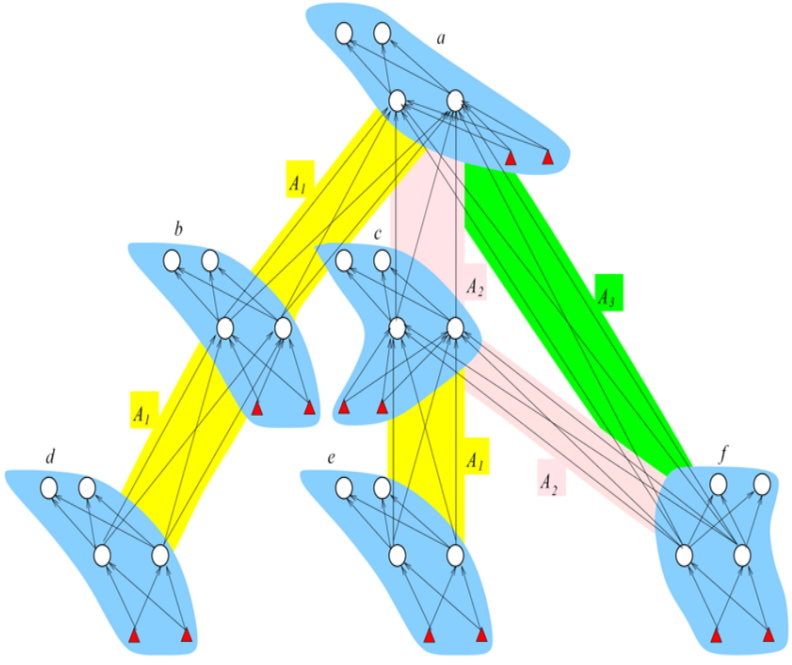
Graph Neural Networks

A constraint-based formulation [a]

[a] In collaboration with Giuseppe Marra, M. Maggini, S. Melacci and M. Gori

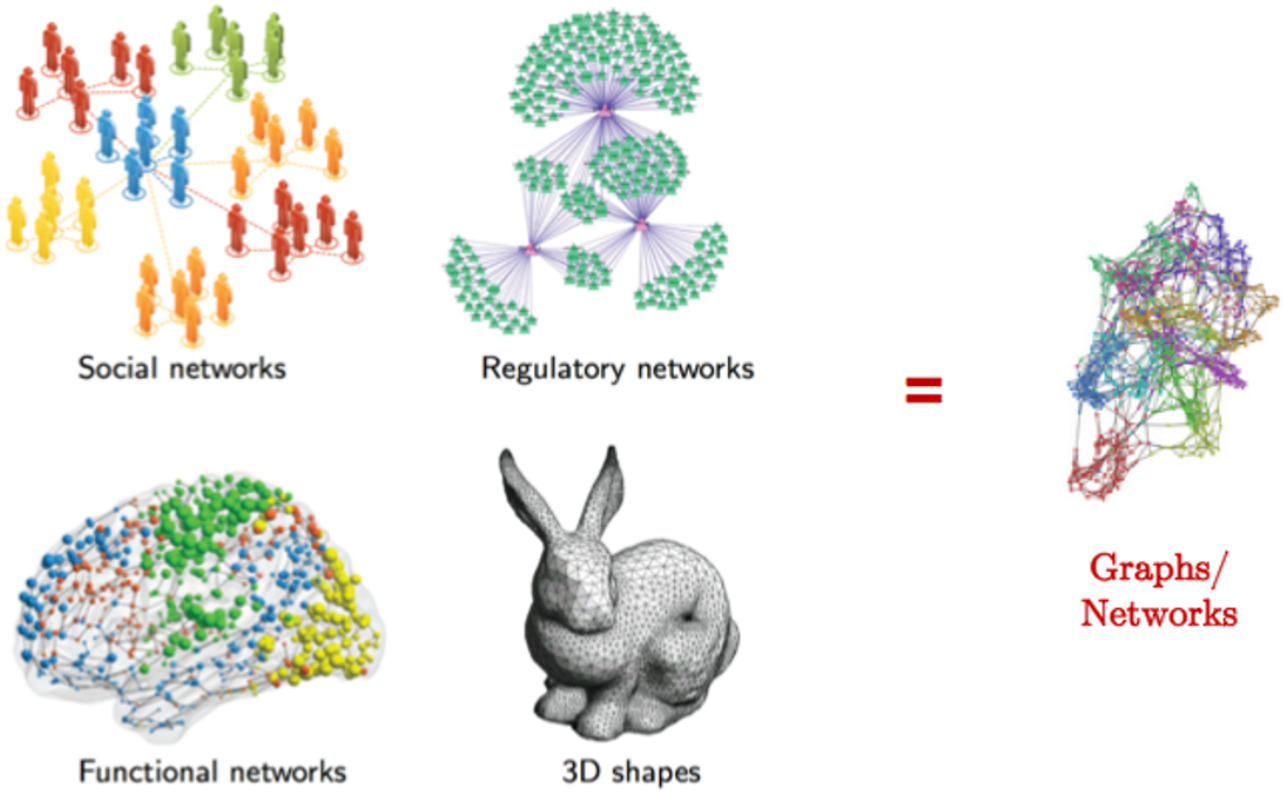
Matteo Tiezzi

DIISM, SAILab
(Siena Artificial
Intelligence Laboratory)



22 July 2019
ACDL Satellite Workshop on Graph Neural Networks

Learning in structured domains



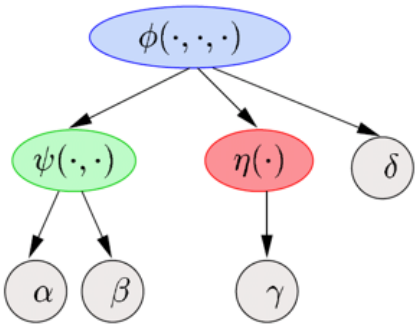
- Non-Euclidean (graph or manifold-structured) data such as social networks, molecular graphs and 3D point clouds in computer vision

Graph-focused Tasks

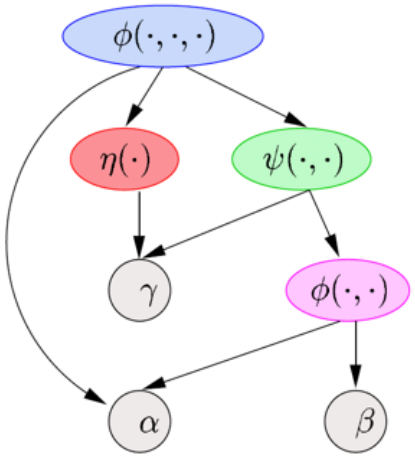
$$\tau(G)$$

On the truth of logic statements

$\phi(\psi(\alpha,\beta),\eta(\gamma),\delta)$

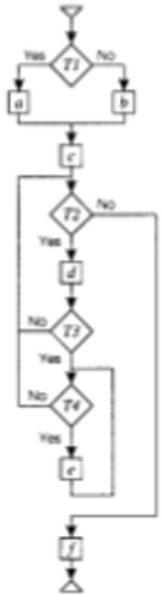


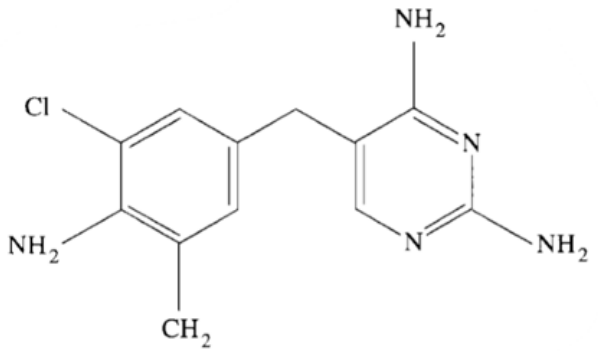
$\phi(\alpha,\eta(\gamma),\psi(\gamma,\phi(\alpha,\beta)))$



Program behavior

```
program name (list);
var
...
begin
if T1 then
        a
else
        b;
c;
while T2 do
        begin
        d;
        if T3 then
        while T4 do
                e
        end;
f
end.
```





Physicochemical behavior

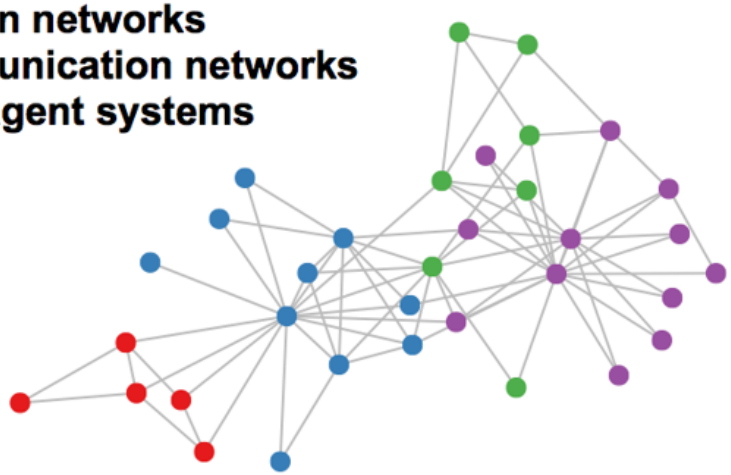
Node-focused Tasks

$$\tau(G, n)$$

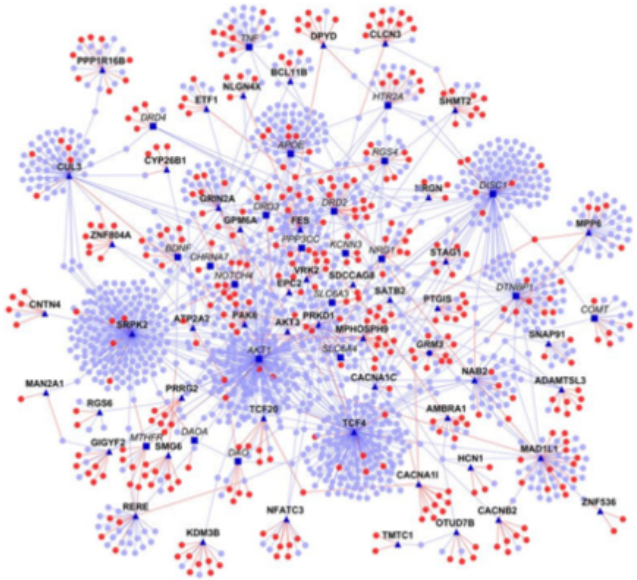
Social nets

here we need to make prediction at node level!

Social networks
Citation networks
Communication networks
Multi-agent systems



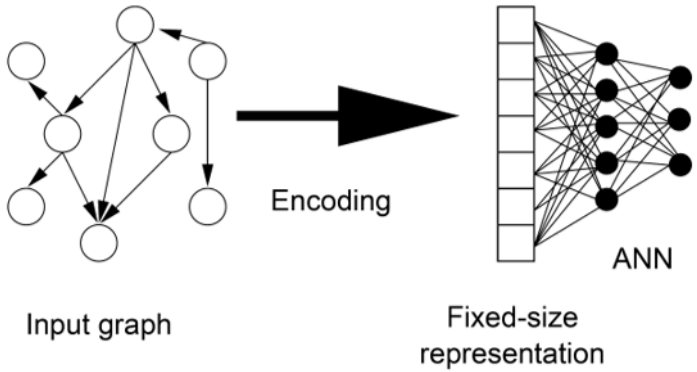
Karate club network



Protein Interaction Network

Graph Representation

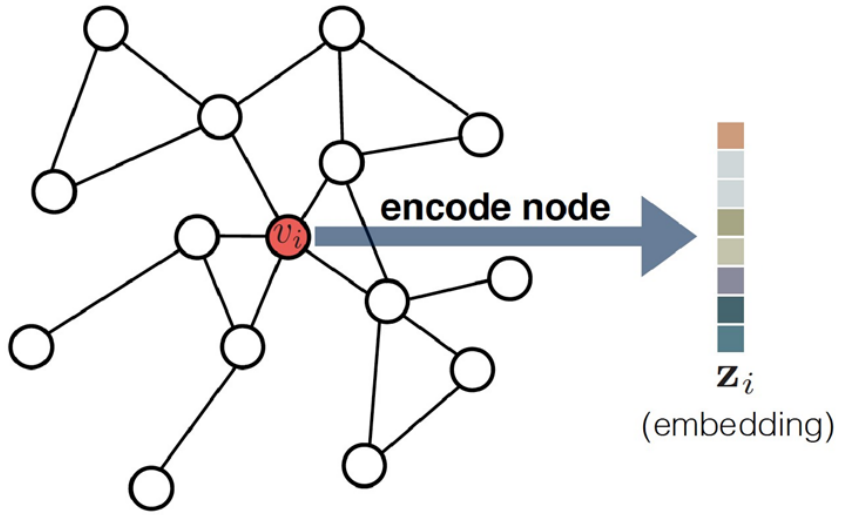
- Traditional machine learning approaches assume to deal with *flat data*
- Aim at obtaining simple representations from complex data structures, relying on summary graph statistics, kernel functions, graph traversals procedures etc.

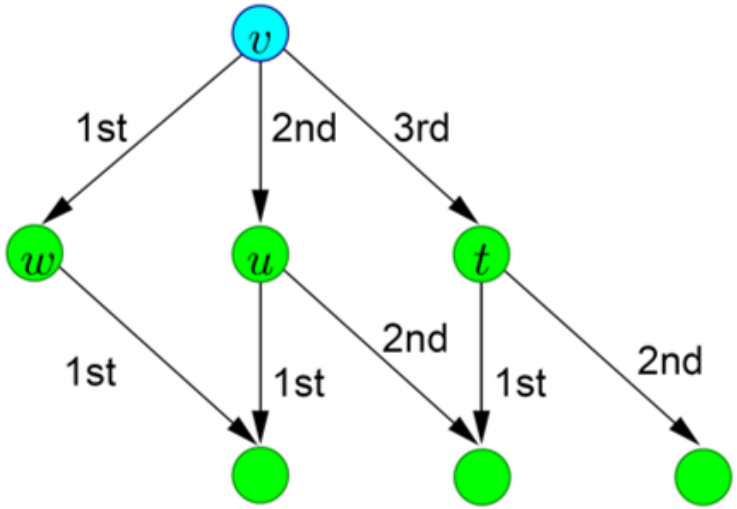


- **Pre-processing step**, using hand-engineered statistics to extract structural information into simpler encodings
- Limited approaches – **loosing useful information, not able to adapt during learning**

Embedding a graph

- Mapping a graph to a real valued vector – concatenating the features stored in each node, following an order derived from the connection topology





- The encoding of the topology by the position of the node inside the vector is not well defined for any category of graph – it holds for Directed Ordered Acyclic Graphs (DOAGs), this does not hold for generic cyclic graphs

The Graph Neural Network Model

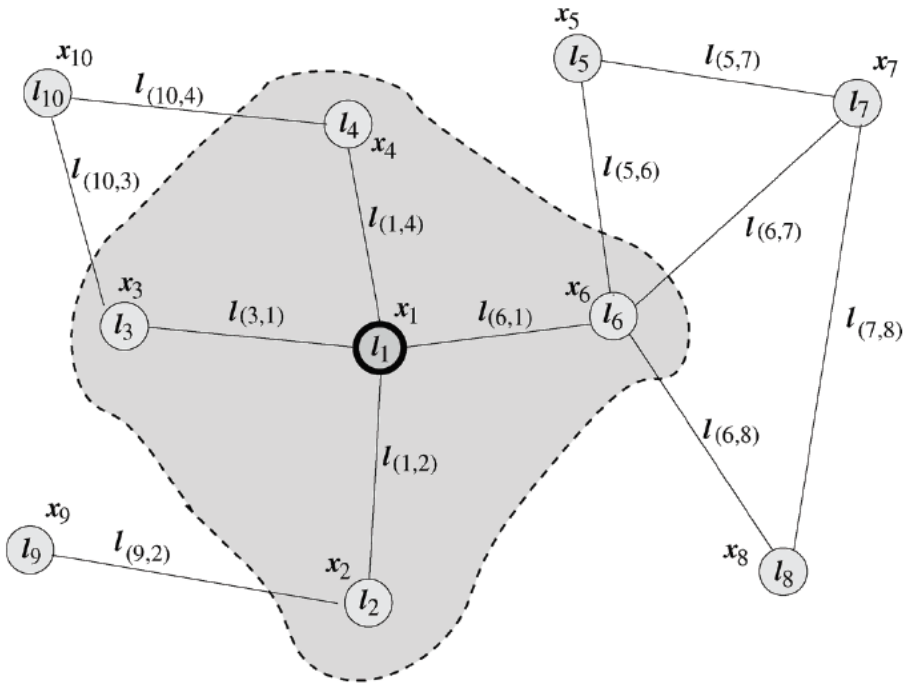
The Graph Neural Network Model (GNN)

- Introduced in [Scarselli et al. 2009], it is able to process graphs directly, without the need of a preprocessing step and without any limitation on the graph type (more general w.r.t Recursive nets, [Frasconi et al. 1998])
- This model exploits neural networks to learn how to encode nodes of a graph for a given task taking into account both the *information local* to each node and the whole *graph topology*.
- The learning process requires, for each epoch, an iterative **diffusion mechanism** up to convergence to a **stable fixed point**

The Graph Neural Network Model

- Given an input graph $G = (V, E)$, where V is a finite set of *nodes* and $E \subseteq V \times V$ collects the *arcs*, GNNs apply a two-phase computation on G
- **Encoding (aggregate) phase** the model computes a state vector for each node in V by (iteratively) combining the states of neighboring nodes (i.e. nodes $u, v \in V$ that are connected by an arc $(u, v) \in E$) – exploiting the **state transition function** f_w
- **Output (readout) phase** the latent representations encoded by the states stored in each node are exploited to compute the model output – exploiting the **output function** g_w

Neighbor-based computation



$$x_1 = f_w(l_1, l_{(1,2)}, l_{(3,1)}, l_{(1,4)}, l_{(6,1)}, x_2, x_3, x_4, x_6, l_2, l_3, l_4, l_6)$$

$l_{co[1]}$ $x_{ne[1]}$ $l_{ne[n]}$

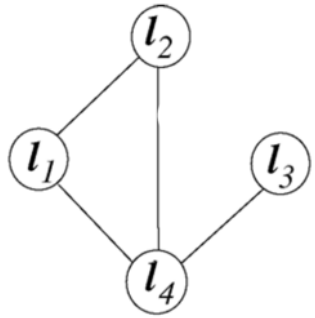
node label connection label neighbor state neighbor label

$$x_n = f_w(l_n, l_{co[n]}, x_{ne[n]}, l_{ne[n]}) \qquad x = F_w(x, l)$$

$$o_n = g_w(x_n, l_n) \qquad o = G_w(x, l_N)$$

equilibrium configuration!

Non-positional graphs

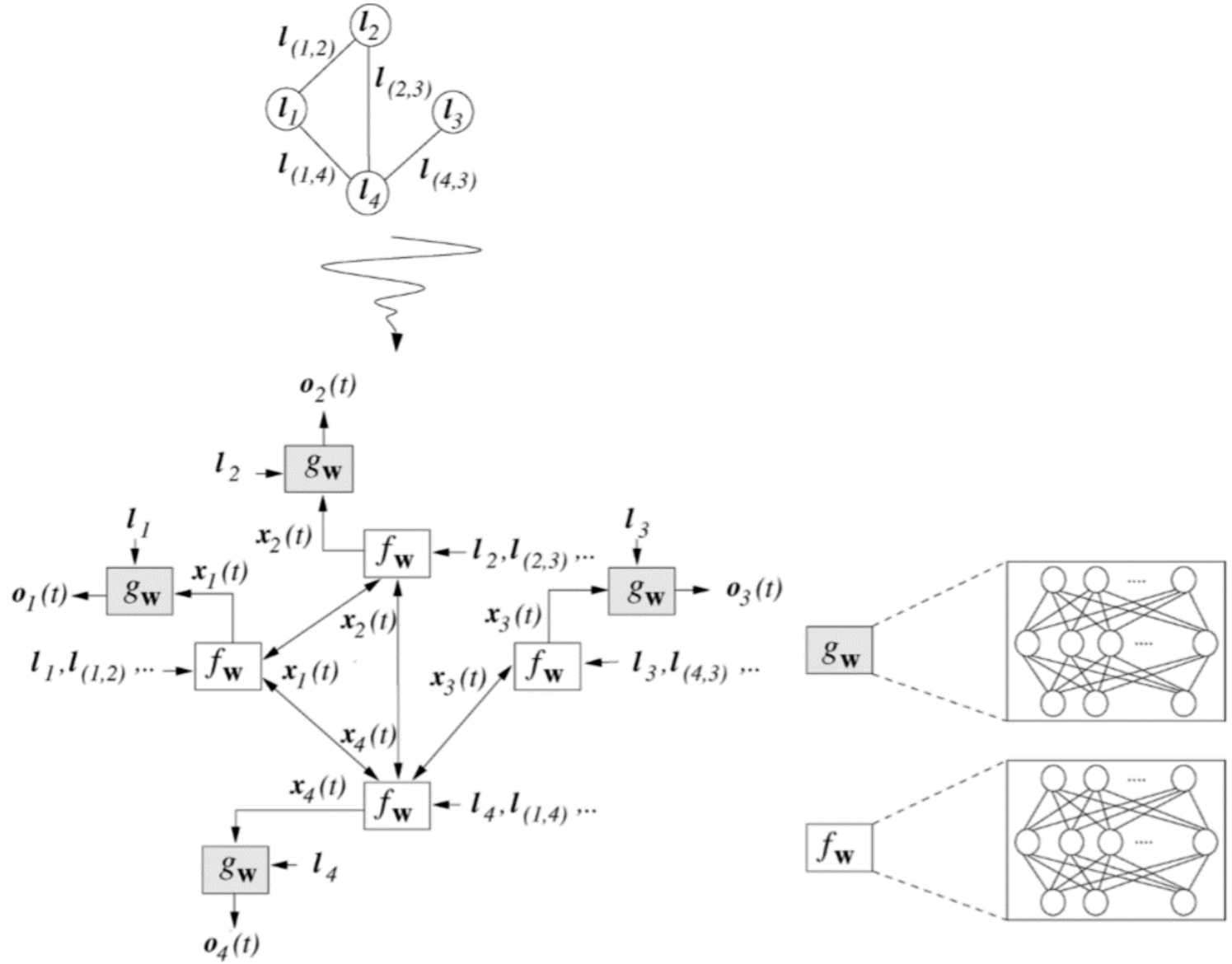


$$x_n^{(t)} = \sum_{(n,v)\in E} f_w(l_n, l_{(n,v)}, x_v^{(t-1)}, l_v)$$

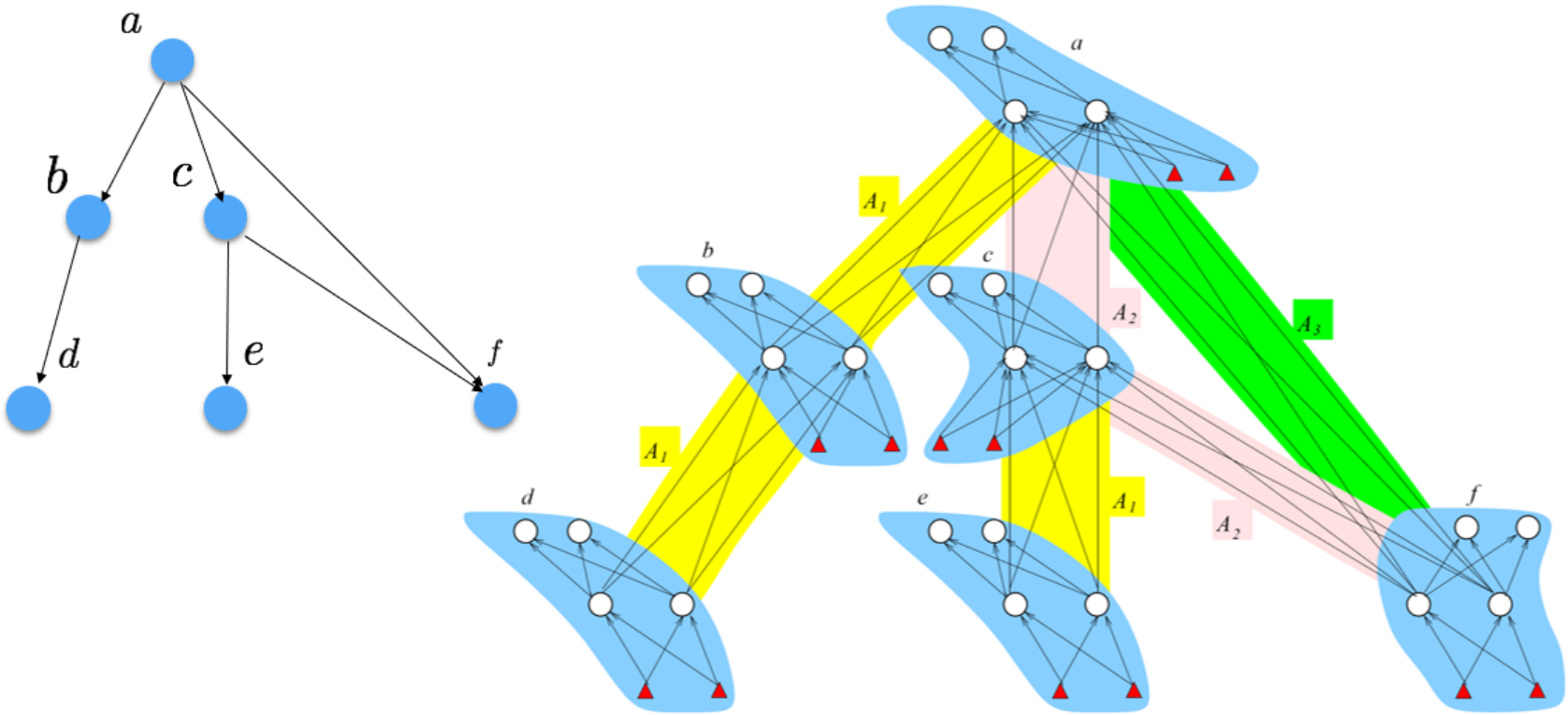
permutation-independent

$$o_n = g_w(x_n^{(T)}, l_n)$$

Graph Encoding



Graph Encoding



From the encoding network to the encoding neural network ...

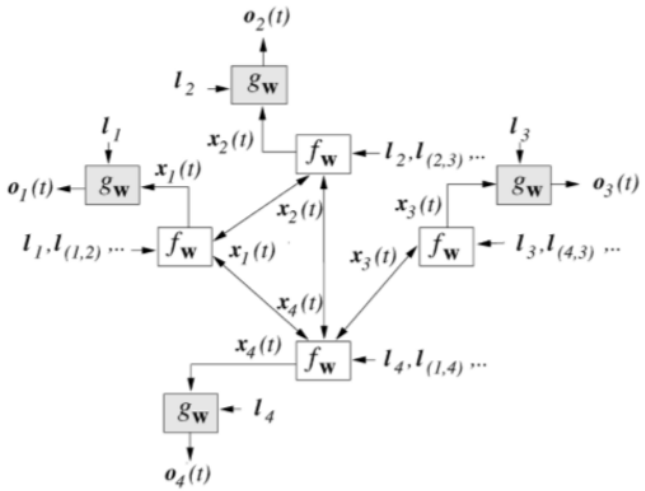
Convergence procedure

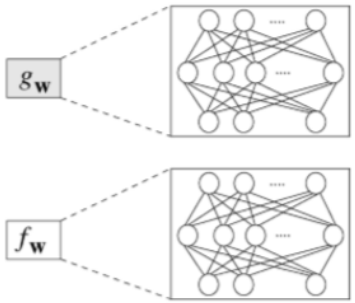
- The recursive application of the state transition function $f_w()$ on the graph nodes yields a diffusion mechanism, whose range depends on T
- In the original GNN model [Scarselli et al. 2009] the convergence procedure is executed until convergence of the state representation, i.e. until $x_n^{(t)} \simeq x_n^{(t-1)}, v \in V$.
- This scheme corresponds to the computation of the *fixed point* of the state transition function $f_w()$ on the input graph. In order to guarantee the convergence of this phase, the transition function is required to be a *contraction map*.
  Banach Fixed Point Theorem

Reaching Equilibrium

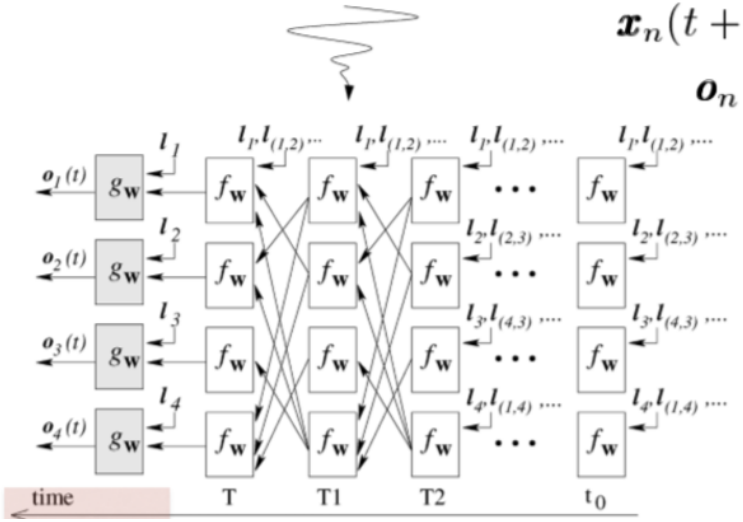
How we get the equilibrium points?

Relaxation to an equilibrium





$$\boldsymbol{x}_n(t+1) = f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}(t), \boldsymbol{l}_{\mathrm{ne}[n]})$$
$$\boldsymbol{o}_n(t) = g_{\boldsymbol{w}}(\boldsymbol{x}_n(t), \boldsymbol{l}_n), \qquad n \in \boldsymbol{N}.$$



$$\boldsymbol{x}(t+1) = F_{\boldsymbol{w}}(\boldsymbol{x}(t), \boldsymbol{l})$$

Lagrangian GNN

GNN: a Constraint formulation

- Basically, the encoding phase, through the iteration of $f_w()$, finds a solution to the fixed point problem defined by the constraint

$$\forall n \in V, x_n = \sum_{(n,v)\in E} f_w(l_n, l_{(n,v)}, x_v, l_v) \tag{1}$$

  In this case, the states encode the information contained in the whole graph.
- This diffusion mechanism is more general than executing only a fixed number of iterations (i.e. stacking a fixed number of layers).
- However, it can be computationally heavy and, hence, many recent GNN architectures apply only a fixed number of iterations for all nodes [Scarselli et al. 2009].

Lagrangian Propagation GNN

- With an approach related to [Carreira-Perpinan et al., 2014] [Taylor et al. 2016], we consider a Lagrangian formulation of the problem by adding free variables corresponding to the node states x_n
- With $L(g_w(x_v), y_v)$ as the loss function (target fitting approximation for node $v \in S$), the formulation of the learning task is

$$\begin{aligned} \text{minimize} \quad & \sum_{v \in S} L(g_w(x_v), y_v) \\ \text{subject to} \quad & \mathcal{G}(x_v - \sum_{(n,v) \in E} f_w(l_v, l_{(v,n)}, x_n, l_n)) = 0, \quad \forall\, v \in V \end{aligned} \tag{2}$$

  and $\mathcal{G}(x)$ is a function characterized by $\mathcal{G}(0) = 0$.
- Apart from classical choices, like $\mathcal{G}(x) = x$ or $\mathcal{G}(x) = x^2$, we can design different function shape with desired properties.

	lin	*lin-*ϵ	*abs*	*abs-*ϵ	*squared*
$\mathcal{G}(x)$	x	$\max(x, \epsilon) - \max(-x, \epsilon)$	$\lvert x \rvert$	$\max(\lvert x \rvert - \epsilon, 0)$	x^2
Unilateral	$\times$	$\times$	✓	✓	✓
ϵ-insensitive	$\times$	✓	$\times$	✓	$\times$

Optimization procedure: search for saddle points

- The constrained optimization problem can be solved in the Lagrangian framework by introducing for each constraint a Lagrange multiplier λ_v, to define the Lagrangian function $\mathcal{L}(\theta_{f_w}, \theta_{g_w}, X, \Lambda)$ where Λ is the set of the $|V|$ Lagrangian multipliers.
- We can define the unconstrained optimization problem as the search for saddle points in the adjoint space $(\theta_{f_w}, \theta_{g_w}, X, \Lambda)$ as:

$$\min_{\theta_{f_w}, \theta_{g_w}, X} \max_{\Lambda} \mathcal{L}(\theta_{f_w}, \theta_{g_w}, X, \Lambda)$$

  where θ_{f_w} and θ_{g_w} are the weights of the MLPs implementing the state transition function and the output function,
- The problem can be solved by gradient *descent* with respect to the variables $\theta_{f_w}, \theta_{g_w}, X$ and gradient *ascent* with respect to the Lagrange multipliers Λ.

Optimization procedure: search for saddle points

- The diffusion mechanism of the state computation is enforced by means of the constraints.
- The learning algorithm is based on a mixed strategy where :
  - Backpropagation is used to efficiently update the weights of the neural networks that implement the state transition and output functions
  - The diffusion mechanism evolves gradually by enforcing the convergence of the state transition function to a fixed point by means of the constraints.

Reaching Equilibrium

$$\mathcal{L}(\theta_{f_a}, \theta_{f_r}, X, \Lambda) = \sum_{v \in S} [L(f_r(x_v|\theta_{f_r}), y_v) + \\ + \lambda_v \mathcal{G}\left(x_v - f_a(x_{\text{ne}[v]}, l_{\text{ne}[v]}, l_{(v,\text{ch}[v])}, l_{(\text{pa}[v],v)}, x_v, l_v|\theta_{f_a})\right)]$$

$$\min_{\theta_{f_a}, \theta_{f_r}, X} \max_{\Lambda} \mathcal{L}(\theta_{f_a}, \theta_{f_r}, X, \Lambda)$$

$$\frac{\partial \mathcal{L}}{\partial x_v} = L' f'_{r,v} + \lambda_v \mathcal{G}'_v (1 - f'_{a,v}) - \sum_{w: v \in ne[w]} \lambda_w \mathcal{G}'_w f'_{a,w}$$

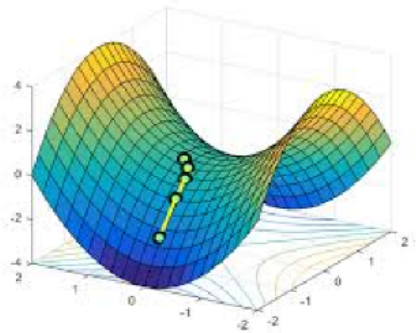
$$\frac{\partial \mathcal{L}}{\partial \theta_{f_a}} = -\sum_{v \in S} \lambda_v \mathcal{G}'_v f'_{a,v}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_{f_r}} = \sum_{v \in S} L' f'_{r,v}$$

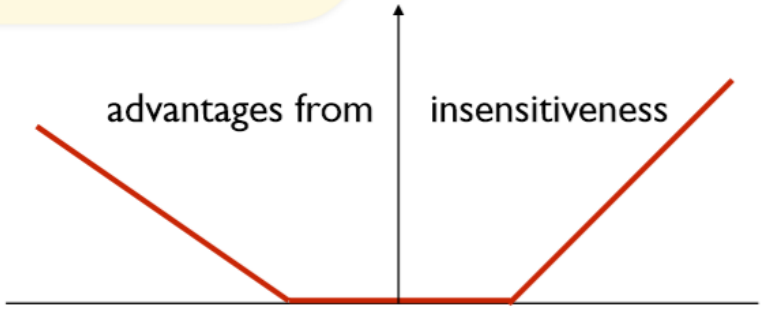
gradient descent

$$\frac{\partial \mathcal{L}}{\partial \lambda_v} = \mathcal{G}_v$$

gradient ascent



better suited for generalization



Experiments

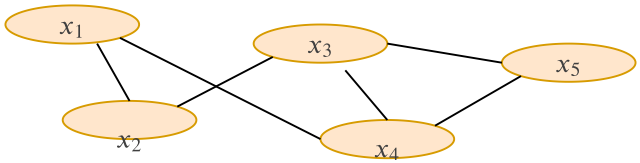
Experiments: node classification benchmarks

Table: Accuracies on the artificial datasets, for the proposed model (Lagrangian Propagation GNN - LP-GNN) and the standard GNN model for different settings.

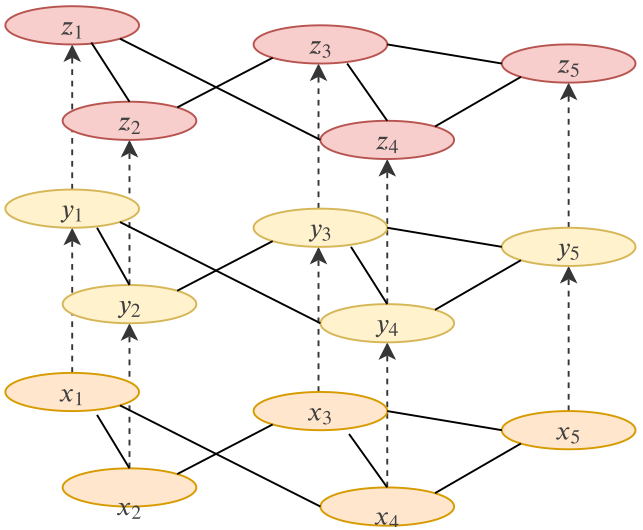
Model			Subgraph		Clique	
	$\mathcal{G}$	ϵ	*Acc(avg)*	*Acc(std)*	*Acc(avg)*	*Acc(std)*
LP-GNN	*abs*	0.00	96.25	0.96	88.80	4.82
		0.01	**96.30**	0.87	88.75	5.03
		0.10	95.80	0.85	85.88	4.13
	lin	0.00	95.94	0.91	84.61	2.49
		0.01	95.94	0.91	85.21	0.54
		0.10	95.80	0.85	85.14	2.17
	squared	-	96.17	1.01	**93.07**	2.18
Scarselli et al. 2009	-	-	95.86	0.64	91.86	1.12

Evolutions: MultiLayer Lagrangian GNNs

- Even though the state transition function can be implemented by Deep Neural Networks, the GNN is still shallow.
- A multilayer version of our algorithm can be easily introduced by simply adding the corresponding constraints
- Label of hidden nodes are simply the states of the nodes of the layer below.
- The new constraints add new variables (e.g. y_i, z_i, etc.)



(a) A shallow GNN.



(b) A deep GNN.

- The optimization problem is easily updated into:

$$\begin{aligned}
\text{minimize} \quad & \sum_{v \in S} L(g_w(z_v), t_v) \\
\text{subject to} \quad & \mathcal{G}(x_v - \sum_{(v,n)\in E} f_w^0(x_n, l_v, l_n)) = 0, \quad \forall\, v \in V \qquad (3) \\
& \mathcal{G}(y_v - \sum_{(v,n)\in E} f_w^1(y_n, x_n, x_v)) = 0, \quad \forall\, v \in V \qquad (4) \\
& \mathcal{G}(z_v - \sum_{(v,n)\in E} f_w^2(z_n, y_n, y_v)) = 0, \quad \forall\, v \in V \qquad (5)
\end{aligned}$$

Experiments: graph classification benchmarks

Table: Test set classification average accuracies and standard deviations for the graph classification benchmarks.

Datasets	IMDB-B	IMDB-M	MUTAG	PROT.	PTC	NCI1
# graphs	1000	1500	188	1113	344	4110
# classes	2	3	2	2	2	2
Avg # nodes	19.8	13.0	17.9	39.1	25.5	29.8
DCNN	49.1	33.5	67.0	61.3	56.6	62.6
PatchySan	71.0 $\pm$ 2.2	45.2 $\pm$ 2.8	92.6 $\pm$ 4.2	75.9 $\pm$ 2.8	60.0 $\pm$ 4.8	78.6 $\pm$ 1.9
DGCNN	70.0	47.8	85.8	75.5	58.6	74.4
AWL	74.5 $\pm$ 5.9	51.5 $\pm$ 3.6	87.9 $\pm$ 9.8	–	–	–
GIN	75.1 $\pm$ 5.1	52.3 $\pm$ 2.8	89.4 $\pm$ 5.6	76.2 $\pm$ 2.8	64.6 $\pm$ 7.0	82.7 $\pm$ 1.7
GNN	60.9 $\pm$ 5.7	41.1 $\pm$ 3.8	88.8 $\pm$ 11.5	76.4 $\pm$ 4.4	61.2 $\pm$ 8.5	51.5 $\pm$ 2.6
LP-GNN	71.2 $\pm$ 4.7	46.6 $\pm$ 3.7	87.7 $\pm$ 6.5	75.7 $\pm$ 4.5	61.7 $\pm$ 6.8	64.5 $\pm$ 2.2
LP-GNN-Multi*	76.2 $\pm$ 3.2	50.8 $\pm$ 2.0	89.4 $\pm$ 7.2	77.2 $\pm$ 3.9	67.9 $\pm$ 7.2	69.0 $\pm$ 1.7

Semisupervised Learning in Graph Domains

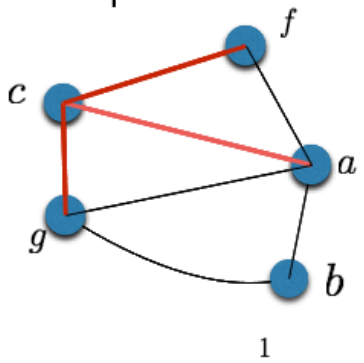
- In supervised node classification tasks, all the nodes in the training data needs to be labeled.
- Simple extension to tackle the Semisupervised tasks. In particular, by splitting the nodes into supervised and unsupervised, we can:
  - On **supervised** nodes, enforce state transition constraints and the objective function
  - On **unsupervised** nodes, enforce state transition constraints only
- This will allow to exploit the topological patterns of unsupervised data to make state transition function converge also for unsupervised nodes.

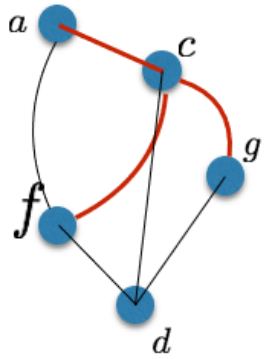
When new graphs come ...

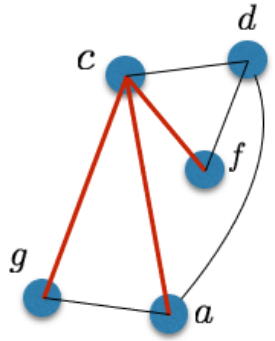
semi-supervised learning

constraints on state propagation

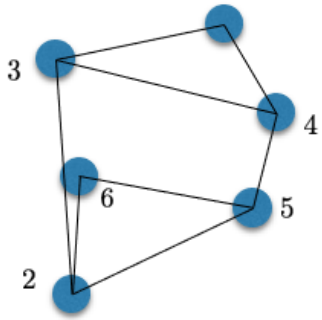
constraints on supervision







constraints on state propagation



you can always enforce state propagation

$$\forall n \in V, x_n = \sum_{(n,v)\in E} f_w(l_n, l_{(n,v)}, x_v, l_v)$$

Conclusions

- GNN learning task casted as a constrained optimization problem allows us to avoid the explicit computation of the fixed point needed to encode the graph.
- Jointly optimize the model weights and the state representation without the need of separate phases.
- Simplify the computational scheme of GNNs and allows us to incorporate alternative strategies in the fixed point optimization by the choice of the constraint function
- The constraint-based scheme can be extended to all the other methods proposed in the literature that exploit more sophisticated architectures.