Graph Neural Networks

Naeemullah Khan

naeemullah.khan@kaust.edu.sa

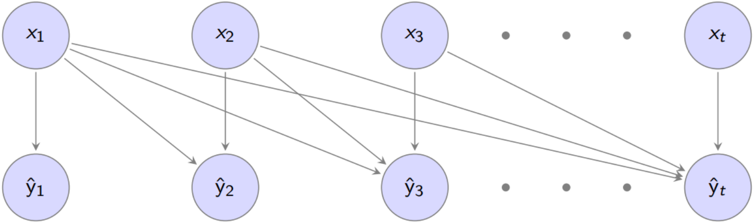
KAUST Academy
King Abdullah University of Science and Technology

February 1, 2023

Recap - GCNNs

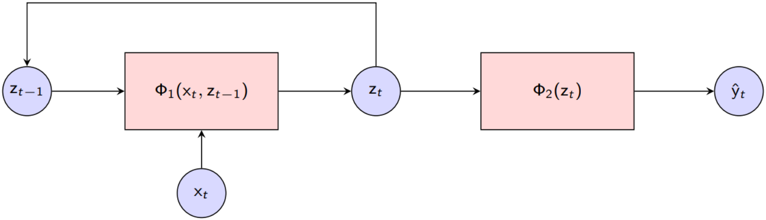
- Images can be represented as grids in space. Generalize convolutional filters to graphs.
- Graphs can be represented in form of matrices. Giving us graph Shift operators and making it easier to operate over them.
- Graph signal is a vector in which each component x_i is associated with node i.
- Graph signal, graph shift operator and graph filter make up the ingredients for the graph neural network.
- Graph signal is the input. Graph Shift operator is a parameter. We can also treat it as input if we want to consider different graphs. Graph filters are trainable parameters.
- A GCNN is composed of multiple graph perceptrons (graph filter + activation function).
- Individuals graph filters can be replaced with filter banks to learn multiple features.

- GCNNs are architectures specialized in learning data defined over graph supports.
- Several processes have a sequential nature. To learn from them, we need dedicated architectures.
- Often, we want to learn properties of a sequence.
- Predictions on a sequence depends on observation histories $y^t = \Phi(x_t, x_{t-1}, \cdots, x_1)$.



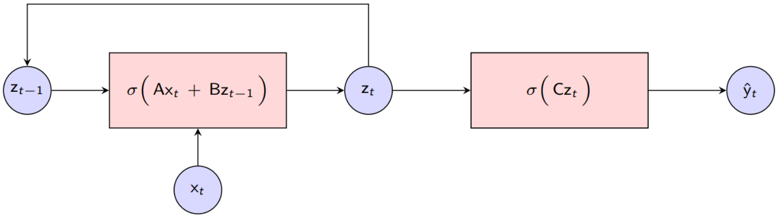
Recurrent Neural Networks (RNNs)

- A recurrent neural network is made up of two separate learning parametrizations.
  - $\Phi_1(x_t, z_{t-1}) \Rightarrow$ From observed state x_t and hidden state z_{t-1} to hidden state update z.
  - $\Phi_2(z_t) \Rightarrow$ From updated hidden state z_t to output estimate $\hat{y}_t$.
- It is a recurrent neural network because hidden states are fed-back as inputs for the next time step.

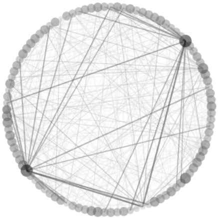


Recurrent Neural Networks (RNNs) (cont.)

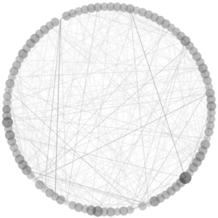
- We can use one perceptron to update the the hidden state $\Rightarrow \Phi_1(x_t, z_{t-1}) = \sigma(Ax_t + Bz_{t-1})$.
- And a second perceptron to predict the output $\Rightarrow \Phi_2(z_t) = \sigma(Cz_t)$.
- Number of trainable parameters $\equiv$ Entries of A, B and C.Does not depend on the time index t.



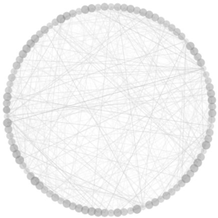
- We define Graph Recurrent Neural Networks (GRNNs) as particular cases of RNNs.
- Consider a time varying process x_t in which each of the signals is supported on shift operator $\mathbf{S}$.



x_{t-2}



x_{t-1}

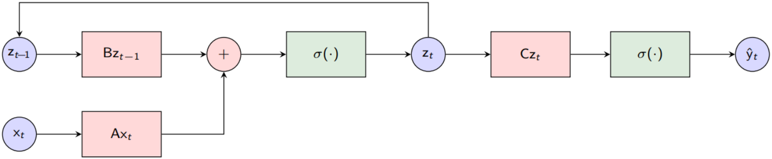


x_t

Graph Recurrent Nueral Networks (cont.)

- A graph recurrent neural network (GRNN) combines
  - A GNN because x_t is supported on a graph.
  - An RNN because x_t is a sequence.

- An RNN has a hidden state z_t updated with the perceptron $\Rightarrow z_t = \sigma(Ax_t + Bz_{t-1})$.
- And it has an output prediction $\hat{y}_t$ given by the perceptron $\Rightarrow y_t = \sigma(Cz_t)$.



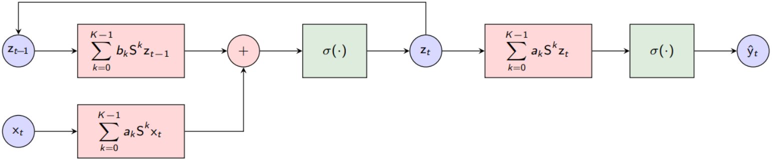
- The observed state x_t and the output y_t are graph signals supported on the graph shift operator $\mathbf{S}$.
- The hidden state z_t is constructed to be a graph signal supported on the graph shift operator $\mathbf{S}$.

- Hidden and observed state are propagated through graph filters to update the hidden state.

$$A = A(S) = \sum_{k=0}^{K-1} a_k S^k \qquad B = B(S) = \sum_{k=0}^{K-1} b_k S^k$$

- The state update is

$$z_t = \sigma\left[A(S)x_t + B(S)z_{t-1}\right] = \left[\sum_{k=0}^{K-1} a_k S^k x_t + \sum_{k=0}^{K-1} b_k S^k z_{t-1}\right]$$

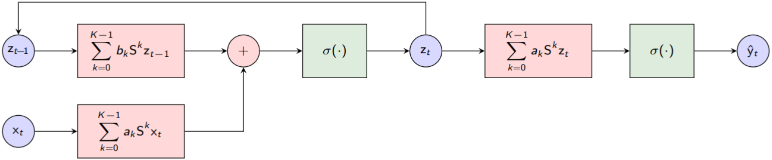


- The hidden state z_t is propagated through a graph filter to make a prediction $\hat{y}_t$ of the output y_t.

$$C = C(S) = \sum_{k=0}^{K-1} C_k S^k$$

- The prediction of the output y_t is given by

$$\hat{y}_t = \left[\sum_{k=0}^{K-1} c_k S^k z_t \right]$$



Graph Recurrent Nueral Networks (cont.)

- A GRNN is made up a hidden state update perceptron and an output prediction perceptron.

$$z_t = \left[\sum_{k=0}^{K-1} a_k S^k x_t + \sum_{k=0}^{K-1} b_k S^k z_{t-1} \right] \qquad \hat{y}_t = \sigma\left[C(S) z_t\right] = \left[\sum_{k=0}^{K-1} c_k S^k z_t \right]$$

- Each of these filters can be replaced by a MIMO filter bank to yield a GRNN with multiple features.

$$Z_t = \left[\sum_{k=0}^{K-1} S^k X_t A_k + \sum_{k=0}^{K-1} S^k Z_{t-1} B_k \right] \qquad \hat{Y}_t = \left[\sum_{k=0}^{K-1} S^k Z_t C_k \right]$$

- Multiple-feature hidden state Z_t permits larger dimensionality relative to observed states

Gating in GRNNs

- Like RNNs, GRNNs may also experience the problem of vanishing/exploding gradients.
- We address it by adding gating operators to GRNNs.
- Gates are scalars in [0, 1] acting on the current input and on the previous state. They control how much of the input and past time information should be taken into account.
- The value of each gate is updated at every step of the sequence.
- This allows creating paths through time with derivatives that neither vanish nor explode and creates dependency paths that allow encoding both short and long term dependencies.

Gating in GRNNs (cont.)

- We consider two types of gates here: Input Gate and Forget Gate.

$$Z_t = \sigma\left(\hat{\mathcal{Q}}\{\mathcal{A}_S(X_t)\} + \check{\mathcal{Q}}\mathcal{B}_S(Z_{t-1})\right)$$

- Input Gate operator $\hat{\mathcal{Q}} : \mathbb{R}^{N\times H} \to \mathbb{R}^{N\times H}$
  - controls the importance of the input X_t at time t.
- Forget Gate operator $\check{\mathcal{Q}} : \mathbb{R}^{N\times H} \to \mathbb{R}^{N\times H}$
  - controls the importance of the state Z_t at time t.

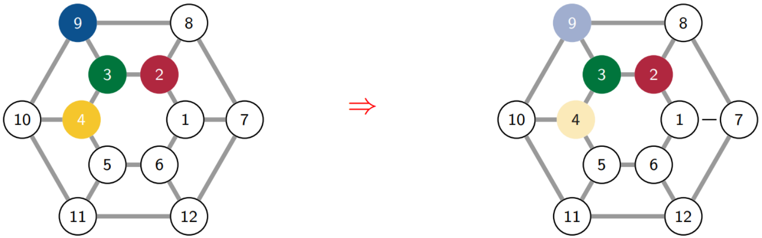
- First type of gating for GRNNs is time gating.
- Time gating multiplies the input and the state by scalar gates $\hat{q} \in [0, 1]$ and $\check{q} \in [0, 1]$.
- A single scalar gate is applied to the whole graph signal i.e., same gate value for all nodes.

Spatial Gating

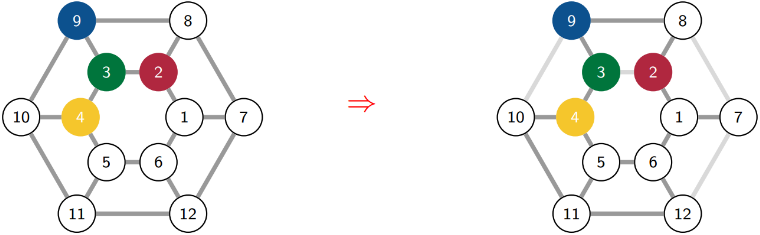
- Spatial imbalances can cause gradients to vanish in space as Some nodes/paths might get assigned more importance than others in long range exchanges.
- For example, graphs with community structure, where some nodes are highly connected within clusters.
  - Gradients of Z_T depend on successive products of $B(S) \Rightarrow$ successive products of S.
  - For large T, the matrix entries in S^T with highly connected nodes will get densely populated.
  - Overshadows community structure $\Rightarrow$ can't encode long processes that are local on the graph.

Spatial Gating (cont.)

- Spatial gating strategies help encode long range spatial dependencies in graph processes.
- Node and edge structure of the graph can allow for spatial gating.
- Node gating: One input and one forget gate for each node of the graph.



- Edge gating: One input and one forget gate for each edge of the graph.



- Node gating operators correspond to multiplication of the input and state by diagonal matrices.
  - The diagonals are the input and forget vector gates $\hat{q} \in [0,1]^N$ and $\check{q} \in [0,1]^N$.
  - A scalar gate applied to each nodal component of the signal i.e., different gate values for each node
- Node gating operators correspond to elementwise multiplication of the shift operator by gate matrices.
  - The matrices multiplying the GSOs are the input and forget matrix gates $\hat{Q} \in [0,1]^{N \times N}$ and $\check{Q} \in [0,1]^{N \times N}$.
  - Separate gate for each edge i.e., control the amount of information transmitted across edges.
- Parameters of input and forget gate operators are the outputs of GRNNs themselves.

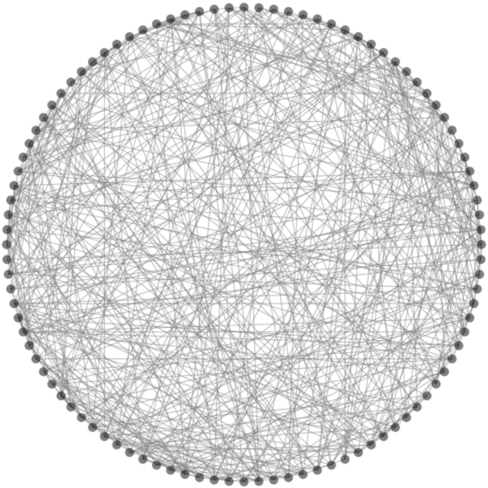
Application - Epidemic Modeling with GRNNs

- Model the spread of an infectious disease over a friendship network as a graph process.
- Graph is a symmetric friendship network corresponding to a high school in France.
- Model the spread of the disease on the graph using Susceptible-Infectious-Removed (SIR) model.
- Compare the performance of a GRNN, a RNN, and a GNN in predicting infections after 8 days.

Friendship Network

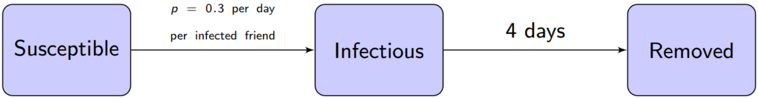
- Real-world friendship network corresponding to 134 students from a high school in Marseille.
- Each node of the graph represents a student.
- Friendships are modeled as symmetric unweighted edges.
- Isolated nodes are removed to make the graph fully connected.
- Assumption: friends are likely to be in contact with each other.

Friendship Network (cont.)



Susceptible-Infectious-Removed (SIR) Disease Model

- Process starts with random seed infections on day 0. Probability $p_{seed} = 0.05$.
- Each person is in one of the three SIR states. updated each day with the following rules.
- **Susceptible**: can get the disease from an infected friend with probability $p_{inf} = 0.3$.
- **Infectious**: can spread the disease for 4 days after being infected, after which they recover.
- **Removed**: have overcome the disease and can no longer spread it or contract it.



Problem Setup

- **Problem**: given the node states, goal is to predict whether each node will be infected in 8 days.
- **Input**: graph process x_t where, at each time t, $[x_t]_i$ is given by

$$[x_t]_i = \begin{cases} 0, & \text{if student i is susceptible} \\ 1, & \text{if student i is infectious} \\ 2, & \text{if student i is removed} \end{cases}$$

- **Output**: binary graph process y_t. Our goal is only to track infections.

$$[y_t]_i = \begin{cases} 0, & \text{if student i is susceptible or removed} \\ 1, & \text{if student i is infectious} \end{cases}$$

- Given $x_t, x_{t+1}, \cdots, x_{t+7}$, we want to predict $y_{t+8}, y_{t+9}, \cdots, y_{t+15} \Rightarrow$ binary node classification.

Objective Function

- Accuracy is not a good performance metric $\Rightarrow$ does not distinguish true positives and true negatives.
- In epidemic tracking, true positives are more important than true negatives $\Rightarrow$ maximize F1 score.

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

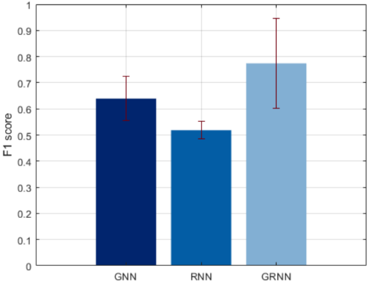
- Precision $=$ True Positive/Predicted Positive (Proportion of correct positive predictions)
- Recall $=$ True Positive/All Actual Positive (Proportion of correctly predicted positives)
- Loss function we minimize is $1 - F1 \Rightarrow$ trade-off between minimizing FPs and FNs.

Objective Function (cont.)

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Results

- We compare a GRNN with a GNN and a RNN, all with roughly the same number of parameters.
  - In the GNN, the time instants become input features $\Rightarrow$ parameters depend on T.
  - In the RNN, the nodal components become input features $\Rightarrow$ parameters depend on N



- GRNN improves upon RNN and GNN $\Rightarrow$ exploits both spatial and temporal structure of the data.

Summary

- Several processes have a sequential nature. To learn from them, we need dedicated architectures.
- Recurrent Neural Networks (RNNs) are designed to work with sequential data.
- We define Graph Recurrent Neural Networks (GRNNs) as particular cases of RNNs.
- Hidden and observed state are propagated through graph filters to update the hidden states and to predict outputs.
- Time and spatial Gating is used to deal with the problem of vanishing gradients in GRNNs.

References

These slides have been adapted from

- Alejandro Ribeiro, UPenn EE5140: Graph Neural Networks
- Jure Leskovec, Stanford CS224W: Machine Learning with Graphs