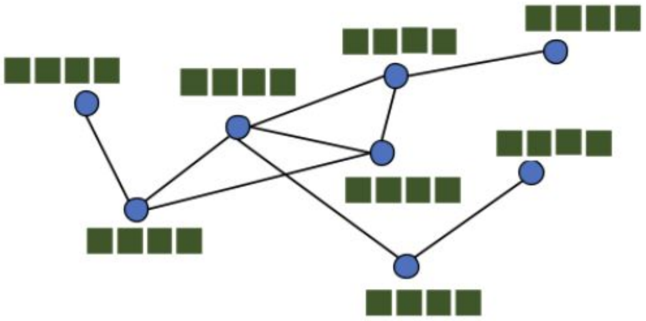
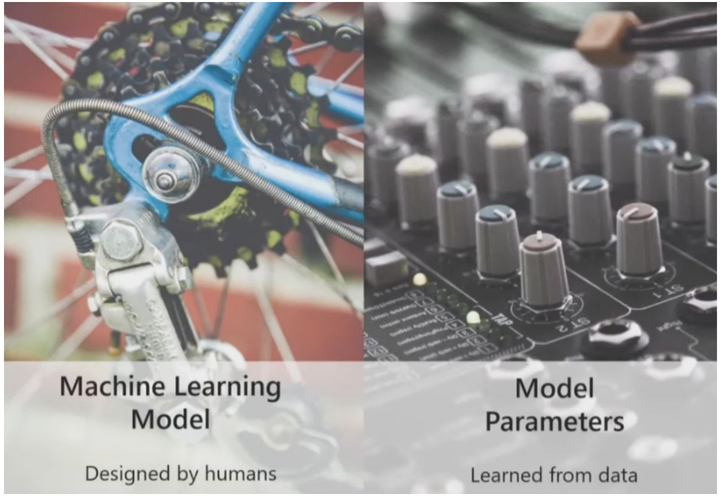


Graph Neural Networks Models

Jay Urbain, PhD - 10/12/2022

Machine Learning

- We design a model
- The model represents something in the world
- Learn parameters through data



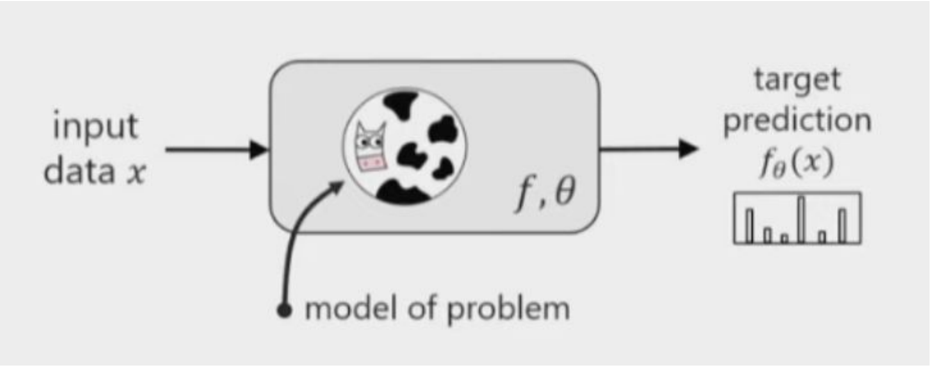
"All models are wrong, some are useful" – George Box

Supervised Machine Learning

Given an IID dataset: $\{(x_1, y_1), \ldots, (x_N, y_N)\}$

Pick θ that minimize the Loss

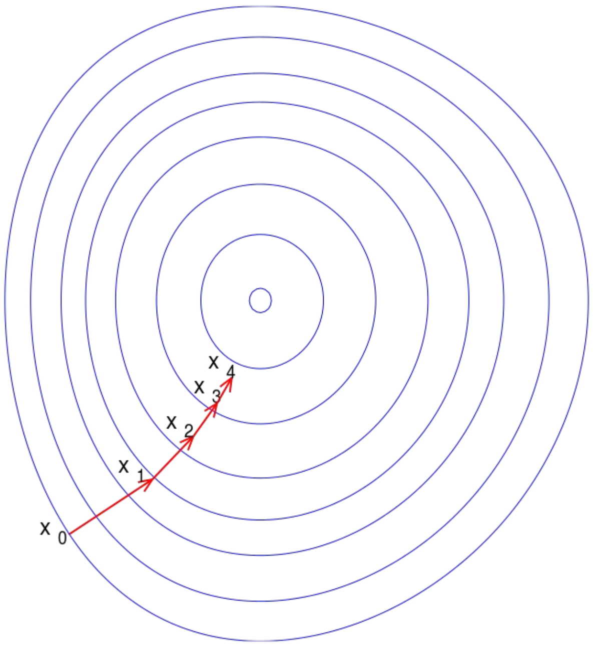
$$L(\theta) = \frac{1}{N} \sum_i L(f_\theta(x_i), y_i)$$



Gradient descent learning of model parameters

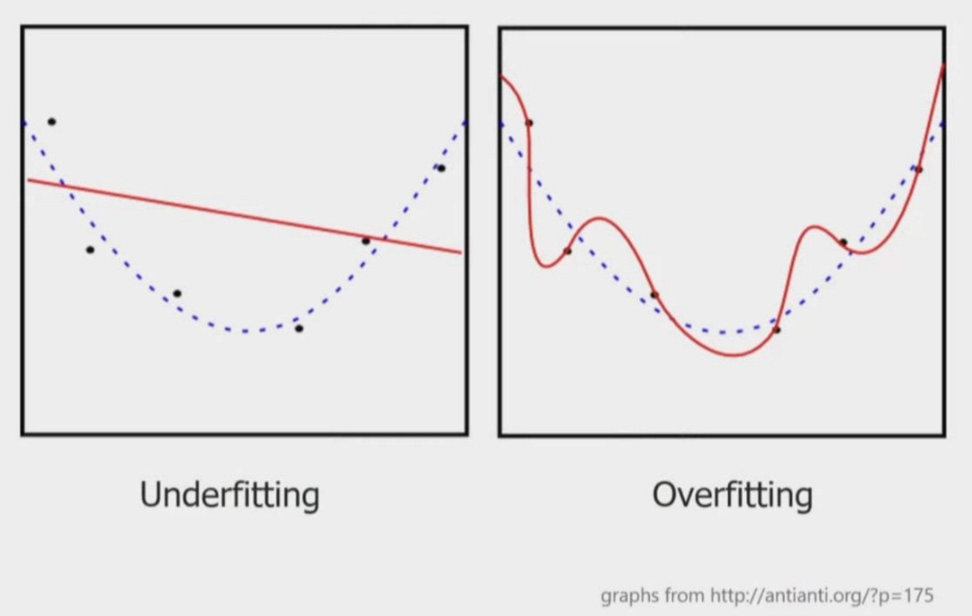
While not converged, update parameters:

$$\theta = \theta - \gamma \frac{\partial L}{\partial \theta}$$



Generalization

Goal: Generalize to unseen data.



Distributed vector representation

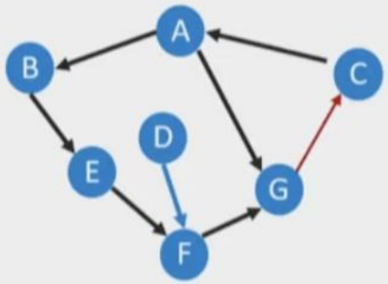
Distributed representation - Meaning is distributed among components of a vector.

$\boldsymbol{r} = E\,\mathbb{I}_w$ with E a DxV matrix

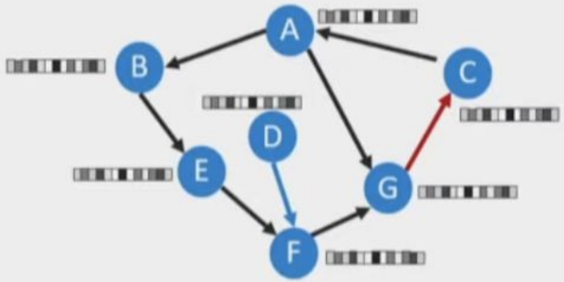
"vocabulary"

Graph neural networks

- Where does the graph come from? Modeling decision.
- You, as a human have defined a graph.
- Each node has an encoded representation.



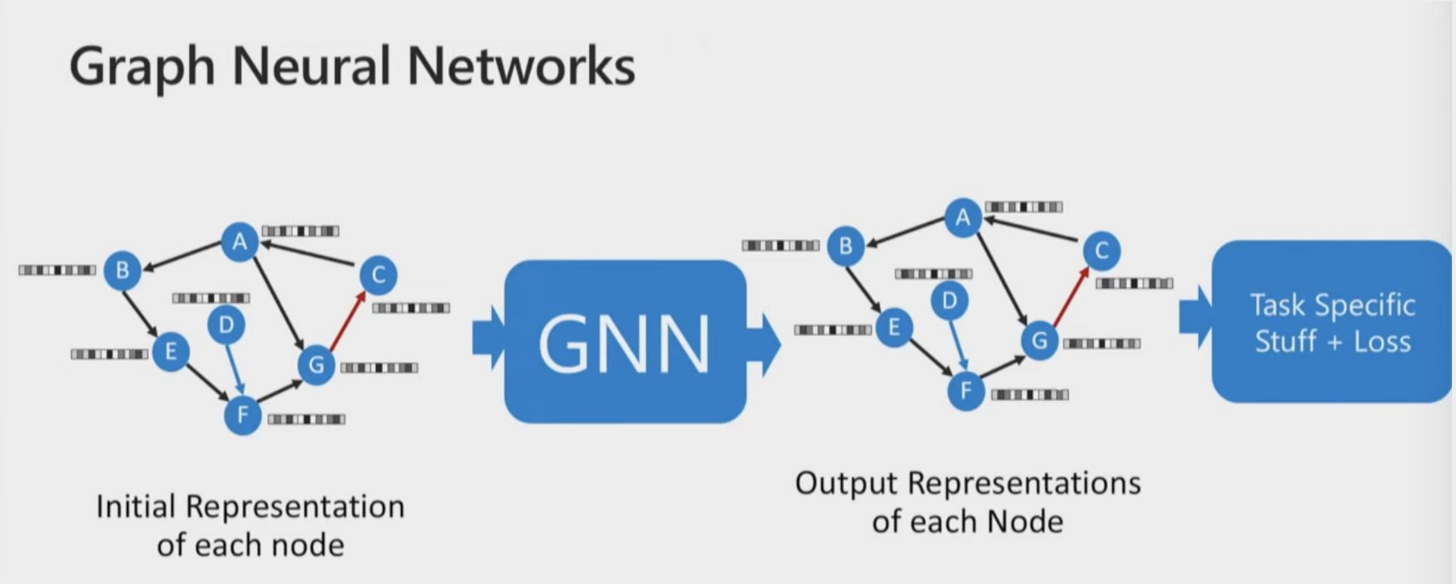
Graph Representation of Problem



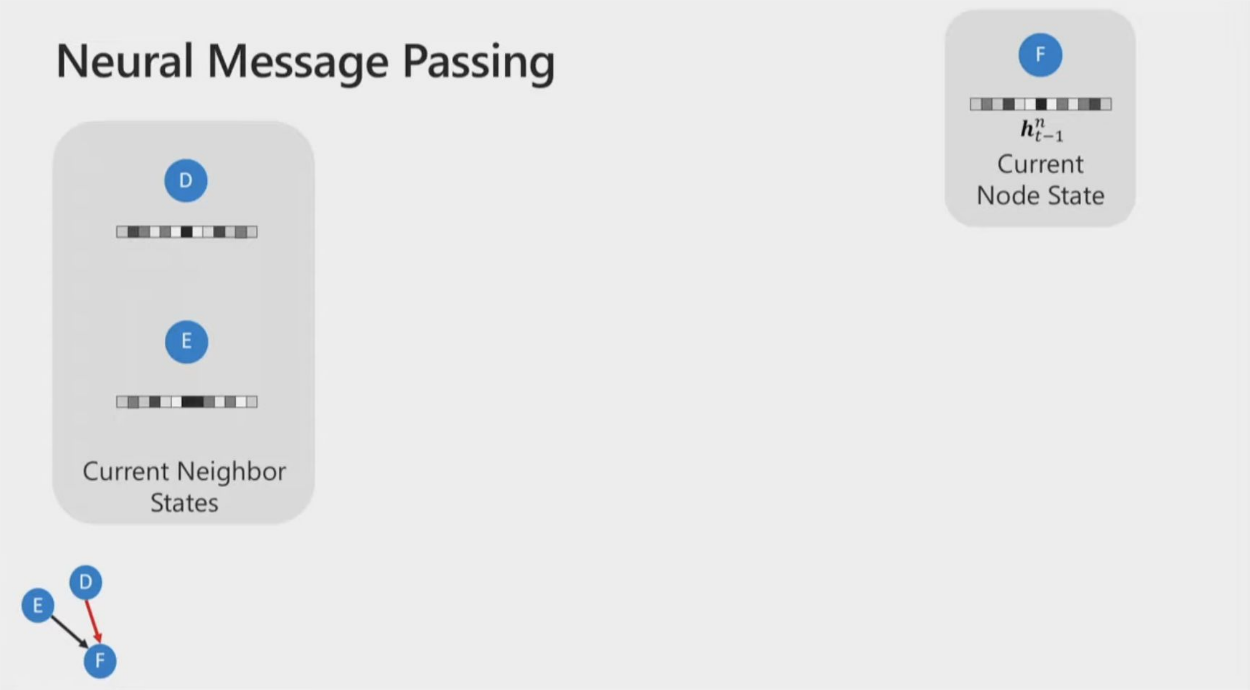
Initial Representation of each node

Graph neural networks

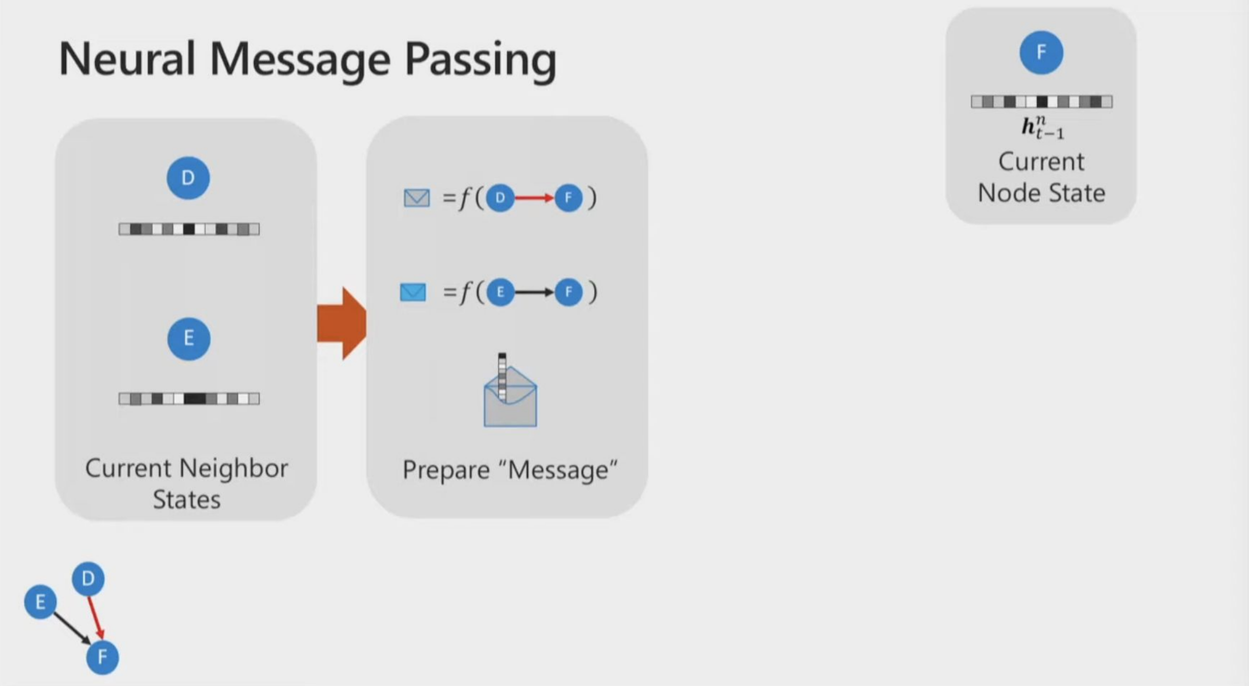
What we want to do



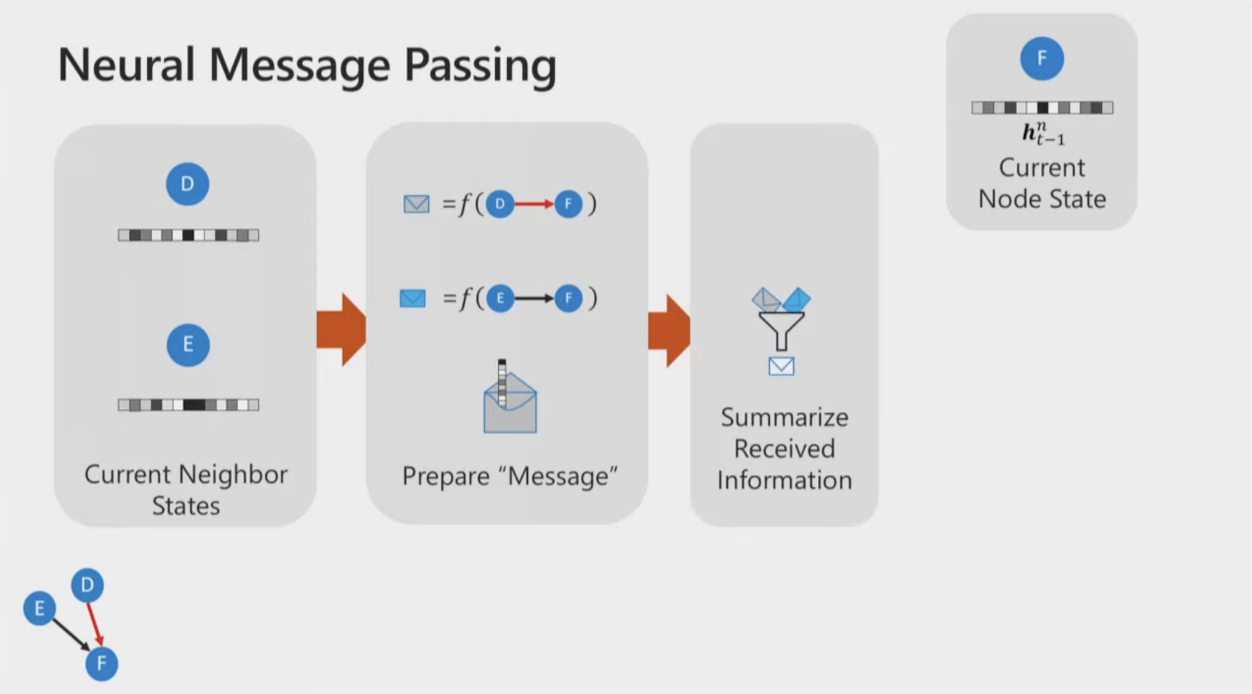
Neural message passing



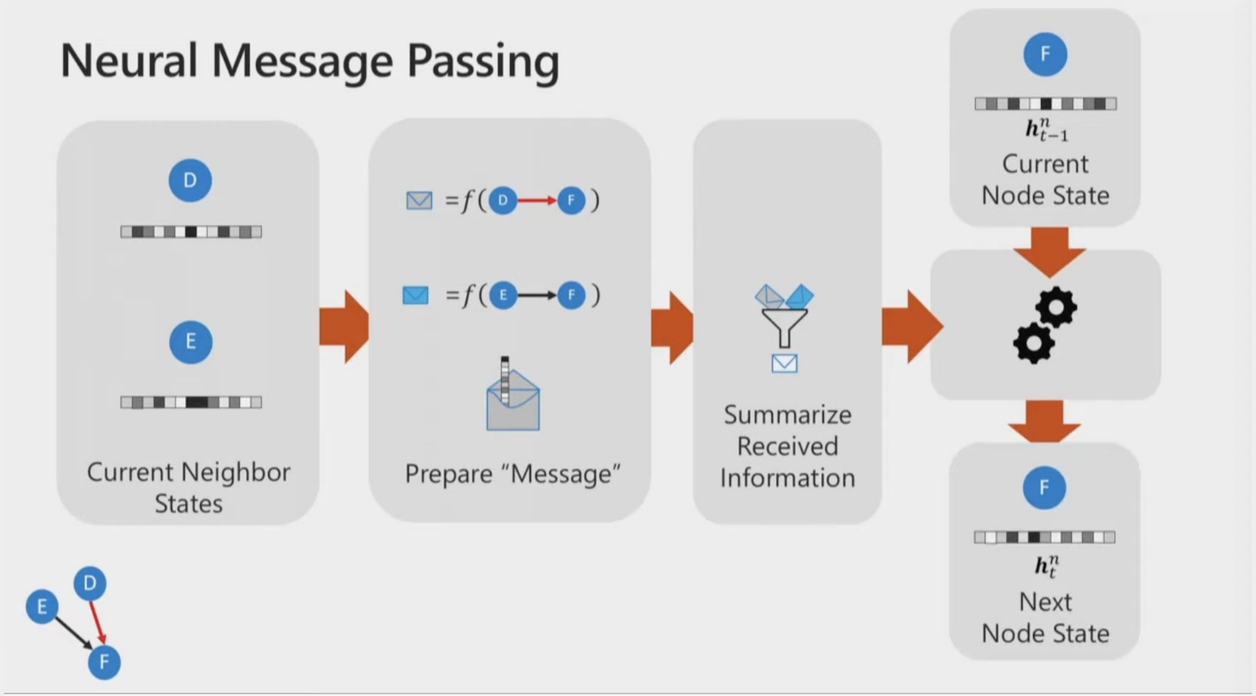
Neural message passing



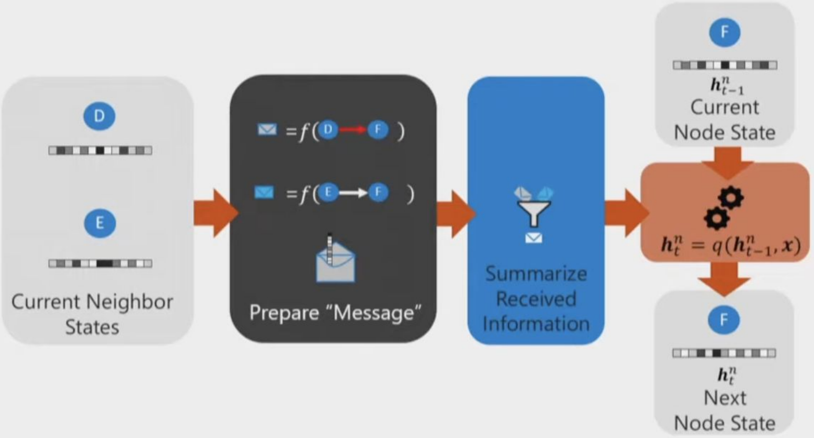
Neural message passing



Neural message passing



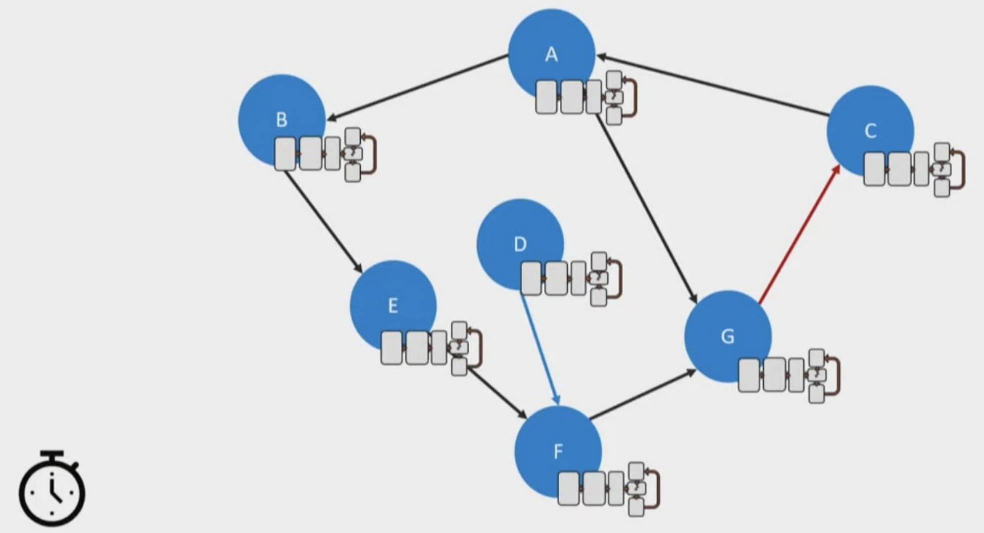
Each node now has information from its neighbors



$$h^n_t = q\left(h^n_{t-1}, \bigcup_{\forall n_j : n \xrightarrow{k} n_j} f_t\left(h^n_{t-1}, k, h^{n_j}_{t-1}\right)\right)$$

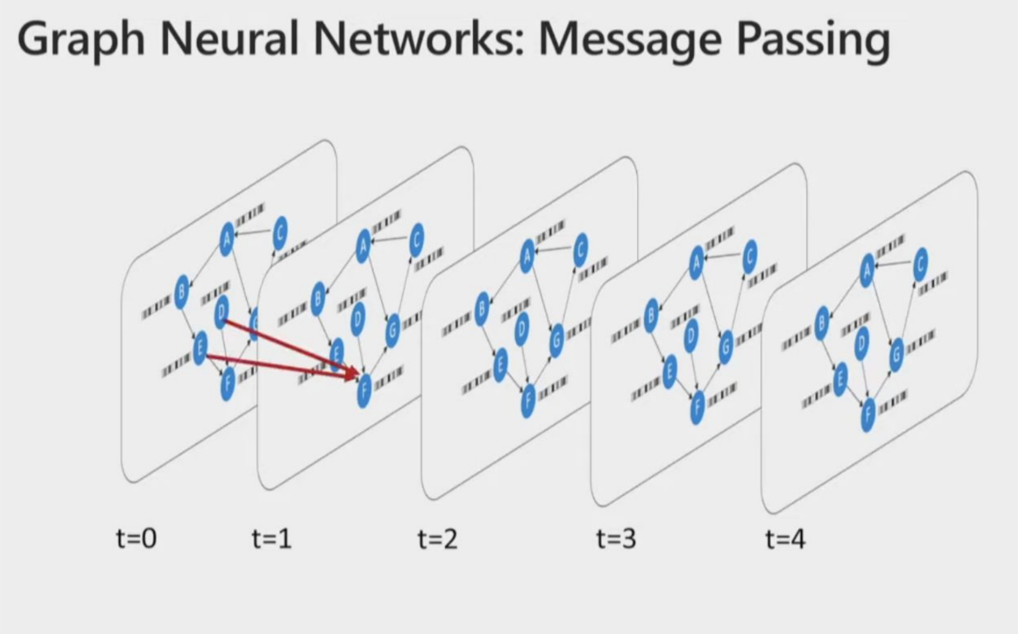
U: Could be any permutation function for combining the network.

Each node now has information from its neighbors



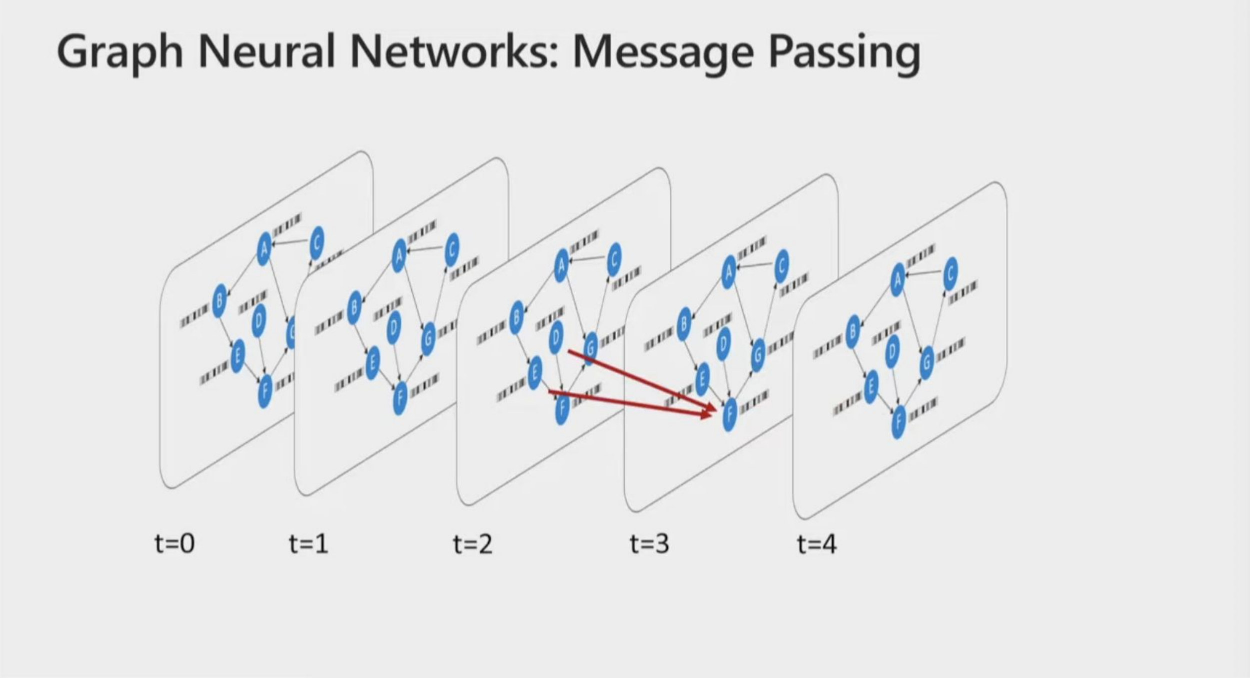
Works like a CPU clock

Graph Neural Message Passing



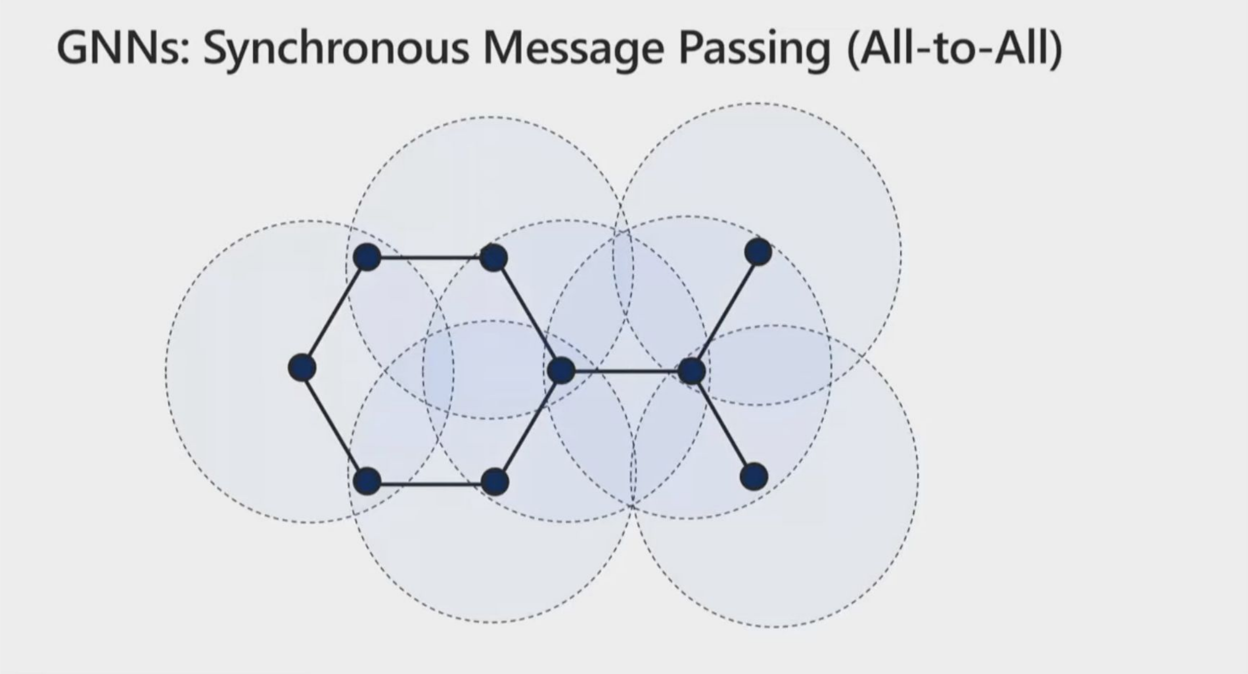
Works like a CPU clock

Graph Neural Message Passing



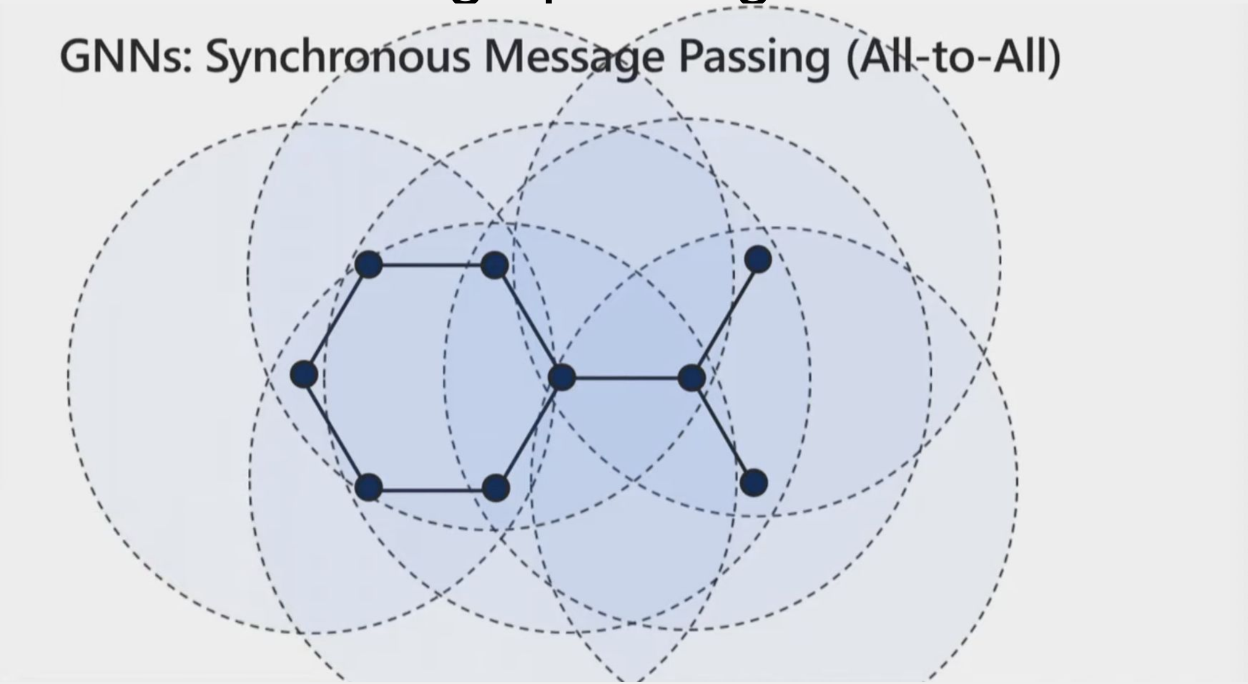
Simulation: https://distill.pub/2021/gnn-intro/

Synchronous message passing



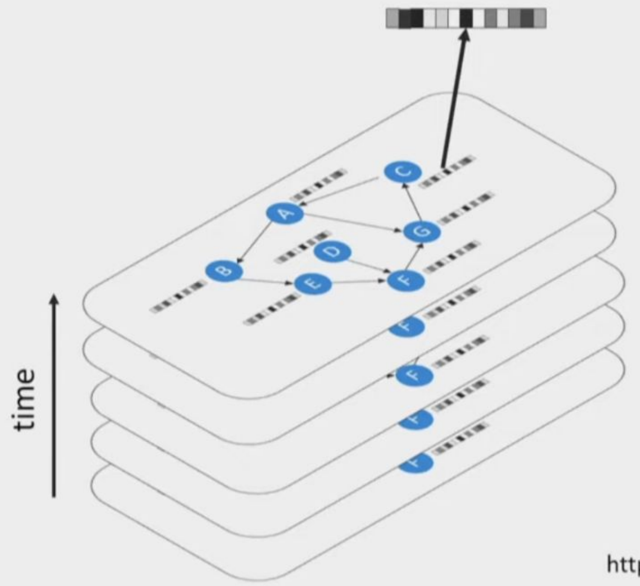
At first step node learns about itself and its neighbor

Synchronous message passing



At next step node learns about its neighbors neighbors!

Graph neural network output

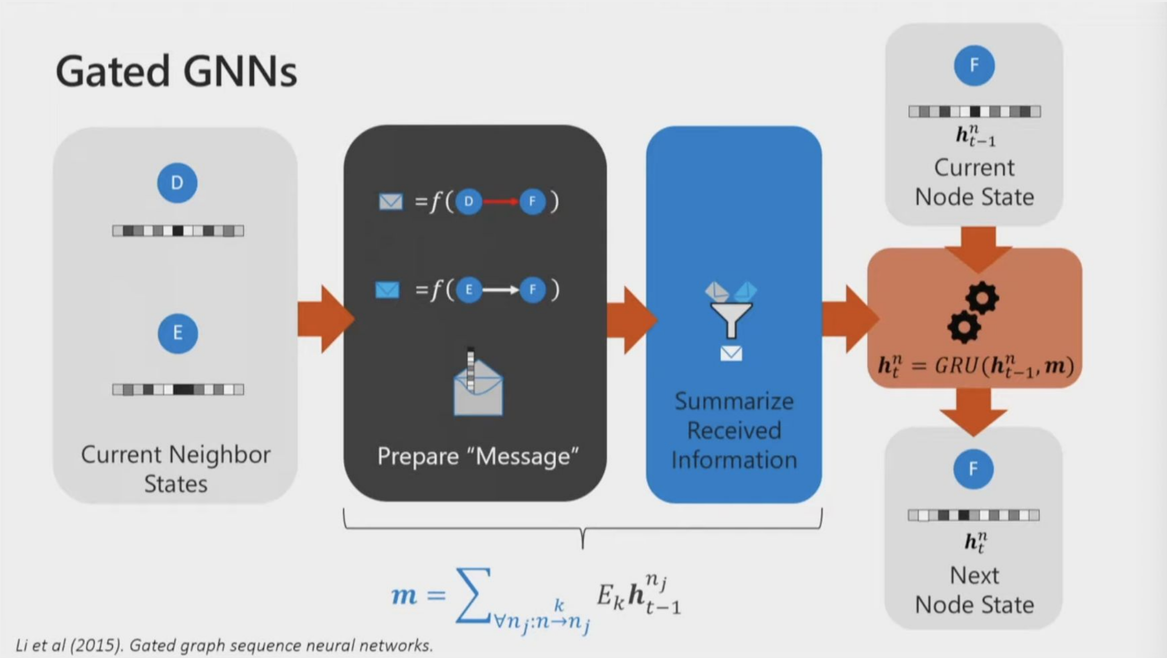


- node selection
- node classification
- graph classification

https://github.com/microsoft/tf-gnn-samples/

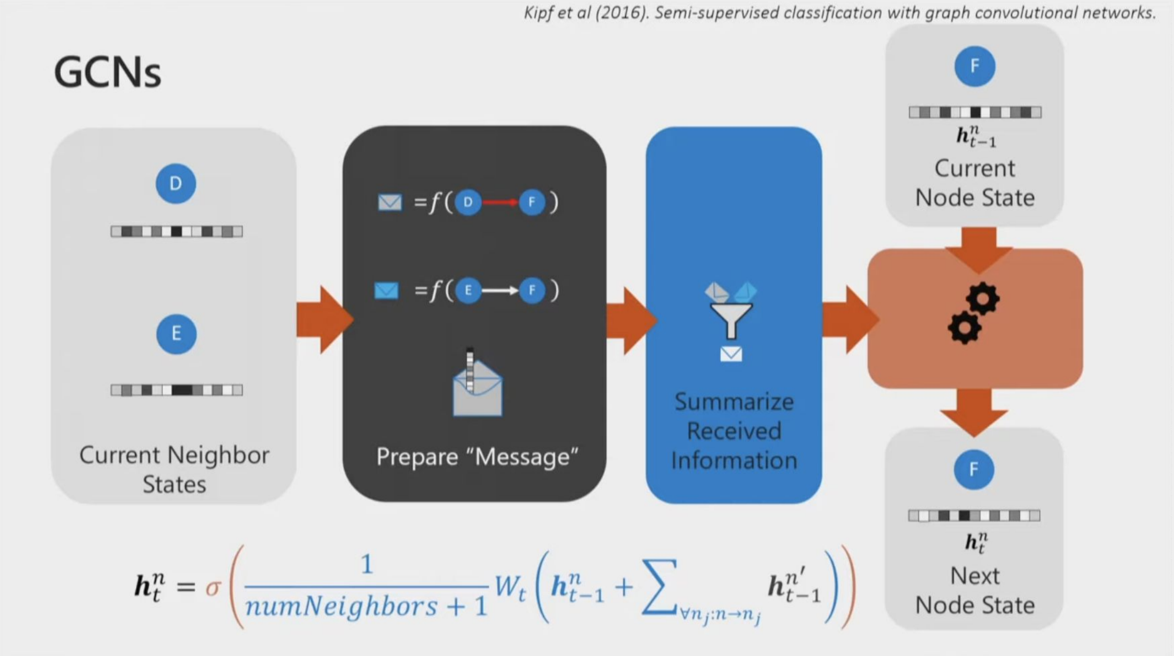
Also link prediction, subgraph similarity, etc.

Gated GNN



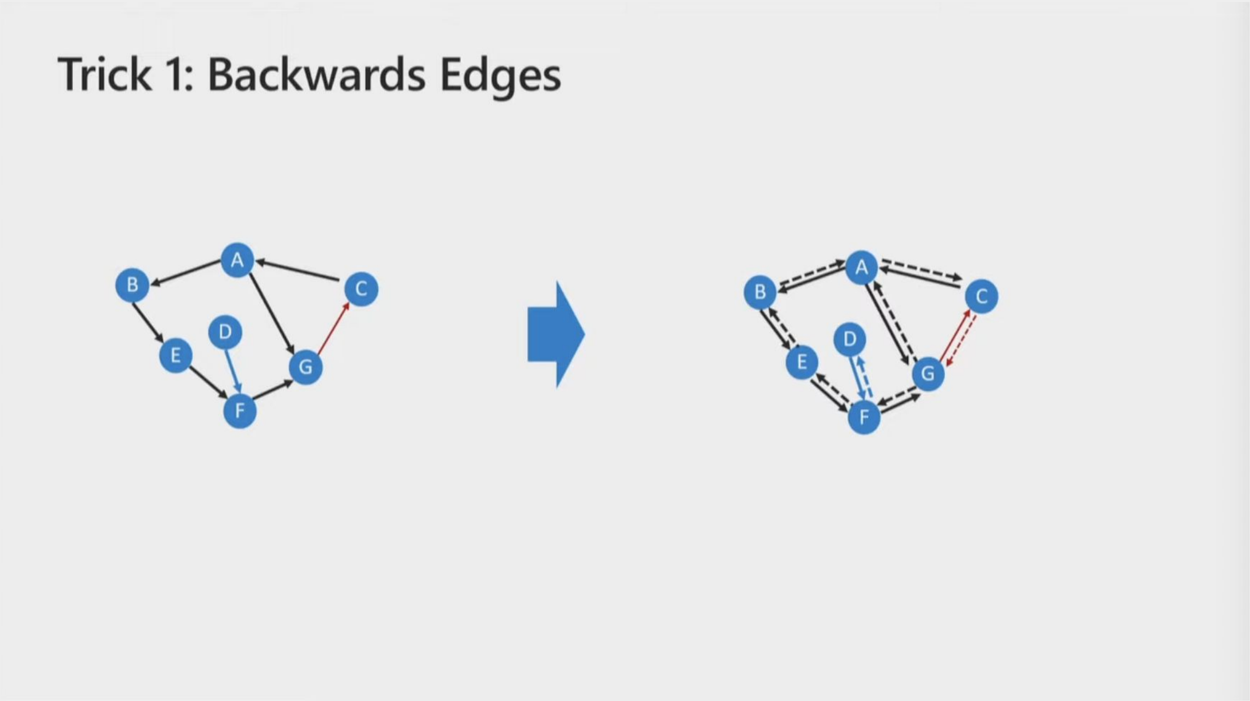
Permutation invariant combination.

Graph convolutional networks



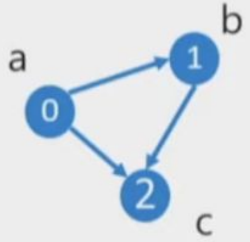
https://www.cs.toronto.edu/~yujiali/files/talks/iclr16_ggnn_talk.pdf

Trick: Backward edges



Adjacency Matrix

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \ \mathrm{N} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$



$$N \, x \, A = \begin{bmatrix} 0 \\ a \\ a + b \end{bmatrix}$$

GGNN as Matrix Operation

Node States

$$H_t = \begin{bmatrix} \boldsymbol{h}_t^{n_0} \\ \vdots \\ \boldsymbol{h}_t^{n_K} \end{bmatrix} \quad \texttt{(num\_nodes x D)}$$

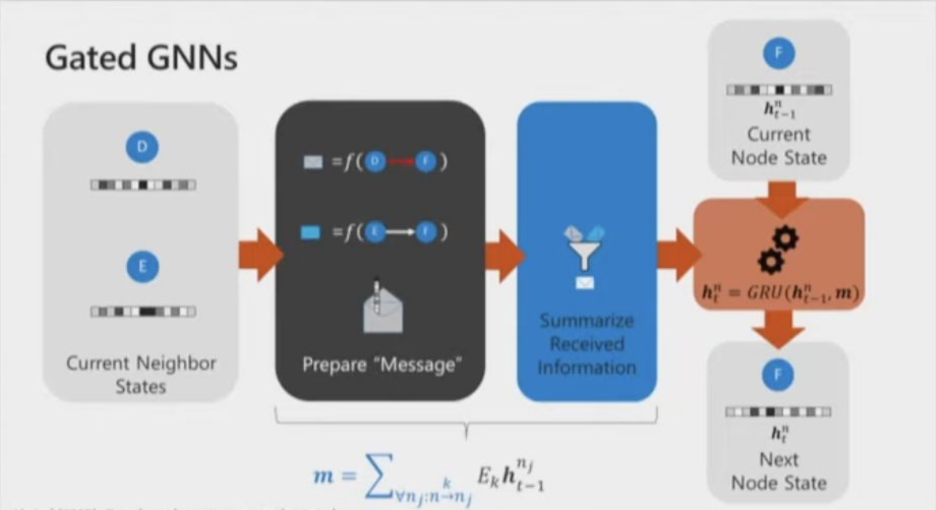
Messages to-be sent

$$M_t^k = E_k H_t \quad \texttt{(num\_nodes x M)}$$

Received Messages

$$R_t = \sum_k A M_t^k \quad \texttt{(num\_nodes x M)}$$

Update $H_{t+1} = GRU(H_t, R_t)$



Li et al (2015). Gated graph sequence neural networks.

GNN Operation

GGNN as Matrix Operation

<u>Node States</u>

$$H_t = \begin{bmatrix} \boldsymbol{h}_t^{n_0} \\ \vdots \\ \boldsymbol{h}_t^{n_K} \end{bmatrix} \quad \texttt{(num\_nodes x D)}$$

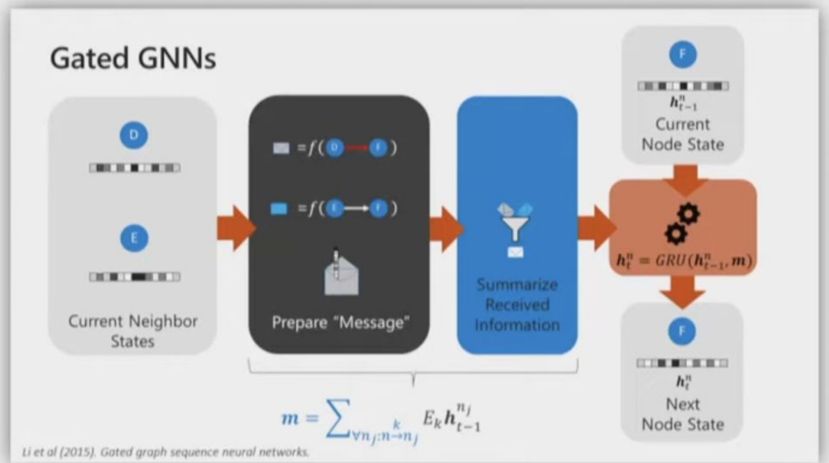
<u>Messages to-be sent</u>

$$M_t^k = E_k H_t \quad \texttt{(num\_nodes x M)}$$

<u>Received Messages</u>

$$R_t = \sum_k A M_t^k \quad \texttt{(num\_nodes x M)}$$

<u>Update</u> $H_{t+1} = GRU(H_t, R_t)$



If we used a vanilla RNN

$$H_{t+1} = \sigma(\mathrm{U}H_t + \mathrm{W}R_t)$$

Expressing Matrix Operations as Code

einsum

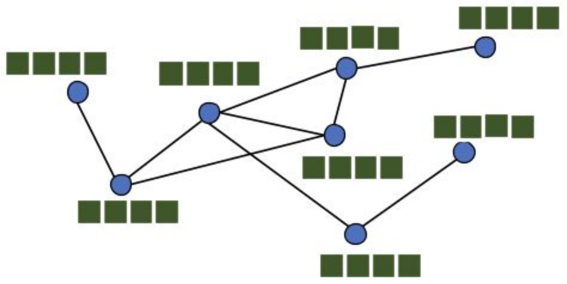
```
C=np.einsum('bd,qd->bq`, A, B)
```
$C_{b,q} = \sum_d A_{b,d} B_{q,d}$

```
D=np.einsum('abc,be,abq->cqe', A, B,C)
```
$D_{c,q,e} = \sum_b \sum_a A_{a,b,c} B_{b,e} C_{a,b,q}$

Pseudocode

```
def GGNN(initial_node_states, adj):
  node_states = initial_node_states  # [N, D]

  for i in range(num_steps):
     messages = {}
     for k in range(num_message_types):
       messages[k] = einsum('nd,dm->nm', edge_transform, node_states) # [N, M]

     received_messages = zeros(num_nodes, M) # [N, M]
     for k in range(num_message_types):
       received_messages += einsum('nm,nl->lm', messages[k], adj[k])

     node_states = GRU(node_states, received_messages)

   return node_states
```

SKIP - Graphs and graph signals



Graph Signal: $f : \mathcal{V} \to \mathbb{R}^{N \times d}$

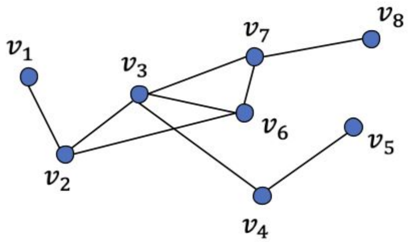
$$\mathcal{V} = \{v_1, \ldots, v_N\}$$

$$\mathcal{E} = \{e_1, \ldots, e_M\}$$

$$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$$

$$\mathcal{V} \longrightarrow \begin{bmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ f(5) \\ f(6) \\ f(7) \\ f(8) \end{bmatrix}$$

SKIP - Graph representations



Adjacency Matrix: $A[i, j] = 1$ if v_i is adjacent to v_j
$A[i, j] = 0$, otherwise

Adjacency Matrix

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$\boldsymbol{A}$

Spectral graph theory. American Mathematical Soc.; 1997.