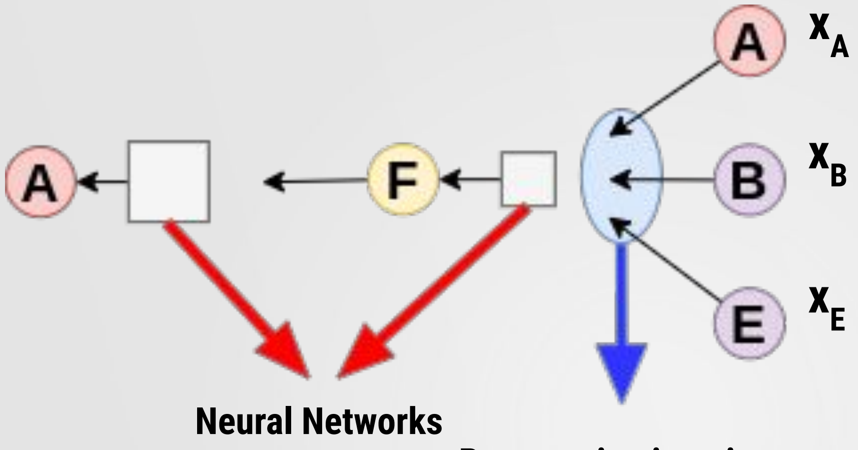
Recurrent Graph Neural Networks

Giovanni Pellegrini[1,2,3]

SML[1] Lab, University of Trento, Italy
TIM[2]
EIT DIGITAL[3]

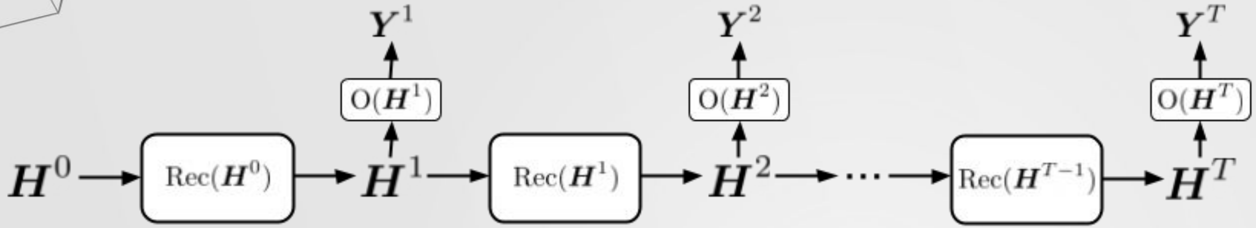
01 GNNs so far

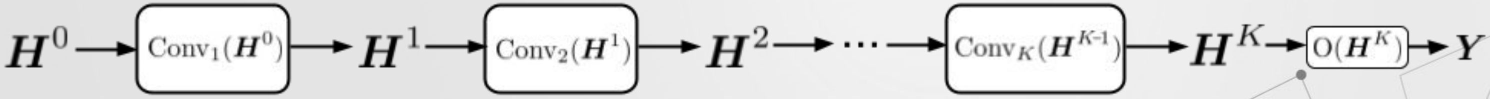


<u>Permutation invariant</u>
Aggregation

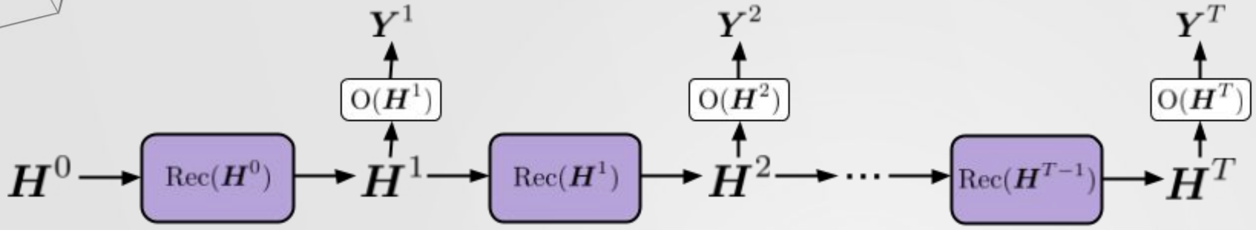
Sum
Average
Max

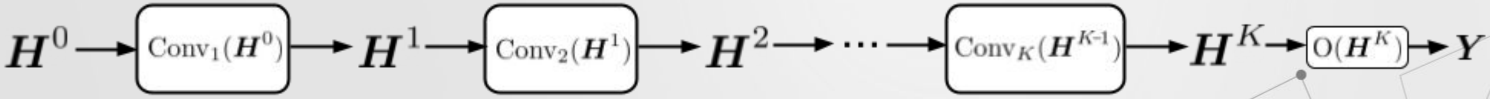
02 Recurrent VS Convolutional



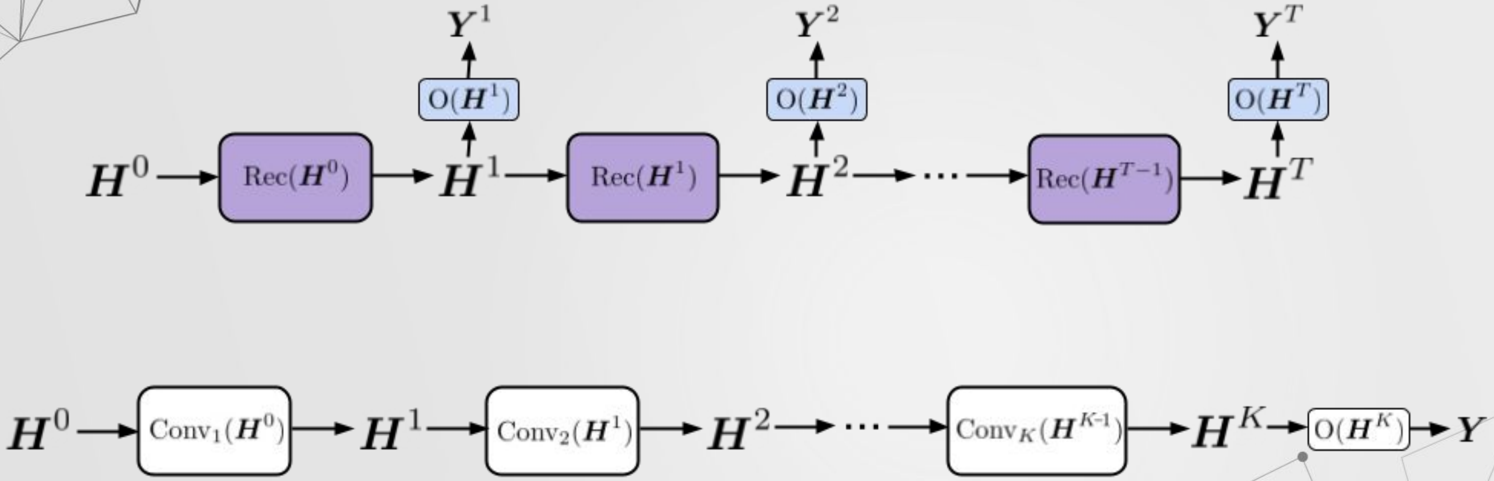


02 Recurrent VS Convolutional

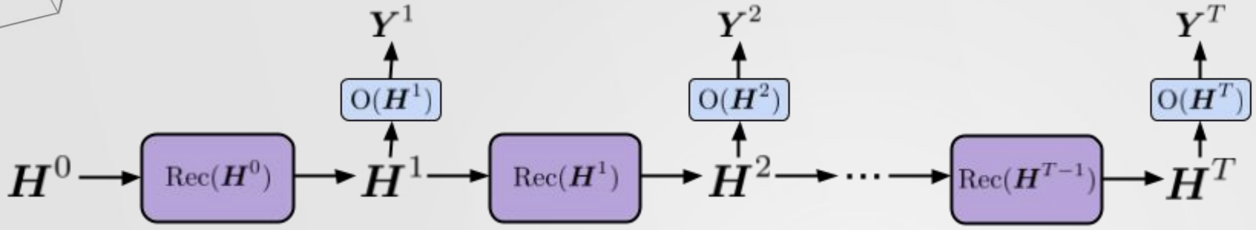


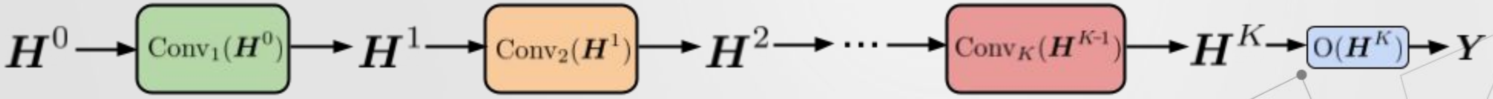


02 Recurrent VS Convolutional

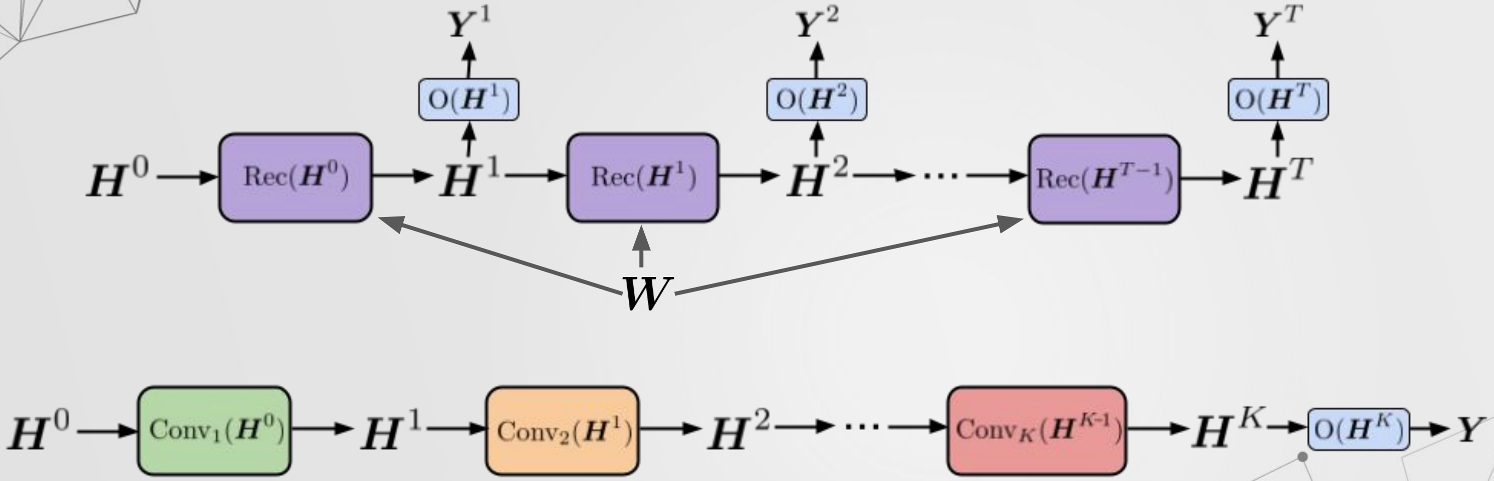


02 Recurrent VS Convolutional





02 Recurrent VS Convolutional



02 Recurrent VS Convolutional

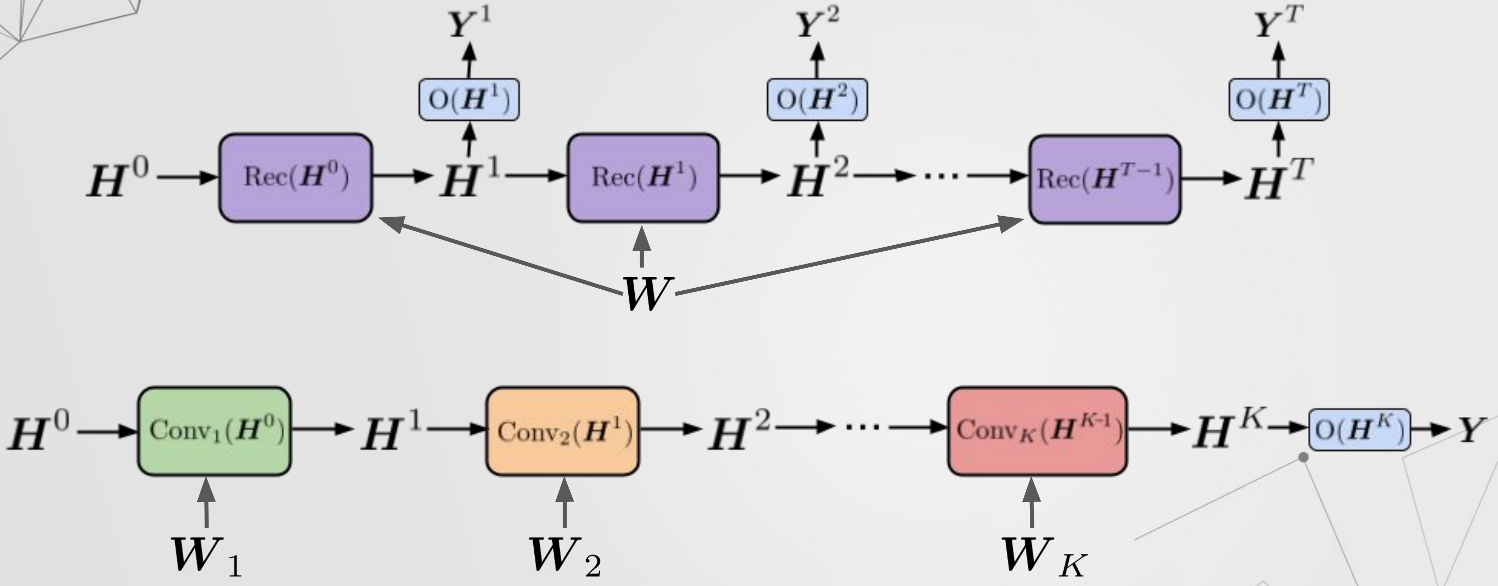


TABLE OF CONTENTS

01 GNNs so far

02 Recurrent VS Convolutional

03 The Graph Neural Network Model

04 Gated Graph NN

05 Gated Graph Sequence NN

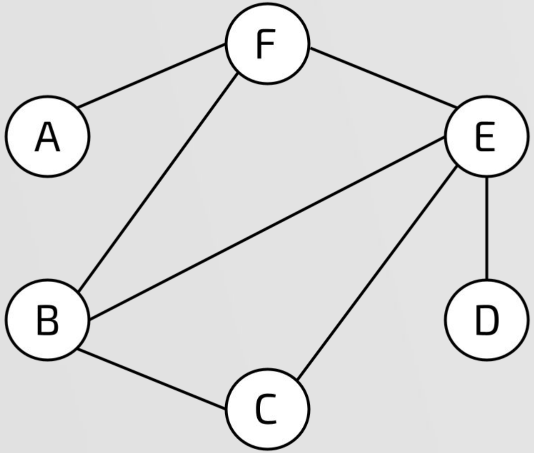
06 PyG Tutorial

03 Graph Neural Network Model[1]

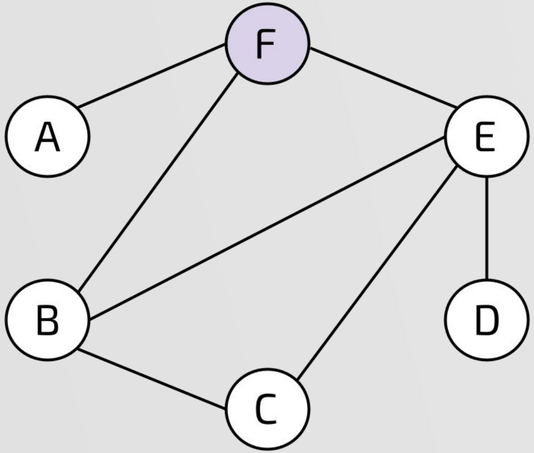
- Pioneer work on Graph Neural Network
- Diffusion mechanism (convolution)
- General framework for graph processing

[1] Scarselli, et al., *The graph neural network model*, IEEE Transactions on Neural Networks, 2009

03 Graph Neural Network Model

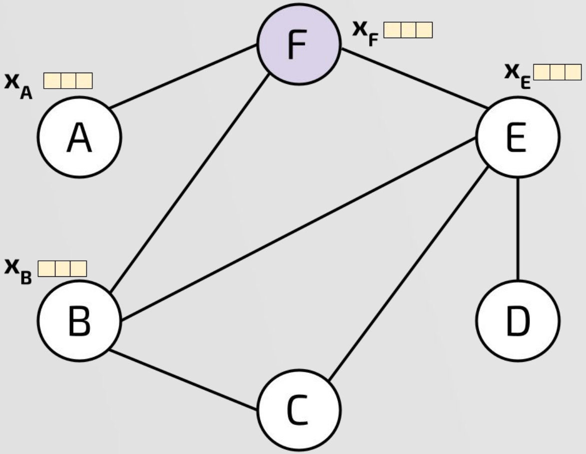


03 Graph Neural Network Model



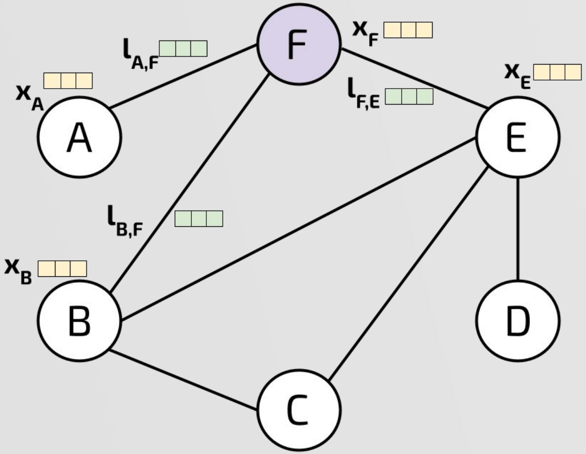
03 Graph Neural Network Model

$$\boldsymbol{x}_v = \mathbb{R}^d$$



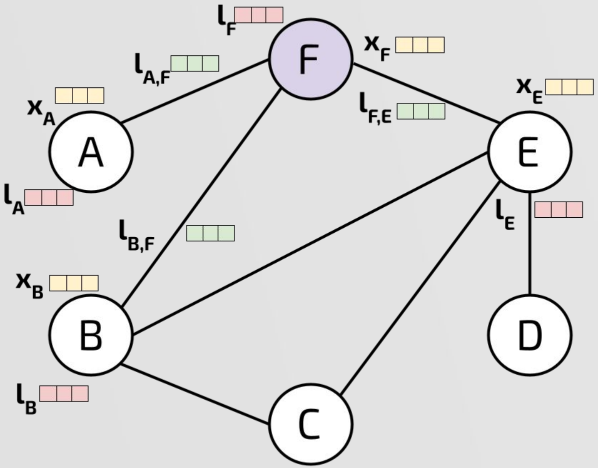
03 Graph Neural Network Model

$$\boldsymbol{x}_v = \mathbb{R}^d$$
$$\boldsymbol{l}_{v,u} = \mathbb{R}^{l_E}$$



03 Graph Neural Network Model

$$\boldsymbol{x}_v = \mathbb{R}^d$$
$$\boldsymbol{l}_{v,u} = \mathbb{R}^{l_E}$$
$$\boldsymbol{l}_v = \mathbb{R}^{l_N}$$

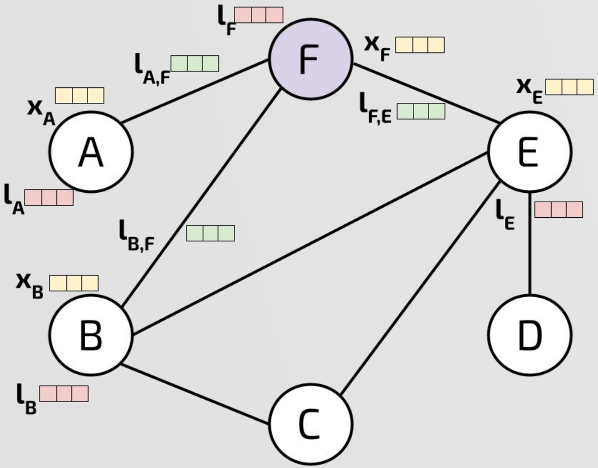


03 Graph Neural Network Model

$$\boldsymbol{x}_v = \mathbb{R}^d$$
$$\boldsymbol{l}_{v,u} = \mathbb{R}^{l_E}$$
$$\boldsymbol{l}_v = \mathbb{R}^{l_N}$$

$co(v)$ = edges connected to v

$ne(v)$ = neighbours of v

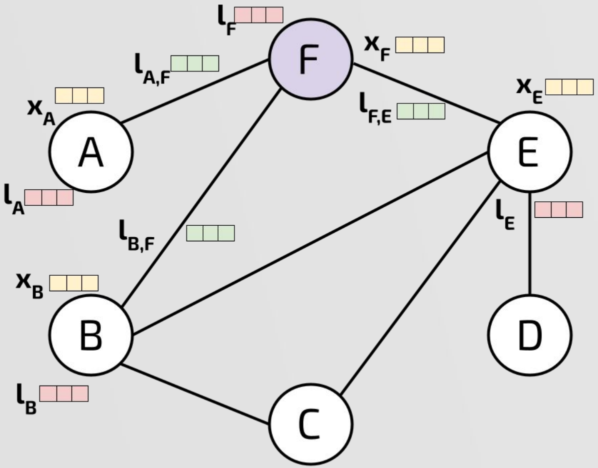


03 Graph Neural Network Model

$$\boldsymbol{x}_v = \mathbb{R}^d$$
$$\boldsymbol{l}_{v,u} = \mathbb{R}^{l_E}$$
$$\boldsymbol{l}_v = \mathbb{R}^{l_N}$$

$co(v)$ = edges connected to v

$ne(v)$ = neighbours of v



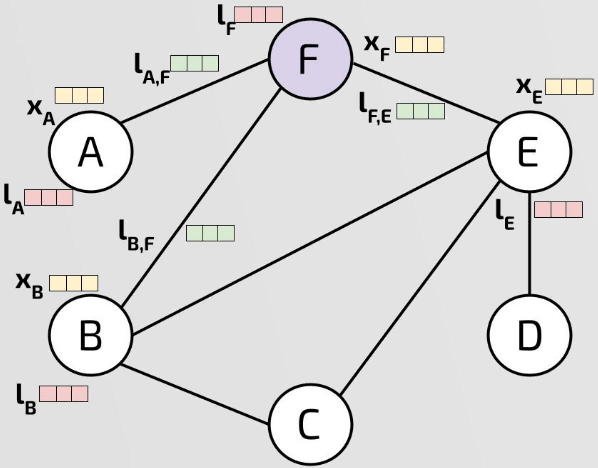
$$\boldsymbol{x}_v^{t+1} = f_{\boldsymbol{w}}(\boldsymbol{l}_v, \boldsymbol{l}_{co(v)}, \boldsymbol{x}^t_{ne(v)}, \boldsymbol{l}_{ne(v)})$$

03 Graph Neural Network Model

$$\boldsymbol{x}_v = \mathbb{R}^d$$
$$\boldsymbol{l}_{v,u} = \mathbb{R}^{l_E}$$
$$\boldsymbol{l}_v = \mathbb{R}^{l_N}$$

$co(v)$ = edges connected to v

$ne(v)$ = neighbours of v



$$\boldsymbol{x}_v^{t+1} = f_{\boldsymbol{w}}(\boldsymbol{l}_v, \boldsymbol{l}_{co(v)}, \boldsymbol{x}_{ne(v)}^t, \boldsymbol{l}_{ne(v)})$$

$$\boldsymbol{o}_v^t = g_{\boldsymbol{w}}(\boldsymbol{x}_v^t, \boldsymbol{l}_v)$$

03 Graph Neural Network Model

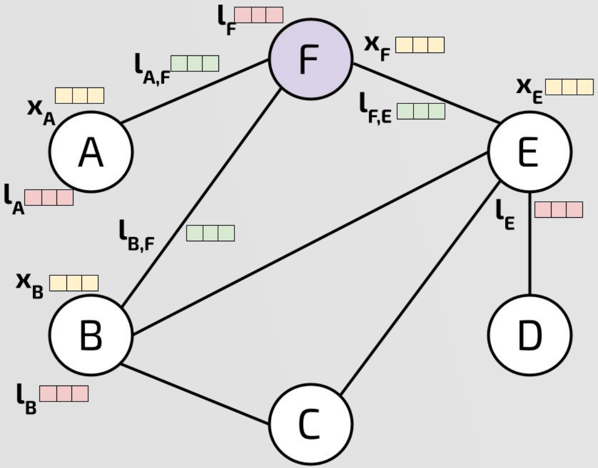
$$\boldsymbol{x}_v = \mathbb{R}^d$$

$$\boldsymbol{l}_{v,u} = \mathbb{R}^{l_E}$$

$$\boldsymbol{l}_v = \mathbb{R}^{l_N}$$

$co(v)$ = edges connected to v

$ne(v)$ = neighbours of v



$$\boldsymbol{x}_v^{t+1} = f_{\boxed{\boldsymbol{w}}}(\boldsymbol{l}_v, \boldsymbol{l}_{co(v)}, \boldsymbol{x}_{ne(v)}^t, \boldsymbol{l}_{ne(v)})$$

learnable parameters

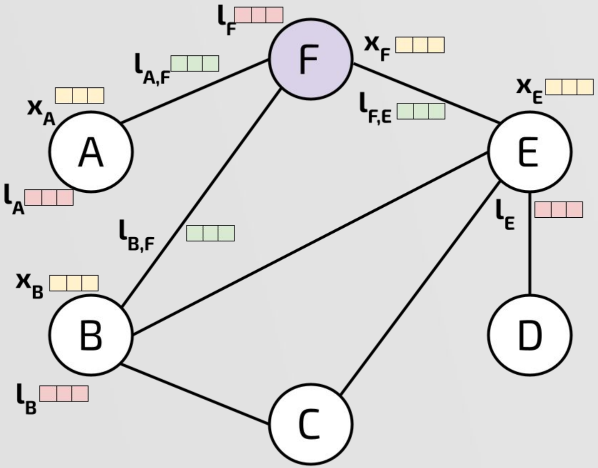
$$\boldsymbol{o}_v^t = g_{\boxed{\boldsymbol{w}}}(\boldsymbol{x}_v^t, \boldsymbol{l}_v)$$

03 Graph Neural Network Model

$$\boldsymbol{x}_v = \mathbb{R}^d$$
$$\boldsymbol{l}_{v,u} = \mathbb{R}^{l_E}$$
$$\boldsymbol{l}_v = \mathbb{R}^{l_N}$$

$co(v)$ = edges connected to v

$ne(v)$ = neighbours of v



$$\boldsymbol{x}_v^{t+1} = f_{\boxed{\boldsymbol{w}}}(\boldsymbol{l}_v, \boldsymbol{l}_{co(v)}, \boldsymbol{x}_{ne(v)}^t, \boldsymbol{l}_{ne(v)})$$

learnable parameters

$$\boldsymbol{o}_v^t = g_{\boxed{\boldsymbol{w}}}(\boldsymbol{x}_v^t, \boldsymbol{l}_v)$$

$f_{\boldsymbol{w}}$ = transition function

$g_{\boldsymbol{w}}$ = output function

03 Graph Neural Network Model

Repeated forward of the transition function create an **encoding network** $\varphi_{\boldsymbol{w}}$

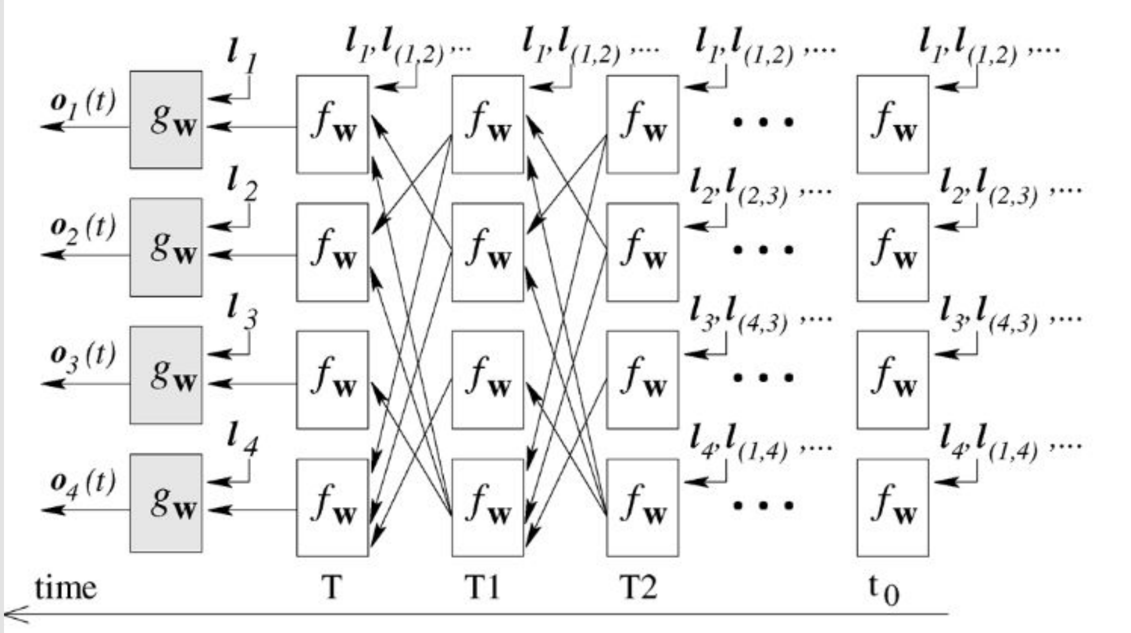


Image taken from the original publication

03 Graph Neural Network Model

Goal: converge to a unique solution for $\mathbf{x}_v$ and $\mathbf{o}_v$

03 Graph Neural Network Model

Goal: converge to a unique solution for $\mathbf{x}_v$ and $\mathbf{o}_v$

If the transition function $f_{\boldsymbol{w}}$ is a **contraction mapping**, there exists a fixed point solution

$$\|\boldsymbol{x}_v^{t+1} - \boldsymbol{x}_v^{t}\| < \epsilon$$

03 Graph Neural Network Model

Goal: converge to a unique solution for $\mathbf{x}_v$ and $\mathbf{o}_v$

If the transition function $f_{\boldsymbol{w}}$ is a **contraction mapping**, there exists a fixed point solution

$$\|\boldsymbol{x}_v^{t+1} - \boldsymbol{x}_v^t\| < \epsilon$$

If $f_{\boldsymbol{w}}$ is a NN, to ensure the contraction mapping a penalty based on the norm of the Jacobian is added to the loss function

MAIN

initialize $\boldsymbol{w}$;

$\boldsymbol{x}$=**Forward**($\boldsymbol{w}$);

repeat

$\frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{w}}$=BACKWARD($\boldsymbol{x}, \boldsymbol{w}$);

$\boldsymbol{w}$=$\boldsymbol{w} - \lambda \cdot \frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{w}}$;

$\boldsymbol{x}$=FORWARD($\boldsymbol{w}$);

until (a stopping criterion);

return $\boldsymbol{w}$;

end

03 Graph Neural Network Model

```
MAIN
    initialize w;
    x=Forward(w);
    repeat
        ∂e_w/∂w = BACKWARD(x, w);
        w = w − λ · ∂e_w/∂w;
        x = FORWARD(w);
    until (a stopping criterion);
    return w;
end
```

```
FORWARD(w)
    initialize x(0), t = 0;
    repeat
        x(t + 1) = F_w(x(t), l);
        t = t + 1;
    until ||x(t) − x(t − 1)|| ≤ ε_f
    return x(t);
end
```

03 Graph Neural Network Model

MAIN
    initialize $\boldsymbol{w}$;
    $\boldsymbol{x}$=**Forward**($\boldsymbol{w}$);
    repeat
        $\frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{w}}$=BACKWARD($\boldsymbol{x}, \boldsymbol{w}$);
        $\boldsymbol{w}=\boldsymbol{w} - \lambda \cdot \frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{w}}$;
        $\boldsymbol{x}$=FORWARD($\boldsymbol{w}$);
    until (a stopping criterion);
    return $\boldsymbol{w}$;
end

FORWARD($\boldsymbol{w}$)
    initialize $\boldsymbol{x}(0)$, $t = 0$;
    repeat
        $\boldsymbol{x}(t+1) = F_{\boldsymbol{w}}(\boldsymbol{x}(t), \boldsymbol{l})$;
        t=$t+1$;
    until $\|\boldsymbol{x}(t) - \boldsymbol{x}(t-1)\| \leq \varepsilon_f$
    return $\boldsymbol{x}(t)$;
end

BACKWARD($\boldsymbol{x}$,$\boldsymbol{w}$)
    $\boldsymbol{o} = G_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l}_{\boldsymbol{N}})$;
    $\boldsymbol{A} = \frac{\partial F_{\boldsymbol{w}}}{\partial \boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{l})$;
    $\boldsymbol{b} = \frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{o}} \cdot \frac{\partial G_{\boldsymbol{w}}}{\partial \boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{l}_{\boldsymbol{N}})$;
    initialize $\boldsymbol{z}(0)$, t=0;
    repeat
        $\boldsymbol{z}(t) = \boldsymbol{z}(t+1) \cdot \boldsymbol{A} + \boldsymbol{b}$;
        t=$t-1$;
    until $\|\boldsymbol{z}(t-1) - \boldsymbol{z}(t)\| \leq \varepsilon_b$;
    $\boldsymbol{c} = \frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{o}} \cdot \frac{\partial G_{\boldsymbol{w}}}{\partial \boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l}_{\boldsymbol{N}})$;
    $\boldsymbol{d} = \boldsymbol{z}(t) \cdot \frac{\partial F_{\boldsymbol{w}}}{\partial \boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l})$;
    $\frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{w}} = \boldsymbol{c} + \boldsymbol{d}$;
    return $\frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{w}}$;
end

03 Graph Neural Network Model

```
MAIN
    initialize w;
    x=Forward(w);
    repeat
        ∂e_w/∂w = BACKWARD(x, w);
        w = w − λ · ∂e_w/∂w;
        x = FORWARD(w);
    until (a stopping criterion);
    return w;
end
```

```
FORWARD(w)
    initialize x(0), t = 0;
    repeat
        x(t + 1) = F_w(x(t), l);
        t = t + 1;
    until ||x(t) − x(t − 1)|| ≤ ε_f
    return x(t);
end
```

```
BACKWARD(x,w)
    o = G_w(x, l_N);
    A = ∂F_w/∂x (x, l);
    b = ∂e_w/∂o · ∂G_w/∂x (x, l_N);
    initialize z(0), t=0;
    repeat
        z(t) = z(t + 1) · A + b;
        t = t − 1;
    until ||z(t − 1) − z(t)|| ≤ ε_b;
    c = ∂e_w/∂o · ∂G_w/∂w (x, l_N);
    d = z(t) · ∂F_w/∂w (x, l);
    ∂e_w/∂w = c + d;
    return ∂e_w/∂w;
end
```

Almeida - Pineda algorithm

04 Gated Graph Neural Network[2]

- Adaptation of the GNNM
- Uses GRU (Gated Recurrent Units) as transition function
- Iterate over T timesteps (instead of until convergence)
- Uses BTT (BackProp Through Time) to compute gradient

[1] Li et al, *Gated graph sequence neural networks*, ICLR, 2015.

04 Gated Graph Neural Network

Node annotations: node embeddings with additional information

Propagation Model:

$$\mathbf{h}_v^{(1)} = [\boldsymbol{x}_v^\top, \mathbf{0}]^\top \tag{1}$$

$$\mathbf{a}_v^{(t)} = \mathbf{A}_{v:}^\top \left[\mathbf{h}_1^{(t-1)\top} \dots \mathbf{h}_{|\mathcal{V}|}^{(t-1)\top}\right]^\top + \mathbf{b} \tag{2}$$

$$\mathbf{z}_v^t = \sigma\left(\mathbf{W}^z \mathbf{a}_v^{(t)} + \mathbf{U}^z \mathbf{h}_v^{(t-1)}\right) \tag{3}$$

$$\mathbf{r}_v^t = \sigma\left(\mathbf{W}^r \mathbf{a}_v^{(t)} + \mathbf{U}^r \mathbf{h}_v^{(t-1)}\right) \tag{4}$$

$$\widetilde{\mathbf{h}_v^{(t)}} = \tanh\left(\mathbf{W}\mathbf{a}_v^{(t)} + \mathbf{U}\left(\mathbf{r}_v^t \odot \mathbf{h}_v^{(t-1)}\right)\right) \tag{5}$$

$$\mathbf{h}_v^{(t)} = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{(t-1)} + \mathbf{z}_v^t \odot \widetilde{\mathbf{h}_v^{(t)}}. \tag{6}$$

04 Gated Graph Neural Network

Node annotations: node embeddings with additional information

Propagation Model:

Node state

Node annotation

$$\boxed{\mathbf{h}_v^{(1)}} = [\boxed{\boldsymbol{x}_v^\top}, \mathbf{0}]^\top \tag{1}$$

$$\mathbf{a}_v^{(t)} = \mathbf{A}_{v:}^\top \left[\mathbf{h}_1^{(t-1)\top} \ldots \mathbf{h}_{|\mathcal{V}|}^{(t-1)\top}\right]^\top + \mathbf{b} \tag{2}$$

$$\mathbf{z}_v^t = \sigma\left(\mathbf{W}^z \mathbf{a}_v^{(t)} + \mathbf{U}^z \mathbf{h}_v^{(t-1)}\right) \tag{3}$$

$$\mathbf{r}_v^t = \sigma\left(\mathbf{W}^r \mathbf{a}_v^{(t)} + \mathbf{U}^r \mathbf{h}_v^{(t-1)}\right) \tag{4}$$

$$\widetilde{\mathbf{h}_v^{(t)}} = \tanh\left(\mathbf{W} \mathbf{a}_v^{(t)} + \mathbf{U}\left(\mathbf{r}_v^t \odot \mathbf{h}_v^{(t-1)}\right)\right) \tag{5}$$

$$\mathbf{h}_v^{(t)} = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{(t-1)} + \mathbf{z}_v^t \odot \widetilde{\mathbf{h}_v^{(t)}}. \tag{6}$$

04 Gated Graph Neural Network

Node annotations: node embeddings with additional information

Propagation Model:

Node state

Node annotation

$$\boxed{\mathbf{h}_v^{(1)}} = [\boxed{\boldsymbol{x}_v^\top}, \mathbf{0}]^\top \quad (1)$$

$$\mathbf{a}_v^{(t)} = \mathbf{A}_{v:}^\top \left[\mathbf{h}_1^{(t-1)\top} \dots \mathbf{h}_{|\mathcal{V}|}^{(t-1)\top}\right]^\top + \mathbf{b} \quad (2)$$

$$\mathbf{z}_v^t = \sigma\left(\mathbf{W}^z \mathbf{a}_v^{(t)} + \mathbf{U}^z \mathbf{h}_v^{(t-1)}\right) \quad (3)$$

$$\mathbf{r}_v^t = \sigma\left(\mathbf{W}^r \mathbf{a}_v^{(t)} + \mathbf{U}^r \mathbf{h}_v^{(t-1)}\right) \quad (4)$$

$$\widetilde{\mathbf{h}_v^{(t)}} = \tanh\left(\mathbf{W}\mathbf{a}_v^{(t)} + \mathbf{U}\left(\mathbf{r}_v^t \odot \mathbf{h}_v^{(t-1)}\right)\right) \quad (5)$$

$$\mathbf{h}_v^{(t)} = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{(t-1)} + \mathbf{z}_v^t \odot \widetilde{\mathbf{h}_v^{(t)}}. \quad (6)$$

$$\mathrm{o}_\mathrm{v} = \mathrm{g}(\mathbf{h}_\mathrm{v}{}^{(\mathrm{T})}, \mathrm{x}_\mathrm{v})$$

05 Gated Graph Sequence Neural Network

What if we want to produce sequences of output values?

05 Gated Graph Sequence Neural Network

What if we want to produce sequences of output values?

GGSNN : multiple GGNNs operate in sequence to produce $\boldsymbol{o}_v^{(1)}, \ldots, \boldsymbol{o}_v^{(k)}$

05 Gated Graph Sequence Neural Network

What if we want to produce sequences of output values?

GGSNN : multiple GGNNs operate in sequence to produce $\boldsymbol{o}_v^{(1)}, \ldots, \boldsymbol{o}_v^{(k)}$

$\mathcal{F}_x^{(k)}$ Computes $\mathbf{X}^{(k+1)}$ from $\mathbf{X}^{(k)}$

$\mathcal{F}_o^{(k)}$ Computes $\mathbf{o}^{(k)}$ from $\mathbf{X}^{(k)}$

05 Gated Graph Sequence Neural Network

What if we want to produce sequences of output values?

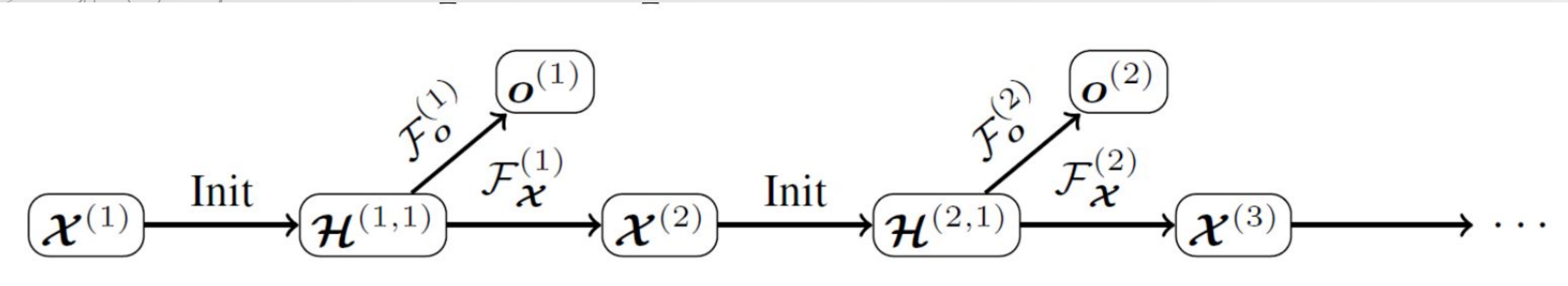
GGSNN : multiple GGNNs operate in sequence to produce $\boldsymbol{o}_v^{(1)}, \ldots, \boldsymbol{o}_v^{(k)}$

$\mathcal{F}_x^{(k)}$ Computes $\mathbf{X}^{(k+1)}$ from $\mathbf{X}^{(k)}$

$\mathcal{F}_o^{(k)}$ Computes $\mathbf{o}^{(k)}$ from $\mathbf{X}^{(k)}$

$\mathcal{F}^{(k)}$ implements both the **transition** and **output** function!

05 Gated Graph Sequence Neural Network



$$\boldsymbol{X}^{(k)} = [\boldsymbol{x}_1^{(k)}, \dots, \boldsymbol{x}_v^{(k)}]$$

$$\boldsymbol{H}^{(k,t)} = [\boldsymbol{h}_1^{(k,t)}, \dots, \boldsymbol{h}_v^{(k,t)}]$$

k = length of the sequence

t = timesteps

06 PyG tutorial

Let's go to the Jupyter notebook….