Gated Graph Sequence Neural Network

Deepdog 08.25.21

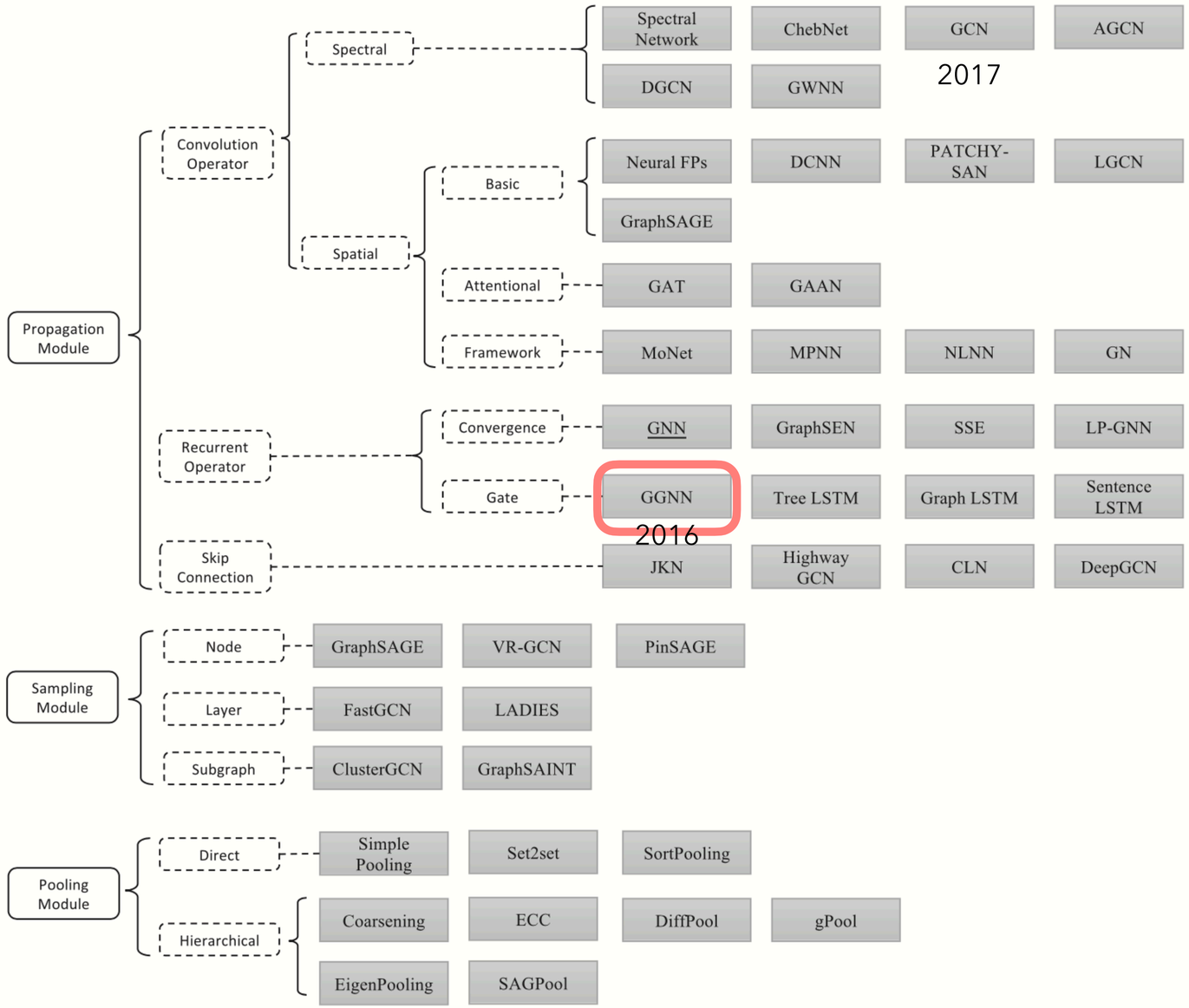


Fig. 3. An overview of computational modules.

Motivation

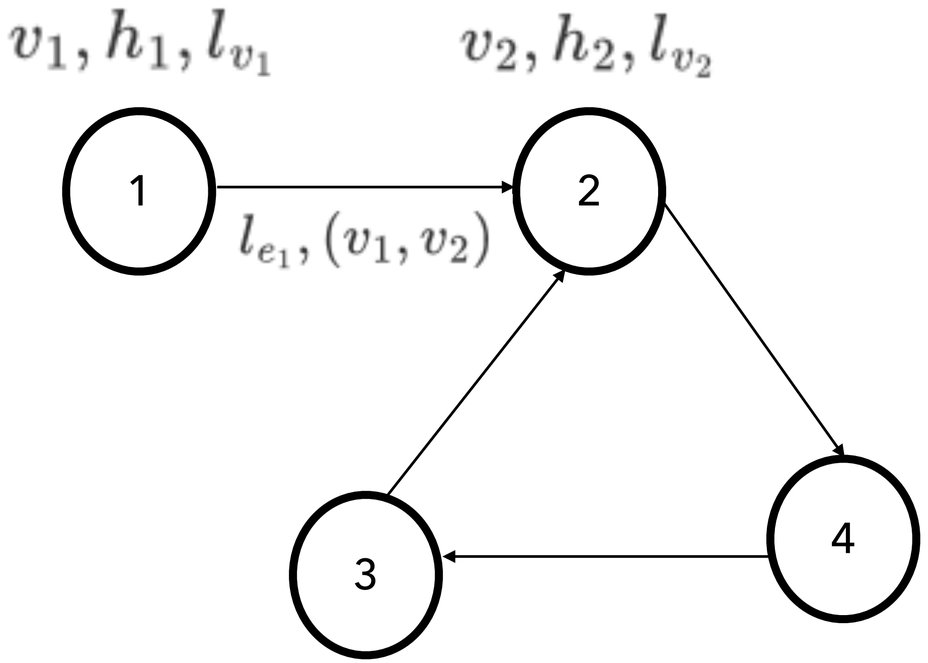
- Previous work on feature learning for graph-structured inputs has focused on models that produce single outputs
- Our motivating application comes from program verification and requires outputting logical formulas

Feature learning on graphs

1. Learning a representation of the input graph
2. Learning representation of the internal state during the process of producing a sequence of outputs
    - How to learn features on the graph that encode the partial output sequence that has already been produced (e.g., the path so far if outputting a path) and that
    - Still needs to be produced (e.g., the remaining path)

Graph Neural Networks (Gori et al., 2005; Scarselli et al., 2009)

$$G = (V, E)$$
$$v \in V$$
$$e = (v, v') \in V \times V$$
$$(v, v') : v \to v'$$



$$\mathbf{h}_v \in \mathbb{R}^D : node\ vector$$
$$l_v \in \{1, \ldots, L_v\} : nodel\ labels$$
$$l_e \in \{1, \ldots, L_e\} : edge\ labels$$

$$\mathbf{h}_S = \{\mathbf{h}_v | v \in S\} : S\ is\ set\ of\ nodes$$
$$l_S = \{l_e | e \in S\} : S\ is\ set\ of\ edges$$

$$IN(v) = \{v' | (v', v) \in E\} : returns\ the\ set\ of\ predecessor\ nodes$$
$$Out(v) = \{v' | (v, v') \in E\} : returns\ the\ set\ of\ sucessor\ nodes$$

$$IN(v_2) = \{v_1, v_3\}$$

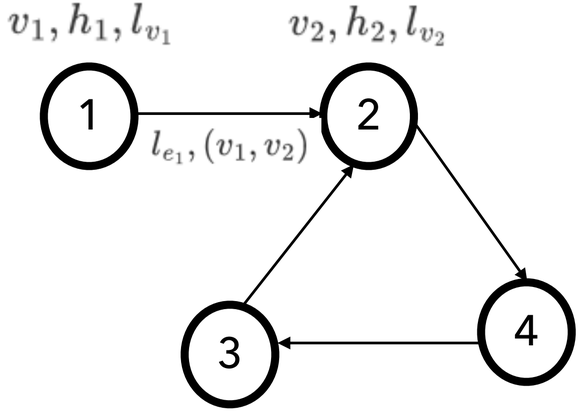
$$Out(v_2) = \{V_4\}$$

$$Co(v_2) = \{(v_2, v_4), (v_1, v_2), (v_3, v_2)\}$$

$$NBR(v) = IN(v) \cup Out(v)$$

$$Co(v) = \{(v', v'') \in E | v = v' \vee v = v''\}$$

GNNs
- Propagation Model



$$h_v^{(0)} : set\ to\ arbitrary\ values$$

$$\mathbf{h}_v^t = f^*(l_v, l_{Co(v)}, l_{NBR(v)}, \mathbf{h}_{NBR(v)}^{(t-1)})$$

$$f^*(l_v, l_{Co(v)}, l_{NBR(v)}, \mathbf{h}_{NBR(v)}^{(t-1)}) = \sum_{v' \in IN(v)} f(l_v, l_{(v',v)}, l_{v'}, \mathbf{h}_{v'}^{t-1}) + \sum_{v' \in Out(v)} f(l_v, l_{(v,v')}, l_{v'}, \mathbf{h}_{v'}^{t-1})$$

$$f(l_v, l_{(v',v)}, l_{v'}, \mathbf{h}_{v'}^t) = \mathbf{A}^{(l_v, l_{(v',v)}, l_{v'})} \mathbf{h}_{v'}^{(t-1)} + \mathbf{b}^{(l_v, l_{(v',v)}, l_{v'})}$$

Where f() is either a linear function of h or a neural network

$$IN(v_2) = \{v_1, v_3\}$$

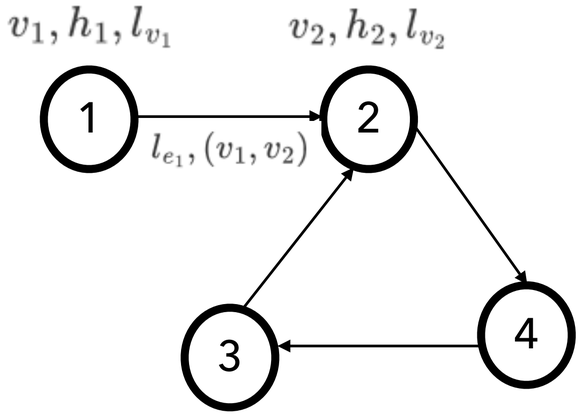
$$Out(v_2) = \{V_4\}$$

$$Co(v_2) = \{(v_2, v_4), (v_1, v_2), (v_3, v_2)\}$$

$$NBR(v) = IN(v) \cup Out(v)$$

$$Co(v) = \{(v', v'') \in E | v = v' \vee v = v''\}$$

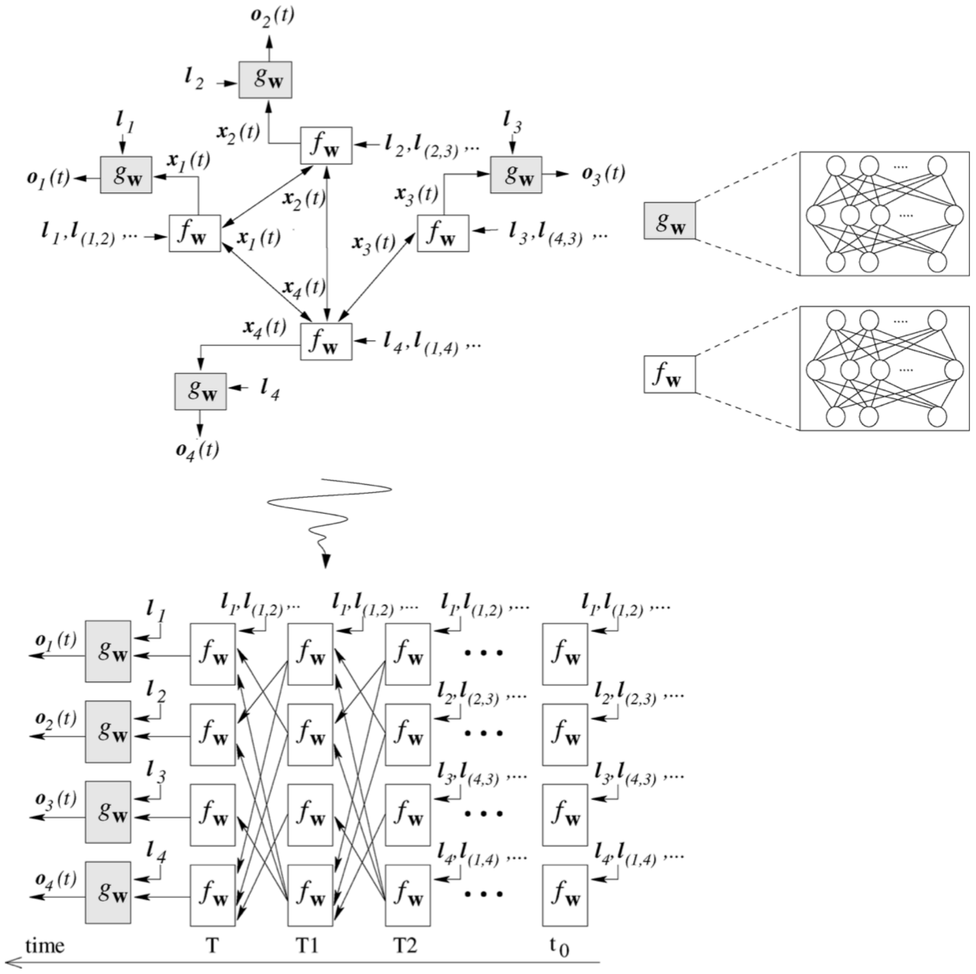
GNNs
- Output model and Learning



$$O_v = g(\mathbf{h}_v^{(T)}, l_v)$$

- g() can be a neural net

GNNs
- Output model and Learning




```
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
```

- x(t)를 T 시점까지 계속 업데이트
  Banach's fixed point theorem에 의해서 $x(T) \simeq x(T-1)$

- loss를 계산하고 W의 gradient를 계산
  이때 W는 last step에 대한 gradient로 update.

Scarselli, F., Gori, M., Tsoi, A., Hagenbuchner, M. & Monfardini, G. 2009, 'The graph neural network model', IEEE Transactions on Neural Networks, vol. 20, no. 1, pp. 61-80.

GNNs
- Limitations

Learning is done via the Almeida-Pineda algorithm (Almeida, 1990; Pineda, 1987), which works by running the propagation to convergence, and then computing gradients based upon the converged solution. This has the advantage of not needing to store intermediate states in order to compute gradients. The disadvantage is that parameters must be constrained so that the propagation step is a contraction map. This is needed to ensure convergence, but it may limit the expressivity of the model. When $f(\cdot)$ is a neural network, this is encouraged using a penalty term on the 1-norm of the network's Jacobian. See Appendix A for an example that gives the intuition that contraction maps have trouble propagating information across a long range in a graph.

GNNs
- Limitations

- Fixed point까지 반복적으로 학습해야하는 방법이 비효율적이다.
  이 가정을 제거하면 multi-layer GNN으로 node와 이웃 nodes의 더 안정적인 representation을 얻을 수 있다.

- GNNs은 같은 parameter를 재사용한다.
  이렇게 hidden states를 sequential하게 update하는 경우는 GRU나 LSTM같은 RNN kernels를 사용할 수 있다.

- GNNs에는 edge에 대한 정보를 충분히 활용하지 못 하고 있다.

Gated Graph Neural Networks

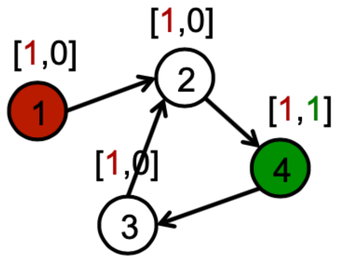
- GNNs을 변형하여 sequential outputs을 수행하는 모델.
- The biggest modification of GNNs is that we use Gated Recurrent Units (Cho et al., 2014) and
- unroll the recurrence for a fixed number of steps T and
- Use back propagation through time in order to compute gradients

Gated Graph Neural Networks
- Node annotations

Initialization

Problem specific node annotations in $\mathbf{h}_v^{(0)}$

Example reachability problem: can we go from A to B?



output yes

It is easy to learn a propagation model that copies and adds the first bit to a node's neighbor.

It is easy to learn an output model that outputs yes if it sees the [■, ■] pattern, otherwise no

https://youtu.be/_Uj-tNjhVDQ

Gated Graph Neural Networks
- Propagation model

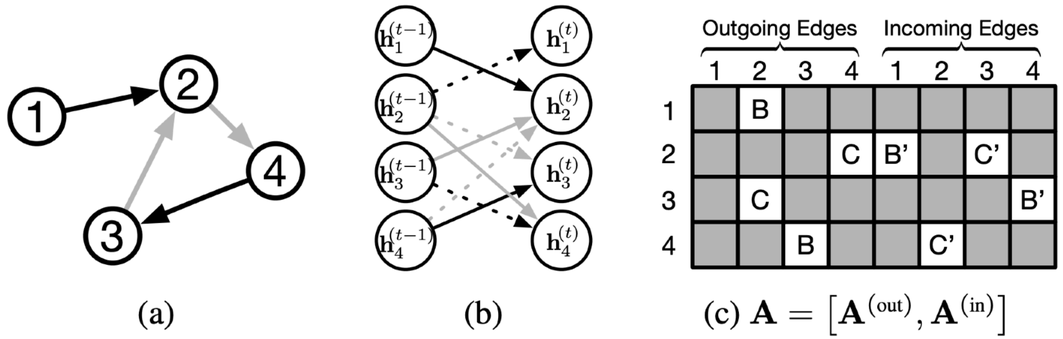


Figure 1: (a) Example graph. Color denotes edge types. (b) Unrolled one timestep. (c) Parameter tying and sparsity in recurrent matrix. Letters denote edge types with B' corresponding to the reverse edge of type B. B and B' denote distinct parameters.

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

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

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

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

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

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

Gated Graph Neural Networks
- Output models

Node selection

$$o_v = g(h_v^{(T)}, x_v)$$

Graph-level outputs

$$h_g = tanh(\sum_{v \in V} \sigma(i(h_v^{(T)}, x_v)) \odot tanh(j(h_v^{(T)}, x_v)))$$

- $\sigma(i(h_v^{(T)}, x_v))$ acts as a soft attention mechanism that decides which nodes are relevant to the current graph level task

Gated Graph Sequence Neural Networks

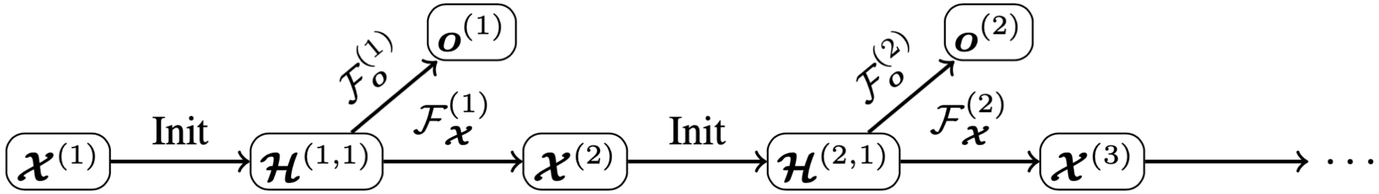
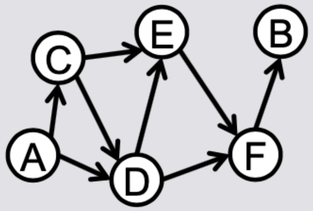


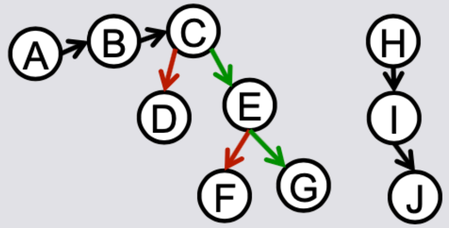
Figure 2: Architecture of GGS-NN models.

Shortest path from A to B?



A – D – F – B

What are the structures in this graph?



List(A,C) ∧ Tree(C) ∧ List(H, J)

https://youtu.be/_Uj-tNjhVDQ

Gated Graph Sequence Neural Networks

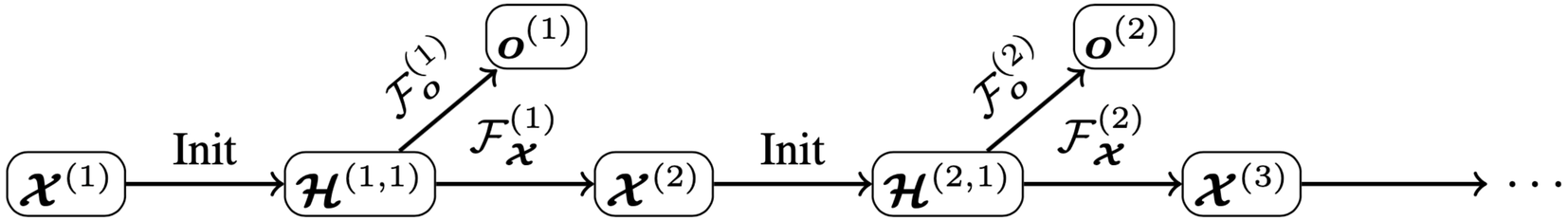


Figure 2: Architecture of GGS-NN models.

- In GGS-NNs, several GG-NNs operate in sequence to produce an output sequence $o^{(1)}, \ldots, o^{(K)}$
- For the k^{th} output step, the matrix of node annotations : $\chi^{(k)} = [x_1^{(k)}, \ldots, x_{|V|}^{(k)}]^T \in \mathbb{R}^{|V| \times L_v}$
- Two GG-NNs contain a propagation model and and output model.

1. $F_o^{(k)} : \chi^{(k)} \longrightarrow o^{(k)}$
2. $F_\chi^{(k)} : \chi^{(k)} \longrightarrow \chi^{(k+1)}$

- In the propagation models, we denote the matrix of node vectors at the t^{th} propagation step of the k^{th} output step as

$$\boldsymbol{H}^{(k,t)} = [h_1^{(k,t)}; \ldots; h_{|V|}^{(k,t)}]^T \in \mathbb{R}^{|V| \times D}$$

  - As before, in step k, we set $\boldsymbol{H}^{(k,1)}$ by 0-extending $\chi^{(k)}$ per node.

Gated Graph Sequence Neural Networks

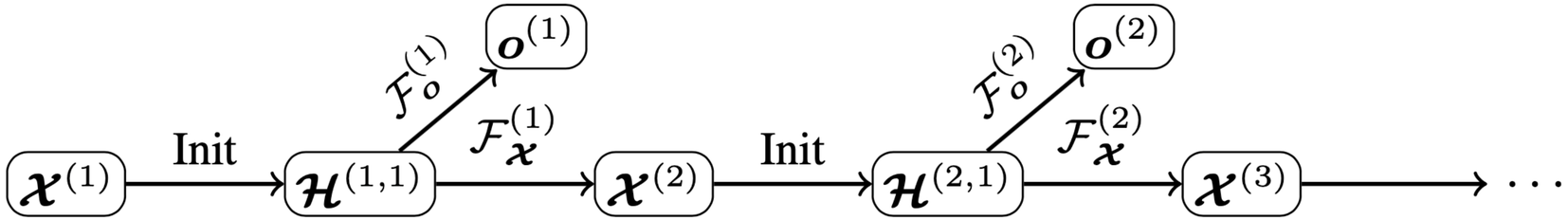


Figure 2: Architecture of GGS-NN models.

- Node annotation output model for predicting $\chi^{(k+1)}$ from $H^{(k,T)}$
- The prediction is done for each node independently using a NN $j(h_v^{(k,T)}, x_v^{(k)})$

$$x_v^{(k+1)} = \sigma(j(h_v^{(k,T)}, x_v^{(k)})$$

Gated Graph Sequence Neural Networks

When to stop

At each prediction step, a separate output GG-NN is used to make a graph-level binary classification prediction on whether to continue or stop.