i at time t;
- $Z' = \sum_{t \in \mathcal{T}'} \sum_{v_i \in \mathcal{V}'(t)} w_i(t)$ - normalising factor;
- $\mathcal{V}'(t)$ and $\mathcal{T}'$ - training node set and training time set.

We take the choice of weights to be

$$w_i(t) \propto \rho\left(k_i, x_i, \Phi_i, x_{\mathcal{N}_i}, \Phi_{\mathcal{N}_i}, \Omega_{i\mathcal{N}_i}\right)^{-\lambda},$$

where k_i is the degree of node v_i in G.

Four Types of Synthetic Dynamics

Four Types of Synthetic Dynamics

- *Simple contagion* dynamics: discrete-time susceptible-infected-susceptible, so $\mathcal{S} = \{S, I\} = \{0, 1\}$.

Four Types of Synthetic Dynamics

- *Simple contagion* dynamics: discrete-time susceptible-infected-susceptible, so $\mathcal{S} = \{S, I\} = \{0, 1\}$. Have $\alpha(I)$ as the infection probability function, where I is the number of infected neighbours.

Four Types of Synthetic Dynamics

- *Simple contagion* dynamics: discrete-time susceptible-infected-susceptible, so $\mathcal{S} = \{S, I\} = \{0, 1\}$. Have $\alpha(I)$ as the infection probability function, where I is the number of infected neighbours. Recovery probability is β.

Four Types of Synthetic Dynamics

I *Simple contagion* dynamics: discrete-time susceptible-infected-susceptible, so $\mathcal{S} = \{S, I\} = \{0, 1\}$. Have $\alpha(I)$ as the infection probability function, where I is the number of infected neighbours. Recovery probability is β.

II *Complex contagion* dynamics: nonmonotonic infection function $\alpha(I)$.

Four Types of Synthetic Dynamics

I *Simple contagion* dynamics: discrete-time susceptible-infected-susceptible, so $\mathcal{S} = \{S, I\} = \{0, 1\}$. Have $\alpha(I)$ as the infection probability function, where I is the number of infected neighbours. Recovery probability is β.

II *Complex contagion* dynamics: nonmonotonic infection function $\alpha(I)$.

III *Interacting contagion* dynamics (with two diseases): $\mathcal{S} = \{S_1S_2, I_1S_2, S_1I_2, I_1I_2\} = \{0, 1, 2, 3\}$.

Four Types of Synthetic Dynamics

I *Simple contagion* dynamics: discrete-time susceptible-infected-susceptible, so $\mathcal{S} = \{S, I\} = \{0, 1\}$. Have $\alpha(I)$ as the infection probability function, where I is the number of infected neighbours. Recovery probability is β.

II *Complex contagion* dynamics: nonmonotonic infection function $\alpha(I)$.

III *Interacting contagion* dynamics (with two diseases): $\mathcal{S} = \{S_1S_2, I_1S_2, S_1I_2, I_1I_2\} = \{0, 1, 2, 3\}$.

IV *Metapopulation* dynamics: status of individuals are gathered by geographical regions. E.g. Deterministic metapopulation dynamics with constant population size, and have S, I or R for each individual.

Performance

- We compare GNN predictions $\hat{y}_i(t)$ with corresponding target $y_i(t)$.
- Use Pearson correlation coefficient r between predictions and targets. Compute error as $1 - r$ for each degree class k.
- The GNN outperforms the MLE on Erdos-Renyi and Barabasi-Albert networks.

Performance

- We compare GNN predictions $\hat{y}_i(t)$ with corresponding target $y_i(t)$.
- Use Pearson correlation coefficient r between predictions and targets. Compute error as $1 - r$ for each degree class k.
- The GNN outperforms the MLE on Erdos-Renyi and Barabasi-Albert networks.

A **Barabasi-Albert** network is one in which we begin with m_0 nodes fully connected to each other. Nodes are then added one at a time, and are each connected to $m \leq m_0$ existing nodes. A connection with an existing node i is made with probability

$$p_i = \frac{k_i}{\sum_j k_j}.$$

So we tend to see certain nodes become "hubs" and others become relatively isolated.

Performance

A Graphical Interlude

Graph Neural Network Architecture

First transform state of every node, x_i, with shared multilayer perceptron using

$$\hat{f}_{in} : \mathcal{S} \to \mathbb{R}^d$$
$$x_i \mapsto \hat{f}_{in}(x_i) =: \xi_i$$

or alternatively, if the attributes Φ_i are available

$$\hat{f}_{in} : \mathcal{S} \times \mathbb{R}^Q \to \mathbb{R}^d.$$

Graph Neural Network Architecture

First transform state of every node, x_i, with shared multilayer perceptron using

$$\hat{f}_{in} : \mathcal{S} \to \mathbb{R}^d$$
$$x_i \mapsto \hat{f}_{in}(x_i) =: \xi_i$$

or alternatively, if the attributes Φ_i are available

$$\hat{f}_{in} : \mathcal{S} \times \mathbb{R}^Q \to \mathbb{R}^d.$$

We aggregate features using a modified attention mechanism

$$\nu_i = \hat{f}_{att}\left(\xi_i, \xi_{\mathcal{N}_i}\right).$$

Graph Neural Network Architecture

First transform state of every node, x_i, with shared multilayer perceptron using

$$\hat{f}_{in} : \mathcal{S} \to \mathbb{R}^d$$
$$x_i \mapsto \hat{f}_{in}(x_i) =: \xi_i$$

or alternatively, if the attributes Φ_i are available

$$\hat{f}_{in} : \mathcal{S} \times \mathbb{R}^Q \to \mathbb{R}^d.$$

We aggregate features using a modified attention mechanism

$$\nu_i = \hat{f}_{att}\left(\xi_i, \xi_{\mathcal{N}_i}\right).$$

In addition, we have another MLP for the edge attributes

$$\psi_{ij} = \hat{f}_{edge}(\Omega_{ij}).$$

Graph Neural Network Architecture

First transform state of every node, x_i, with shared multilayer perceptron using

$$\hat{f}_{in} : \mathcal{S} \to \mathbb{R}^d$$
$$x_i \mapsto \hat{f}_{in}(x_i) =: \xi_i$$

or alternatively, if the attributes Φ_i are available

$$\hat{f}_{in} : \mathcal{S} \times \mathbb{R}^Q \to \mathbb{R}^d.$$

We aggregate features using a modified attention mechanism

$$\nu_i = \hat{f}_{att}\left(\xi_i, \xi_{\mathcal{N}_i}\right).$$

In addition, we have another MLP for the edge attributes

$$\psi_{ij} = \hat{f}_{edge}(\Omega_{ij}).$$

The outcome of each node is

$$\hat{y}_i = \hat{f}_{out}(\nu_i).$$

Graph Neural Network Architecture

Focusing on the attention, let

Graph Neural Network Architecture

Focusing on the attention, let

$$a_{ij} = \sigma\left[\mathcal{A}\left(\xi_i\right) + \mathcal{B}\left(\xi_j\right) + \mathcal{C}\left(\psi_{ij}\right)\right],$$

where $\sigma(.)$ is the usual sigmoid function.

Graph Neural Network Architecture

Focusing on the attention, let

$$a_{ij} = \sigma\left[\mathcal{A}\left(\xi_i\right) + \mathcal{B}\left(\xi_j\right) + \mathcal{C}\left(\psi_{ij}\right)\right],$$

where $\sigma(.)$ is the usual sigmoid function. Therefore, $a_{ij} \in (0, 1)$, where $a_{ij} = 0$ implies that the state of v_j has no effect on the outcome of v_i, and $a_{ij} = 1$ implies the effect is maximal.

Graph Neural Network Architecture

Focusing on the attention, let

$$a_{ij} = \sigma\left[\mathcal{A}\left(\xi_i\right) + \mathcal{B}\left(\xi_j\right) + \mathcal{C}\left(\psi_{ij}\right)\right],$$

where $\sigma(.)$ is the usual sigmoid function. Therefore, $a_{ij} \in (0, 1)$, where $a_{ij} = 0$ implies that the state of v_j has no effect on the outcome of v_i, and $a_{ij} = 1$ implies the effect is maximal. The attention is then

$$\nu_i = \hat{f}_{att}(\xi_i, \xi_{\mathcal{N}_i}) = \xi_i + \sum_{v_j \in \mathcal{N}_i} a_{ij}\xi_j.$$

Other important details

- The rectified Adam algorithm was used to optimise the hyperparameters.
- When $\mathcal{S}$ is discrete and countable, can simplify inputs to $\rho(.)$, which we estimate as

$$\rho(k, x, l) = \frac{1}{|\mathcal{V}|\, T} \sum_{i=1}^{|\mathcal{V}|} \mathbb{1}\left(k_i = k\right) \times \sum_{t=1}^{T} \mathbb{1}\left(x_t(t) = x\right) \mathbb{1}\left(l_i(t) = l\right).$$

- When we have continuous states, we cannot estimate ρ directly, and so we instead use

$$w_i(t) = \left[P(k_i)\Sigma\left(\Phi_i, \Omega_i | k_i\right) \Pi\left(\bar{x}(t)\right)\right]^{-\lambda}.$$