Zang and Wang 2019: Neural Dynamics on Complex Networks

Minqi Pan

April 4, 2020

Neural Dynamics on Complex Networks

- AAAI 2020, Best Paper of "The 1st International Workshop on Deep Learning on Graphs: Methodologies and Applications", Feb 8th, 2020
- Chengxi Zang and Fei Wang
- Weill Cornell Medicine

Outline

Outline

The Differential Equation System

$$\frac{dX(t)}{dt} = f(X(t), G, W(t), t)$$

- $X(t) \in \mathbb{R}^{n \times d}$: the state (node feature values) of a dynamic system consisting of n linked nodes at time $t \in [0, \infty)$, and each node is characterized by d dimensional features
- $f : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d}$: a function governing the dynamics of the system, which could be either linear or nonlinear
- $G = (\mathcal{V}, \mathcal{E})$: the network structure capturing how the nodes are linked to each other
- $W(t)$: the parameters which control how the system evolves over time
- $X(0) = X_0$: the initial state of this system at time $t = 0$

Semantic Labels

- $Y(X, \Theta, t) \in \{0, 1\}^{n \times k}$: the semantic labels of the nodes at time t
- Θ: the parameters of this classification function

Problem #1: Network Dynamics Learning

- Given a graph G and the observations of the states of system:

$$\{X(\hat{t_1}), X(\hat{t_2}), \ldots, X(\hat{t_T}) : 0 \leqslant t_1 \leqslant \cdots \leqslant t_T\}$$

- t_1 to t_T are arbitrary physical time stamps, possibly irregularly sampled with different observational time intervals
- How to learn the continous-time dynamics $\frac{dX(t)}{dt}$ on complex networks from empirical data? Can we learn differential equation systems $\frac{dX(t)}{dt} = f(X(t), G, W(t), t)$ to generate or predict continuous-time dynamics $X(t)$ at arbitrary physical time t?
    - "extrapolation prediction": when $t > t_T$
    - "interpolation prediction": when $t < t_T$ and $t \neq \{t_1, \ldots, t_T\}$

Problem #2: Structured Sequence Learning

- A special case of the problem of Network Dynamics Learning
- $t_1, t_2, \ldots, t_T$ are sampled regularly with equal time intervals
- Emphasizing on sequential order instead of arbitrary physical time
- The goal is to exptrapolate next m steps:

$$X[t_T + 1], \ldots, X[t_T + m]$$

Problem #3: One-snapshot Learning

- A special case of the problem of Network Dynamics Learning
- How to learn the semantic labels of $Y(X(t_T))$ at the moment $t = t_T$ for each node?
- Emphasizing on a specific moment
- Without loss of generality, we focus on the moment at the terminal time t_T
- The function Y can be a mapping from the nodes' states (e.g. humidity) to their labels (e.g. taking umbrella or not)

Network Dynamics #1: Heat Diffusion

- Let $\overrightarrow{x_i(t)} \in \mathbb{R}^{d \times 1}$ be d dimensional features of node i at time t
- Thus

$$X(t) = \begin{bmatrix} \vdots \\ \overrightarrow{x_i(t)} \\ \vdots \end{bmatrix}$$

- The heat diffusion dynamics governed by Newton's law of cooling

$$\frac{d\overrightarrow{x_i(t)}}{dt} = -k_{i,j} \sum_{j=1}^{n} A_{i,j}(\overrightarrow{x_i} - \overrightarrow{x_j})$$

which states that the rate of heat change of node i is proportional to the difference of the temperature between node i and its neighbors with heat capacity matrix A

Network Dynamics #2: Mutualistic Interaction

- The mutualistic differential equation systems capture the abundance $\overrightarrow{x_i(t)}$ of species i in ecology:

$$\frac{d\overrightarrow{x_i(t)}}{dt} = b_i + \overrightarrow{x_i}\left(1 - \frac{\overrightarrow{x_i}}{k_i}\right)\left(\frac{\overrightarrow{x_i}}{c_i} - 1\right) + \sum_{j=1}^{n} A_{i,j} \frac{\overrightarrow{x_i}\overrightarrow{x_j}}{d_i + e_i\overrightarrow{x_i} + h_j\overrightarrow{x_j}}$$

  - with incoming migration term b_i
  - with logistic growth with population capacity k_i
  - with Allee effect with cold-start threshold c_i
  - with mutualistic interaction term with interaction network A
- For brevity, the operations between vectors are element-wise

Network Dynamics #3: Gene Regulatory

- Governed by Michaelis-Menten equation

$$\frac{d\overrightarrow{x_i(t)}}{dt} = -b_i \overrightarrow{x_i(t)}^f + \sum_{j=1}^{n} A_{i,j} \frac{\overrightarrow{x_j}^h}{\overrightarrow{x_j}^h + 1}$$

  - the 1st term models degradation when $f = 1$ or dimerization when $f = 2$
  - the 2nd term captures genetic activation tuned by the Hill coefficient h

Complex Networks

1. "Grid" where each node is connected with 8 neighbors
2. "Random" generated by Erdós and Rényi model
3. "Power-law" generated by Albert-Barabási model
4. "Small-world" generated by Watts-Strogatz model
5. "Community" generated by random partitionmodel

Visualization

- To visualize dynamics on complex networks over time is not trivial
- We firsts generate a network with n nodes by aforementioned network models
- The nodes are re-ordered according to the community detection method by Newman
- Each node has a unique label from 1 to n
- We layout these nodes on a 2-dimensional $\sqrt{n} \times \sqrt{n}$ grid and each grid point $(r, c) \in \mathbb{N}^2$ represents the i^{th} node where $i = r\sqrt{n} + c + 1$
- Thus, nodes' states $X(t) \in \mathbb{R}^{n \times d}$ at time t when $d = 1$ can be visualized as a scalar field function $X : \mathbb{N}^2 \to \mathbb{R}$ over the grid

General Framework

$$\underset{W(t),\Theta(T)}{\arg\min} \ \mathcal{L} = \int_0^T \mathcal{R}\left(X(t), G, W, t\right) dt + \mathcal{S}\left(Y(X(T), \Theta)\right)$$

$$\text{subject to } \frac{dX(t)}{dt} = f(X(t), G, W, t), X(0)$$

- $\mathcal{R}(X(t), G, W, t)$: the running loss of the dynamics on graph at time t
- $\mathcal{S}(Y(X(T), \Theta))$: the terminal semantic loss at time T
- By integrating $\frac{dX(t)}{dt} = f(X(t), G, W, t)$ over time t from initial state X_0, a.k.a. solving the initial value problem for this differential equation system, we can get the continous-time dynamics $X(t) = x(0) + \int_0^T f(X(\tau), G, W, \tau)d\tau$ at arbitrary time moment $t > 0$

As an Optimal Control Problem

- By solving the above optimization problem
  - Obtain the best control parameters $W(t)$ for differential equation system $\frac{dX}{dt} = f(X, G, W, t)$
  - Obtain the best classification parameters Θ for semantic function $Y(X(t), \Theta)$
- Differences from the traditional Optimal Control framework: We model the differential equation systems

$$\frac{dX}{dt} = f(X, G, W, t)$$

by graph neural networks

In a Dynamical System View

- By integration $\frac{dX}{dt} = f(X, G, W, t)$ over continuous time, namely

$$X(t) = X(0) + \int_0^t f(X(\tau), G, W, \tau)d\tau$$

  we get our differential deep learning models
- Our differential deep learning models can be a time-varying coefficient dynamical system where $W(t)$ changes over time
- Or a constant coefficient dynamical system when W is constant over time for parameter sharing

Further Encoding (1)

$$\begin{aligned}
\underset{W(t),\Theta(T)}{\arg\min}\ \mathcal{L} &= \int_0^T \mathcal{R}\left(X(t), G, W, t\right) dt + \mathcal{S}\left(Y(X(T), \Theta)\right) \\
\text{subject to } X_h(t) &= f_{\text{encode}}(X(t)) \\
\frac{dX_h(t)}{dt} &= f(X_h(t), G, W, t), X_h(0) \\
X(t) &= f_{\text{decode}}(X_h(t))
\end{aligned}$$

- To further increase the express ability of our model, we can encode the network signal $X(t)$ from the original space to $X_h(t)$ in hidden space (usually with a different number of dimensions), and learn the dynamics in such a space

Further Encoding (2)

$$\begin{aligned}
\underset{W(t),\Theta(T)}{\arg\min} \quad & \mathcal{L} = \int_0^T \mathcal{R}\left(X(t), G, W, t\right) dt + \mathcal{S}\left(Y(X(T), \Theta)\right) \\
\text{subject to} \quad & X_h(t) = f_{\text{encode}}(X(t)) \\
& \frac{dX_h(t)}{dt} = f(X_h(t), G, W, t), X_h(0) \\
& X(t) = f_{\text{decode}}(X_h(t))
\end{aligned}$$

- The 1st constraint transforms $X(t)$ into hidden space $X_h(t)$
- The 2nd constraint is the governing dynamics in the hidden space
- The 3rd constraint decodes the hidden signal back to the original space

Further Encoding (3)

$$\underset{W(t),\Theta(T)}{\arg\min}\ \mathcal{L} = \int_0^T \mathcal{R}\left(X(t), G, W, t\right) dt + \mathcal{S}\left(Y(X(T), \Theta)\right)$$

$$\text{subject to } X_h(t) = f_{\text{encode}}(X(t))$$

$$\frac{dX_h(t)}{dt} = f(X_h(t), G, W, t), X_h(0)$$

$$X(t) = f_{\text{decode}}(X_h(t))$$

- The design of $f_{\text{encode}}, f, f_{\text{decode}}$ are flexible to be any neural structure, e.g. Softmax as the decoder for classficiation
- We denote this model as "NDCN"

Discrete Layers vs. Continuous Layers

- The deep learning methods with L hidden neural layers f_* are

$$X[L] = f_L \circ \cdots \circ f_2 \circ f_1(X[0]),$$

which are iterated maps with an integer number of discrete layers and thus cannot learn continous-time dynamics $X(t)$ at arbitrary time

- In contrast, our model

$$X(t) = X(0) + \int_0^t f(X(\tau), G, W, \tau) d\tau$$

can have contiunous layers with a real number t depth corresponding to continous-time dynamics

Solving the Initial Value Problem

- Integrate the differential equation systems over time by numerical methods
- The numerical methods can approximate continuous-time dynamics

$$X(t) = X(0) + \int_0^t f(X(\tau), G, W, \tau) d\tau$$

  at arbitrary time t accurately with guaranteed error
- In order to learn the learnable parameters W, we back-propogate the gradients of the loss function w.r.t. the control parameters $\frac{\partial \mathcal{L}}{\partial W}$ over the numerical integration process backwards in an end-to-end manner, and solve the optimization problem by stochastic gradient descent methods

Outline

The Continous-time Setting

- The observational times t_1 to t_T of the observed states of system

$$\{X(\hat{t}_1), X(\hat{t}_2), \ldots, X(\hat{t}_T) : 0 \leqslant t_1 \leqslant \cdots \leqslant t_T\}$$

  are arbitrary physical time stamps which are irregularly sampled with different observational time intervals
- Extrapolation prediction is to predict

$$X(t)$$

  at arbitrary physical time moment t when $t > t_T$
- Interpolation prediction is to predict

$$X(t)$$

  when $t < t_T$ and $t \neq \{t_1, \ldots, t_T\}$

Model Instance (1)

$$\underset{W_*, b_*}{\arg\min} \ \mathcal{L} = \int_0^T |X(t) - \hat{X(t)}| dt$$

$$\text{subject to } X_h(t) = \tanh(X(t)W_e + b_e)W_0 + b_0$$

$$\frac{dX_h(t)}{dt} = \text{ReLU}(\Phi X_h(t)W + b), X_h(0)$$

$$X(t) = X_h(t)W_d + b_d$$

- Loss: emphasizing on running loss only; use ℓ_1-norm loss as the running loss $\mathcal{R}$
- $|\cdot|$: ℓ_1-norm loss (element-wise absolute value difference) between $X(t)$ and $\hat{X(t)}$ at time $t \in [0, T]$

Model Instance (2)

$$\underset{W_*, b_*}{\arg\min} \ \mathcal{L} = \int_0^T |X(t) - \hat{X(t)}| dt$$

$$\text{subject to } X_h(t) = \tanh(X(t)W_e + b_e)W_0 + b_0$$

$$\frac{dX_h(t)}{dt} = \mathrm{ReLU}(\Phi X_h(t)W + b), X_h(0)$$

$$X(t) = X_h(t)W_d + b_d$$

- The encoding function: two fully connected neural layers with a nonlinear hidden layer as the encoding function
- the linear decoding function: for regression tasks in the original signal space

Model Instance (3)

$$\arg\min_{W_*, b_*} \mathcal{L} = \int_0^T |X(t) - \hat{X(t)}| dt$$

$$\text{subject to } X_h(t) = \tanh(X(t)W_e + b_e)W_0 + b_0$$

$$\frac{dX_h(t)}{dt} = \text{ReLU}(\Phi X_h(t)W + b), X_h(0)$$

$$X(t) = X_h(t)W_d + b_d$$

- $\hat{X(t)} \in \mathbb{R}^{n \times d}$: the supervised dynamic information available at time stamp t
    - in the semi-supervised case the missing information can be padded by 0

Model Instance (4)

$$\underset{W_*, b_*}{\arg\min} \ \mathcal{L} = \int_0^T |X(t) - \hat{X(t)}| dt$$

$$\text{subject to } X_h(t) = \tanh(X(t)W_e + b_e)W_0 + b_0$$

$$\frac{dX_h(t)}{dt} = \mathsf{ReLU}(\Phi X_h(t)W + b), X_h(0)$$

$$X(t) = X_h(t)W_d + b_d$$

- $\Phi = D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}} \in \mathbb{R}^{n \times n}$: graph diffusion operator to model the instantaneous network dynamics in the hidden space, which is the normalized graph Laplacian
  - $A \in \mathbb{R}^{n \times n}$: the adjacency matrix of the network
  - $D \in \mathbb{R}^{n \times n}$: the corresponding node degree matrix

Model Instance (5)

$$\underset{W_*, b_*}{\arg\min}\ \mathcal{L} = \int_0^T |X(t) - \hat{X(t)}| dt$$

subject to $X_h(t) = \tanh(X(t)W_e + b_e)W_0 + b_0$

$$\frac{dX_h(t)}{dt} = \mathsf{ReLU}(\Phi X_h(t)W + b), X_h(0)$$

$$X(t) = X_h(t)W_d + b_d$$

- $W \in \mathbb{R}^{d_e \times d_e}$ and $b \in \mathbb{R}^{n \times d_e}$: shared parameters (namely, the weights and bias of a linear connection layer) over time $t \in [0, T]$
- $W_e \in \mathbb{R}^{d \times d_e}$ and $W_0 \in \mathbb{R}^{d_2 \times d}$: for decoding
- b_e, b_0, b, b_d: the biases at the corresponding layer

Model Instance (6)

$$\arg\min_{W_*, b_*} \mathcal{L} = \int_0^T |X(t) - \hat{X(t)}| dt$$

$$\text{subject to } X_h(t) = \tanh(X(t)W_e + b_e)W_0 + b_0$$

$$\frac{dX_h(t)}{dt} = \text{ReLU}(\Phi X_h(t)W + b), X_h(0)$$

$$X(t) = X_h(t)W_d + b_d$$

- We learn the parameters

$$W_e, W_0, W, W_d, b_e, b_0, b, b_d$$

from empirical data so that we can learn X in a data-driven manner

Model Instance (7)

$$\underset{W_*,b_*}{\arg\min}\ \mathcal{L} = \int_0^T |X(t) - \hat{X(t)}|dt$$

$$\text{subject to } X_h(t) = \tanh(X(t)W_e + b_e)W_0 + b_0$$

$$\frac{dX_h(t)}{dt} = \text{ReLU}(\Phi X_h(t)W + b), X_h(0)$$

$$X(t) = X_h(t)W_d + b_d$$

- $\frac{dX(t)}{dt}$: a single neural layer at time moment t
- $X(t)$ at arbitrary time t is achieved by integrating $\frac{dX(t)}{dt}$ over time, leading to a continous-time deep neural network:

$$X(t) = X(0) + \int_0^t \text{ReLU}(\Phi X(\tau)W + b)d\tau$$

Outline

Baselines

- There are no baselines for learning continous-time dynamics on complex networks
- Thus we compare the ablation models of NDCN
- By investigating ablation models we show that NDCN is a minimum model for this task

Baseline #1

- Keep the loss function the same
- The model without encoding and decoding functions
- Thus no hidden space:

$$\frac{dX(t)}{dt} = \text{ReLU}(\Phi X(t)W + b),$$

- Namely ODE-GNN, which learns the dynamics in the original signal space $X(t)$

Baseline #2

- Keep the loss function the same
- The model without graph diffusion operator

$$\Phi : \frac{dX_h(t)}{dt} = \mathsf{ReLU}(X_h(t)W + b),$$

- I.e. an ODE Neural Network, which can be though as a continous-time version of forward residual neural network

Baseline #3

- Keep the loss function the same
- The model without control parameters W:

$$\frac{dX_h(t)}{dt} = \text{ReLU}(\Phi X_h(t))$$

- No linear connnection layer between t and $t + dt$ where $dt \to 0$
- Thus indicating a determined dynamics to spread signals

Experimental Setup (1)

- We generate underlying networks with 400 nodes by Network Dynamics #1-#3 and Complex Networks #1-#5
- We set the initial value $X(0)$ the same for all the experiments
- Thus different dynamics are only due to their different dynamic rules and underlying networks

Experimental Setup (2)

- We irregularly sample 120 snapshots of the continuous-time dynamics

$$\{X(\hat{t_1}), \ldots, X(\hat{t}_{120}) : 0 \leqslant t_1 < \cdots < t_{120} \leqslant T\}$$

  where the time intervals between $t_1, \ldots, t_{120}$ are different
- Training: Randomly choose 80 snapshots from $X(\hat{t_1})$ to $X(\hat{t}_{100})$
- Interpolation testing: the left 20 snapshots from $X(\hat{t_1})$ to $X(\hat{t}_{100})$
- Extrapolation testing: use the 20 snapshots from $X(\hat{t}_{101})$ to $X(\hat{t}_{120})$

Experimental Setup (3)

- We use Dormand-Prince method to get the ground truth dynamics
- We use Euler method in the forward process of our NDCN
- We evaluate the results by ℓ_1 loss and normalized ℓ_1 loss (normalized by the mean element-wise value of $\hat{X(t)}$) and they lead to the same conclusion
- Results are the mean and standard deviation of the loss over 20 independent runs for 3 dynamic laws on 5 different networks by each method

Results (Visual)

- We find that one dyanmic law may behave quite different on different networks
  - Heat dynamics may gradually die out to be stable but follow different dynamic pattern on different networks
  - Gene dynamics are asymptotically stable on grid but unstable on random networks or community networks
  - Both gene regulation dynamics and biological mutualistic dynamics show very bursty patterns on power-law networks
- NDCN learns all these different network dynamics veryt well

Results (Quantitative)

- Each quantitative result is the normalized ℓ_1 error with standard deviation (in percentage %) from 20 runs for 3 dynamics on 5 networks by each method
- NDCN captures different dynamics on various complex networks accurately
- NDCN outperforms all the continuous-time baselines by a large margin
- NDCN potentially serves as a minimum model in learning contiunous-time dynamics on complex networks

Outline

Baselines

- We compare our model with the temporal-GNN models
  - Temporal-GNN are usually combinations of RNN models and GNN models
  - Temporal-GNN models are usually used for next few step prediction and cannot be used for interpolation task (say, to predict $X[t_{1.23}]$)
- We use GCN as a graph structure extractor
- We use LSTM/GRU/RNN to learn the temporal relationship between ordered structured sequences

Baseline #1

- We keep the loss function the same
- LSTM-GNN: the temporal-GNN with LSTM cell

$$X[t+1] = \mathsf{LSTM}(\mathsf{GCN}(X[t], G))$$

Baseline #2

- We keep the loss function the same
- GRU-GNN: the temporal-GNN with GRU cell

$$X[t+1] = \mathsf{GRU}(\mathsf{GCN}(X[t], G))$$

Baseline #1

- We keep the loss function the same
- RNN-GNN: the temporal-GNN with RNN cell

$$X[t+1] = \mathsf{RNN}(\mathsf{GCN}(X[t], G))$$

Experimental Setup

- We regularly sample 100 snapshots of the continuous-time network dynamics

$$\{X[\hat{t}_1], \ldots, X[\hat{t}_{100}] : 0 \leqslant t_1 < \cdots < t_{120} \leqslant T\}$$

  where the time intervals between $t_1, \ldots, t_{100}$ are the same
- Training: use first 80 snapshots $X[\hat{t}_1], \ldots, X[\hat{t}_{80}]$
- Prediction/Extrapolation Testing: use the left 20 snapshots $X[\hat{t}_{81}], \ldots, X[\hat{t}_{100}]$
- We use 5 and 10 for hidden dimension of GCN and RNN models respectively

Results

- GRU-GNN model works well in mutualistic dynamics on random network and community network
- NDCN predicts different dynamics on these complex networks accurately
- NDCN outperforms the baselines in almost all the settings
- NDCN captures the structure and dynamics in a much more succinct way
- NDCN only has 901 parameters to learn, compared to $24k, 64k, 84k$ of RNN-GCN, GRU-GNN, LSTM-GNN, respectively

Outline

Learning the Semantic Labels at the Terminal Time

- Existing GNNs (s.o.t.a. in graph semi-supervised classification task) usually adopt 1 or 2 hidden layers
- NDCN follows the perspective of a dynamical system and goes beyond an integer number L of hidden layers in GNNS to a real number depth t of hidden layers, implying continuous-time dynamics on the graph
- By integration continous-time dynamics on the graph over time, we get a more fine-grained forward process
- Thus NDCN model shows very competitive even better results compared with s.o.t.a. GNN models which may have sophisticated parameters (e.g. attention)

Model Instance (1)

$$\begin{aligned}
\underset{W_e,b_e,W_d,b_d}{\arg\min} \quad & \mathcal{L} = \int_0^T \mathcal{R}(t)dt - \sum_{i=1}^{n}\sum_{k=1}^{c} \hat{Y}_{i,k}(T)\log Y_{i,k}(T) \\
\text{subject to } & X_h(0) = \tanh(X(0)W_e + b_e) \\
& \frac{dX_h(t)}{dt} = \text{ReLU}(\Phi X_h(t)) \\
& X(T) = \text{Softmax}(X_h(T)W_d + b_d)
\end{aligned}$$

- Loss: terminal semantic loss $\mathcal{S}(Y(T))$ modeled by the cross-entropy loss for classification task

Model Instance (2)

$$\underset{W_e, b_e, W_d, b_d}{\arg\min} \quad \mathcal{L} = \int_0^T \mathcal{R}(t)dt - \sum_{i=1}^{n}\sum_{k=1}^{c} \hat{Y}_{i,k}(T) \log Y_{i,k}(T)$$

$$\text{subject to } X_h(0) = \tanh(X(0)W_e + b_e)$$

$$\frac{dX_h(t)}{dt} = \text{ReLU}(\Phi X_h(t))$$

$$X(T) = \text{Softmax}(X_h(T)W_d + b_d)$$

- $Y(T) \in \mathbb{R}^{n \times c}$: the label distributions of nodes at time $T \in \mathbb{R}$ whose
  - $Y_{i,k}(T)$: the probability of the node $i = 1, \ldots, n$ with label $k = 1, \ldots, c$ at time T
- $\hat{Y}(T) \in \mathbb{R}^{n \times c}$: the supervised information (again missing information can be padded by 0) observed at $t = T$

Model Instance (3)

$$\begin{aligned}\underset{W_e,b_e,W_d,b_d}{\arg\min} \quad & \mathcal{L} = \int_0^T \mathcal{R}(t)dt - \sum_{i=1}^{n}\sum_{k=1}^{c} \hat{Y}_{i,k}(T)\log Y_{i,k}(T) \\ \text{subject to } & X_h(0) = \tanh(X(0)W_e + b_e) \\ & \frac{dX_h(t)}{dt} = \text{ReLU}(\Phi X_h(t)) \\ & X(T) = \text{Softmax}(X_h(T)W_d + b_d)\end{aligned}$$

- We use differential equation system $\frac{dX(t)}{dt} = \text{ReLU}(\Phi X(t))$ to spread the graph signals over continuous time $[0, T]$, i.e.,

$$X_h(T) = X_h(0) + \int_0^T \text{ReLU}(\Phi X_h(t))$$

Model Instance (4)

$$\underset{W_e,b_e,W_d,b_d}{\arg\min} \quad \mathcal{L} = \int_0^T \mathcal{R}(t)dt - \sum_{i=1}^{n}\sum_{k=1}^{c} \hat{Y}_{i,k}(T)\log Y_{i,k}(T)$$

$$\text{subject to } X_h(0) = \tanh(X(0)W_e + b_e)$$

$$\frac{dX_h(t)}{dt} = \text{ReLU}(\Phi X_h(t))$$

$$X(T) = \text{Softmax}(X_h(T)W_d + b_d)$$

- Compared with the continuous-time model instance, we only have supervised information from one shapshot at time $t = T$
- Thus we model the running loss as the ℓ_2-norm regularizer of the learnable parameters to avoid overfitting: $\int_0^T \mathcal{R}(t)dt = \lambda(|W_e|_2^2 + |b_e|_2^2 + |W_d|_2^2 + |b_d|_2^2)$

Model Instance (5)

$$\underset{W_e,b_e,W_d,b_d}{\arg\min} \quad \mathcal{L} = \int_0^T \mathcal{R}(t)dt - \sum_{i=1}^{n}\sum_{k=1}^{c} \hat{Y}_{i,k}(T)\log Y_{i,k}(T)$$

$$\text{subject to } X_h(0) = \tanh(X(0)W_e + b_e)$$

$$\frac{dX_h(t)}{dt} = \text{ReLU}(\Phi X_h(t))$$

$$X(T) = \text{Softmax}(X_h(T)W_d + b_d)$$

- We adopt the diffusion operator $\Phi = \tilde{D}^{-\frac{1}{2}}(\alpha I + (1-\alpha)A)\tilde{D}^{-\frac{1}{2}}$ where A is the adjacency matrix, D is the degree matrix and $\tilde{D} = \alpha I + (1-\alpha)D$ keeps Φ normalized

Model Instance (6)

$$\underset{W_e, b_e, W_d, b_d}{\arg\min} \quad \mathcal{L} = \int_0^T \mathcal{R}(t)dt - \sum_{i=1}^{n} \sum_{k=1}^{c} \hat{Y}_{i,k}(T) \log Y_{i,k}(T)$$

$$\text{subject to } X_h(0) = \tanh(X(0)W_e + b_e)$$

$$\frac{dX_h(t)}{dt} = \mathsf{ReLU}(\Phi X_h(t))$$

$$X(T) = \mathsf{Softmax}(X_h(T)W_d + b_d)$$

- The parameter $\alpha \in [0, 1]$ tunes nodes' adherence to their previous information or their neighbors' collective opinion
- We use α as a hyper-parameter here for simplicity and we can make it as a learnable parameter later

Model Instance (7)

- The differential equation system $\frac{dX}{dt} = \Phi X$ follows the dynamics of averaging the neighborhood opinion as

$$\frac{d\overrightarrow{x_i(t)}}{dt} = \frac{\alpha}{(1-\alpha)d_i + \alpha}\overrightarrow{x_i(t)} + \sum_j^n A_{i,j}\frac{1-\alpha}{\sqrt{(1-\alpha)d_i + \alpha}\sqrt{(1-\alpha)d_j + \alpha}}\overrightarrow{x_j(t)}$$

  for node i
- When $\alpha = 0$, Φ averages the nieghbors as normalized random walk
- When $\alpha = 1$, Φ captures exponential dynamics without network effects
- When $\alpha = 0.5$, Φ averages both neighbors and itself

Outline

Results (1)

- NDCN outperforms many s.o.t.a. GNN models
- We report the mean and standard deviation of our results for 100 runs
- Cora dataset: terminal time $T = 1.2, \alpha = 0$
- Citeseer dataset: $T = 1.0, \alpha = 0.8$
- Pubmed dataset: $T = 1.1, \alpha = 0.4$

Results (2)

- NDCN gives better classification accuracy at terminal time $T \in \mathbb{R}^+$ by capturing the continous-time network dynamcis to diffuse network siignals
- For all the three datasets their accuracy curves follow rise and fall patterns aroundd the best terminal time
- When the terminal time T is too small or too large, the accuracy degenerates because the features of nodes are in under-diffusion or over-diffusion states, implying the necessity in capturing continuous-time dynamics
- In contrast, previous GNNs can only have an discrete number of layers which cannot capture the continuous-time network dynamics accurately