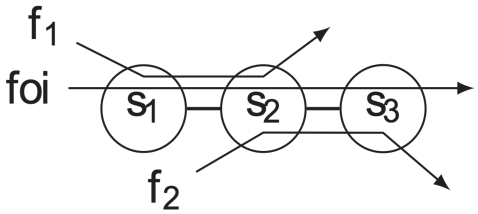
DeepFP: *Predicting flow prolongations in FIFO analysis using Graph neural networks*

Karim Hadidane

Motivation

Derive an end-to-end delay bound for **foi**

Nesting :
a sequence of servers is called nested if any two flows
have disjoint paths or one of them is included in the other's path

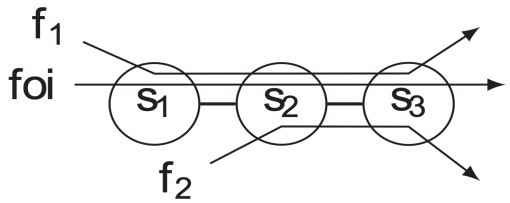


Non-nested

Least Upper Delay Bound (LUDB)

Input: network model

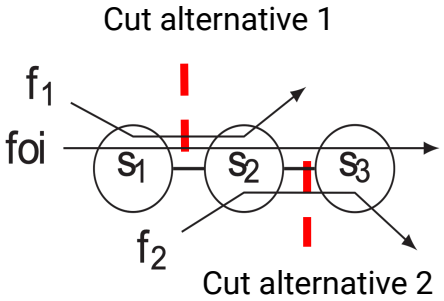
1) (if non nested) Derive all cuts (into nested subtandems)
2) Compute nesting tree (backtracking step)
3) Remove cross flows step by step and compute end-to-end service curve
4) Compute end-to-end delay bound



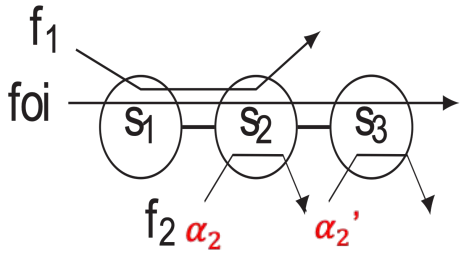
Nested

F. Geyer, A. Scheffler and S. Bondorf, "Tightening Network Calculus Delay Bounds by Predicting Flow Prolongations in the FIFO Analysis," 2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2021, pp. 157-170, doi: 10.1109/RTAS52030.2021.00021.

LUDB: Nested vs Non-nested tandems



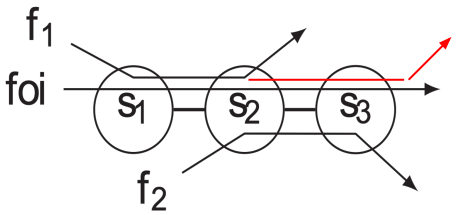
Cut alternative 2



$$h(\alpha_{foi}, ((\beta_1 \otimes (\beta_2 \ominus_\theta \alpha_2)) \ominus_\theta \alpha_1) \otimes (\beta_3 \ominus_\theta \alpha_2')$$

$$\alpha_2' = \alpha_2 \oslash (\beta_2 \ominus_\theta ((\alpha_{foi} + \alpha_1) \oslash \beta_1))$$

Flow Prolongation



$$h(\alpha_{foi} + \alpha_1, \beta_1 \otimes ((\beta_2 \otimes \beta_3) \ominus_\theta \alpha_2))$$

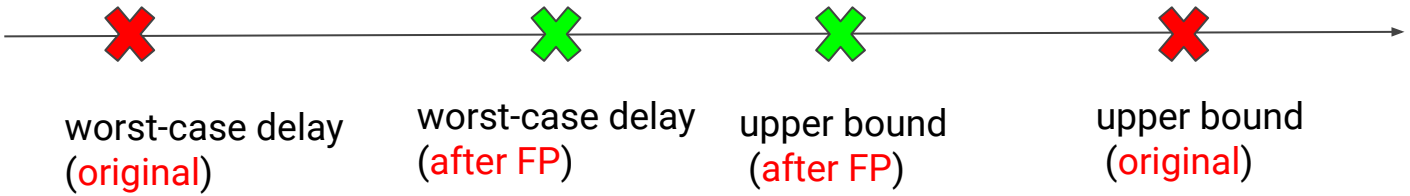
deconvolution: $(f \oslash g)(d) = \sup_{u \geq 0} \{f(d+u) - g(u)\}$

leftover service curve: $\beta^1(t,\theta) = \beta \ominus_\theta \alpha_2 = [\beta(t) - \alpha_2(t-\theta)]^\uparrow \cdot 1_{\theta \geq 0}$

F. Geyer, A. Scheffler and S. Bondorf, "Tightening Network Calculus Delay Bounds by Predicting Flow Prolongations in the FIFO Analysis," 2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2021, pp. 157-170, doi: 10.1109/RTAS52030.2021.00021.

Flow prolongation

- **Possible improvement in delay bounds tightness**
- **Delay bounds derived still valid for original model**

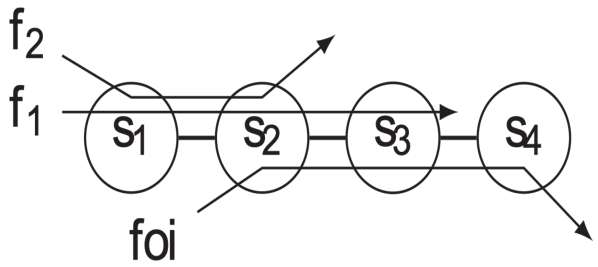


- **Tradeoff between adding interference and nesting benefits**
- **tighter delay bounds, but not always !**

The Problem

Assumptions:

- Feed-forward network
- Servers with rate-latency service curves
- Flows with token-bucket arrival curves



Improve delay bounds using flow prolongations

1- **beneficial?**
2- if so, **best flow prolongations?**

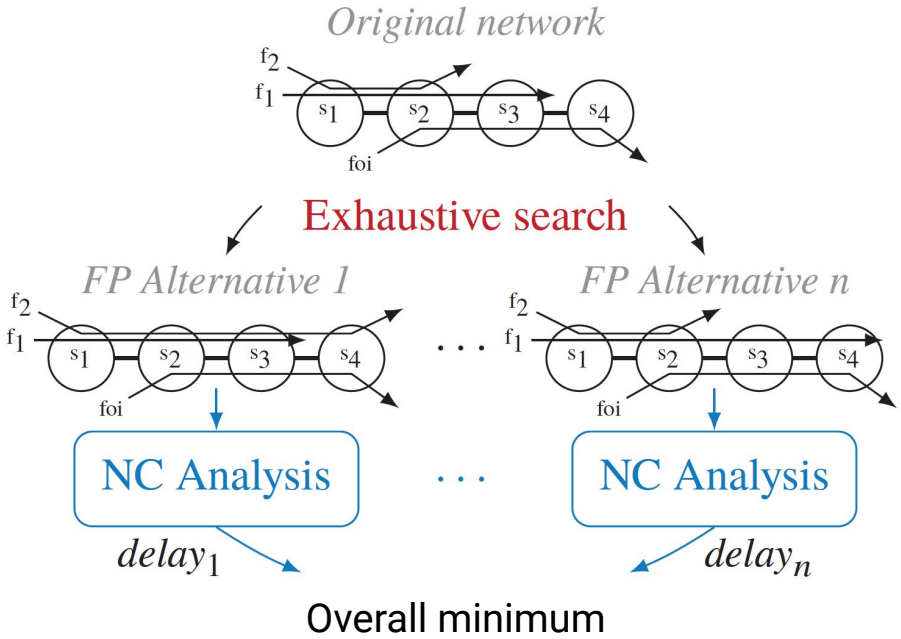
Exhaustive search

Try all possible alternatives,
take the minimum delay bound

$$O(n^m)$$

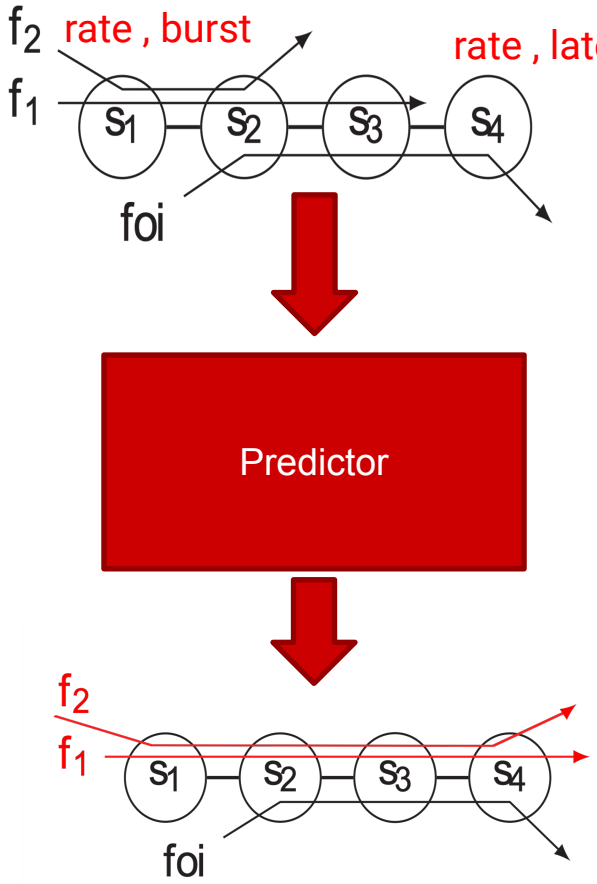
n : # of servers
m : # of cross flows

Not scalable !



F. Geyer, A. Scheffler and S. Bondorf, "Tightening Network Calculus Delay Bounds by Predicting Flow Prolongations in the FIFO Analysis," 2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2021, pp. 157-170, doi: 10.1109/RTAS52030.2021.00021.

Predicting sinks of cross-flows



For each cross-flow, predict sink

Graph Neural Network (GNN)

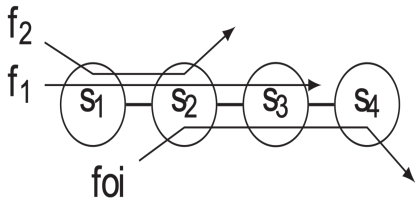
Neural Network for Graph

- Input: Graph G, node n
- Output: Score

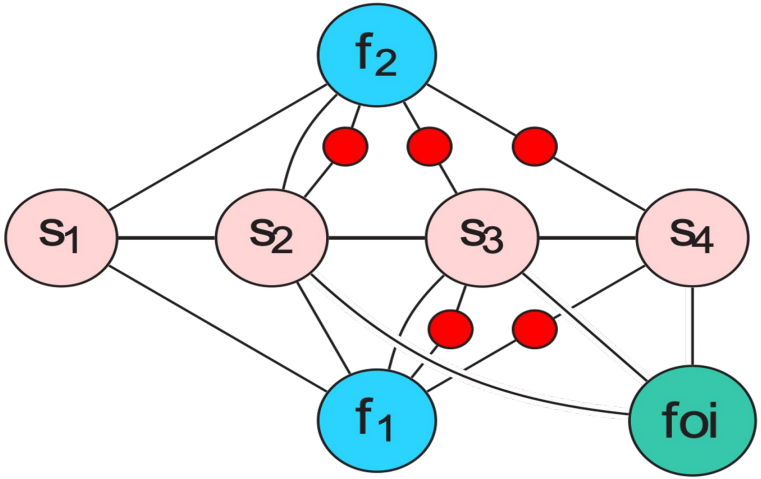
Learning from examples a function $\tau_1(G, n)$

$\tau(G, n)$

Predicting if FP beneficial



Original network



Transformed graph G

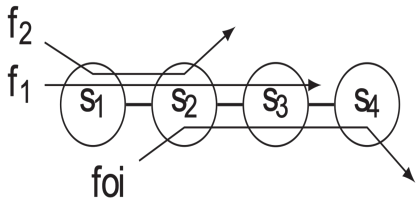
$\tau_1(G, n) \in [0,1]$

Output of GNN

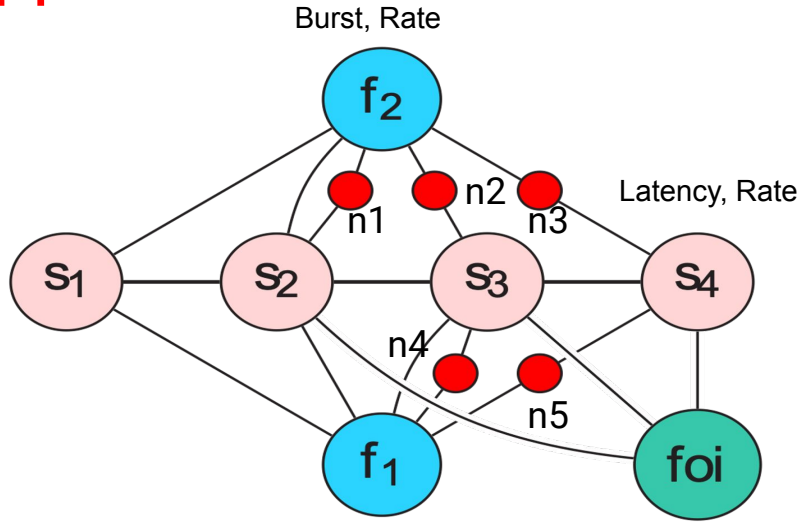
If beneficial, use GNN on next slide

If > 0.5 , beneficial
Otherwise , no

Predicting best FP



Original network



Transformed graph G

GNN

$\tau(G,.)$

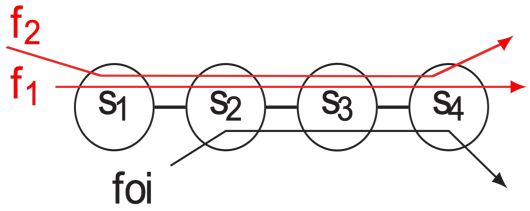
$\in [0,1]$

$$\begin{bmatrix} \tau(G,n1) \\ \tau(G,n2) \\ \tau(G,n3) \\ \tau(G,n4) \\ \tau(G,n5) \end{bmatrix}$$

Output for prolongation nodes

Post-processing step

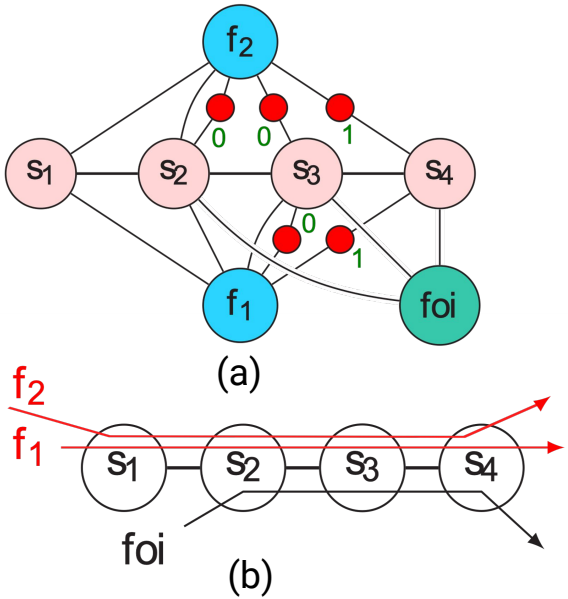
(Only one prolongation node set to 1 (per cross flow))



Output network

Dataset

- networks (token-bucket flows, rate-latency servers, flow of interest)
- Best flow prolongation (found by exhaustive search)

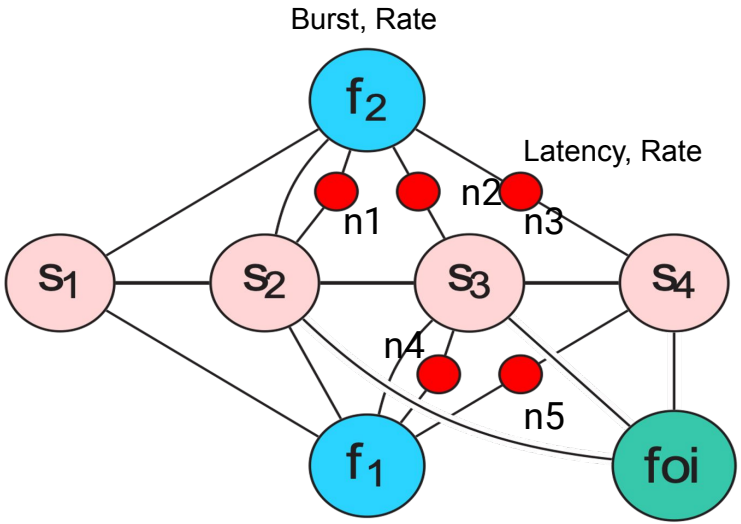


Parameter	Min	Max	Mean
# of servers	4	10	7.8
# of flows	5	35	24.5
# of cross-flows	1	21	4.1
# of prolong. comb. (PMOO-FP_{foi})	2	4024	16.8
# of prolong. comb. (DEBORAH-FP_{foi})	2	131072	247.1
Flow path length	3	9	4.1
Number of nodes in graph	11	128	43.3

Table II: Statistics about the generated dataset

a) Example of graph with targets coded, b) its corresponding network

Training



$\tau(G,.)$

$$\begin{bmatrix} \tau(G,n1) \\ \tau(G,n2) \\ \tau(G,n3) \\ \tau(G,n4) \\ \tau(G,n5) \end{bmatrix}$$

Update weights

$t_{n_i} \in \{0,1\}$

$$\begin{bmatrix} t_{n1} \\ t_{n2} \\ t_{n3} \\ t_{n4} \\ t_{n5} \end{bmatrix}$$

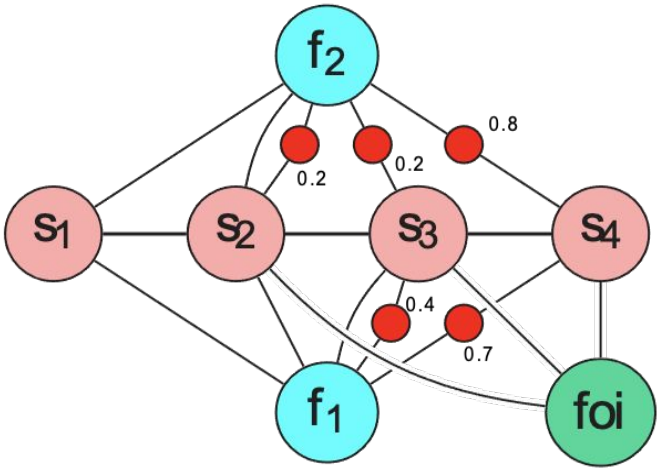
$loss(\tau, t)$

Recap

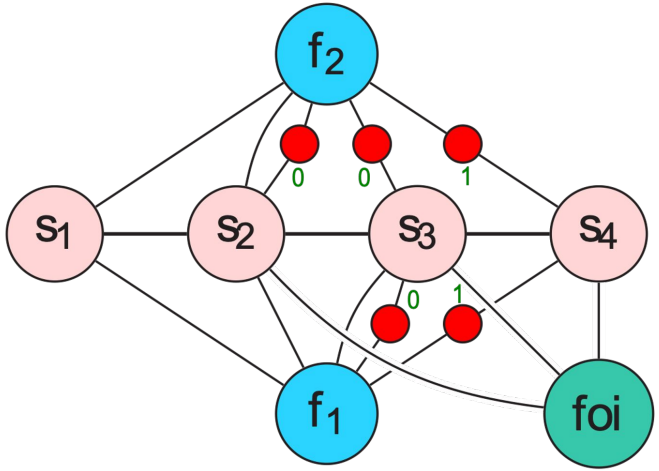
- GNN outputs score between 0 and 1 for prolongation nodes
- GNN outputs processed later to map to network model
- GNN trained using loss function between output (between 0 and 1) and target (0 or 1) per node

DeepFP

Maximal score defines the sink for a cross-flow



Output of gnn



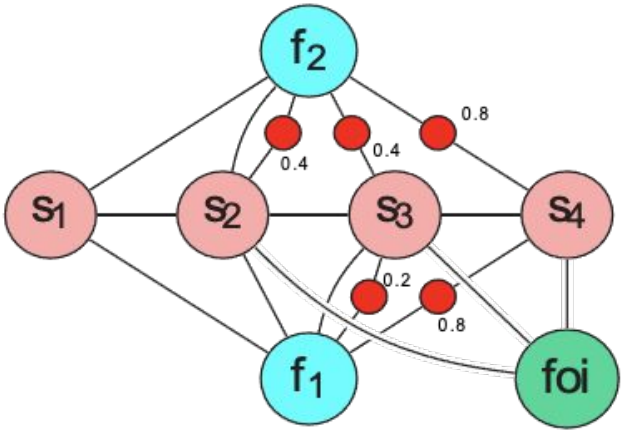
Output of DeepFP

Extending DeepFP: $DeepFP_k$

Sample the sink using a probability computed from GNN scores (each cross-flow)
Generate k combinations

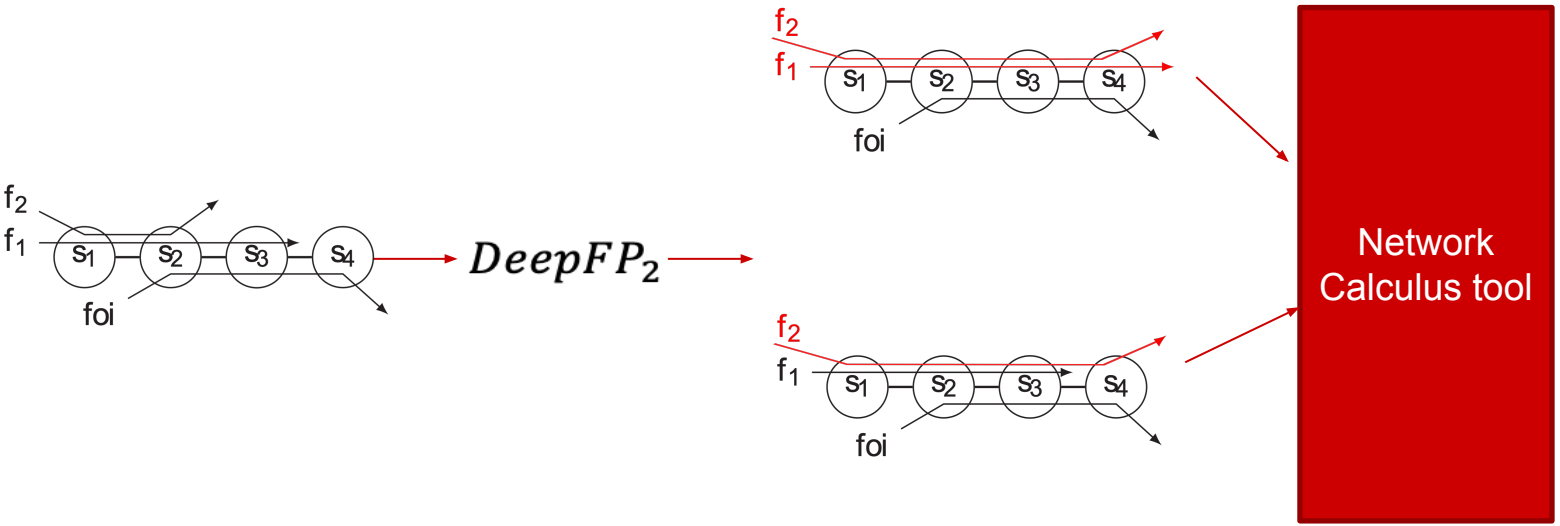
$$prob = \frac{GNN\ score}{sum\ of\ scores\ per\ cross\ flow}$$

Example:



Cross-flow	GNN output scores	Probability distribution
f1	Node n1: 0.4 Node n2 : 0.4 Node n3: 0.8	Node n1: 25% Node n2: 25% Node n3: 50%
f2	Node n4: 0.2 Node n5: 0.8	Node n4: 20% Node n5: 80%

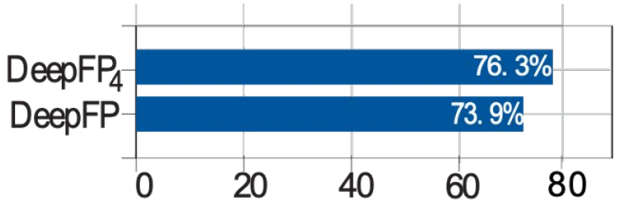
$DeepFP_2$



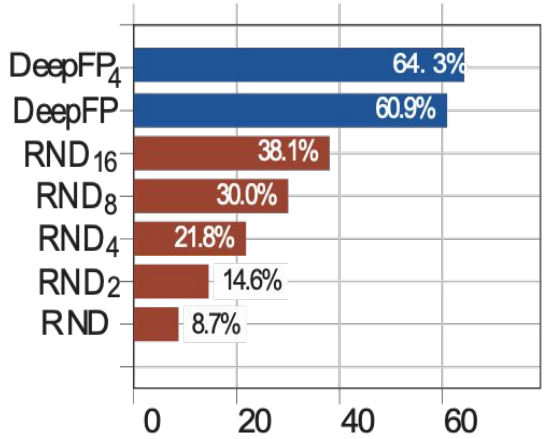
Combination with min delay bound

Results:

Accuracy: ratio of networks where the computed end-to-end delay bound is equal to the best end-to end delay bound



Accuracy (our model)



Accuracy (reported by the paper)

Summary:

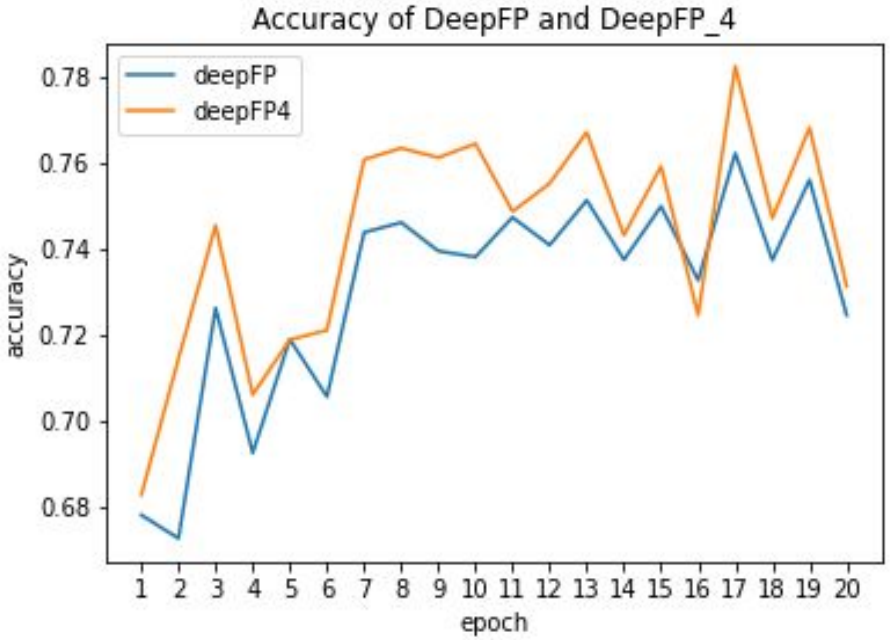
- DeepFP is an approach to make Flow Prolongation scale.
- FP can considerably tighten NC delay bounds derived for FIFO multiplexing networks.
- The proposed GNN-based DeepFP allows to apply it to larger networks.
- We achieve an accuracy of 73.9% with DeepFP

Thank you :)

Loss function : Cross entropy

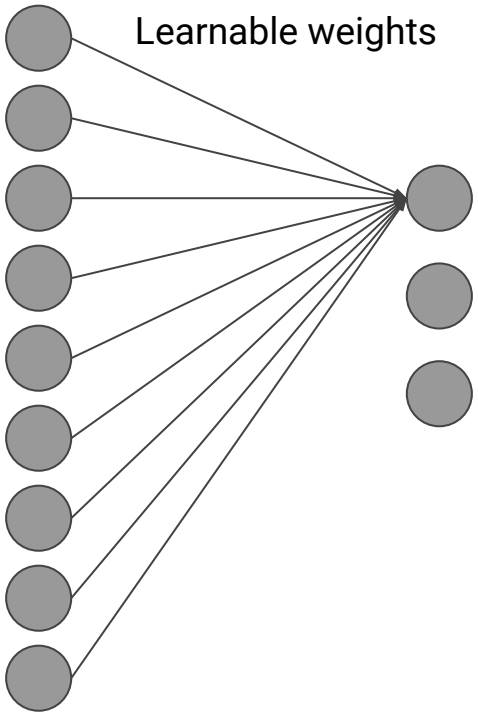
$$-t_n \log \tau(G, n) - (1 - t_n) \log(1 - \tau(G, n))$$

Cross-entropy loss increases as the predicted score diverges from the actual target label.


Accuracy of DeepFP and DeepFP_4
deepFP
deepFP4
accuracy
epoch
0.78
0.76
0.74
0.72
0.70
0.68
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Layer 1: Init Feed Forward

Node features



x_v : input feature vector

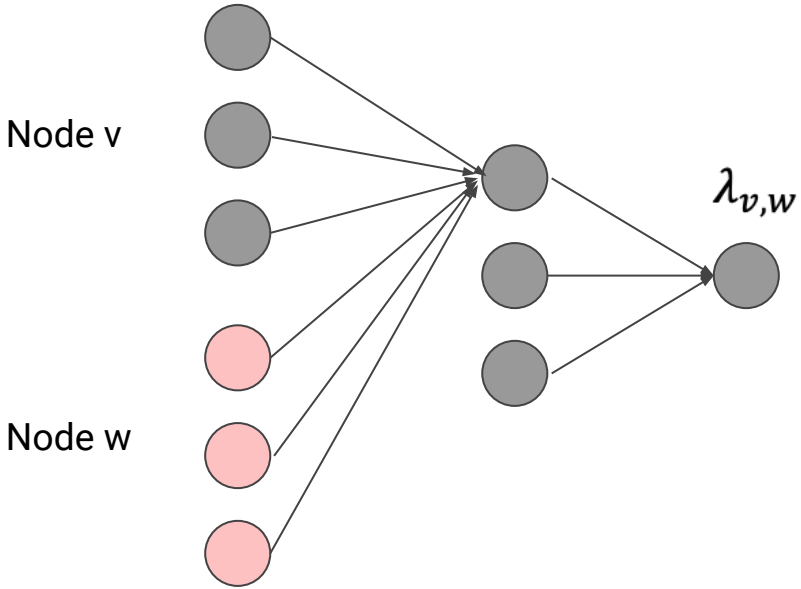
h_v : output vector

$$h_v^{(0)} = \sigma(W_1 x_v + b_1)$$

Edge attention

Goal: assign an importance score to each neighbor node

Let node w a neighbor of v



$$\lambda_{v,w} = \sigma(W_2(W_1 h_v + b_1) + b_2)$$

Gated Recurrent Unit (GRU):

Inputs: old state $h_v^{(t)}$, message $m_v^{(t+1)}$

Step 1: compute reset gate

$$r = \sigma(m_v^{(t+1)} W_{rm} + h_v^{(t)} W_{rh} + b_r)$$

Step 2: compute update gate

$$z = \sigma(m_v^{(t+1)} W_{zm} + h_v^{(t)} W_{zh} + b_z)$$

Step 3: compute candidate hidden state

$$\widetilde{h_v} = \tanh(m_v^{(t+1)} W_{hm} + (r \odot h_v^{(t)}) W_{hh} + b_h)$$

Step 4: output the new value

$$h_v^{(t+1)} = z \odot h_v^{(t)} + (1 - z) \odot \tilde{h}_v$$

Layer 2: Memory unit + Edge attention

Step 1: Assign weights to each neighbor (Edge attention layer) $\lambda_{v,w}$

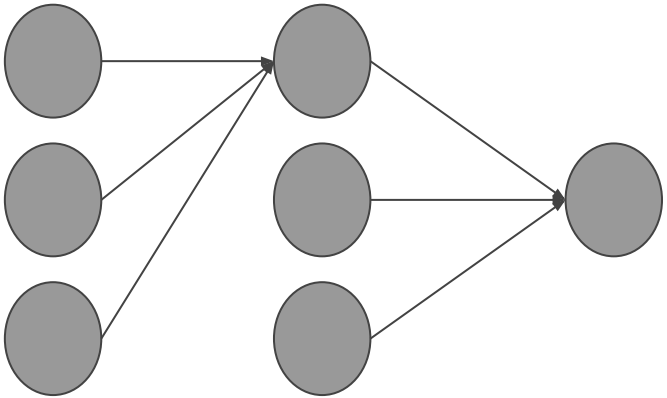
Step 2: Compute a message from nodes in the neighborhood $m_v = \sum_{w \in N(v)} \lambda_{v,w} \, h_w$

Step 3: update the current feature vector of the node $h_v = GRU(h_v, m_v)$

Step 4: Go to step 1

Repeat for fixed number T of iterations

Layer 3: Output



$$o_v = \sigma(W_{o2}(W_{o1}h_v^T + b_{o1}) + b_{o2})$$

$$o_v \in [0,1]$$

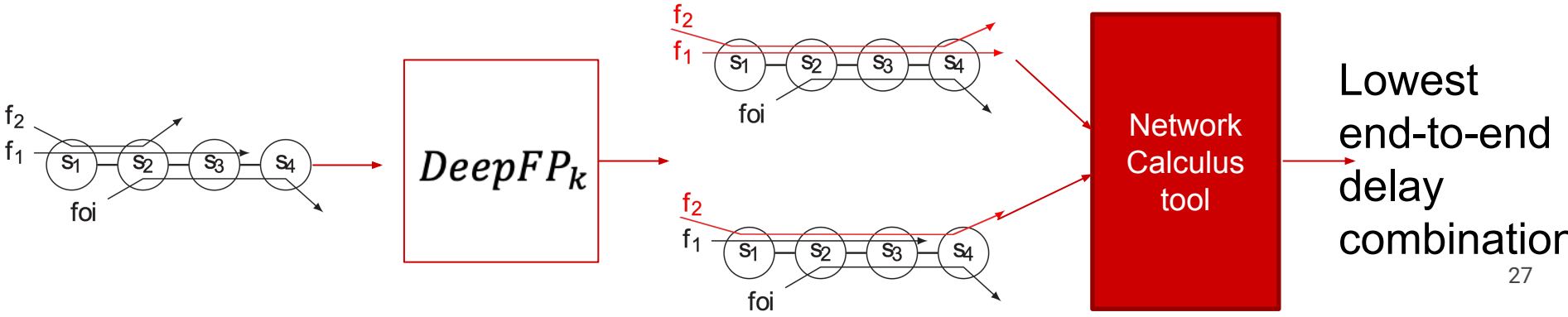
$$t_v: target\ of\ node\ v \in \{0,1\}$$

Loss of prolongation nodes in a graph G :

$$\sum_{v \in P(G)} -t_v \log(o_v) - (1 - t_v)\log(1 - o_v)$$

Backpropagate the loss and update all the weights

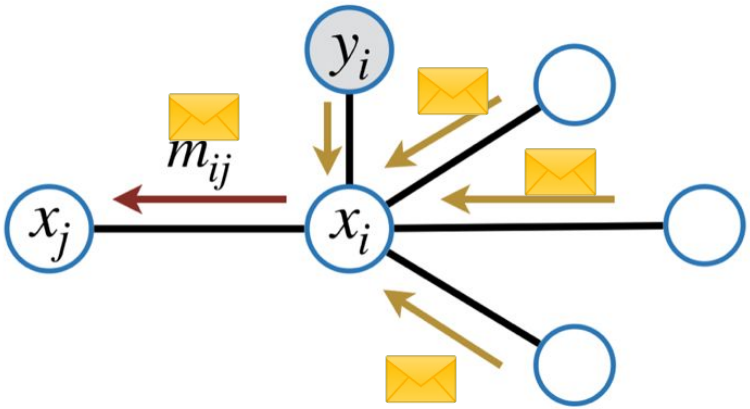
Current model and Future work



Graph neural network: Message Passing

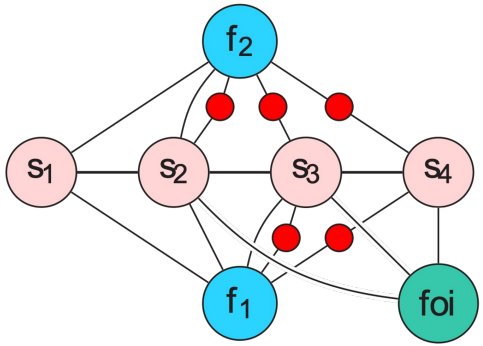
Neural Networks (NNs) to compute nodes' representation for graph-structured data

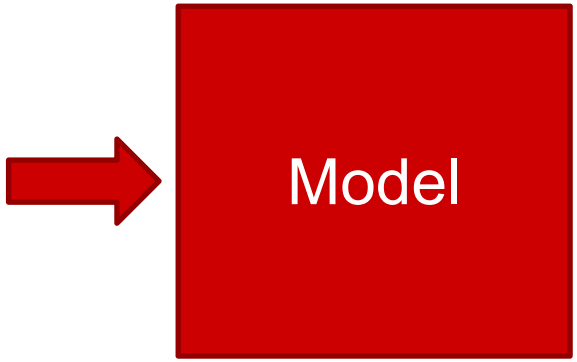
A node learns its representation by talking with its **local neighborhood**

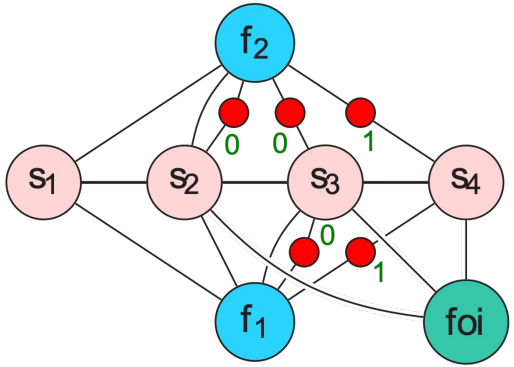


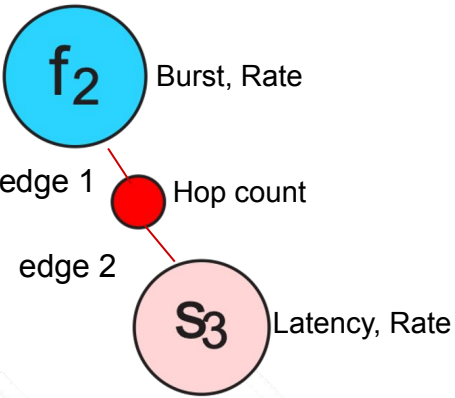
http://www.annamgrim.com/research.html

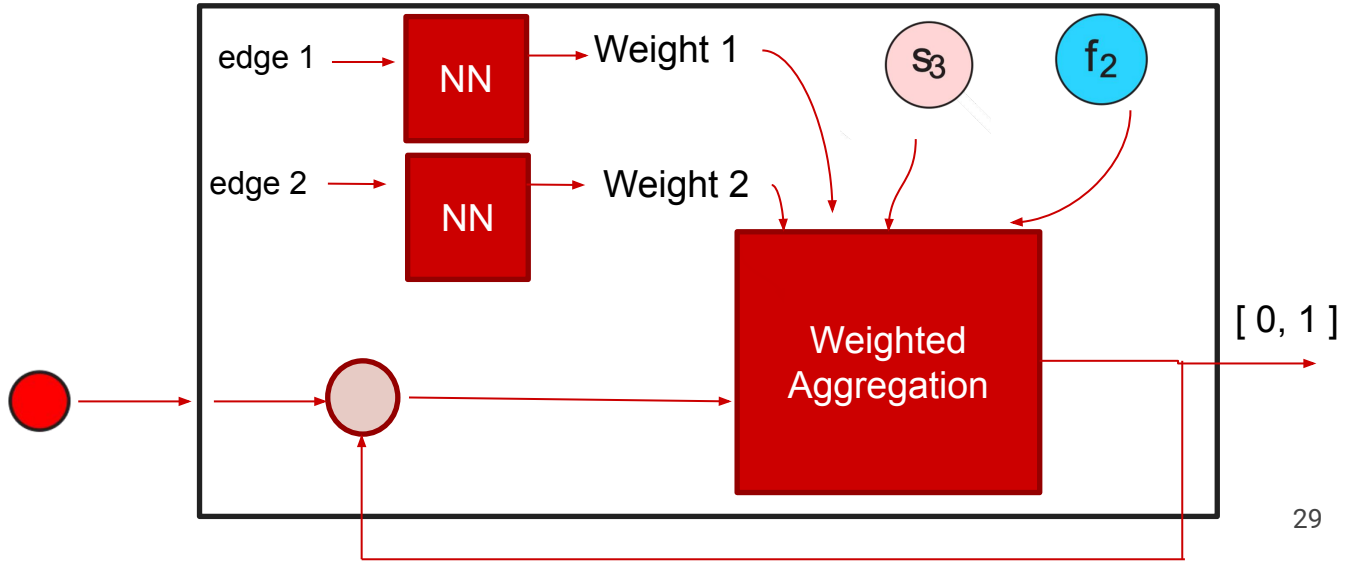
Learning on nodes



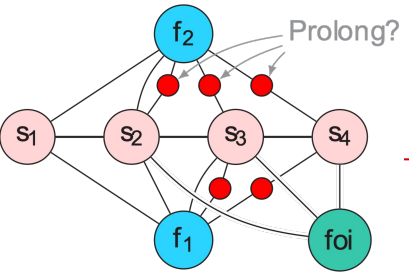


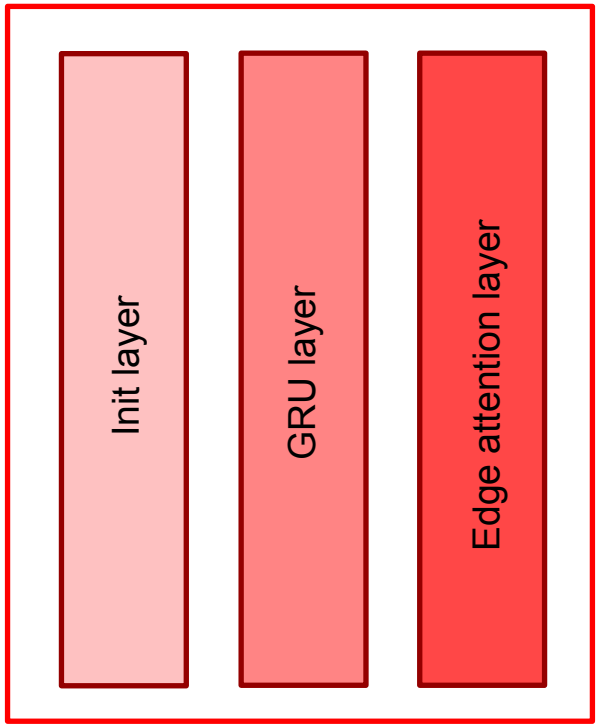






Our model: DeepFP





Deep learning:
learn node representations

Classification

Node classification:
foi node

Node classification:
Prolongation nodes

Our solution

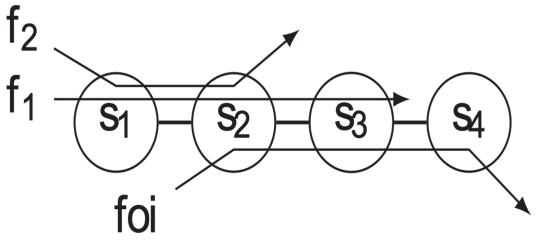
High level goal:

Implementing a solution to an optimization problem
with a comparable performance to exhaustive search based methods (using neural networks)

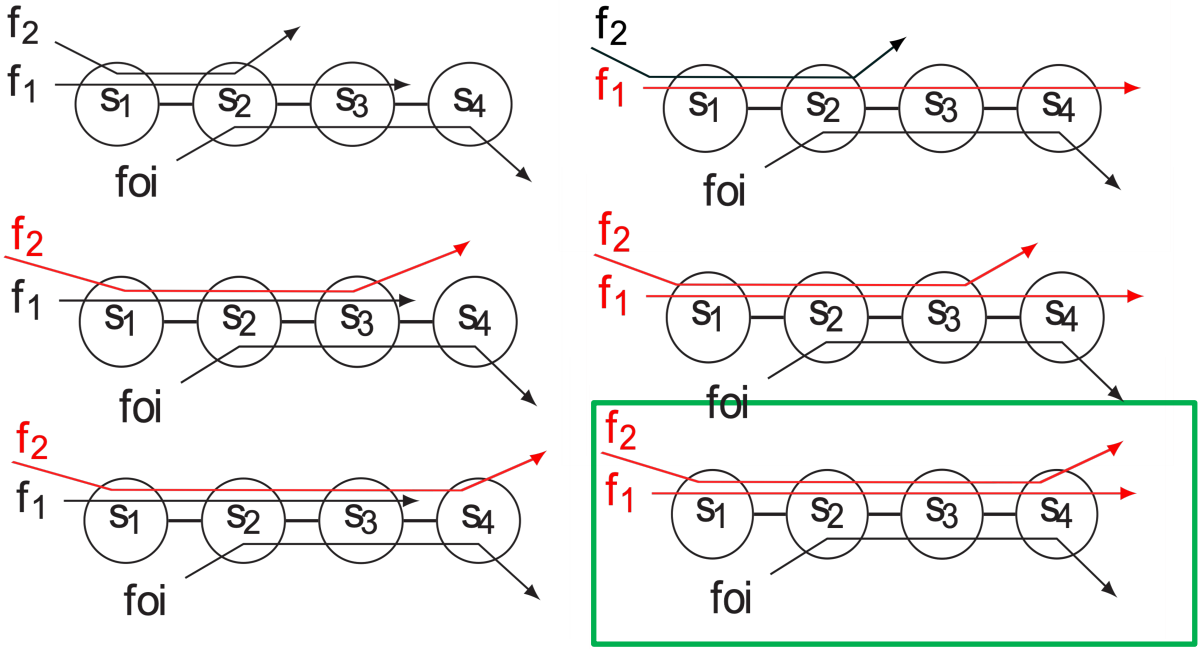
DeepFP:

- Predict, in a given scenario, if flow prolongation is beneficial
- Predict the best flow prolongations

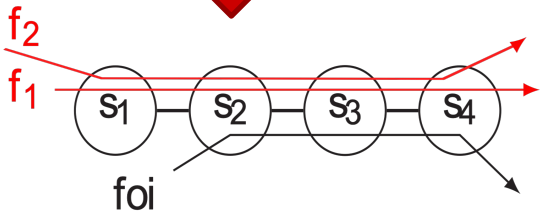
DeepFP



Six possible alternatives



DeepFP



Learning objective

Suppose we would like to learn a function f on the graph G

$$f(X,A) = \begin{pmatrix} f(x_1,A) \\ f(x_2,A) \\ . \\ . \\ . \\ f(x_N,A) \end{pmatrix} = \begin{pmatrix} s_1 \\ s_2 \\ . \\ . \\ . \\ s_N \end{pmatrix}$$

N : the number of nodes of G
A : adjacency matrix of G
s_i: score of node i, $s_i \in [0,1]$

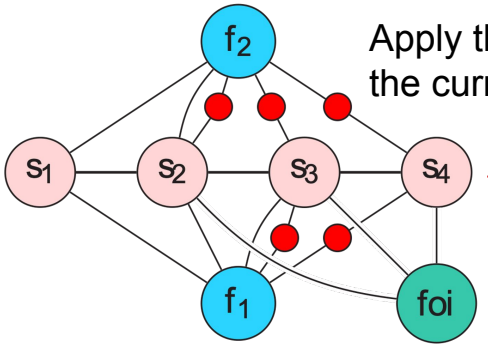
Out of N nodes, P are prolongation nodes

The objective is to learn f for the P prolongation nodes and the foi node

Scores of other nodes are not relevant: not measured in the error

Training

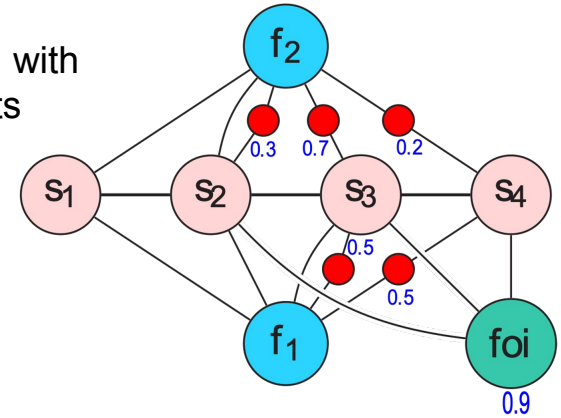
For each graph in the training dataset



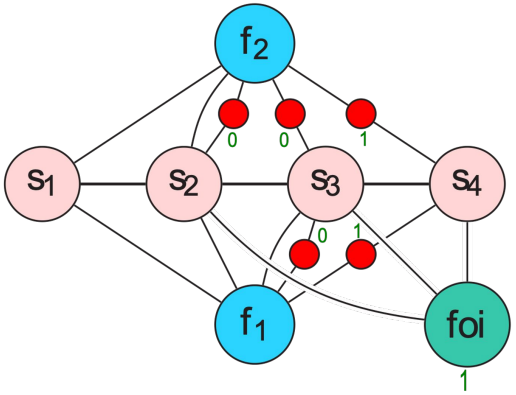
Apply the function with the current weights

(1)

Output



Target



Compare (compute loss)

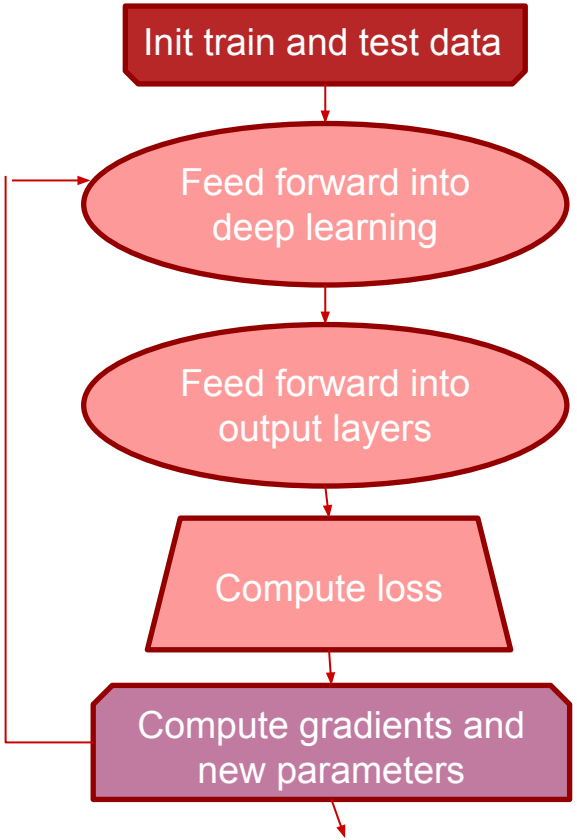
(2)

$$\begin{bmatrix} 0.3 \\ 0.7 \\ 0.2 \\ 0.5 \\ 0.5 \\ 0.9 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

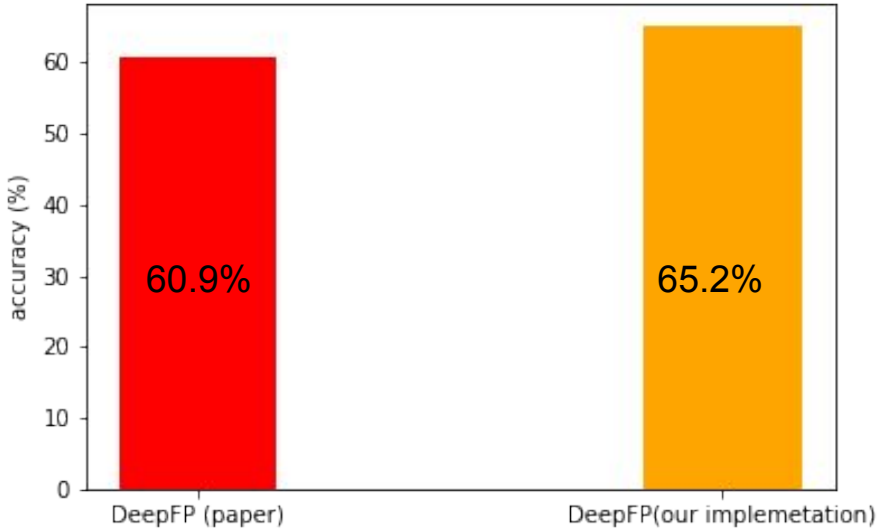
(3)

Update weights

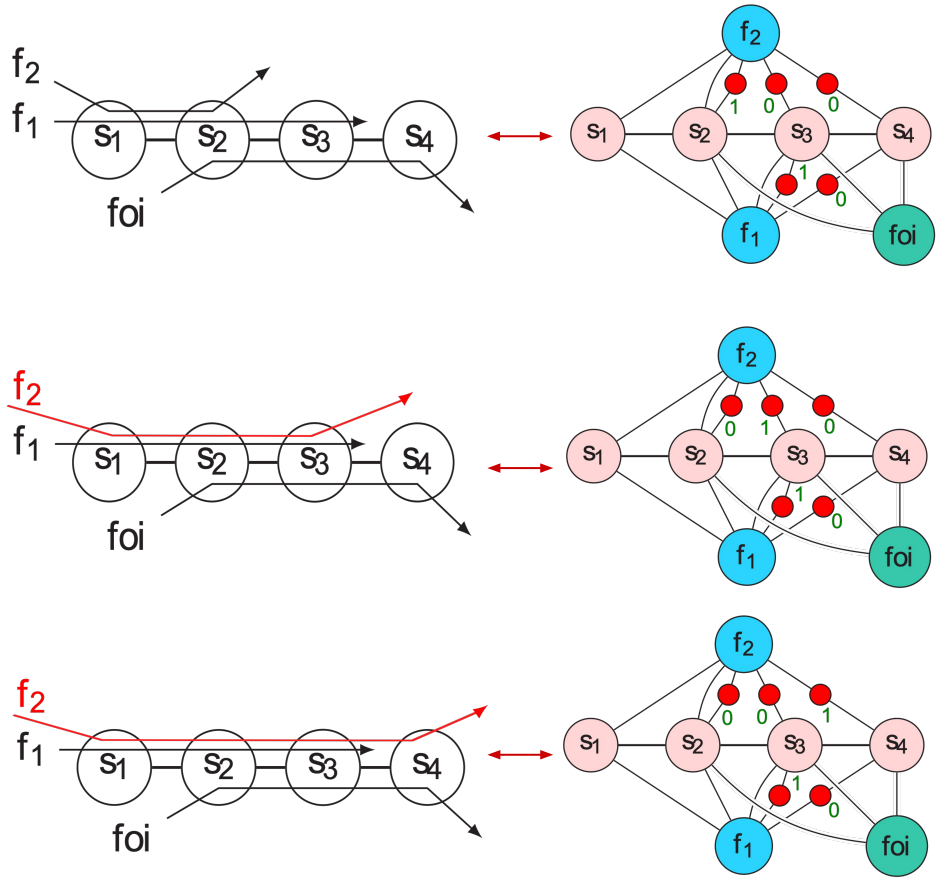
Training and Results

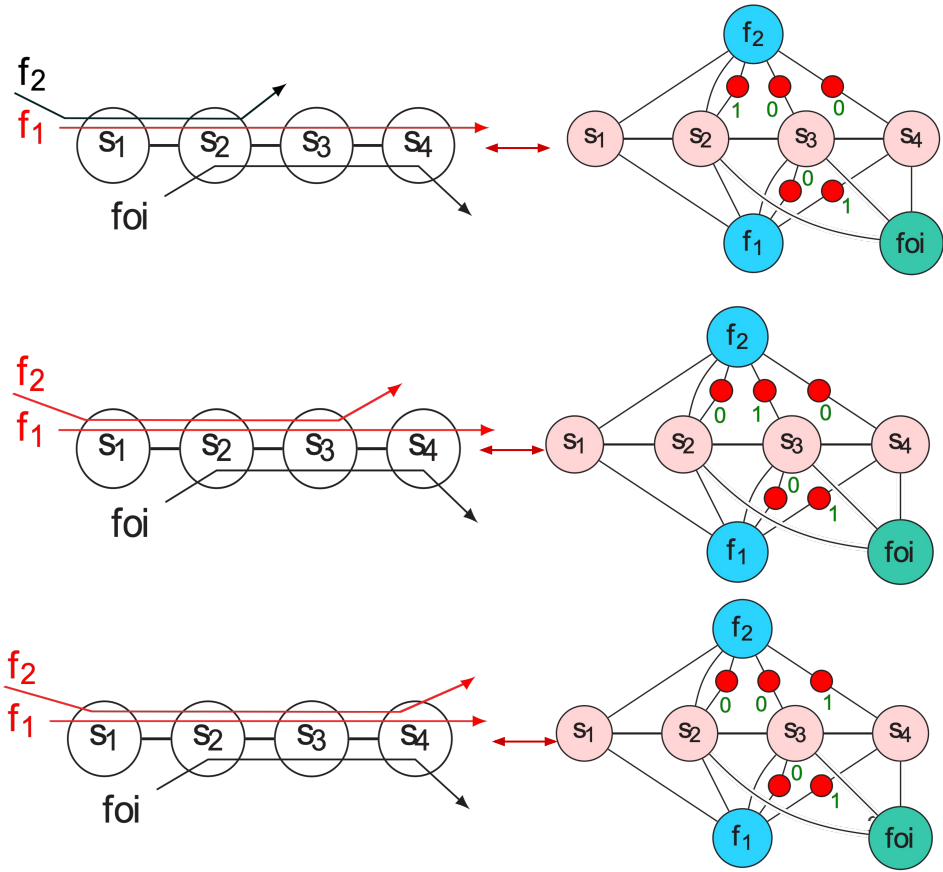


Compute metrics on test : **accuracy**



Mapping from networks to graphs





DeepFP

Init: $h_v^{(0)} = \sigma(W_1 x_v + b_1)$