The Graph Neural Network Model

PRESENTED BY ANTHONY BOYKO

Why Graphs?

Graphs can be used to mimic many different problem domains:

- Proteomics, image analysis, scene description, software engineering, NLP, etc..

How are Graphs Structured?

In their simplest form it includes single nodes.

However more complex graph structures include trees, acyclic graphs, or cyclic graphs

Machine Learning with Graphs (Before GNNs)

Goal is to learn some function $f(G,n)\epsilon R^m$

Applications split into two domains:

1. **Graph-focused** where the function f is independent of the node *n* and uses a classifier (or regressor) on a graph structured data set.
2. **Node-focused** where the function f is dependent of the node *n* the classification depends on the properties of each node.

Example Applications of Graphs

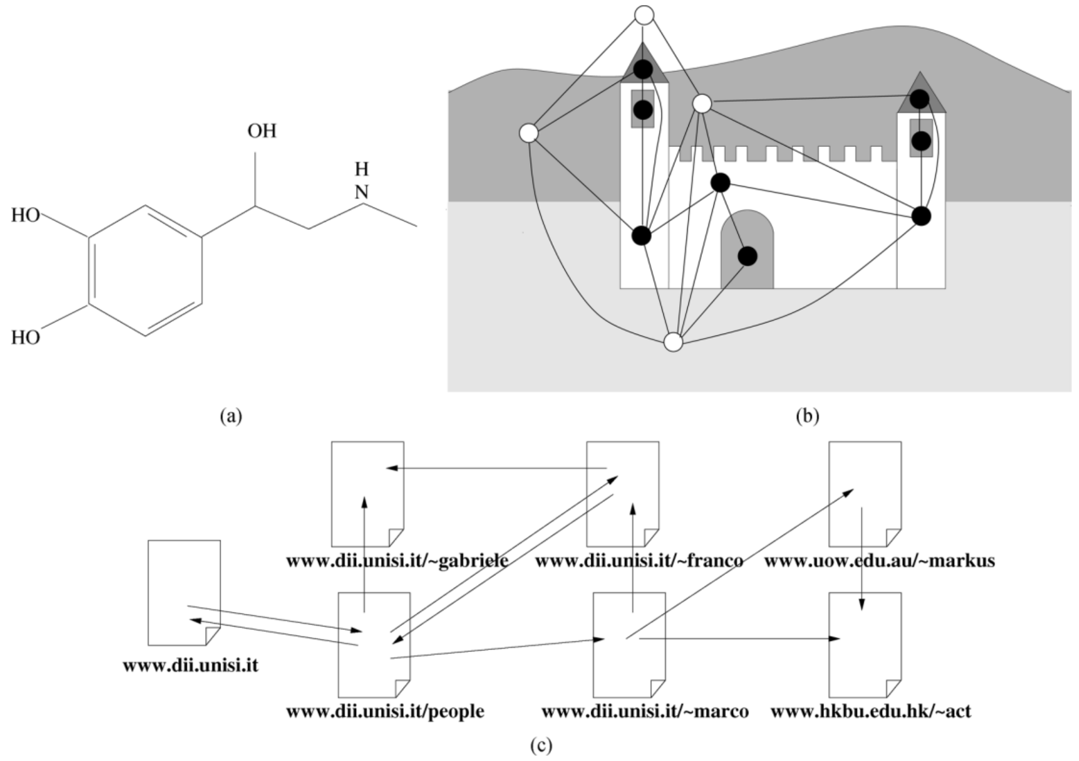
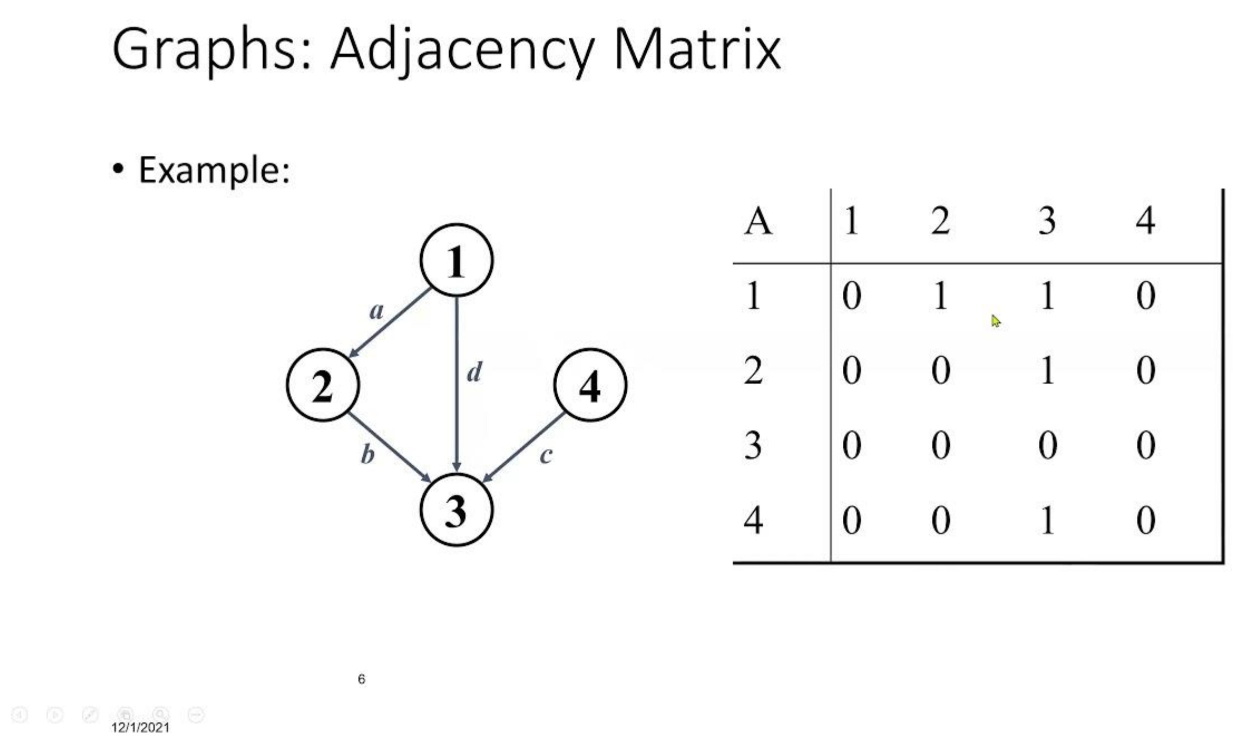


Fig. 1. Some applications where the information is represented by graphs: (a) a chemical compound (adrenaline), (b) an image, and (c) a subset of the web.

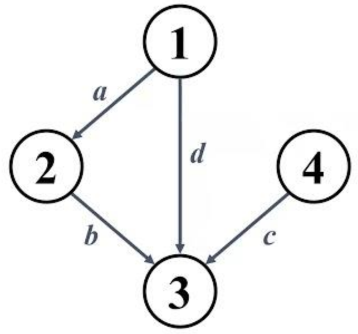
Traditional steps

Preprocess the input graph into a vector of reals (embedded the input)



Graphs: Adjacency Matrix

- Example:



A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

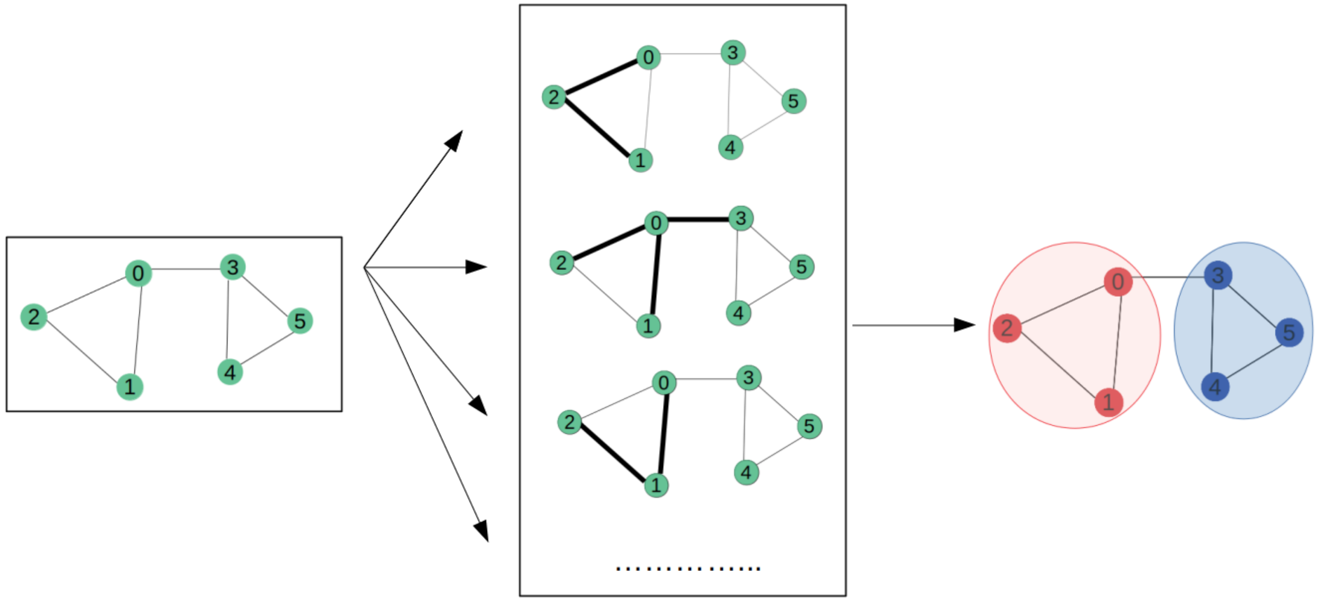
6

12/1/2021

Traditional steps

Run a regression or classification, etc…

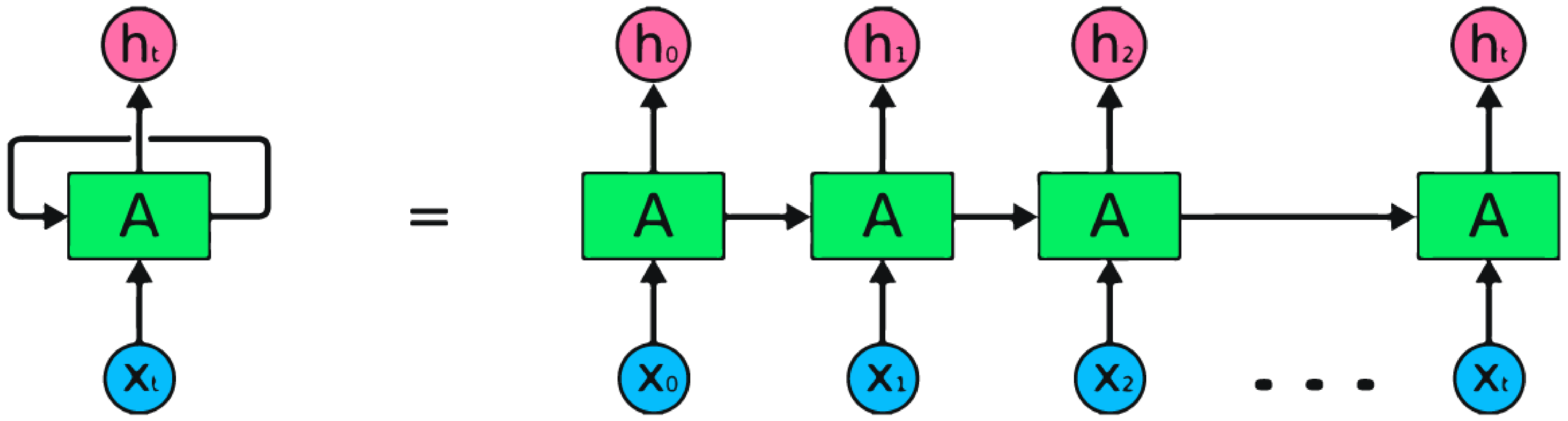
- Note we may lose important information like topological dependency of information on each node
- RNNs and Markov chains have been used a way to address the potential loss of graph structured information



Where GNN’s extended RNN’s

GNN’s can handle a more general class of graphs such as cyclic, directed, and undirected graphs.

Can be applied to node-focused applications without the additional preprocessing step



Where GNN’s extend random walks

GNN’s add a learning algorithm to the deterministic nature of a random walk

GNN’s can be applied to a larger class of processes

GNN Model notation

Graph $\mathrm{G} = (N, E)$ consists of a set of nodes N and a set of edges E

Each node n has a set of neighbors $ne[n]$ (nodes connected to n)

Each node n has a set edges denoted $co[n]$ (edges for node n)

Labels l_n or $l_{(n1,n2)}$ can be used to assign information to nodes or edges respectively. (Think class of the node, color, etc…)

Learning Set

Let's say our input isn’t just a single graph but rather a set of graphs then

Let $G_i = (N_i, E_i) \in \overline{G}$ be such that each G_i is a set of nodes and edges within the set of graphs $\overline{G}$

Learning Set

Let T be a set of pairs $\{(n_{i,j}, t_{i,j}) | n_{i,j} \in N_i, t_{i,j} \in R^m, 1 \leq j \leq q$

Where $n_{i,j}$ represents the jth node from the ith graph

Where $t_{i,j}$ represents the associated target we are trying to learn

Learning Set

We define the learning set then as $L = (G, T)$

Or $L = \{(G_i, n_{i,j}, t_{i,j})|, G_i = (N_i, E_i) \in \overline{G}, n_{i,j} \in N_i, t_{i,j} \in R^m, 1 \leq i \leq p, 1 \leq j \leq q_i$

Output functions

Let $x_n = f_w\left(l_n, l_{co[n]}, x_{ne[n]}, l_{ne[n]}\right)$ be our local transition function for state x_n

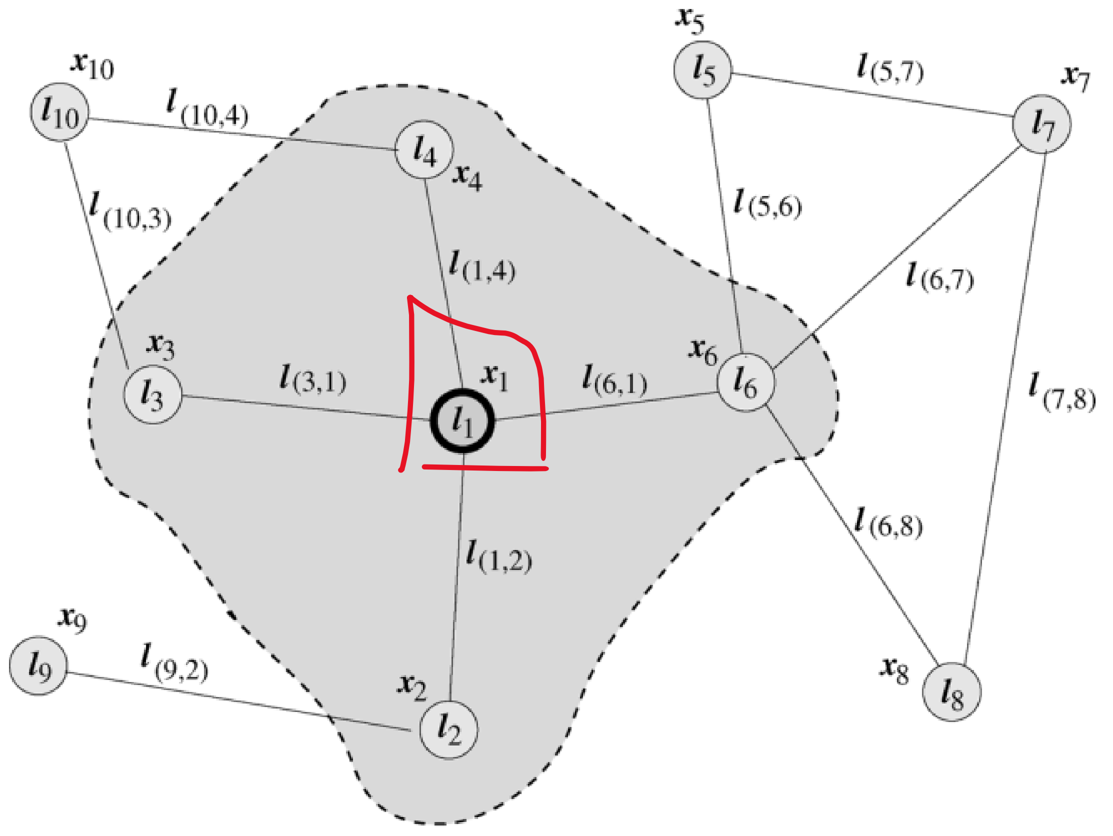
Let $o_n = g_w(x_n, l_n)$ be our local output function for the nth node

The ith State

Here is an example of a 10-node graph being represented in the GNN notation

Also how to calculate the state for x_1

Include graphics to highlight parts of the image



$$\underline{x_1} = f_w(\underline{l_1}, \underbrace{l_{(1,2)}, l_{(3,1)}, l_{(1,4)}, l_{(6,1)}}_{l_{co[1]}}, \underbrace{x_2, x_3, x_4, x_6}_{x_{ne[1]}}, \underbrace{l_2, l_3, l_4, l_6}_{l_{ne[n]}})$$

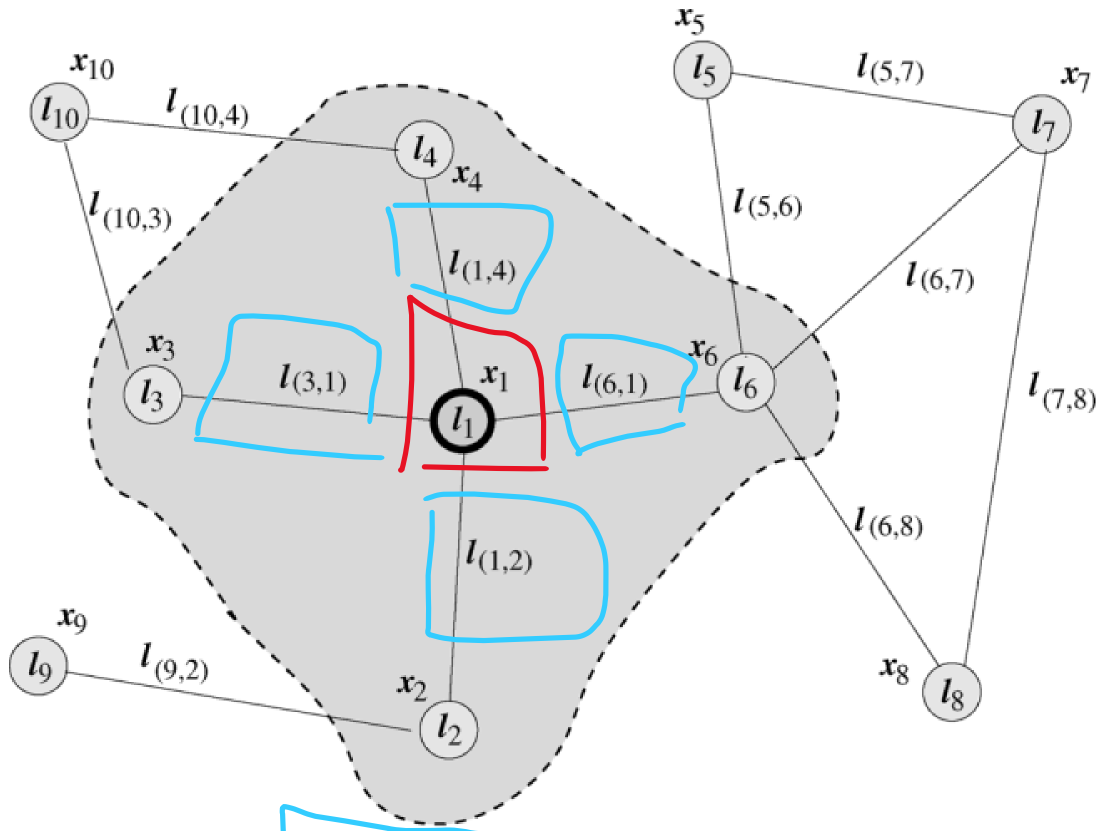
Fig. 2. Graph and the neighborhood of a node. The state $\boldsymbol{x}_1$ of the node 1 depends on the information contained in its neighborhood.

The ith State

Here is an example of a 10-node graph being represented in the GNN notation

Also how to calculate the state for x_1

Include graphics to highlight parts of the image



$$\underline{x_1} = f_w(\underline{l_1}, \underbrace{\overbrace{l_{(1,2)}, l_{(3,1)}, l_{(1,4)}, l_{(6,1)}}}_{l_{co[1]}}, \underbrace{x_2, x_3, x_4, x_6}_{x_{ne[1]}}, \underbrace{l_2, l_3, l_4, l_6}_{l_{ne[n]}})$$

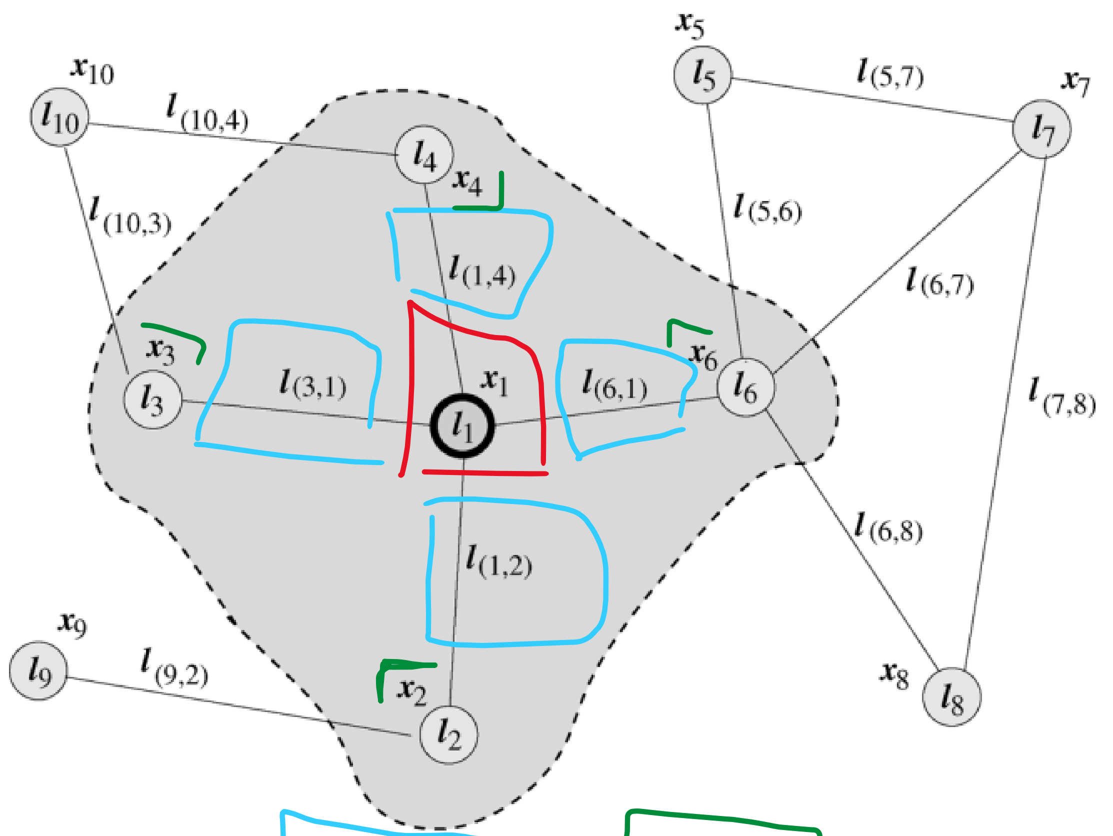
Fig. 2. Graph and the neighborhood of a node. The state $\boldsymbol{x}_1$ of the node 1 depends on the information contained in its neighborhood.

The ith State

Here is an example of a 10-node graph being represented in the GNN notation

Also how to calculate the state for x_1

Include graphics to highlight parts of the image



$$x_1 = f_w(l_1, \underbrace{l_{(1,2)}, l_{(3,1)}, l_{(1,4)}, l_{(6,1)}}_{l_{co[1]}}, \underbrace{x_2, x_3, x_4, x_6}_{x_{ne[1]}}, \underbrace{l_2, l_3, l_4, l_6}_{l_{ne[n]}})$$

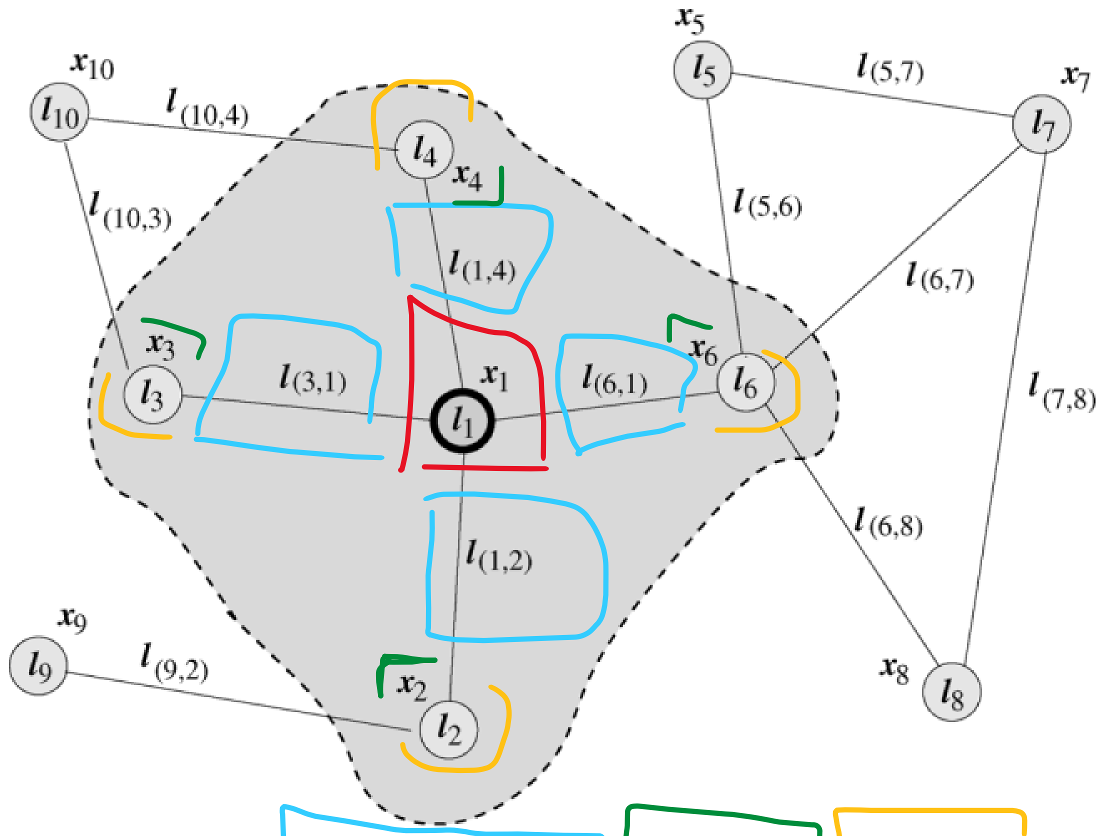
Fig. 2. Graph and the neighborhood of a node. The state $\boldsymbol{x}_1$ of the node 1 depends on the information contained in its neighborhood.

The ith State

Here is an example of a 10-node graph being represented in the GNN notation

Also how to calculate the state for x_1

Include graphics to highlight parts of the image



$$x_1 = f_w(l_1, \underbrace{l_{(1,2)}, l_{(3,1)}, l_{(1,4)}, l_{(6,1)}}_{l_{co[1]}}, \underbrace{x_2, x_3, x_4, x_6}_{x_{ne[1]}}, \underbrace{l_2, l_3, l_4, l_6}_{l_{ne[n]}})$$

Fig. 2. Graph and the neighborhood of a node. The state x_1 of the node 1 depends on the information contained in its neighborhood.

Remarks

1. We can adjust f_w our needs.
   - I.e. remove $l_{ne[n]}$ when calculating the current state because that is already captured by including $x_{ne[n]}$.
   - Or including a neighbour multiple links away
2. ALL equations used focus on use with undirected graphs.
   - Add a variable representing direction to f_w for each arc to handle the directed case

Output functions condensed

We can rewrite the output functions as follows:

1. $x = F_w(x, l)$
2. $o = G_w(x, l_N)$

(Everything is converted from node level to graph level)

Output Map

Let φ_w be a map of $\overline{G} \times \mathrm{N}$ size.

Each element in this maps index will represent a node.

Whose value is the output the models output for that node.

Positional

Positional graphs require the transition function f_w **must** include the positions of the neighbors as input.

- Implementation is simplified if $l_{co[n]}, x_{ne[n]}, l_{ne[n]}$ is sorted according to neighbors' position
- Also spots with no neighbours are padded by a special null value (learning is done by gradient descent so we can't use 0)

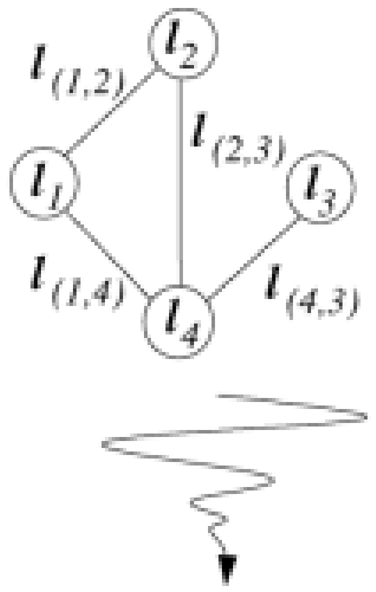
Non-Positional

Non-position graphs we can replace f_w , with:

$$x_n = \sum_{u \in ne[n]} h_w(l_n, l_{(n,u)}, x_u, l_u)$$

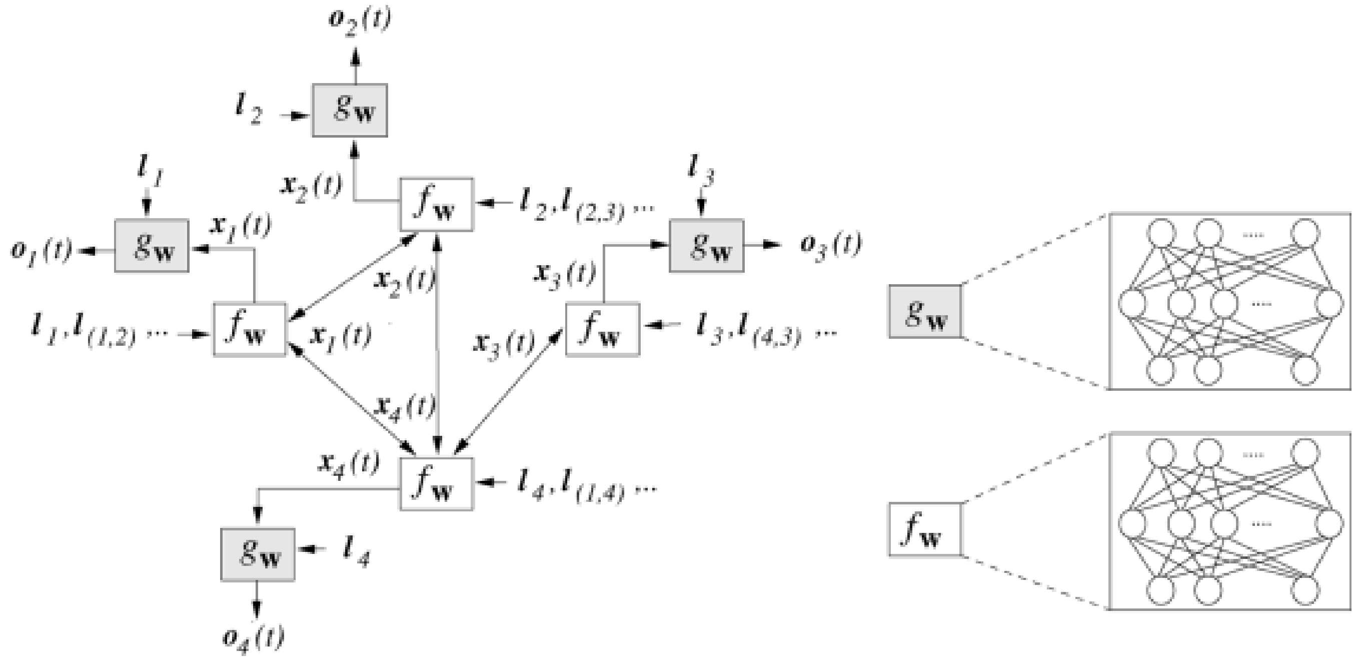
Example of inputs to GNN

First is the representation of the labels in a graph



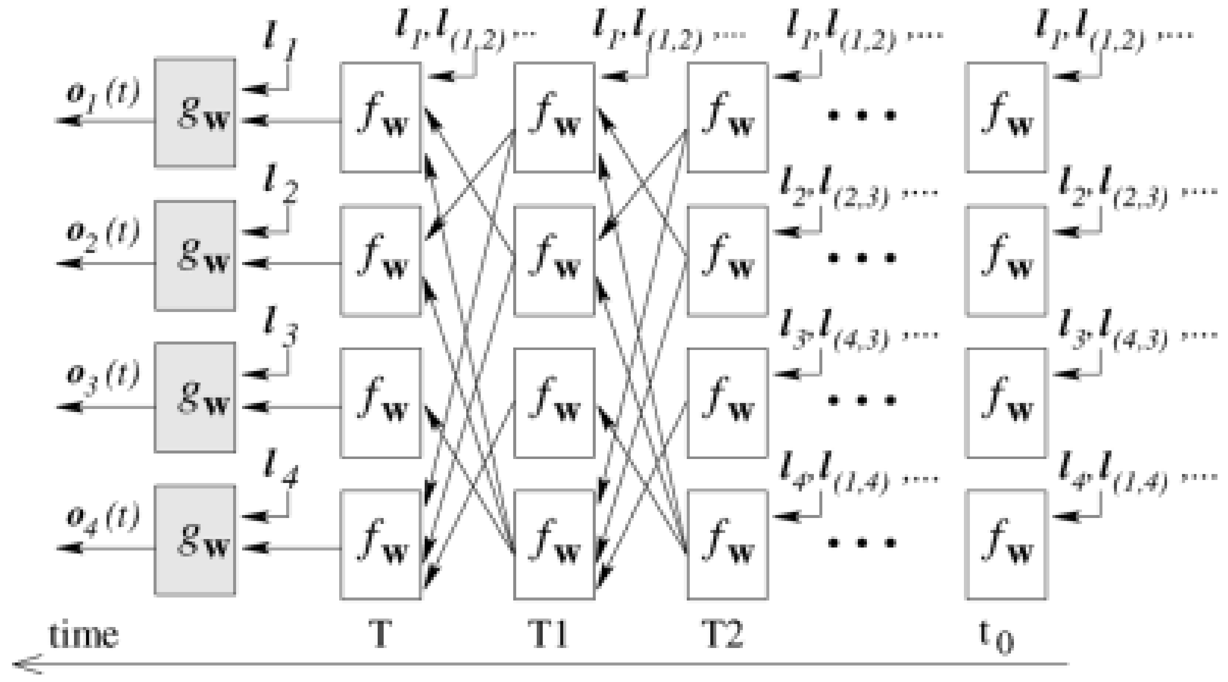
Example of inputs to GNN

Next we add states and output functions (encoding network)



Example of inputs to GNN

Network obtained from unfolding the encoding network



Update Step

iterative scheme (global level):

$$x(t+1) = F_w(x(t), l)$$

Which can be rewritten as (local level):

$$x_n(t+1) = f_w\left(l_n, l_{co[n]}, x_{ne[n]}(t), l_{ne[n]}\right)$$

$$o_n(t) = g_w(x_n(t), l_n)$$

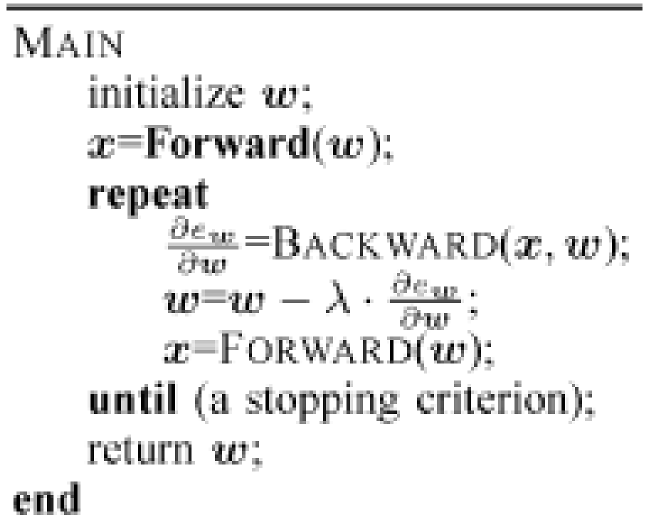
Learning Algorithm

We can pose this learning task as the minimization of a quadratic cost function:

$$e_w = \sum_{i=1}^{p} \sum_{j=1}^{q_i} (t_{i,j} - \varphi_w(G_i, n_{i,j}))^2$$

Learning Algo Main Method

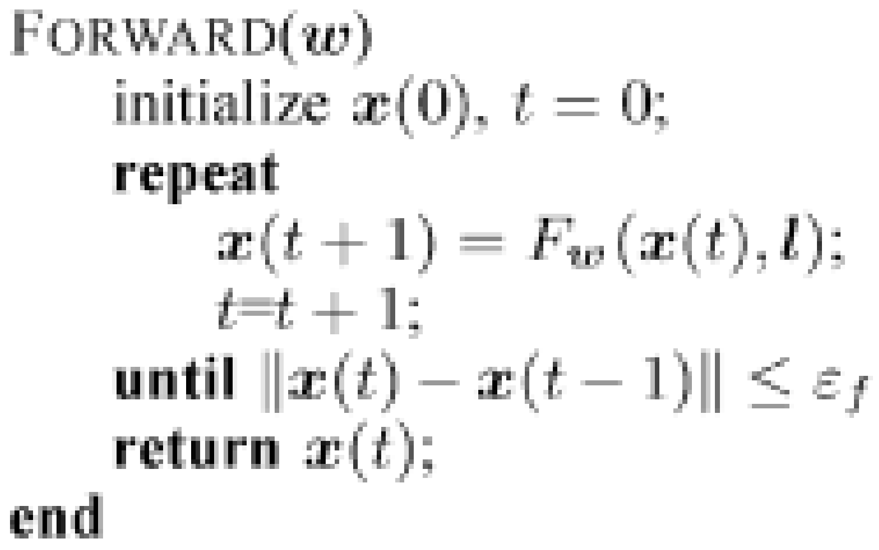
1. Update states iteratively till convergence .
2. The gradient is computed.
3. The weights are updated according to the gradient computed in step 2
4. Repeat till stopping criteria i



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

Learning Algo Forward Method

1. Initialize initial state
2. Calculate the next time steps state
3. Repeat until the difference between timesteps is below some threshold
4. Return last iterations state



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

Results

The Subgraph Matching Problem

The Mutagenesis Problem

Web Page Ranking

The Subgraph Matching Problem

The sub matching problem is where the goal is to see if a particular node belong to a subgraph within G_i

Output is either 1 or -1 depending on existence

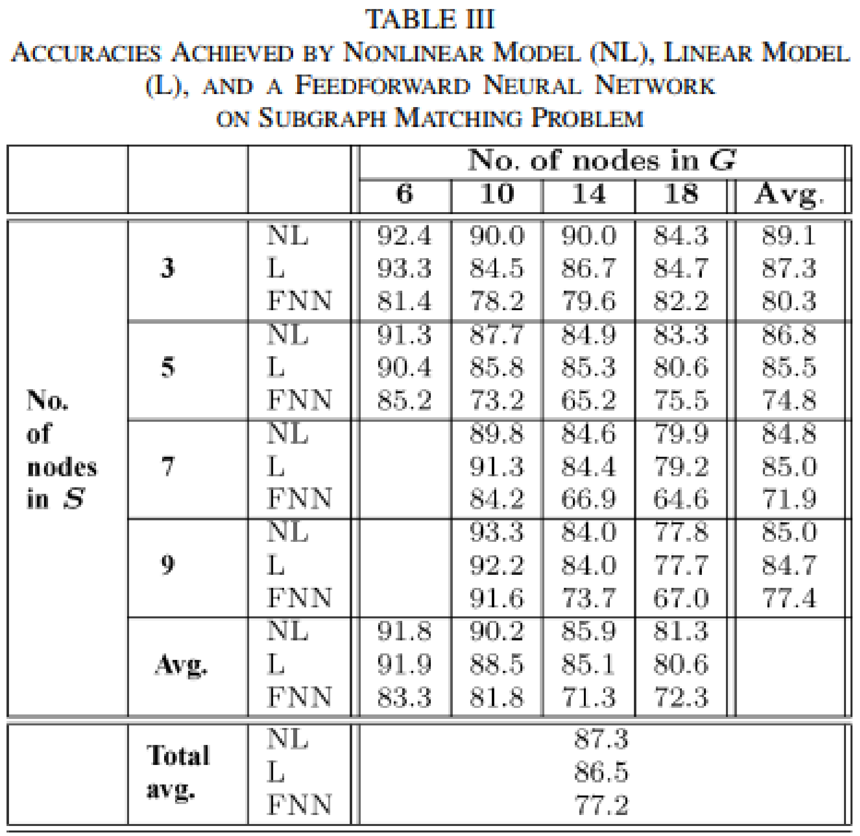


TABLE III
ACCURACIES ACHIEVED BY NONLINEAR MODEL (NL), LINEAR MODEL (L), AND A FEEDFORWARD NEURAL NETWORK ON SUBGRAPH MATCHING PROBLEM

			No. of nodes in G				
			6	10	14	18	Avg.
No. of nodes in S	3	NL	92.4	90.0	90.0	84.3	89.1
		L	93.3	84.5	86.7	84.7	87.3
		FNN	81.4	78.2	79.6	82.2	80.3
	5	NL	91.3	87.7	84.9	83.3	86.8
		L	90.4	85.8	85.3	80.6	85.5
		FNN	85.2	73.2	65.2	75.5	74.8
	7	NL		89.8	84.6	79.9	84.8
		L		91.3	84.4	79.2	85.0
		FNN		84.2	66.9	64.6	71.9
	9	NL		93.3	84.0	77.8	85.0
		L		92.2	84.0	77.7	84.7
		FNN		91.6	73.7	67.0	77.4
	Avg.	NL	91.8	90.2	85.9	81.3	
		L	91.9	88.5	85.1	80.6	
		FNN	83.3	81.8	71.3	72.3	
	Total avg.	NL	87.3				
		L	86.5				
		FNN	77.2				

The Mutagenesis Problem

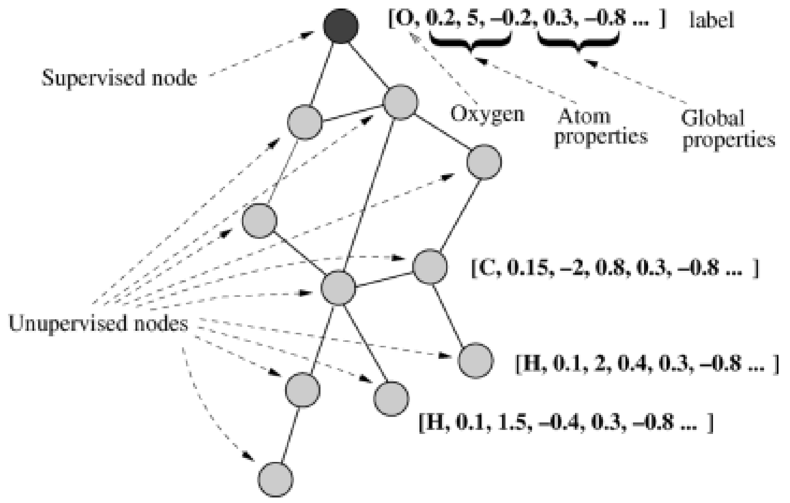

[O, 0.2, 5, –0.2, 0.3, –0.8 ...]
label
Supervised node
Oxygen
Atom properties
Global properties
[C, 0.15, –2, 0.8, 0.3, –0.8 ...]
Unupervised nodes
[H, 0.1, 2, 0.4, 0.3, –0.8 ...]
[H, 0.1, 1.5, –0.4, 0.3, –0.8 ...]

Fig. 6. Atom-bond structure of a molecule represented by a graph with labeled nodes. Nodes represent atoms and edges denote atom bonds. Only one node is supervised.

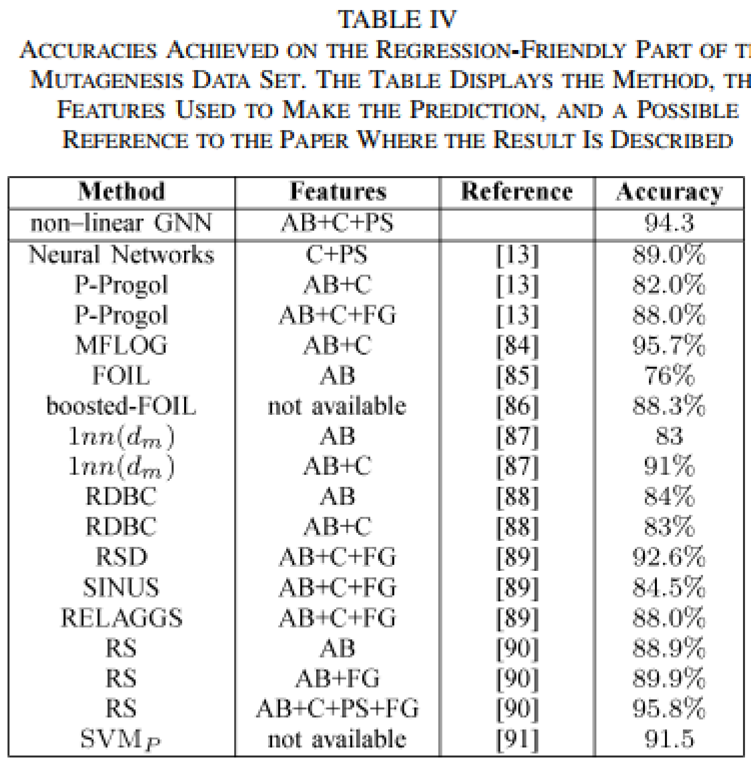


TABLE IV

Accuracies Achieved on the Regression-Friendly Part of the Mutagenesis Data Set. The Table Displays the Method, the Features Used to Make the Prediction, and a Possible Reference to the Paper Where the Result Is Described

Method	Features	Reference	Accuracy
non–linear GNN	AB+C+PS		94.3
Neural Networks	C+PS	[13]	89.0%
P-Progol	AB+C	[13]	82.0%
P-Progol	AB+C+FG	[13]	88.0%
MFLOG	AB+C	[84]	95.7%
FOIL	AB	[85]	76%
boosted-FOIL	not available	[86]	88.3%
$1nn(d_m)$	AB	[87]	83
$1nn(d_m)$	AB+C	[87]	91%
RDBC	AB	[88]	84%
RDBC	AB+C	[88]	83%
RSD	AB+C+FG	[89]	92.6%
SINUS	AB+C+FG	[89]	84.5%
RELAGGS	AB+C+FG	[89]	88.0%
RS	AB	[90]	88.9%
RS	AB+FG	[90]	89.9%
RS	AB+C+PS+FG	[90]	95.8%
SVM$_P$	not available	[91]	91.5

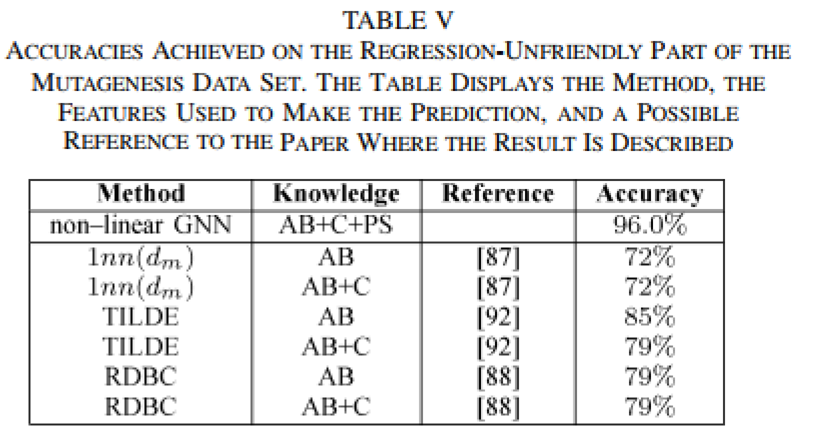


TABLE V

Accuracies Achieved on the Regression-Unfriendly Part of the Mutagenesis Data Set. The Table Displays the Method, the Features Used to Make the Prediction, and a Possible Reference to the Paper Where the Result Is Described

Method	Knowledge	Reference	Accuracy
non–linear GNN	AB+C+PS		96.0%
$1nn(d_m)$	AB	[87]	72%
$1nn(d_m)$	AB+C	[87]	72%
TILDE	AB	[92]	85%
TILDE	AB+C	[92]	79%
RDBC	AB	[88]	79%
RDBC	AB+C	[88]	79%

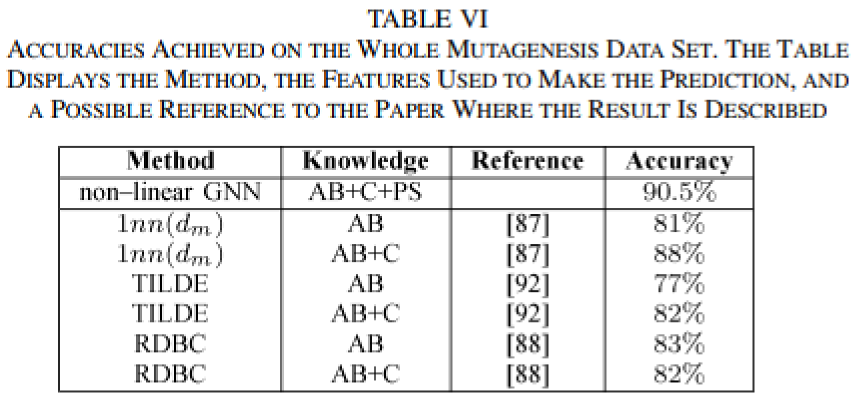


TABLE VI

Accuracies Achieved on the Whole Mutagenesis Data Set. The Table Displays the Method, the Features Used to Make the Prediction, and a Possible Reference to the Paper Where the Result Is Described

Method	Knowledge	Reference	Accuracy
non–linear GNN	AB+C+PS		90.5%
$1nn(d_m)$	AB	[87]	81%
$1nn(d_m)$	AB+C	[87]	88%
TILDE	AB	[92]	77%
TILDE	AB+C	[92]	82%
RDBC	AB	[88]	83%
RDBC	AB+C	[88]	82%

Web Page Ranking

Inspired by Google's PageRank

Wanted to see how pages relate to two given topics

Authority is a measure of the pages content

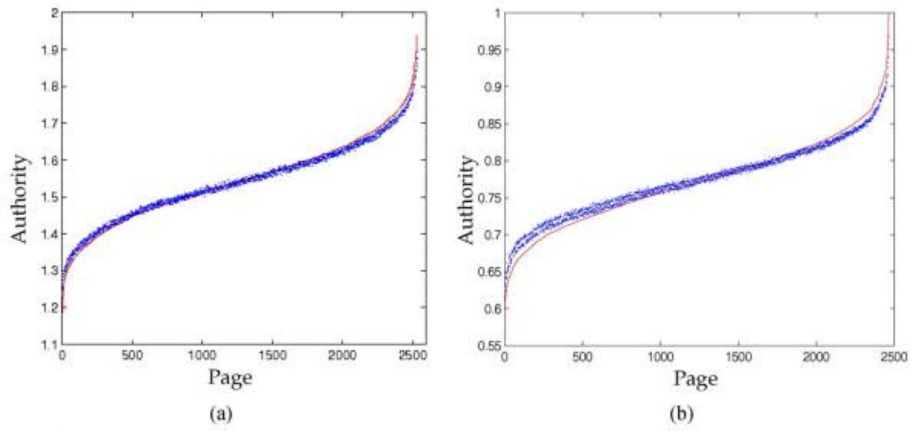


Fig. 7. Desired function τ (the continuous lines) and the output of the GNN (the dotted lines) on the pages that belong to only one topic (a) and on the other pages (b). Horizontal axis stands for pages and vertical axis stands for scores. Pages have been sorted according to the desired value $\tau(\boldsymbol{G}, n)$.

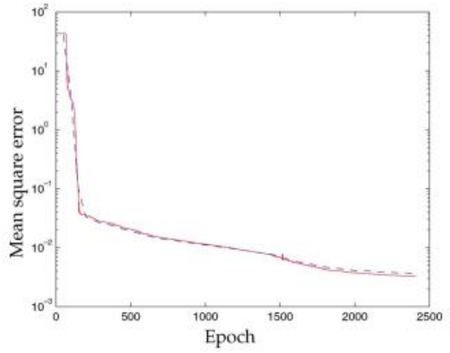


Fig. 8. Error function on the training set (continuous line) and on the validation (dashed line) set during learning phase.

Conclusion

- ❑In conclusion this paper introduces the GNN model
- ❑It is a generalized way to handle problems in the graph space
- ❑Paper used a series of different classification problems as its experiments