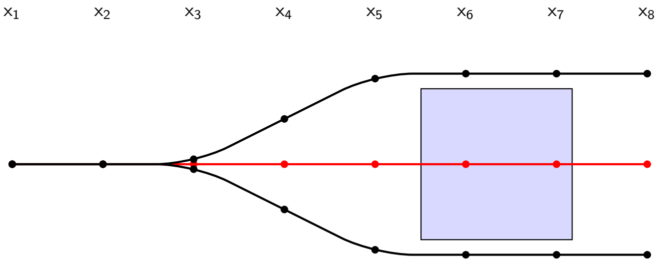
Machine Learning on Sequences

- GNNs are architectures specialized in learning data defined over graph supports
- Several processes have a sequential nature. To learn from them, we need dedicated architectures

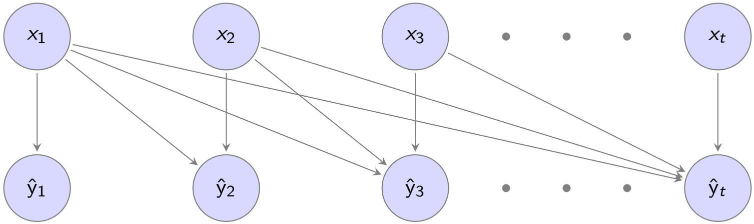
- Often, we want to learn properties of a sequence $\Rightarrow$ Is the particle entering the forbidden area?



- This problem is not just a simple sequence of classifications $\Rightarrow y_t = \phi(x_t)$

- It is a (sequence of) classifications of a sequence $\Rightarrow y_t = \phi(x_{1:t}) = \phi(x_t, x_{t-1}, \ldots, x_1)$

- Predictions on a sequence depends on observation histories $\Rightarrow \hat{y}_t = \Phi(x_t, x_{t-1}, \ldots, x_1)$



- Recurrent neural networks (RNNs) estimate a hidden state to avoid this unbounded memory growth

- A stochastic process (random sequence) is said to be Markov or memoryless if

$$p\left(\mathrm{x}_{t+1} \,\middle|\, \mathrm{x}_{1:t}\right) = p\left(\mathrm{x}_{t+1} \,\middle|\, \mathrm{x}_{t}\right)$$

- It is the same to condition on the current value x_t or conditioning or on the whole trajectory $x_{0:t}$

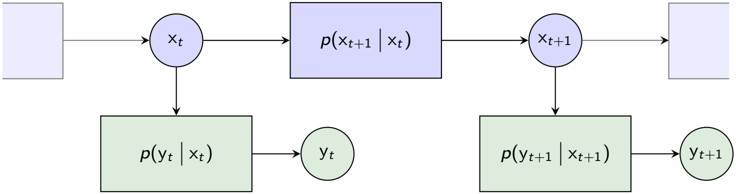
  $\Rightarrow$ The future, given the present, is independent of the past

  $\Rightarrow$ For predicting the future, knowledge of the past is irrelevant

- Outputs (e.g, trajectory categories) are conditionally independent $\Rightarrow p(\mathrm{y}_t \,|\, \mathrm{x}_t) = p(\mathrm{y}_t \,|\, \mathrm{x}_{1:t})$

- In a memoryless Markov Process, learning is equivalent reduce to a sequence of learning problems

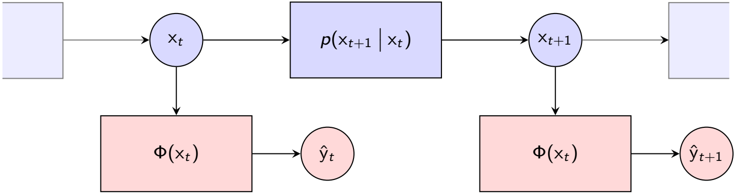
- State evolution is a chain of memoryless transitions. And outputs depend on the current state only



- An AI to predict y_t mimics the conditional distribution of the observations. The past is irrelevant

- In a memoryless Markov Process, learning is equivalent reduce to a sequence of learning problems

- State evolution is a chain of memoryless transitions. And outputs depend on the current state only



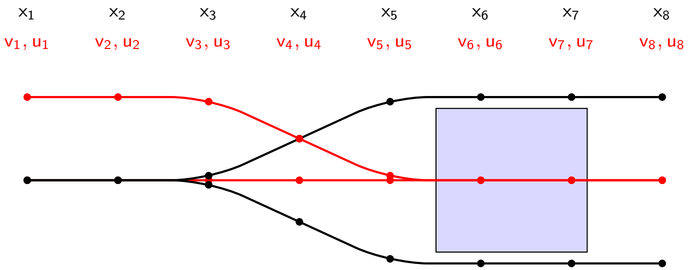
- An AI to predict y_t mimics the conditional distribution of the observations. The past is irrelevant

Recurrent Neural Networks

- Machine Learning in stochastic processes that are not Markov $\Rightarrow$ The past is relevant in learning

All Processes are Markov. However Disguised

- The evolution of the trajectory is not a Markov Process if we observe positions only
- But it is Markov if we have access to velocities and accelerations $\Rightarrow$ Hidden (unobserved) states



- All systems are Markov $\Rightarrow$ We often lack enough information to observe their Markov structure

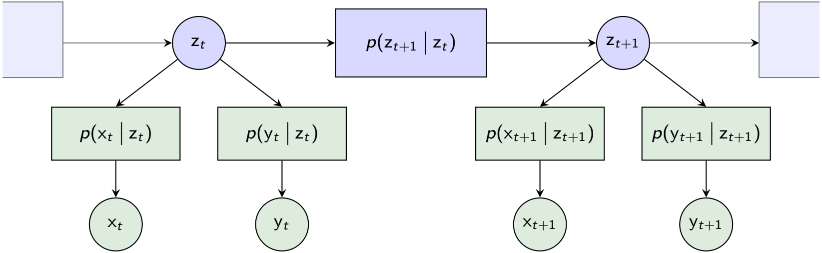
- Stochastic process x_t follows a hidden Markov model if there exists a process z_t such that

$$p\left(z_{t+1} \,\middle|\, z_{1:t}\right) = p\left(z_{t+1} \,\middle|\, z_t\right) \qquad \text{and} \qquad p\left(x_t \,\middle|\, z_t\right) = p\left(x_t \,\middle|\, z_{0:t}\right)$$

- The hidden state z_t is a memoryless Markov stochastic process
- The observed state x_t is conditionally independent. Depends only on the current hidden state z_t
- Outputs are also conditionally independent $\Rightarrow p(y_t \mid z_t) = p(y_t \mid z_{1:t})$

- In a hidden Markov model learning is not equivalent to a sequence of learning problems

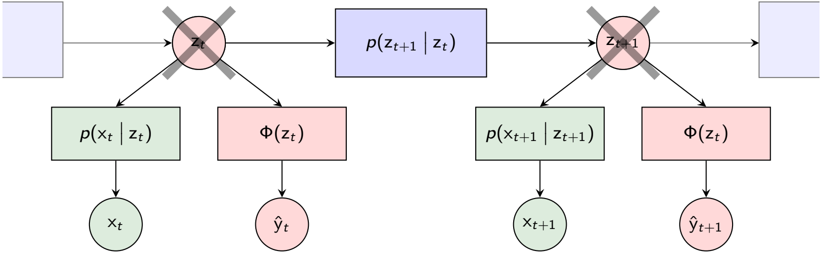
- The AI can try to mimic the conditional distribution $p(y_t \mid z_t)$. But we don't have access to z_t



- Recurrent Neural Network (RNN) $\Rightarrow$ Use observed state x_t to estimate hidden state z_t

- In a hidden Markov model learning is not equivalent to a sequence of learning problems

- The AI can try to mimic the conditional distribution $p(y_t \mid z_t)$. But we don't have access to z_t

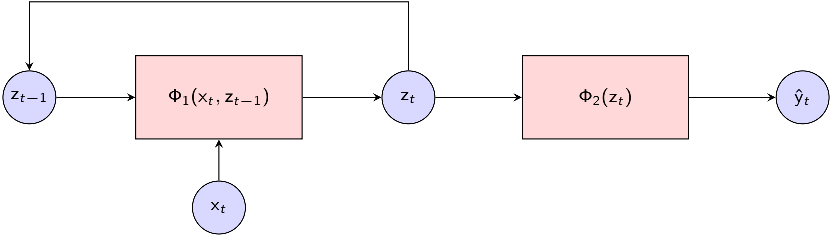


- Recurrent Neural Network (RNN) $\Rightarrow$ Use observed state x_t to estimate hidden state z_t

- A recurrent neural network is made up of two separate learning parametrizations

$\Rightarrow$ $\Phi_1(x_t, z_{t-1})$ $\Rightarrow$ From observed state x_t $\Rightarrow$ and hidden state z_{t-1} $\Rightarrow$ to hidden state update z_t

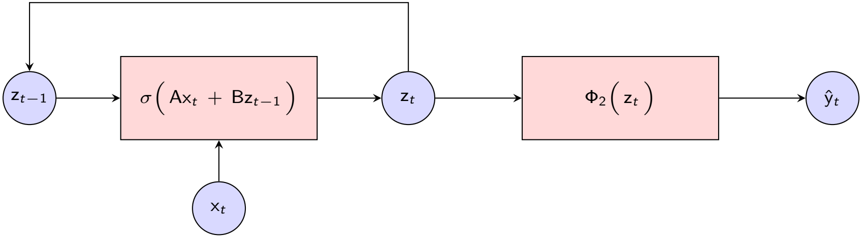
$\Rightarrow$ $\Phi_2(z_t)$ $\Rightarrow$ From updated hidden state z_t $\Rightarrow$ to output estimate $\hat{y}_t$



- It is a recurrent neural network because hidden states are fed-back as inputs for the next time step

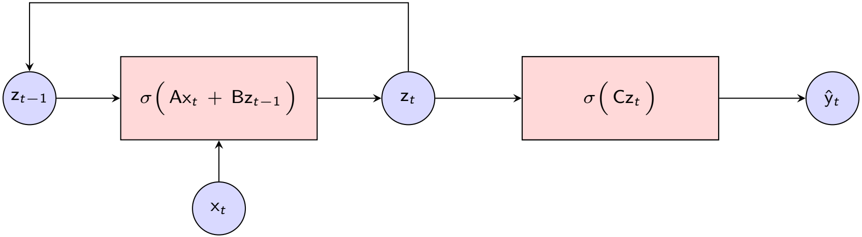
Hidden State Update AI

- Use a perceptron for the AI that updates the hidden state $\Rightarrow \Phi_1\Big(\mathsf{x}_t,\, \mathsf{z}_{t-1}\Big) = \sigma\Big(\mathsf{A}\mathsf{x}_t + \mathsf{B}\mathsf{z}_{t-1}\Big)$



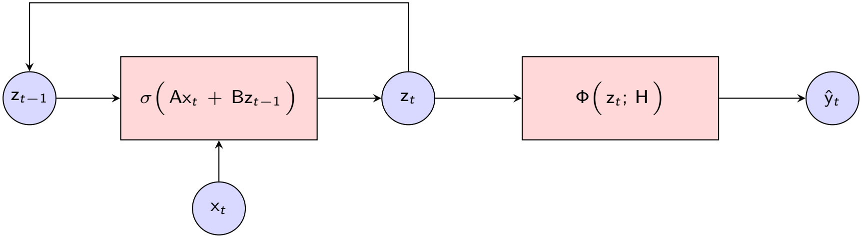
- Number of learnable parameters $\equiv$ Entries of A and B $\Rightarrow$ Does not depend on the time index t

- Use another perceptron for the AI that predicts the output $\Rightarrow \Phi_1\Big(z_t\Big) = \sigma\Big(Cz_t\Big)$



- We can also use a multi-layer neural network for the output prediction AI $\Rightarrow \Phi_1\Big(z_t\Big) = \Phi\Big(z_t;\ H\Big)$

- Use another perceptron for the AI that predicts the output $\Rightarrow \Phi_1\Big(z_t\Big) = \sigma\Big(Cz_t\Big)$



- We can also use a multi-layer neural network for the output prediction AI $\Rightarrow \Phi_1\Big(z_t\Big) = \Phi\Big(z_t;\ H\Big)$

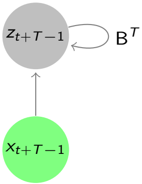
Time Gating

- We discuss the problem of vanishing/exploding gradients in recurrent neural networks

- We introduce gating mechanisms in the form of long short-term memories and gated recurrent units

Vanishing/Exploding Gradients for Long Term Dependencies

- In some tasks, the RNN may have to learn how to model long term dependencies of length T
- This poses a challenge $\Rightarrow$ the Jacobian $\partial z_T / \partial B$ will depend on a chain of multiplications by B



- If eigenvalues of $B \ll 1$, the gradients tend to vanish, leading to exponentially smaller weights B
- If eigenvalues of $B \gg 1$, the gradients tend to explode, leading to exponentially larger weights B

- Consider a simplification of the RNN where we omit the nonlinear function $\sigma(\cdot)$ and the inputs x_t

$$\mathrm{z}_t = \mathrm{B}\mathrm{z}_{t-1}$$

- At time $t = T$, the state variable z_T depends on the Tth power of the matrix B

$$\mathrm{z}_T = \mathrm{B}^T \mathrm{z}_{t-T}$$

- If B admits an eigendecomposition $\mathrm{B} = \mathrm{Q}\Lambda\mathrm{Q}^\top$, the recurrence can be rewritten as

$$\mathrm{z}_T = \mathrm{Q}\Lambda^T\mathrm{Q}^\top \mathrm{z}_{t-T}$$

$\Rightarrow$ Eigenvalues less than one will vanish and eigenvalues greater than one will explode

$\Rightarrow$ Any component of z_{t-T} not aligned with the largest eigenvalues will be discarded

Gating Mechanism

- To address the issue of vanishing gradients, we add a gating mechanism to RNNs
- Gates are scalars in $[0, 1]$ acting on the current input and on the previous state

  $\Rightarrow$ 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

  $\Rightarrow$ Allows creating paths through time with derivatives that neither vanish nor explode

  $\Rightarrow$ Creates dependency paths that allow encoding both short and long term dependencies

- The most popular gated RNN architecture is the Long Short-Term Memory (LSTM) cell
- Three gates: a forget gate $f_t \in [0,1]$, an input gate $g_t \in [0,1]$, and a cell output gate $q_t \in [0,1]$
- Let x_t be the input, z_t the state, and define the internal memory s_t of the LSTM cell
- Memory s_t updated by applying the forget gate to s_{t-1} and the input gate to the state update

$$\mathsf{s}_t = f_t \mathsf{s}_{t-1} + g_t \sigma\left(\mathsf{A}\mathsf{x}_t + \mathsf{B}\mathsf{z}_{t-1}\right)$$

- State z_t updated by applying the cell output gate to the internal cell memory s_t

$$\mathsf{z}_t = q_t \sigma(\mathsf{s}_t)$$

- The Gated Recurrent Unit (GRU) is a second popular gated version of the RNN
- Slight variation of LSTM $\Rightarrow$ single gate $u_t \in [0, 1]$ plays the role of input and forget gates

$$\mathsf{z}_t = u_t \mathsf{z}_{t-1} + (1 - u_t)\sigma\left(\mathsf{A}\mathsf{x}_t + r_t \mathsf{B}\mathsf{z}_{t-1}\right)$$

$\Rightarrow$ Reset gate $r_t \in [0, 1]$ controls contribution of previous state z_{t-1} to updated state

- Besides the LSTM and the GRU, many more variants of gating mechanisms for RNNs exist

- In the LSTM and the GRU, the gates themselves are calculated as the outputs of RNNs
- For example, the forget gate f_t of the LSTM has its own state variable z'_t (as do all the other gates)

$$\mathrm{z}'_t = \sigma\left(\mathrm{A}'\mathrm{x}_t + \mathrm{B}'\mathrm{z}'_{t-1}\right)$$

- The forget gate f_t is then calculated from the input x_t and the state z'_t as

$$f_t = \text{sigmoid}\left(\mathrm{U}\mathrm{x}_t + \mathrm{W}\mathrm{z}'_t\right)$$

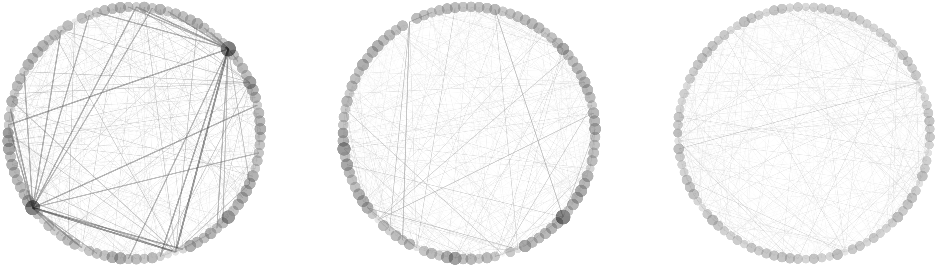
$\Rightarrow$ With U and W linear layers mapping the input and state features to a single scalar

$\Rightarrow$ And the sigmoid activation function ensuring gate values in the $[0, 1]$ interval

Graph Recurrent Neural Networks

- 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 S



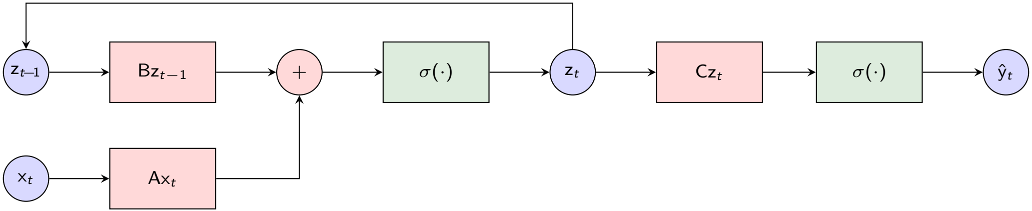
x_{t-2} x_{t-1} x_t

- A graph recurrent neural network (GRNN) combines

$\Rightarrow$ A GNN because x_t is supported on a graph. $\Rightarrow$ An RNN because x_t is a sequence

- An RNN has a hidden state z_t updated with the perceptron $\Rightarrow z_t = \sigma(Ax_t + Bz_{t-1})$

- An it has an output prediction $\hat{y}_t$ given by the perceptron $\Rightarrow \hat{y}_t = \sigma(Cx_t)$



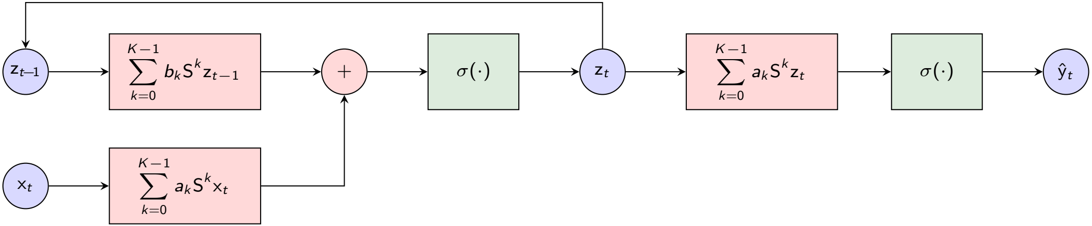
- The observed state x_t and the output y_t are graph signals supported on the graph shift operator S

$\Rightarrow$ The hidden state z_t is constructed to be a graph signal supported on the graph shift operator S

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

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

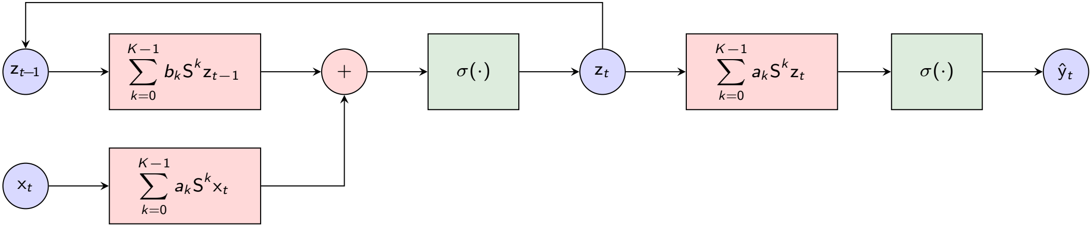
- The state update is $\Rightarrow \mathbf{z}_t = \sigma\Big[\mathbf{A}(\mathbf{S})\mathbf{x}_t + \mathbf{B}(\mathbf{S})\mathbf{z}_{t-1}\Big] = \sigma\Big[\sum_{k=0}^{K-1} a_k \mathbf{S}^k \mathbf{x}_t + \sum_{k=0}^{K-1} b_k \mathbf{S}^k \mathbf{z}_{t-1}\Big]$



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



Multiple Feature GRNNs

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

$$\mathbf{z}_t = \sigma\left[\sum_{k=0}^{K-1} a_k \mathbf{S}^k \mathbf{x}_t + \sum_{k=0}^{K-1} b_k \mathbf{S}^k \mathbf{z}_{t-1}\right] \qquad \hat{\mathbf{y}}_t = \sigma\left[\sum_{k=0}^{K-1} c_k \mathbf{S}^k \mathbf{z}_t\right]$$

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

$$\mathbf{Z}_t = \sigma\left[\sum_{k=0}^{K-1} \mathbf{S}^k \mathbf{X}_t \mathbf{A}_k + \sum_{k=0}^{K-1} \mathbf{S}^k \mathbf{Z}_{t-1} \mathbf{B}_k\right] \qquad \hat{\mathbf{Y}}_t = \sigma\left[\sum_{k=0}^{K-1} \mathbf{S}^k \mathbf{Z}_t \mathbf{C}_k\right]$$

- Multiple-feature hidden state $\mathbf{Z}_t$ permits larger dimensionality relative to observed states

Spatial Gating

- We extend time gating to GRNNs to handle the problem of vanishing/exploding gradients
- We discuss long range graph dependencies and introduce node and edge gating

- Like RNNs, GRNNs may also experience the problem of vanishing/exploding gradients

  $\Rightarrow$ Happens when eigenvalues of $\mathsf{B}(\mathsf{S})$ are much smaller/larger than 1

- Similarly to what we did for RNNs, we address it by adding gating operators to GRNNs

$$\mathsf{Z}_t = \sigma\bigg(\hat{\mathcal{Q}}\left\{\mathcal{A}_\mathsf{S}(\mathsf{X}_t)\right\} + \check{\mathcal{Q}}\left\{\mathcal{B}_\mathsf{S}(\mathsf{Z}_{t-1})\right\} \bigg)$$

- Input gate operator $\hat{\mathcal{Q}} : \mathbb{R}^{N\times H} \to \mathbb{R}^{N\times H}$ $\Rightarrow$ controls the importance of the input X_t at time t

- Forget gate operator $\check{\mathcal{Q}} : \mathbb{R}^{N\times H} \to \mathbb{R}^{N\times H}$ $\Rightarrow$ controls the importance of the state Z_t at time t

- First type of gating for GRNNs is time gating $\Rightarrow$ simple extension of input and forget gates of RNNs
- In the Time-Gated GRNN, the input and forget gate operators are expressed as

$$\hat{\mathcal{Q}}\left\{\mathcal{A}_{\mathsf{S}}(\mathsf{X}_t)\right\} = \hat{q}_t \mathcal{A}_{\mathsf{S}}(\mathsf{X}_t), \qquad \check{\mathcal{Q}}\left\{\mathcal{B}_{\mathsf{S}}(\mathsf{Z}_t)\right\} = \check{q}_t \mathcal{B}_{\mathsf{S}}(\mathsf{Z}_t)$$

- Time gating multiplies the input and the state by scalar gates $\hat{q}_t \in [0,1]$ and $\check{q}_t \in [0,1]$
- A single scalar gate is applied to the whole graph signal $\Rightarrow$ same gate value for all nodes

- Even if eigenvalues of $\mathsf{B}(\mathsf{S}) \sim 1$ spatial imbalances can cause gradients to vanish in space

$\Rightarrow$ Some nodes/paths might get assigned more importance than others in long range exchanges

- **Example**: graphs with community structure, where some nodes are highly connected within clusters

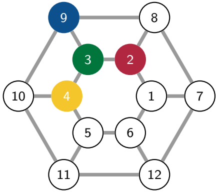
$\Rightarrow$ Gradients of Z_T depend on successive products of $\mathsf{B}(\mathsf{S})$ $\Rightarrow$ successive products of S

$\Rightarrow$ For large T, the matrix entries in S^T with highly connected nodes will get densely populated

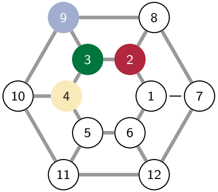
$\Rightarrow$ Overshadows community structure $\Rightarrow$ can't encode long processes that are local on the graph

Spatial Gating

- Node and edge structure of the graph allows for other forms of gating $\Rightarrow$ spatial gating
- Node gating $\Rightarrow$ one input and one forget gate for each node of the graph



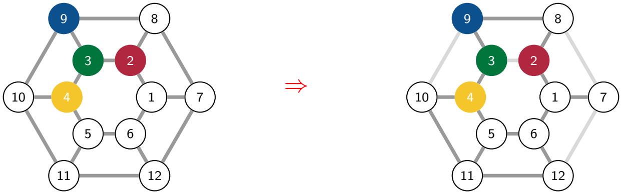
$\Rightarrow$



- Spatial gating strategies help encode long range spatial dependencies in graph processes

- Node and edge structure of the graph allows for other forms of gating $\Rightarrow$ spatial gating

- Edge gating $\Rightarrow$ one input and one forget gate for each edge of the graph



- Spatial gating strategies help encode long range spatial dependencies in graph processes

- In the Node-Gated GRNN, the input gate and forget gate operators are expressed as

$$\hat{\mathcal{Q}}\{\mathcal{A}_{\mathsf{S}}(\mathsf{X}_t)\} = \mathrm{diag}(\hat{\mathsf{q}}_t)\mathcal{A}_{\mathsf{S}}(\mathsf{X}_t), \qquad \check{\mathcal{Q}}\{\mathcal{B}_{\mathsf{S}}(\mathsf{Z}_t)\} = \mathrm{diag}(\check{\mathsf{q}}_t)\mathcal{B}_{\mathsf{S}}(\mathsf{Z}_t)$$

- Gating operators correspond to multiplication of the input and state by diagonal matrices

$\Rightarrow$ The diagonals are the input and forget vector gates $\hat{q}_t \in [0,1]^N$ and $\check{q}_t \in [0,1]^N$

- A scalar gate applied to each nodal component of the signal $\Rightarrow$ different gate values for each node

- In the Edge-Gated GRNN, the input gate and forget gate operators are expressed as

$$\hat{\mathcal{Q}}\left\{\mathcal{A}_{\mathsf{S}}(\mathsf{X}_t)\right\} = \mathcal{A}_{\mathsf{S}\odot\hat{\mathsf{Q}}_t}(\mathsf{X}_t), \qquad \check{\mathcal{Q}}\left\{\mathcal{B}_{\mathsf{S}}(\mathsf{Z}_t)\right\} = \mathcal{B}_{\mathsf{S}\odot\check{\mathsf{Q}}_t}(\mathsf{Z}_t)$$

- Gating operators correspond to elementwise multiplication of the shift operator by gate matrices

  $\Rightarrow$ The matrices multiplying the GSOs are the input and forget matrix gates $\hat{\mathsf{Q}}_t$ and $\check{\mathsf{Q}}_t \in [0,1]^{N\times N}$

- Separate gate for each edge $\Rightarrow$ control the amount of information transmitted across edges

Gate Computation

- Parameters of input and forget gate operators are the outputs of GRNNs themselves
- Input and forget gate states are expressed as

$$\hat{\mathsf{Z}}_t = \hat{\sigma}\Big(\hat{\mathcal{A}}_{\mathsf{S}}(\mathsf{X}_t) + \hat{\mathcal{B}}_{\mathsf{S}}(\hat{\mathsf{Z}}_{t-1})\Big) \qquad \check{\mathsf{Z}}_t = \check{\sigma}\Big(\check{\mathcal{A}}_{\mathsf{S}}(\mathsf{X}_t) + \check{\mathcal{B}}_{\mathsf{S}}(\check{\mathsf{Z}}_{t-1})\Big)$$

$\Rightarrow$ Gate computation takes different forms depending on the type of gating

- In the case of time gating, the gates are calculated as

$$\hat{q}_t = \mathrm{sigmoid}(\hat{\mathsf{c}}^{\mathsf{T}}\mathrm{vec}(\hat{\mathsf{Z}}_t)) \qquad \check{q}_t = \mathrm{sigmoid}(\check{\mathsf{c}}^{\mathsf{T}}\mathrm{vec}(\check{\mathsf{Z}}_t))$$

$\Rightarrow$ Where $\hat{\mathsf{c}} \in \mathbb{R}^{\hat{H}N}$ and $\check{\mathsf{c}} \in \mathbb{R}^{\check{H}N}$ are fully connected layers and the sigmoid ensures gates in $[0, 1]$

Gate Computation

- Parameters of input and forget gate operators are the outputs of GRNNs themselves

- Input and forget gate states are expressed as

$$\hat{\mathsf{Z}}_t = \hat{\sigma}\left(\hat{\mathcal{A}}_{\mathsf{S}}(\mathsf{X}_t) + \hat{\mathcal{B}}_{\mathsf{S}}(\hat{\mathsf{Z}}_{t-1})\right) \qquad \check{\mathsf{Z}}_t = \check{\sigma}\left(\check{\mathcal{A}}_{\mathsf{S}}(\mathsf{X}_t) + \check{\mathcal{B}}_{\mathsf{S}}(\check{\mathsf{Z}}_{t-1})\right)$$

$\Rightarrow$ Gate computation takes different forms depending on the type of gating

- In the case of node gating, the gates are calculated as

$$\hat{\mathsf{q}}_t = \text{sigmoid}\left(\hat{\mathcal{C}}_{\mathsf{S}}(\hat{\mathsf{Z}}_t)\right) \qquad \check{\mathsf{q}}_t = \text{sigmoid}\left(\check{\mathcal{C}}_{\mathsf{S}}(\check{\mathsf{Z}}_t)\right)$$

$\Rightarrow$ Where $\hat{\mathcal{C}}_{\mathsf{S}}$ and $\check{\mathcal{C}}_{\mathsf{S}}$ are graph convolutions and the sigmoid ensures gates in $[0,1]^N$

- Parameters of input and forget gate operators are the outputs of GRNNs themselves
- Input and forget gate states are expressed as

$$\hat{\mathsf{Z}}_t = \hat{\sigma}\left(\hat{\mathcal{A}}_{\mathsf{S}}(\mathsf{X}_t) + \hat{\mathcal{B}}_{\mathsf{S}}(\hat{\mathsf{Z}}_{t-1})\right) \qquad \check{\mathsf{Z}}_t = \check{\sigma}\left(\check{\mathcal{A}}_{\mathsf{S}}(\mathsf{X}_t) + \check{\mathcal{B}}_{\mathsf{S}}(\check{\mathsf{Z}}_{t-1})\right)$$

$\Rightarrow$ Gate computation takes different forms depending on the type of gating

- In the case of edge gating, the gates are calculated as

$$[\hat{\mathsf{Q}}_t]_{ij} = \text{sigmoid}\left(\hat{\mathsf{c}}^{\mathsf{T}}[\boldsymbol{\delta}_i^{\mathsf{T}}\hat{\mathsf{Z}}_t\hat{\mathsf{C}}||\boldsymbol{\delta}_j^{\mathsf{T}}\hat{\mathsf{Z}}_t\hat{\mathsf{C}}]^{\mathsf{T}}\right) \qquad [\check{\mathsf{Q}}_t]_{ij} = \text{sigmoid}\left(\check{\mathsf{c}}^{\mathsf{T}}[\boldsymbol{\delta}_i^{\mathsf{T}}\check{\mathsf{Z}}_t\check{\mathsf{C}}||\boldsymbol{\delta}_j^{\mathsf{T}}\check{\mathsf{Z}}_t\check{\mathsf{C}}]^{\mathsf{T}}\right)$$

$\Rightarrow$ Where $\boldsymbol{\delta}_i$ and $\boldsymbol{\delta}_j$ are N-dimensional Dirac deltas; $\hat{\mathsf{C}} \in \mathbb{R}^{\hat{H}\times\hat{H}'}$, $\check{\mathsf{C}} \in \mathbb{R}^{\check{H}\times\check{H}'}$ are linear layers

$\Rightarrow$ And $\hat{\mathsf{c}} \in \mathbb{R}^{2\hat{H}'\times 1}$ and $\check{\mathsf{c}} \in \mathbb{R}^{2\check{H}'\times 1}$ are f.c. layers applied to concatenation $||$ of features of i and j

Stability of GRNNs

- GRNNs can be seen as a time extension of GNNs, therefore they inherit their stability properties

Definition (Relative perturbation matrices)

Given GSOs $\mathbf{S}$ and $\tilde{\mathbf{S}}$, we define the set of relative perturbation matrices modulo permutation as

$$\mathcal{E}(\mathbf{S},\tilde{\mathbf{S}}) = \left\{\mathbf{E} \in \mathbb{R}^{N\times N} : \mathbf{P}^{\mathsf{T}}\tilde{\mathbf{S}}\mathbf{P} = \mathbf{S} + \mathbf{E}\mathbf{S} + \mathbf{S}\mathbf{E}^{\mathsf{T}}, \mathbf{P} \in \mathcal{P}\right\} \tag{1}$$

where $\mathcal{P} = \left\{\mathbf{P} \in \{0,1\}^{N\times N} : \mathbf{P}\mathbf{1} = \mathbf{1}, \mathbf{P}^{\mathsf{T}}\mathbf{1} = \mathbf{1}\right\}$.

- We consider that the distance between two graphs $\mathbf{S}$ and $\tilde{\mathbf{S}}$ is given by $d(\mathbf{S},\tilde{\mathbf{S}}) = \min_{\mathbf{E}\in\mathcal{E}(\mathbf{S},\hat{\mathbf{S}})} \|\mathbf{E}\|$
- Notice that if $\tilde{\mathbf{S}}$ is a permutation of the shift matrix $\mathbf{S}$, then we have $d(\mathbf{S},\tilde{\mathbf{S}}) = 0$

Definition (Integral Lipschitz filters)

A filter $A(S) = \sum_{k=0}^{K-1} a_k S^k$ is integral Lipschitz if there exists $C > 0$ such that $a(\lambda) = \sum_{k=0}^{K-1} a_k \lambda^k$ satisfies

$$|a(\lambda_2) - a(\lambda_1)| \leq C \frac{|\lambda_2 - \lambda_1|}{|\lambda_1 + \lambda_2|/2} \tag{2}$$

for all $\lambda_1, \lambda_2 \in \mathbb{R}$.

- Integral Lipschitz filters also satisfy $|\lambda a'(\lambda)| \leq C$, where $a'(\lambda)$ is the derivative of $a(\lambda)$
- Recall that the frequency response of integral Lipschitz filters becomes flat for large λ

- We consider a GRNN with $F_X = 1$ input feature, $F_Z = 1$ state feature, and $F_Y = 1$ output feature

$$\mathsf{z}_t = \sigma\big(\mathsf{A}(\mathsf{S})\mathsf{x}_t + \mathsf{B}(\mathsf{S})\mathsf{z}_{t-1}\big) \qquad \hat{\mathsf{y}}_t = \rho\big(\mathsf{C}(\mathsf{S})\mathsf{z}_t\big)$$

(A1) A, B and C are integral Lipschitz with constants C_{A}, C_{B} and C_{C} and $\|\mathsf{A}\| = \|\mathsf{B}\| = \|\mathsf{C}\| = 1$

(A2) Nonlinearities σ and ρ satisfy: $|\sigma(b) - \sigma(a)| \leq |b - a|$ for all $a, b \in \mathbb{R}$, $\sigma(0) = \rho(0) = 0$

(A3) Initial hidden state is identically zero, i.e., $\mathsf{z}_0 = 0$, and the x_t satisfy $\|\mathsf{x}_t\| \leq \|\mathsf{x}\| = 1$ for all t

Theorem (Stability of GRNNs)

Let $\mathsf{S} = \mathsf{V}\Lambda\mathsf{V}^{\mathsf{H}}$ and $\tilde{\mathsf{S}}$ be the GSOs of the original and perturbed graph, and let $\mathsf{E} = \mathsf{U}\mathsf{M}\mathsf{U}^{\mathsf{H}} \in \mathcal{E}(\mathsf{S}, \tilde{\mathsf{S}})$ such that $d(\mathsf{S}, \tilde{\mathsf{S}}) \leq \|\mathsf{E}\| \leq \varepsilon$. Let y_t and $\tilde{\mathsf{y}}_t$ be the outputs of the GRNNs running on S and $\tilde{\mathsf{S}}$ respectively, and satisfying assumptons (A1)-(A3). Then,

$$\min_{\mathsf{P}\in\mathcal{P}} \|\mathsf{y}_t - \mathsf{P}^{\mathsf{T}}\tilde{\mathsf{y}}_t\| \leq C(1+\sqrt{N}\delta)(t^2+3t)\varepsilon \ + \mathcal{O}(\varepsilon^2) \tag{3}$$

where $C = \max\{C_{\mathsf{A}}, C_{\mathsf{B}}, C_{\mathsf{C}}\}$ and $\delta = (\|\mathsf{U}-\mathsf{V}\|+1)^2 - 1$.

Discussion

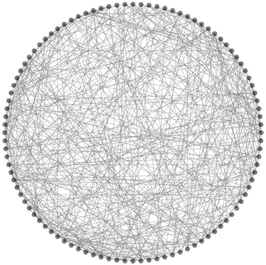
- GRNNs are stable to relative perturbations with constant $C(1+\sqrt{N}\delta)(T^2+3T)$, T process length
- C could be set at a fixed value or learned from data through A, B and C $\Rightarrow$ design parameter
- The term $(1+\delta\sqrt{N})$ is a property of the graph perturbation $\Rightarrow$ cannot be controlled by design
- Eigenvector misalignment $\delta = (\|\mathsf{U}-\mathsf{V}\|+1)^2-1$ measures commutativity of matrices S and E
- Polynomial dependence on T $\Rightarrow$ due to recurrence relationship in the computation of x_t

Epidemic Modeling with GRNNs

- We use a GRNN, a GNN, and a RNN to track an epidemic on a high school friendship network

- 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

- 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



Susceptible-Infectious-Removed (SIR) Disease Model

- Process starts with random seed infections on day 0 $\Rightarrow$ probability $p_{seed} = 0.05$
- Each person is in one of the three SIR states $\Rightarrow$ 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 $\mathbf{x}_t$ where, at each time t, $[\mathbf{x}_t]_i$ is given by

$$[\mathbf{x}_t]_i = \begin{cases} 0, \text{if student } i \text{ is susceptible} \\ 1, \text{if student } i \text{ is infected} \\ 2, \text{if student } i \text{ is removed} \end{cases}$$

- **Output**: binary graph process $\mathbf{y}_t \Rightarrow$ our goal is only to track infections

$$[\mathbf{y}_t]_i = \begin{cases} 0, \text{if student } i \text{ is susceptible or removed} \\ 1, \text{if student } i \text{ is infected} \end{cases}$$

- Given $\mathbf{x}_t, \mathbf{x}_{t+1}, \ldots, \mathbf{x}_{t+7}$, we want to predict $\mathbf{y}_{t+8}, \mathbf{y}_{t+9}, \ldots, \mathbf{y}_{t+15} \Rightarrow$ 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
  $\Rightarrow$ Proportion of correct positive predictions

- Recall $=$ True Positive/All Actual Positive
  $\Rightarrow$ Proportion of correctly predicted positives

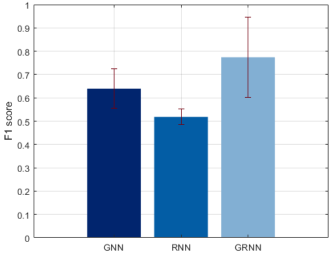
		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

- Loss function we minimize is $1 - F1$ $\Rightarrow$ trade-off between minimizing FPs and FNs

- We compare a GRNN with a GNN and a RNN, all with roughly the same number of parameters

  $\Rightarrow$ In the GNN, the time instants become input features $\Rightarrow$ parameters depend on T

  $\Rightarrow$ 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