Learning and Reasoning with Graph Data: Neural and Statistical-Relational Approaches

GNN and SRL

Manfred Jaeger

Aalborg University

Graph Neural Networks: Basics

From [Wu, Zonghan, et al. "A comprehensive survey on graph neural networks." IEEE transactions on neural networks and learning systems 32.1 (2020): 4-24.] :

TABLE II

TAXONOMY AND REPRESENTATIVE PUBLICATIONS OF GNNS

Category		Publications
Recurrent Graph Neural Networks (RecGNNs)		[15], [16], [17], [18]
Convolutional Graph Neural Networks (ConvGNNs)	Spectral methods	[19], [20], [21], [22], [23], [40], [41]
	Spatial methods	[24], [25], [26], [27], [42], [43], [44] [45], [46], [47], [48], [49], [50], [51] [52], [53], [54], [55], [56], [57], [58]
Graph Autoencoders (GAEs)	Network Embedding	[59], [60], [61], [62], [63], [64]
	Graph Generation	[65], [66], [67], [68], [69], [70]
Spatial-temporal Graph Neural Networks (STGNNs)		[71], [72], [73], [74], [75], [76], [77]

From [Wu, Zonghan, et al. "A comprehensive survey on graph neural networks." IEEE transactions on neural networks and learning systems 32.1 (2020): 4-24.] :

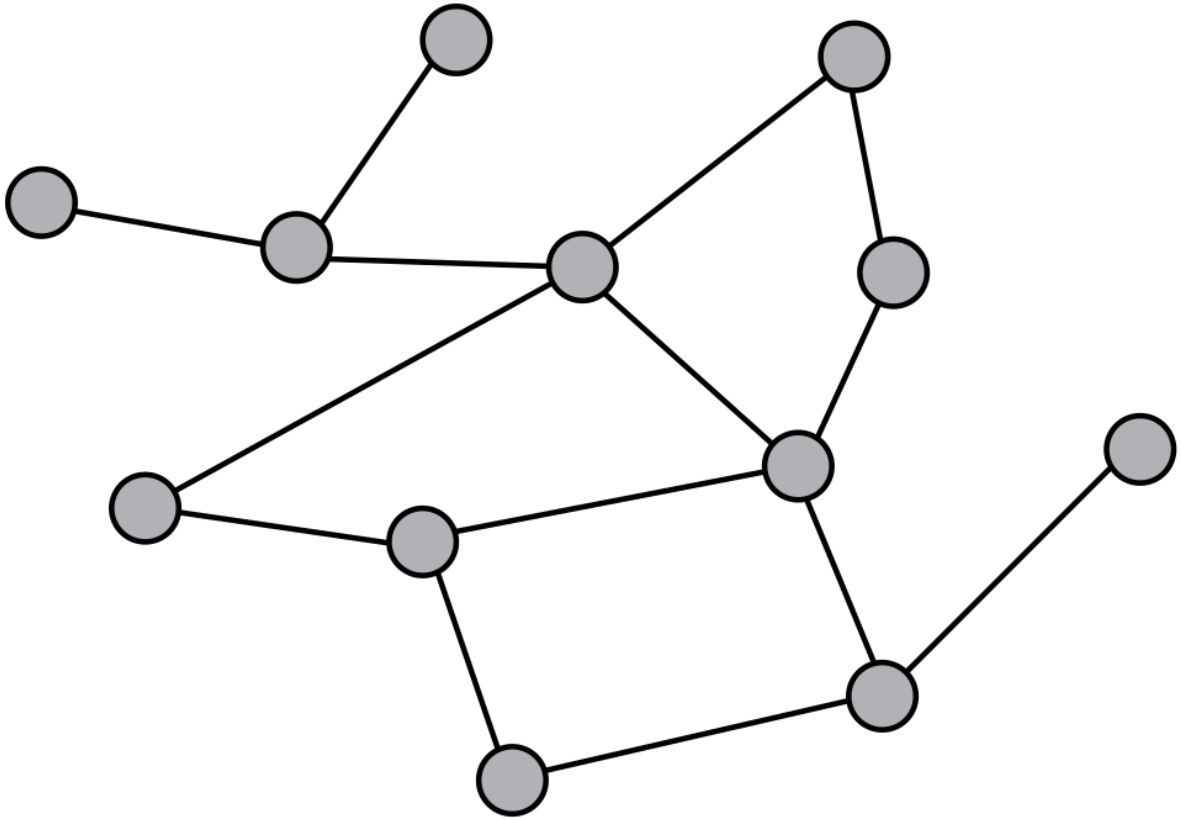
TABLE II

TAXONOMY AND REPRESENTATIVE PUBLICATIONS OF GNNS

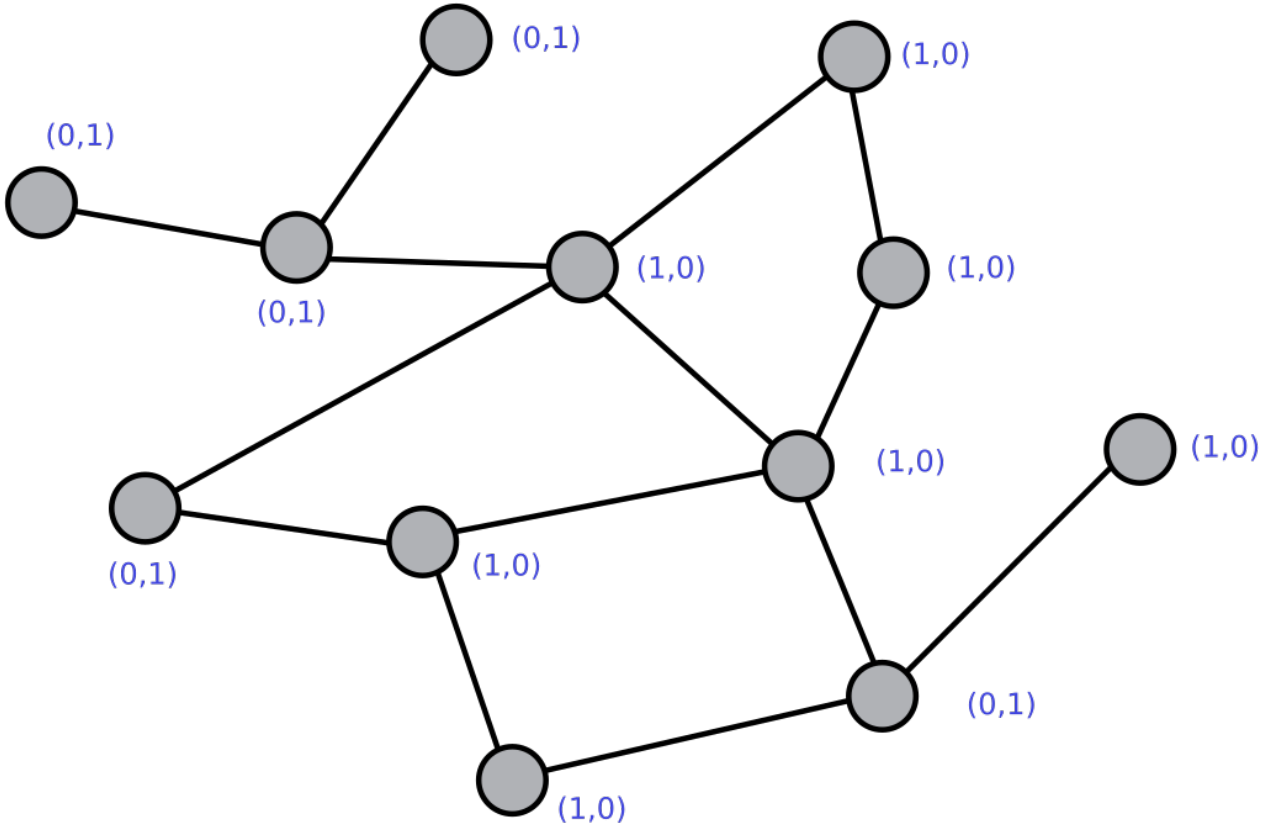
Category		Publications
Recurrent Graph Neural Networks (RecGNNs)		[15], [16], [17], [18]
Convolutional Graph Neural Networks (ConvGNNs)	Spectral methods	[19], [20], [21], [22], [23], [40], [41]
	Spatial methods	[24], [25], [26], [27], [42], [43], [44] [45], [46], [47], [48], [49], [50], [51] [52], [53], [54], [55], [56], [57], [58]
Graph Autoencoders (GAEs)	Network Embedding	[59], [60], [61], [62], [63], [64]
	Graph Generation	[65], [66], [67], [68], [69], [70]
Spatial-temporal Graph Neural Networks (STGNNs)		[71], [72], [73], [74], [75], [76], [77]

Our focus: convolutional-spatial

Iterative construction of node feature (representation, embedding) vectors:

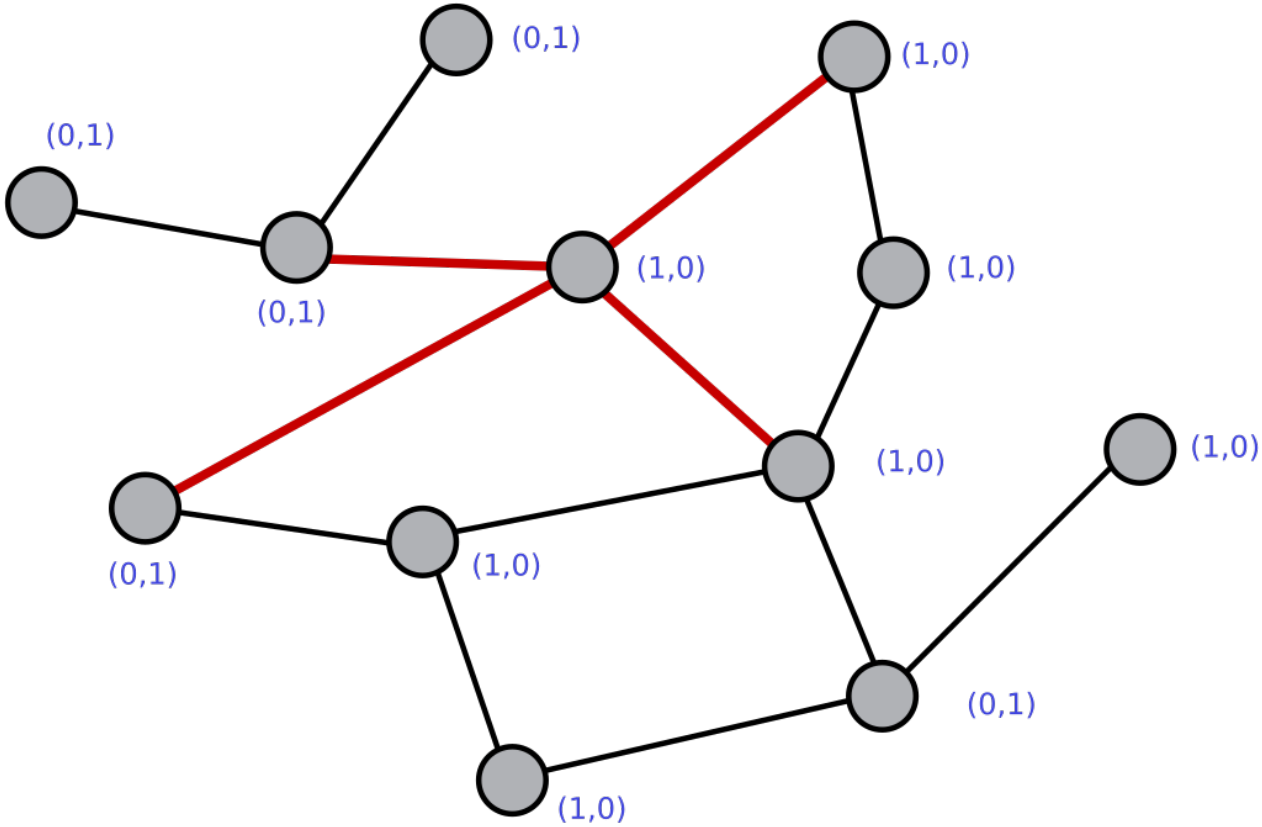


Iterative construction of node feature (representation, embedding) vectors:



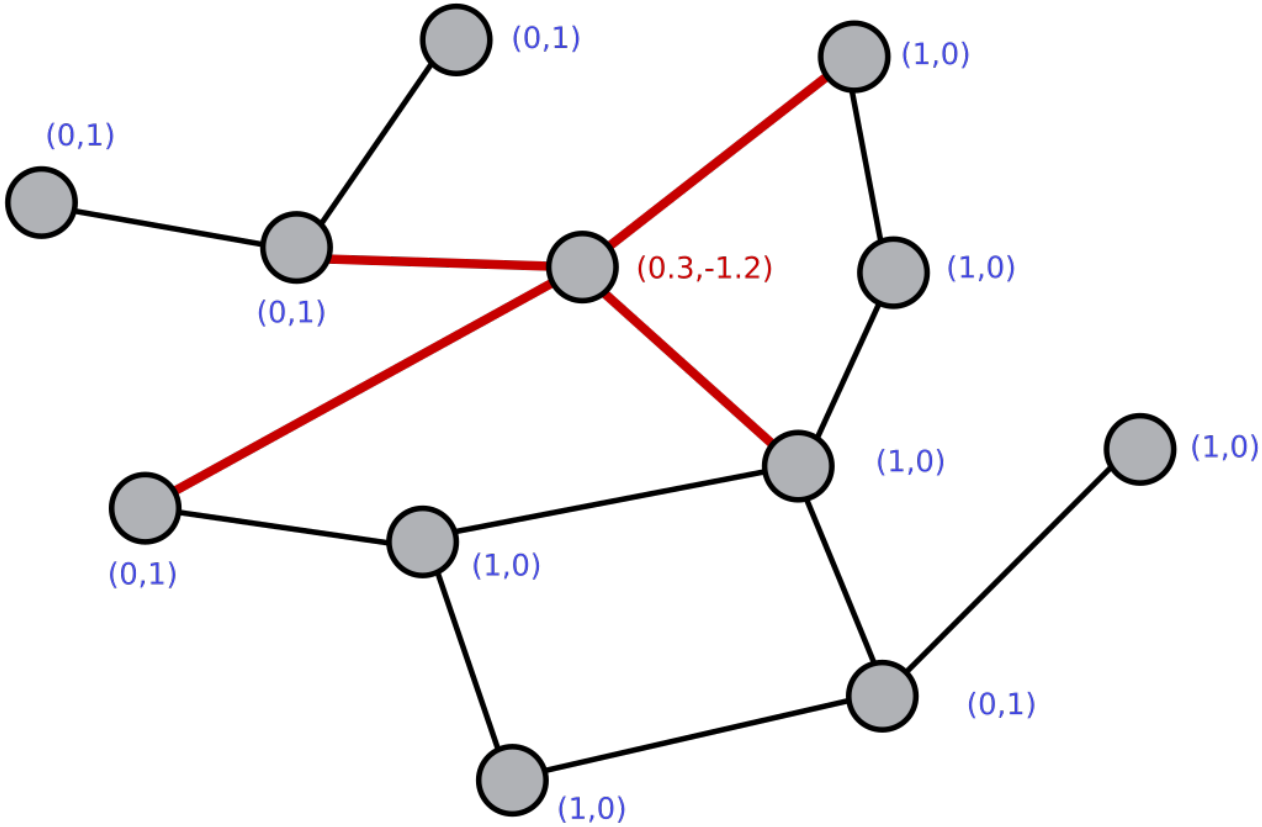
- H_0: initial node feature vectors

Iterative construction of node feature (representation, embedding) vectors:



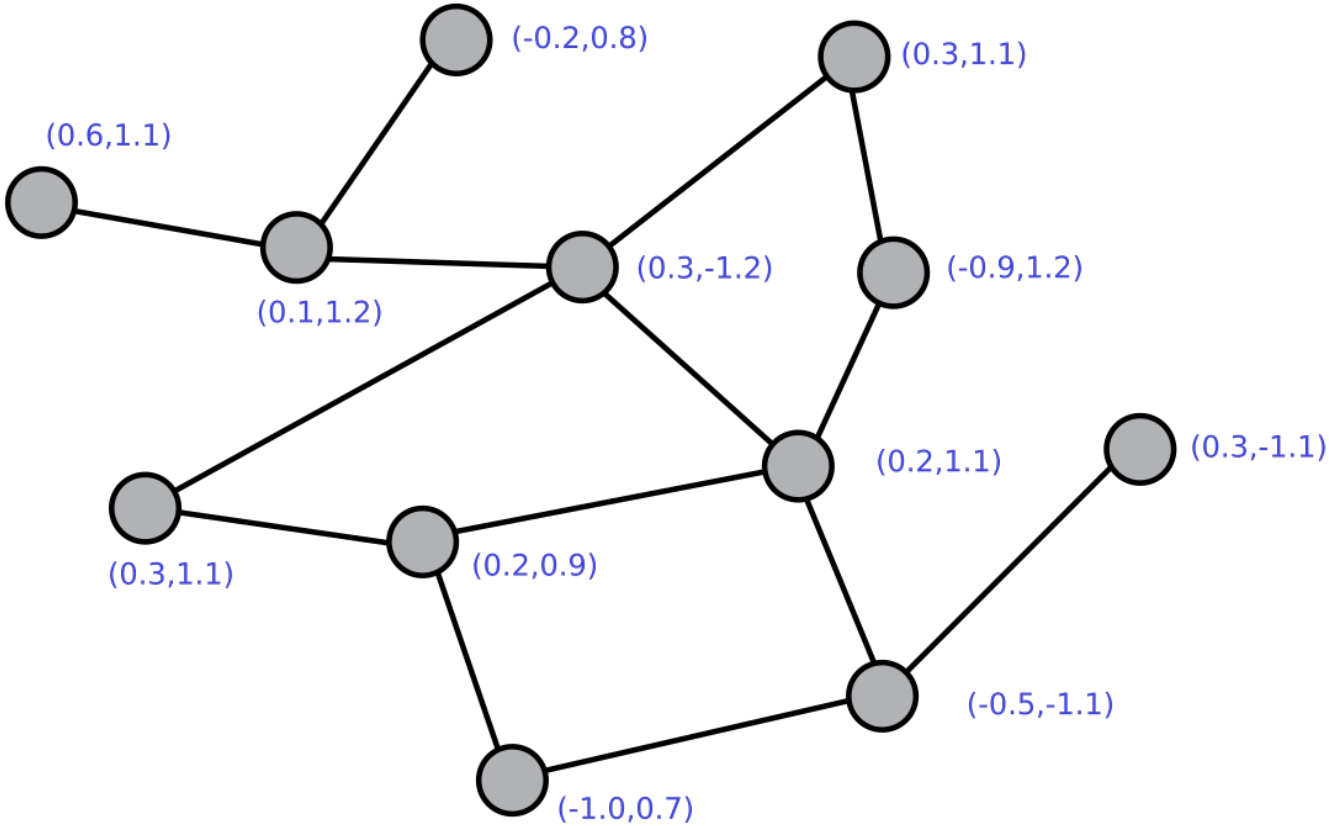
- H_0: initial node feature vectors

Iterative construction of node feature (representation, embedding) vectors:



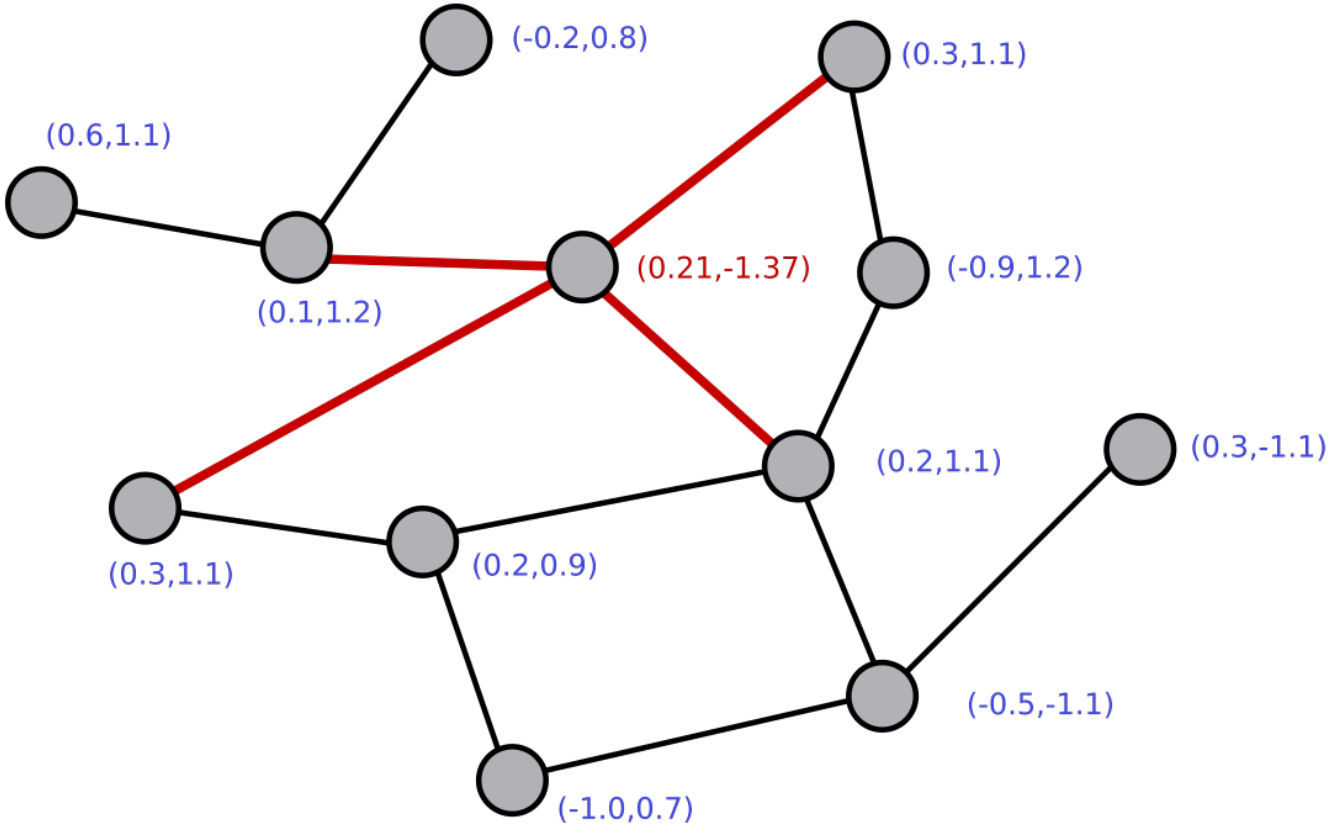
- H_0: initial node feature vectors
- H_1:
  - aggregate neighbors' H_0 features (e.g. sum)
  - apply some (activation) functions

Iterative construction of node feature (representation, embedding) vectors:



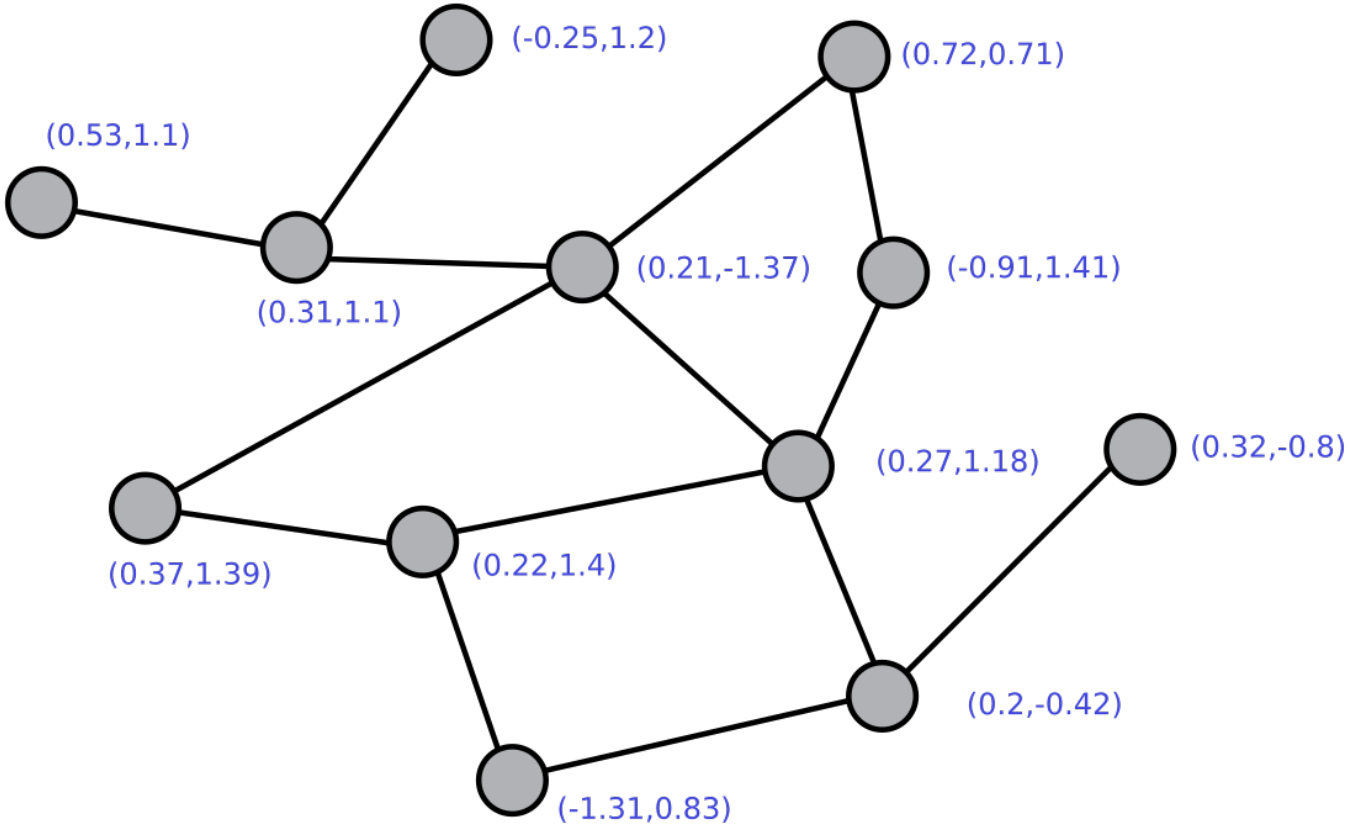
- H_0: initial node feature vectors
- H_1:
  - aggregate neighbors' H_0 features (e.g. sum)
  - apply some (activation) functions

Iterative construction of node feature (representation, embedding) vectors:



- H_0: initial node feature vectors
- H_1:
  - aggregate neighbors' H_0 features (e.g. sum)
  - apply some (activation) functions
- H_2:
  - aggregate neighbors' H_1 features
  - apply some (activation) functions

Iterative construction of node feature (representation, embedding) vectors:



- H_0: initial node feature vectors
- H_1:
  - aggregate neighbors' H_0 features (e.g. sum)
  - apply some (activation) functions
- H_2:
  - aggregate neighbors' H_1 features
  - apply some (activation) functions

$\boldsymbol{h}^k(i)$: d^k-dimensional vector representation of node i at kth iteration (layer).

A basic form of message passing updates:

$$\boldsymbol{h}^0(i) = \text{ initial node feature vector of node } i$$
$$\boldsymbol{h}^{k+1}(i) = f\left(\boldsymbol{W}^k \boldsymbol{h}^k(i) + \boldsymbol{U}^k \sum_{j \in N_i} \boldsymbol{h}^k(j)\right)$$

with ingredients:

- $\boldsymbol{W}^k, \boldsymbol{U}^k$: weight matrices (dimensions: $d^{k+1} \times d^k$)
- f: (nonlinear) activation function (component-wise)

$\boldsymbol{h}^k(i)$: d^k-dimensional vector representation of node i at kth iteration (layer).

A basic form of message passing updates:

$$\begin{aligned}\boldsymbol{h}^0(i) &= \text{ initial node feature vector of node } i \\ \boldsymbol{h}^{k+1}(i) &= f\left(\boldsymbol{W}^k \boldsymbol{h}^k(i) + \boldsymbol{U}^k \sum_{j \in N_i} \boldsymbol{h}^k(j)\right)\end{aligned}$$

with ingredients:

- $\boldsymbol{W}^k, \boldsymbol{U}^k$: weight matrices (dimensions: $d^{k+1} \times d^k$)
- f: (nonlinear) activation function (component-wise)

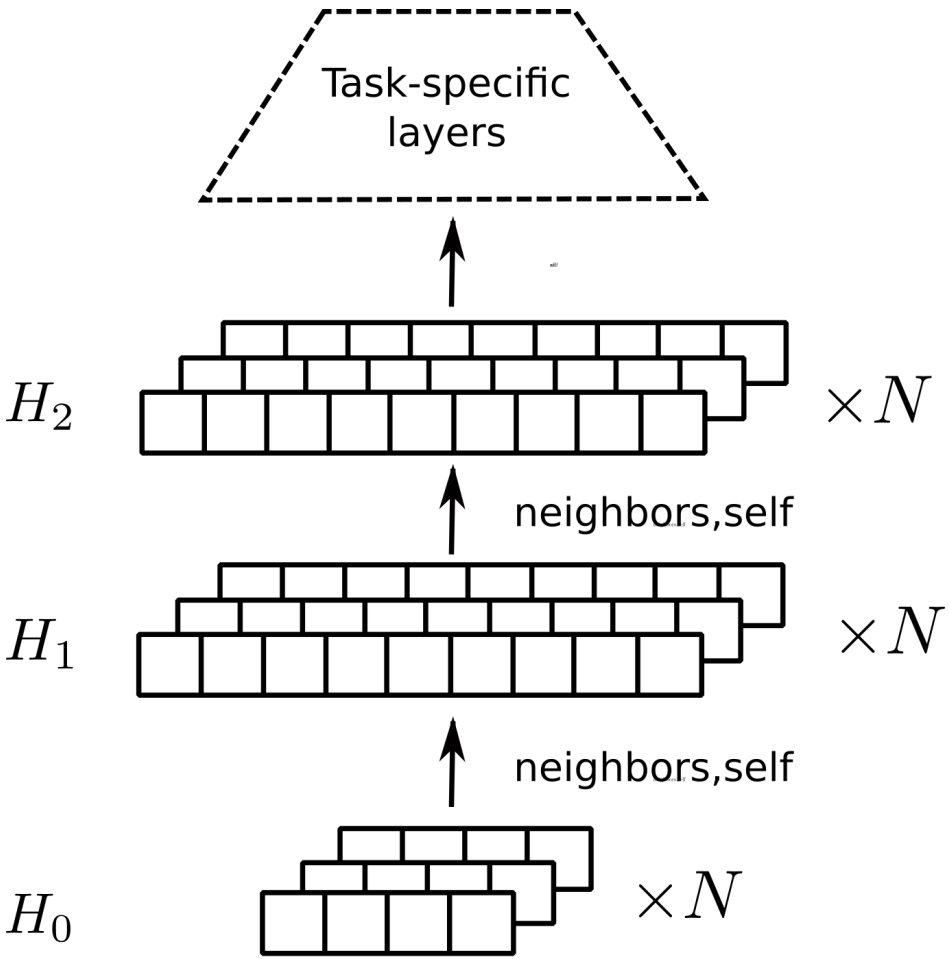
In full matrix notation:

$$\boldsymbol{H}^{k+1} = f\left(\boldsymbol{H}^k (\boldsymbol{W}^k)^T + \boldsymbol{E}\boldsymbol{H}^k (\boldsymbol{U}^k)^T\right)$$

with ingredients:

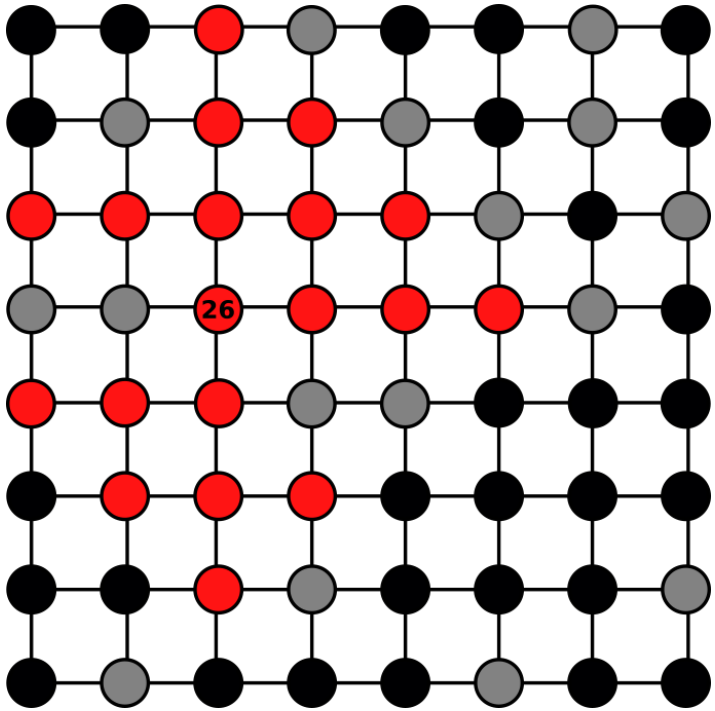
- $\boldsymbol{H}^k, \boldsymbol{H}^{k+1}$: $n \times d^k$ and $n \times d^{k+1}$ matrices
- $\boldsymbol{E}$: $n \times n$ adjacency matrix

Representation as NN architecture/computation graph:



- At each layer: one vector for each node (picture: $n = 3$)
- At top: task-specific transformations of final node representations
- self, neighbors: dependence of vectors in following layer on previous layer

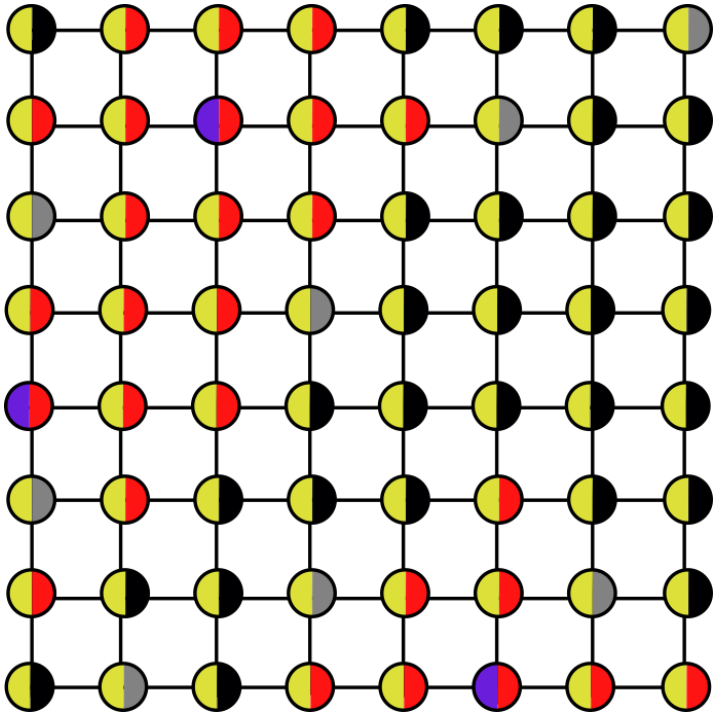
Initial features: node identifiers (typically: one-hot encoded).



Can represent/learn classification rule: node is *red*, if it has distance ≤ 3 to node 26.

➥this only works in transductive settings.

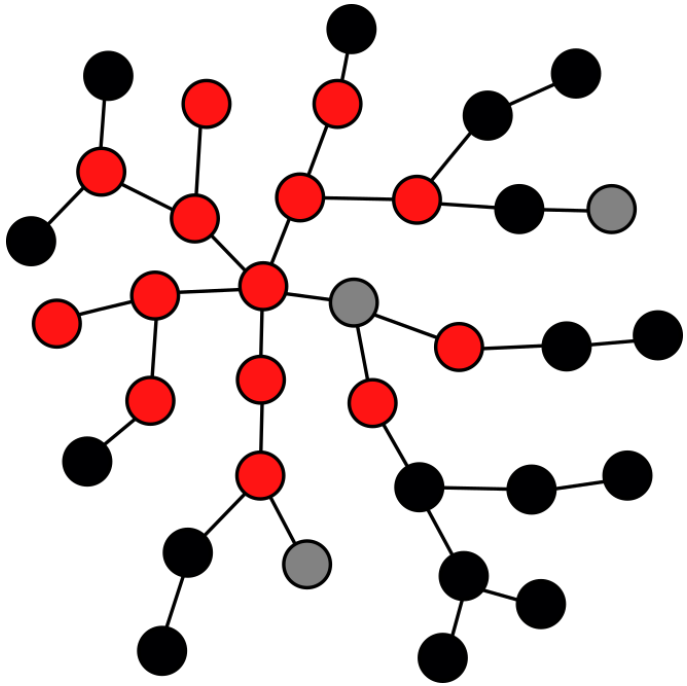
Initial features: node attributes (e.g. *color* $\in \{$*yellow*, *blue*$\}$)



Can represent/learn classification rule: node is *red*, if it has distance ≤ 2 to a blue node.

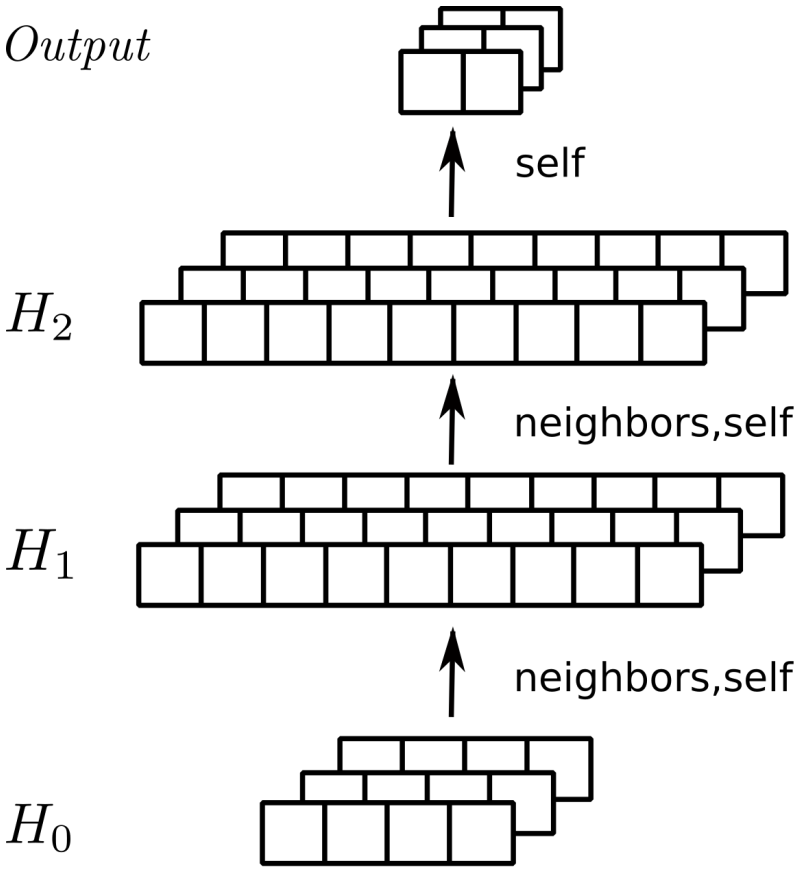
➥this works in inductive settings: rule can be applied to new graphs with yellow/blue nodes.

Initial features: none (then can say e.g.: $\boldsymbol{h}^0(i) = 1$ for all i).

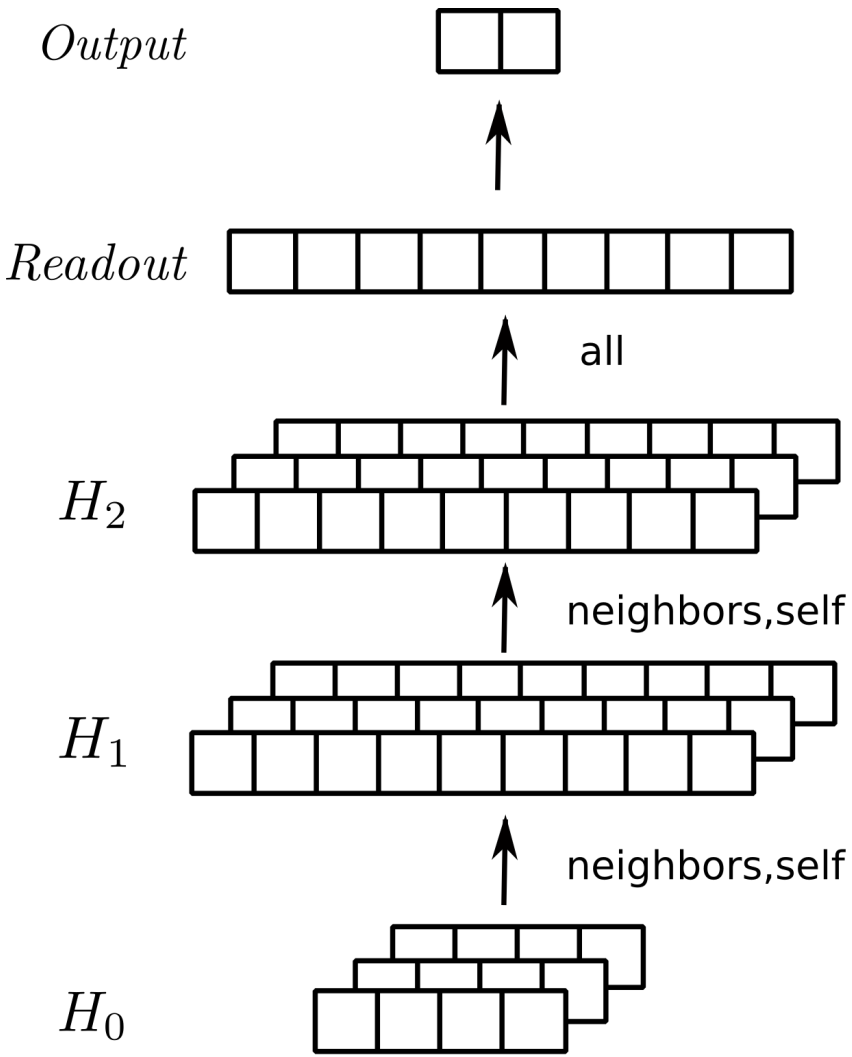


Can represent/learn classification rule: node is *red*, if it has distance ≤ 2 to a node with degree ≥ 5.

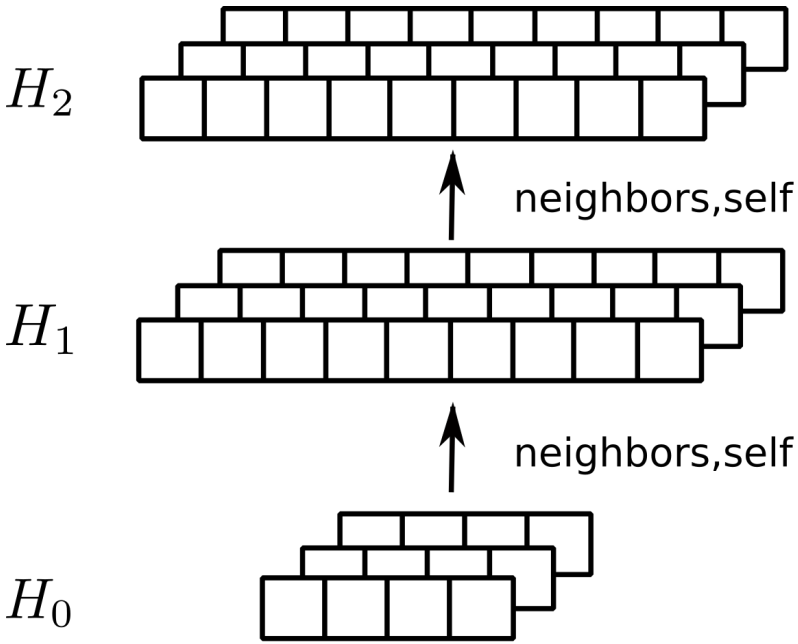
➥this works in inductive settings: rule can be applied to new graphs.



- On top of the final embedding $\boldsymbol{H}^K$: add neural network layers for classifying nodes based on their feature vectors $\boldsymbol{h}^K$
- Loss function: any standard classification loss, e.g. cross-entropy.
- The model is trained "end-to-end": the parameters of the embedding functions are optimized for the particular node classification (or regression) task.



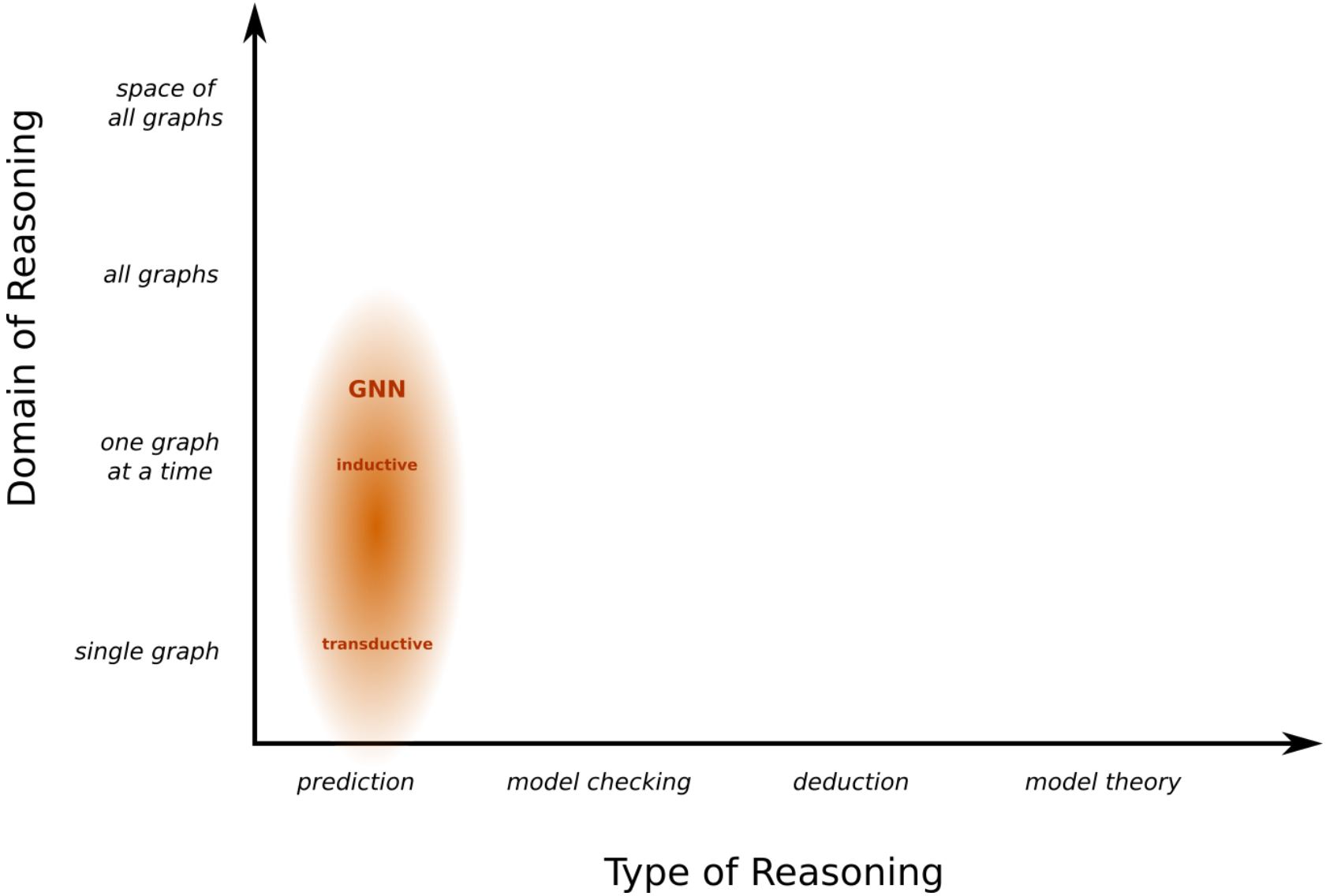
- On top of the final node embedding $\boldsymbol{H}^K$: add a *graph pooling* (a.k.a. *readout*) layer, that aggregates the representations of all nodes in the graph. The same kind of functions for aggregating node representations as in the message passing updates can be used (except no dependence on previous "own" representation $\boldsymbol{h}(i)$).
- On top of the readout layer: add neural network layers for classifying graph.
- Loss function: any standard classification loss, e.g. cross-entropy.
- The model is trained "end-to-end": the parameters of the embedding functions are optimized for the particular node classification (or regression) task.



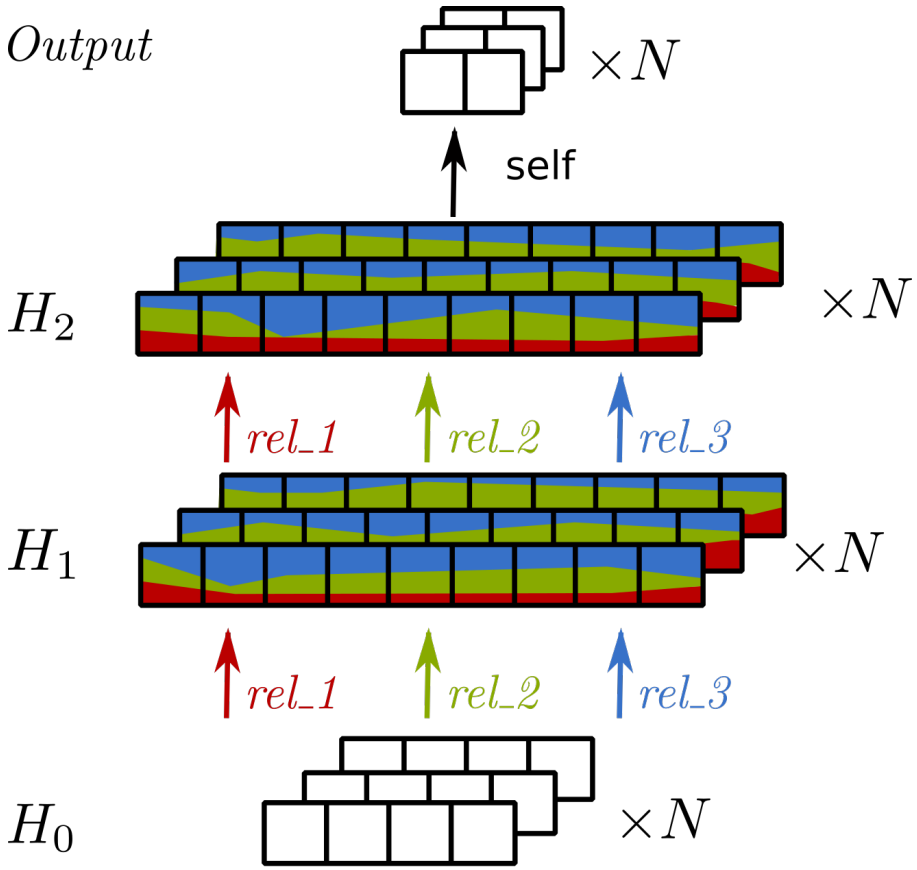
- GNN only computes embeddings
- Trained using reconstruction loss, such as

$$\sum_{i,j \in V} (\boldsymbol{h}(i) \cdot \boldsymbol{h}(j) - \boldsymbol{E}[i,j])^2$$

- Use the score $\boldsymbol{h}(i) \cdot \boldsymbol{h}(j)$ to predict whether there is a (not yet observed) edge between i and j.

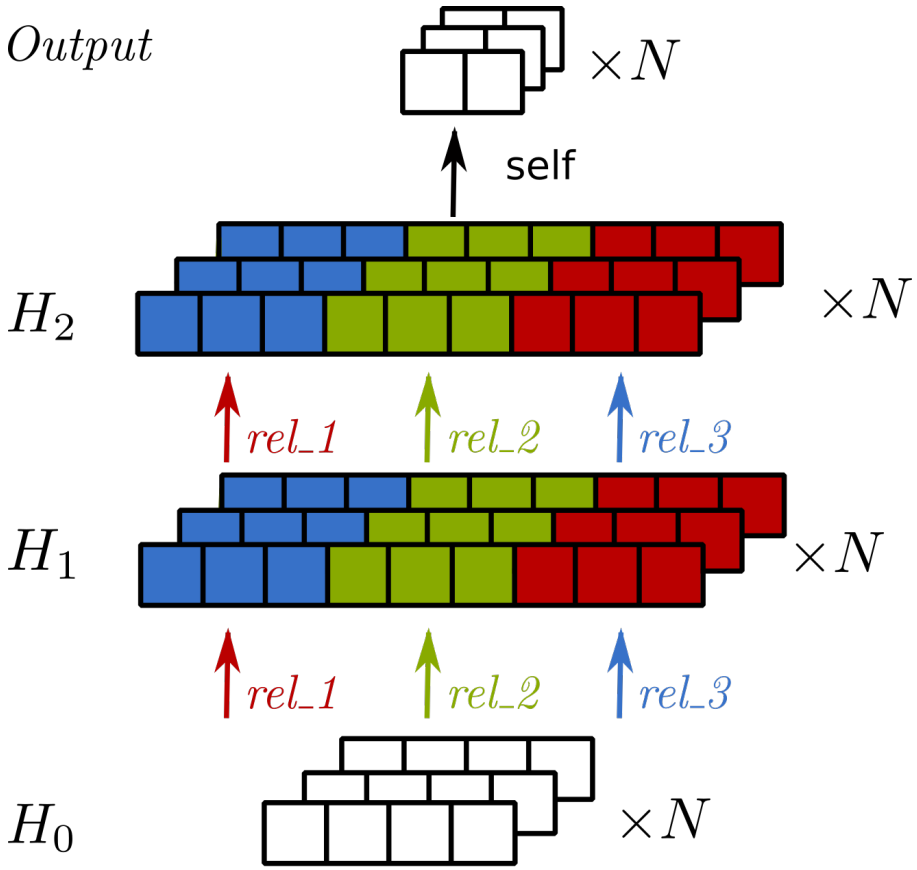

Domain of Reasoning
space of all graphs
all graphs
one graph at a time
single graph
GNN
inductive
transductive
prediction
model checking
deduction
model theory
Type of Reasoning

Graph Neural Networks: Extensions



- aggregate neighbors separately for each relation
- apply function to all aggregates

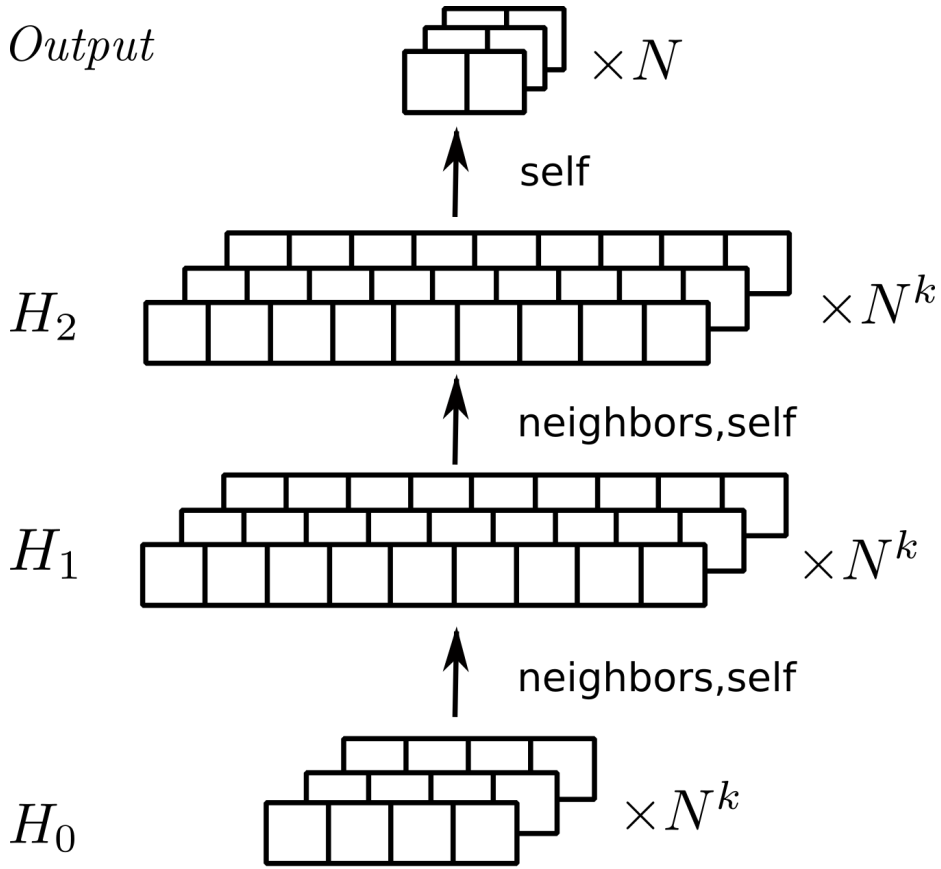
[Schlichtkrull et al.: Modeling Relational Data with Graph Convolutional Networks, 2018]



- aggregate neighbors separately for each relation
- apply function to all aggregates

[Schlichtkrull et al.: Modeling Relational Data with Graph Convolutional Networks, 2018]

Feature vectors learned for k-tuples $\boldsymbol{v}$ of nodes.

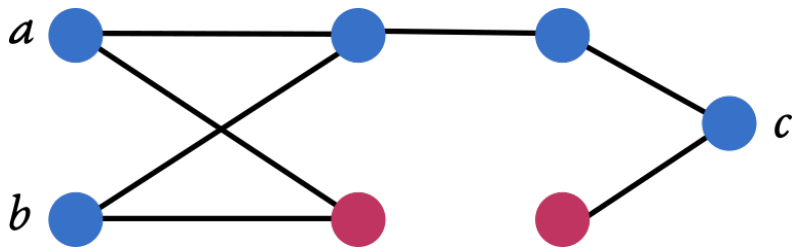


Neighbors for k-tuples defined by $edge^*$-relation:

$edge^*(\boldsymbol{v}, \boldsymbol{v}') \Leftrightarrow \boldsymbol{v}, \boldsymbol{v}'$ identical except for one component j (and $edge(\boldsymbol{v}[j], \boldsymbol{v}'[j])$).

[Morris et al.: Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks, 2019]

Discriminative power: when can two nodes be distinguished by a GNN?



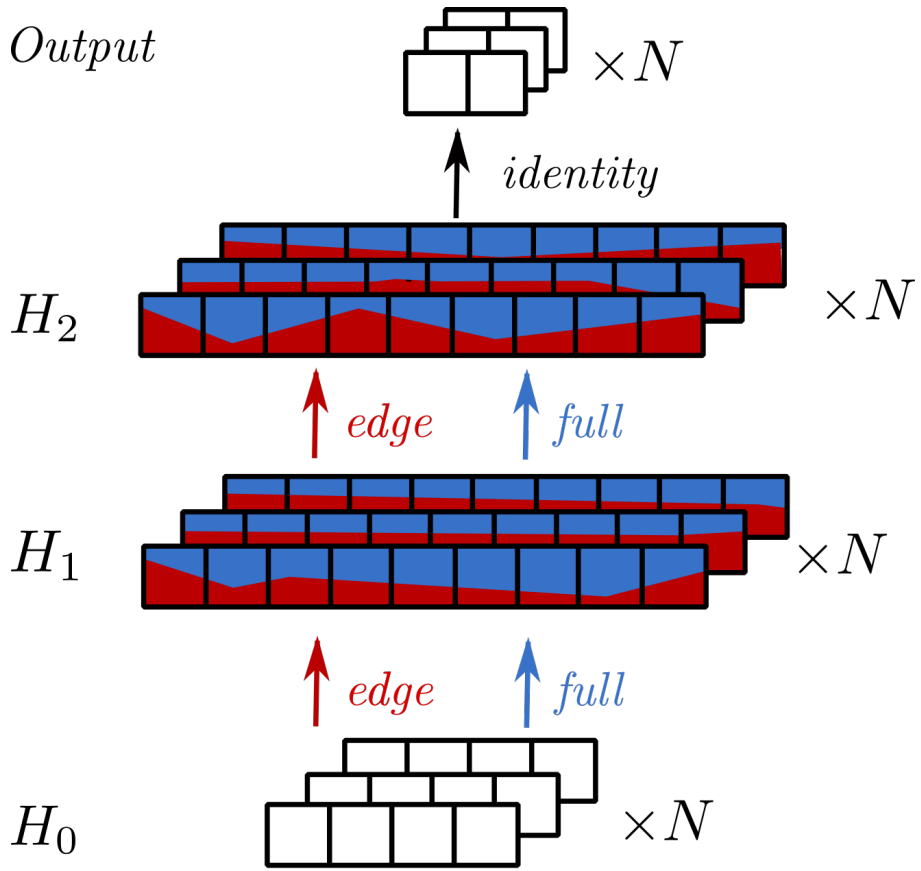
a, b indistinguishable by any GNN.

a, c indistinguishable by 2-layer GNNs, distinguishable by 3-layer GNNs.

➥ l-layer GNNs can only access information in the $l-1$ hop node neighborhood.

➥ (standard) GNNs can not access “global” graph properties for node classification.

Aggregate-Combine-Readout GNN:



- Multi-relational GNN with
  - original *edge* relation
  - a *full* relation that fully connects the graph

[Barceló et al.: The logical expressiveness of graph neural networks, 2020]

Main theorem of [Barceló et al.]:

Every node property that can be expressed in **the two-variable fragment of first-order logic with counting quantifiers** *can be captured by an ACR-GNN.*

- Signature of the logic: *edge* relation, node attributes.

Example

$$\alpha_1(X) \equiv \exists^{[8,10]} Y(\mathit{blue}(Y) \wedge \neg \mathit{edge}(X, Y))$$

Counting in FOL without variable restrictions:

$$\exists^{\geq 3} Y \phi(X, Y) \rightsquigarrow$$
$$\exists Y_1, Y_2, Y_3 (Y_1 \neq Y_2 \wedge Y_1 \neq Y_3 \wedge Y_2 \neq Y_3 \wedge \phi(X, Y_1) \wedge \phi(X, Y_2) \wedge \phi(X, Y_3))$$

Result from [Jaeger, Relational Bayesian Networks, 1997]:

Let $\phi(\boldsymbol{x})$ be a first-order formula over signature $\mathcal{R}$. Then there exists a probability formula $F_\phi(\boldsymbol{x})$ over $\mathcal{R}$, s.t. for every multi-relational graph $G = (V, \boldsymbol{R})$ and every $|\boldsymbol{x}|$-tuple $\boldsymbol{v}$ of nodes: $F_\phi(\boldsymbol{v}) = 1$ iff $\phi(\boldsymbol{v})$ holds in G (and $F_\phi(\boldsymbol{d}) = 0$ otherwise).

Statistical Relational Learning

Statistical Relational Learning, First-Order Probabilistic Models, Probabilistic-Logic Learning, Relational Probabilistic Models, . . . :

Data Perspective

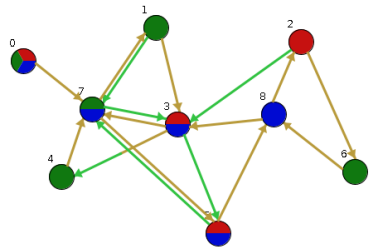
- Model data that is
  - found in relational databases
  - can be described by (first-order) predicate logic

Model Representation

Use formal languages to define probabilistic models for graphs:

- Logic-based representations
- Entity-relationship diagrams (database models)
- Programming languages

Abstract: graphs



Semantics

Predicate logic (relational)

$\forall x(r(x) \rightarrow \exists y(e(x,y) \wedge b(y)))$

$\exists z, x, y \neg(e(x,y) \wedge e(x,z) \wedge (e(y,z))$

. . .

Selected SRL frameworks, 1992-2007:

1992-1995	*Knowledge-based model construction* (Breese,Charniak,Goldman,Poole,Wellman,. . .
1995	*Prism* (Sato)
	Probabilistic Knowledge Bases (Ngo, Haddawy)
1996	*SLP* (Muggleton)
1997	*OOBN* (Pfeffer, Koller)
	MEBN (Laskey, Mahoney)
	RBN (Jaeger)
	ICL (Poole)
1998	*PRM* (Friedman, Getoor, Koller, Pfeffer)
2000	*BLP* (Kersting, De Raedt)
2001	*IBAL* (Pfeffer)
2002	*RMN* (Taskar et al.)
2003	*CLP(BN)* (Cussens,Page,Qazi,Santos Costa)
	RPT (Jensen, Neville)
	LOHMM (Kersting, De Raedt, Raiko)
2004	*LBN* (Blockeel,Bruynooghe,Fierens,Ramon)
	MLN (Richardson,Domingos)
2005	*BLOG* (Milch et al.)
	FOCI (Natarajan et al.)
2007	*ProbLog* (De Raedt, Kimmig, Toivonen)
2007-	Many more; partly transition to **probabilistic programming**

Representatives for main paradigms:

RBN	Directed probabilistic graphical models
MLN	Undirected probabilistic graphical models
ProbLog	(Inductive) logic programming

Markov Logic Networks

A Markov Logic Network consists of a set of pairs (F, w), where

- F is a quantifier-free first-order logic formula
- $w \in \mathbb{R}$ is a *weight*

The **signature** $\mathcal{R}$ of the MLN consists of all relation symbols used in any of the formulas F.

Example

F	w
friends(X,Y)	-0.2
republican(X)	0.1
friends(X, Y) $\Rightarrow$ (*republican*(X) $\Rightarrow$ *republican*(Y))	0.8

$\mathcal{R} = \{friends, republican\}$.

[M. Richardson, P. Domingos: Markov logic networks. *Machine learning* 2006.]

For a given domain (set of nodes) $V = \{1, \ldots, n\}$, the MLN defines a distribution over all graphs $G = (V, \boldsymbol{R})$:

For a formula $F(X, Y)$ define

$$\#(F, G) := |\{(i, j) \in V \times V : F(i, j) \textit{ is true in } G\}|$$

The *weight* of G then is

$$w(G) := \sum_{(F,w)} \#(F, G) \cdot w,$$

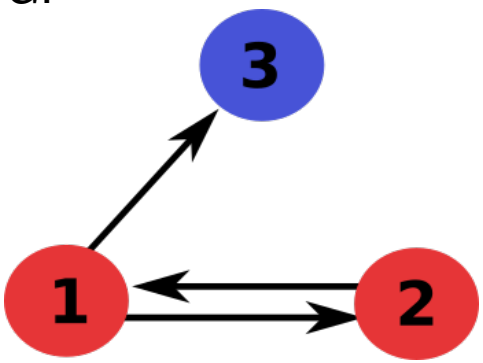
and the *probability of* G is

$$P(G) := \frac{1}{Z} e^{w(G)},$$

with Z the normalizing constant that the sum over all graphs with domain V is 1.

Example

G:



(Red: *republican*)

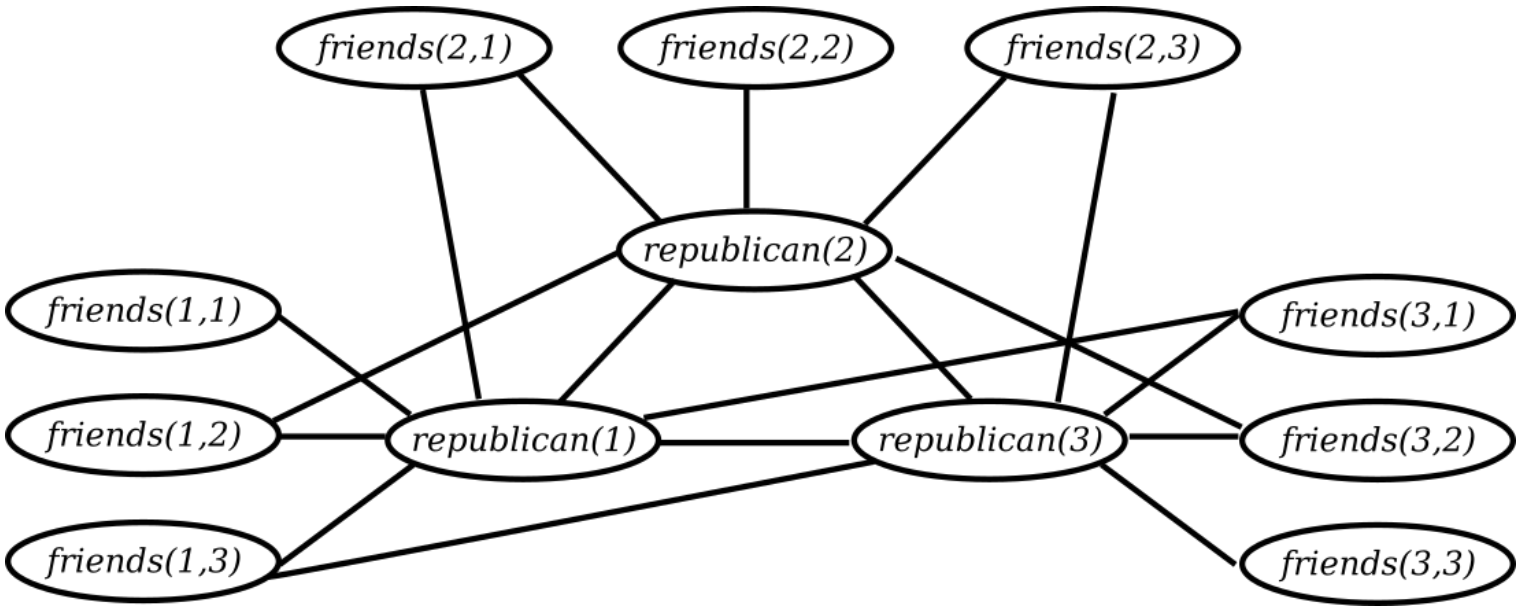
friends(X,Y)	-0.2
republican(X)	0.1
friends(X, Y) $\Rightarrow$ (*republican*(X) $\Rightarrow$ *republican*(Y))	0.8

$w(G) = 3 \cdot (-0.2) + 2 \cdot 0.1 + (6 + 2) \cdot 0.8 = 6.0$

$P(G) =$??? (need to first compute Z, i.e., the weight of all graphs)

The distribution defined by an MLN vor domain V can be represented as a *Markov Random Field (MRF)*

- Nodes are all the ground atoms for the signature $\mathcal{R}$ and the nodes $i \in V$
- Edges represent probabilistic dependencies



MLNs: Pros and Cons

+ Natural models for symmetric dependencies (e.g. homophily)
+ Interpretable model representations
+ Can use expert knowledge or machine learning to construct formulas
- Computational challenges due to Z
- Impact of weight parameters on probabilities can be hard to understand and control

ProbLog

A ProbLog model (program) consists of

- a set of probabilistic atoms
- a set of logical rules (Horn clauses)

Example

$$edge(a, b) : 0.3$$
$$edge(b, c) : 0.3$$

$$path(X, Y) \leftarrow edge(X, Y)$$
$$path(X, Y) \leftarrow edge(X, Z), path(Z, Y)$$

[Kimmig, A., Demoen, B., De Raedt, L., Costa, V. S., & Rocha, R. (2011). On the implementation of the probabilistic logic programming language ProbLog. Theory and Practice of Logic Programming.]

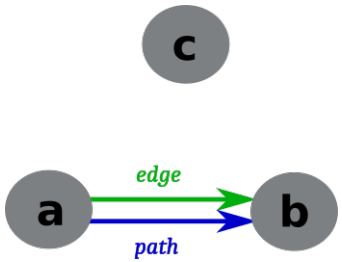
By randomly sampling the ground atoms according to their probability weights

- one obtains a standard logic program LP,
- which defines a unique graph described by all the ground atoms that can be proven from LP

Example

$$P\left(\begin{array}{l} edge(a,b) \\ path(X,Y) \leftarrow edge(X,Y) \\ path(X,Y) \leftarrow edge(X,Z), path(Z,Y) \end{array}\right) = 0.3 \cdot 0.7 = 0.21$$

LP defines graph:



- the probability of a graph is the sum of the probabilities of the LPs that define it.

ProbLog: Pros and Cons

+ Interpretable model representations
+ Can use expert knowledge or machine learning to construct formulas
+ Powerful “transitive closure” (“least fixed point”) expressivity
- Computational challenges due to “all proofs” semantics
- May require non-interpretable, latent relations for probabilistic atoms

SRL Frameworks

An SRL *framework* consists of

- Syntax: a formal representation language for any relational signature $\mathcal{R}$
- Semantics: defines for any domain V, a probability distribution over the space $\mathcal{G}(V, \mathcal{R})$; formally: a mapping
$$V \mapsto P_V \in \Delta\mathcal{G}(V, \mathcal{R})$$
- Inference (reasoning): algorithms for the computation of *conditional probabilities*
$$P_V(A|B) \quad \text{for some } A, B \subseteq \mathcal{G}(V, \mathcal{R})$$
Also: computing *most probable explanation (mpe)*:
$$\max_{G \in \mathcal{G}(V,\mathcal{R})} P_V(G|B)$$
- Learning: methods for learning models from graph data. Typically divided into:
  - Structure learning: determines (logical) structure of the model (here also: knowledge-driven design)
  - Parameter learning: fitting numerical parameters

For probabilistic inference

$$P_V(A|B) = ?$$

usually supported:

- B a set of graphs defined by a conjunction of (negated) atoms
- A a set of graphs defined by a single atom

Examples

$V = \{mary,tom,carl,sue\}$; $P_V(republican(mary)|friends(mary,carl), \neg republican(carl)) = ?$
$V = \{a, b, c, d, e\}$; $P_V(path(a, d)|edge(a, c), path(e, d)) = ?$

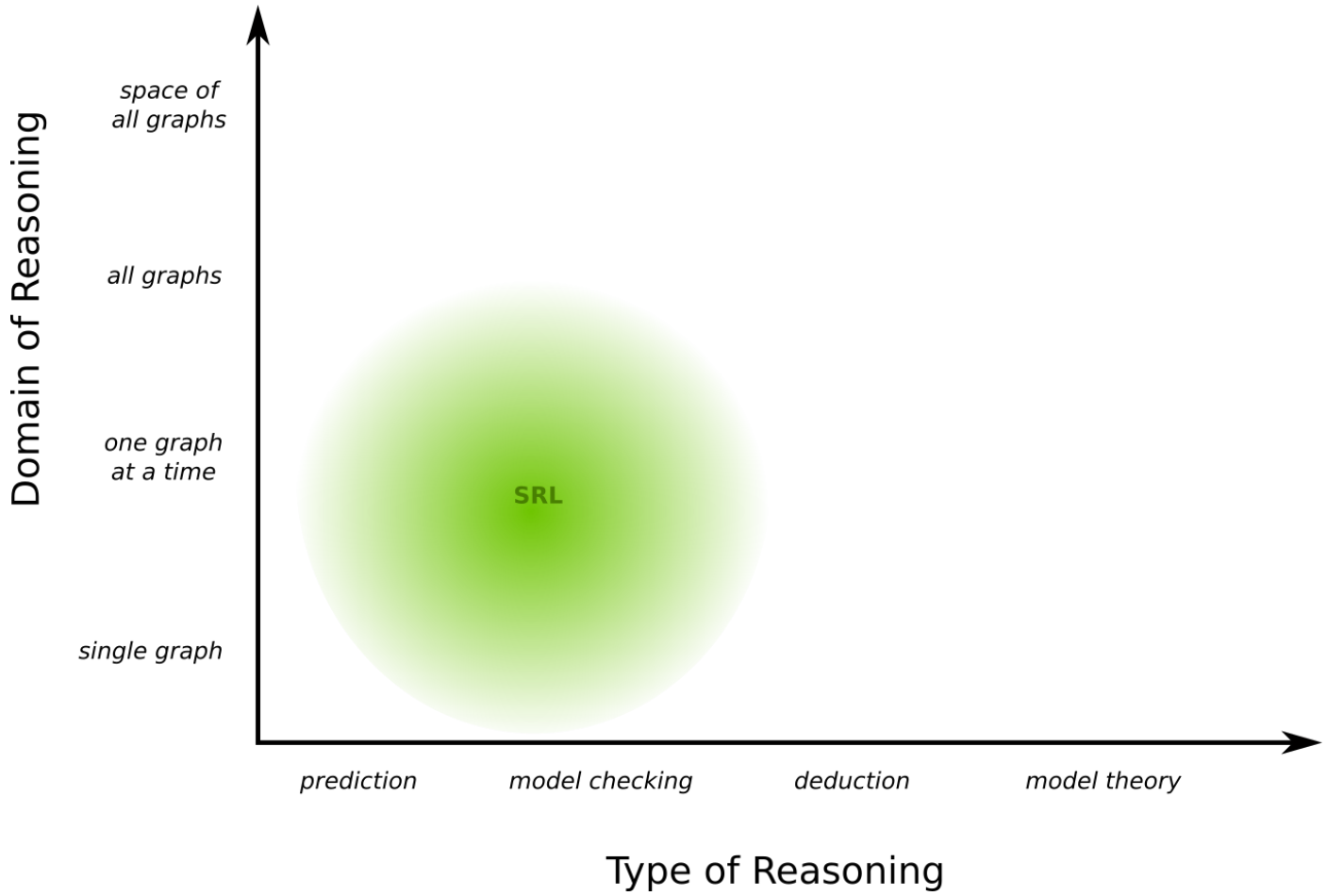
Special case: prediction

E.g.: $\mathcal{R} = \{node\_label, edge\} \cup \mathcal{A}$ (= node attributes).

Node classification then consists of queries

$$V = \{1, \ldots, n\} \quad P_V(node\_label(i)|B),$$

where B is a complete specification of $\{edge\} \cup \mathcal{A}$.



Model checking: checking properties (conditional probabilities) of a particular model (probability distribution P_V).

[Halpern, J. Y., & Vardi, M. Y. (1991). Model checking vs. theorem proving: a manifesto. In Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy]

Relational Bayesian Networks

Chain Rule

For fixed V, P_V is a distribution over values $\boldsymbol{R} = (R_1, \ldots, R_r)$. Let $R_{1:h} := (R_1, \ldots, R_h)$. This distribution can be factored as

$$P_V(\boldsymbol{R}) = P_V(R_1) \cdot P_V(R_2|R_1) \cdot \ldots \cdot P_V(R_h|R_{1:h-1}) \cdot \ldots \cdot P_V(R_r|R_{1,r-1}).$$

Conditional independence of relations

Conditional independencies lead to simplifications:

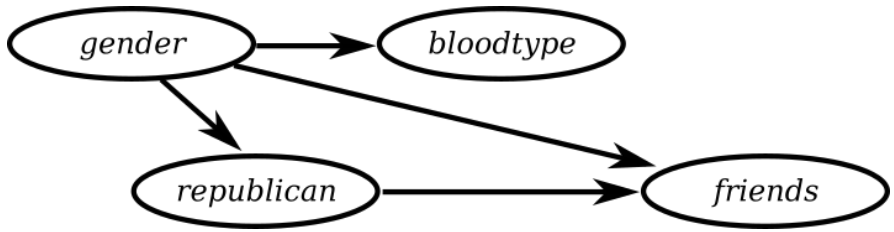
$$P_V(R_h|R_{1:h-1}) = P_V(R_h|Pa(R_h)) \quad \text{for some } Pa(R_h) \subset R_{1:h-1}$$

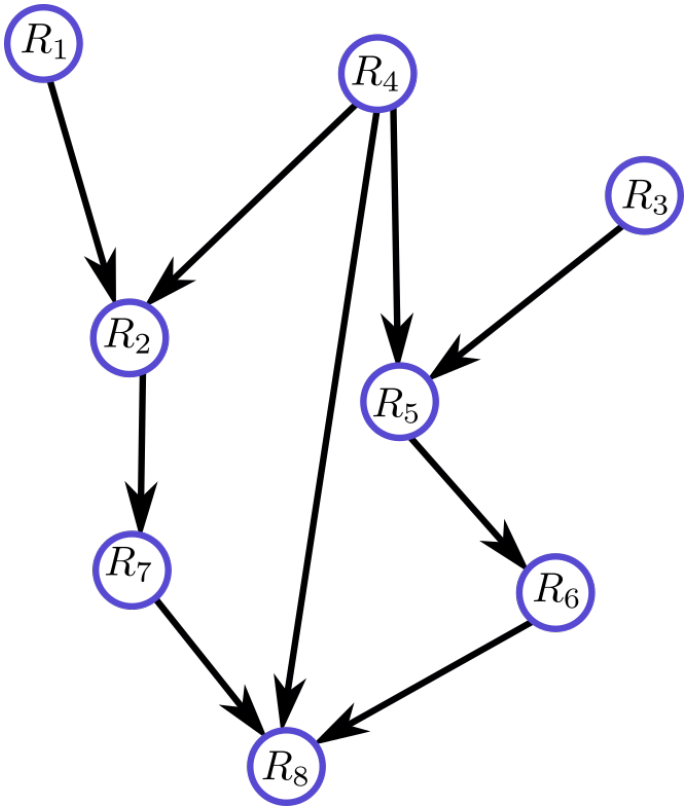
➥directed acyclic graph over relations (*relation DAG*).

$P_V(gender, republican, bloodtype, friends) =$

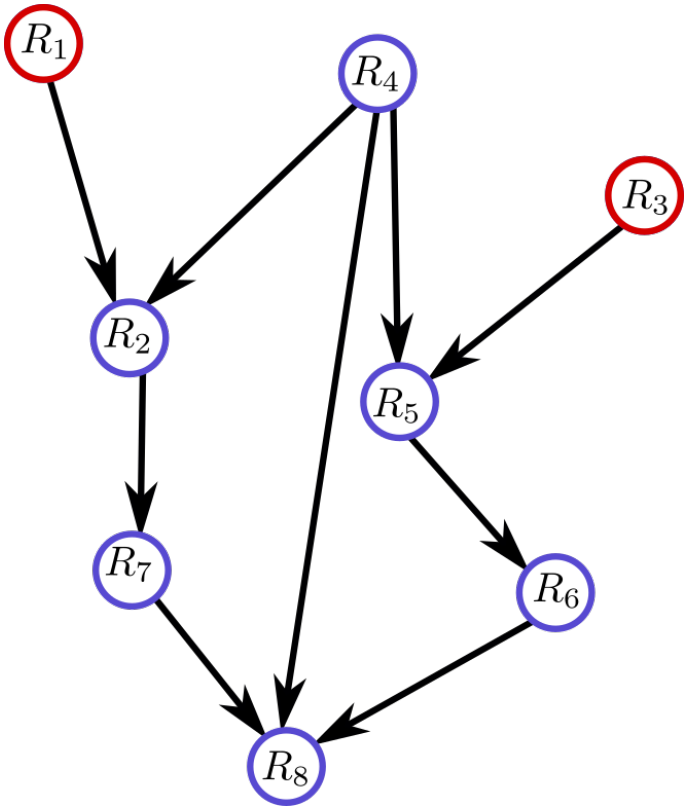
$P_V(gender)P_V(republican|gender)P_V(bloodtype|republican, gender)P_V(friends|bloodtype, republican, gender) \overset{\text{assume}}{=}$

$P_V(gender)P_V(republican|gender)P_V(bloodtype|gender)P_V(friends|republican, gender)$





- Defines full generative probabilistic model for graphs in signature $\mathcal{R}$

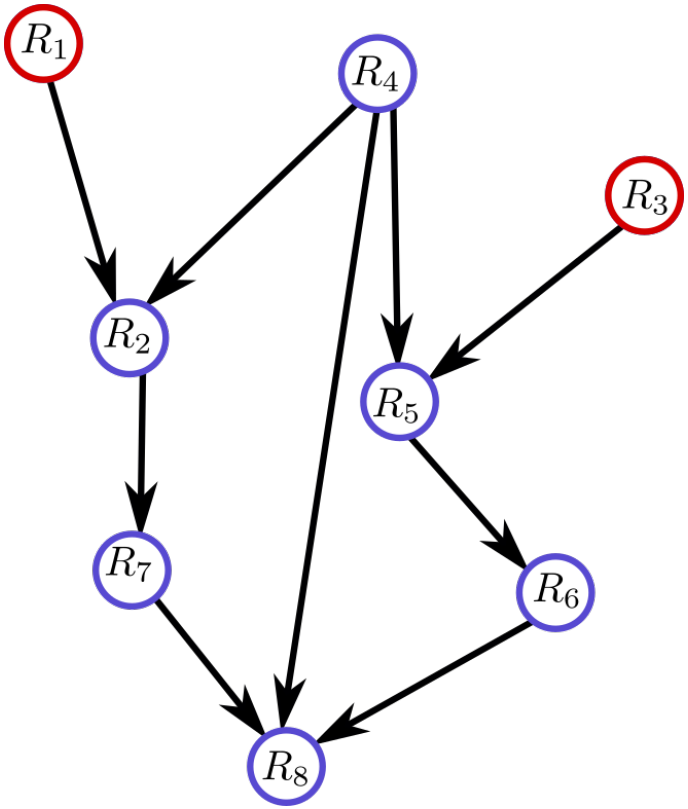


- Defines full generative probabilistic model for graphs in signature $\mathcal{R}$
- Sometimes: assume some relations $R \in \mathcal{R}$ are predefined input relations:

$$\mathcal{R} = \mathcal{R}_{prob} \cup \mathcal{R}_{in}$$

  - make these relations roots in the relation DAG
  - do not define a distribution $P_V(R_h)$ for these relations
  - defines a *conditional* distribution

$$P_V(\boldsymbol{R}_{prob} | \boldsymbol{R}_{in})$$



- Defines full generative probabilistic model for graphs in signature $\mathcal{R}$
- Sometimes: assume some relations $R \in \mathcal{R}$ are predefined input relations:

$$\mathcal{R} = \mathcal{R}_{prob} \cup \mathcal{R}_{in}$$

  - make these relations roots in the relation DAG
  - do not define a distribution $P_V(R_h)$ for these relations
  - defines a *conditional* distribution

$$P_V(\boldsymbol{R}_{prob} | \boldsymbol{R}_{in})$$

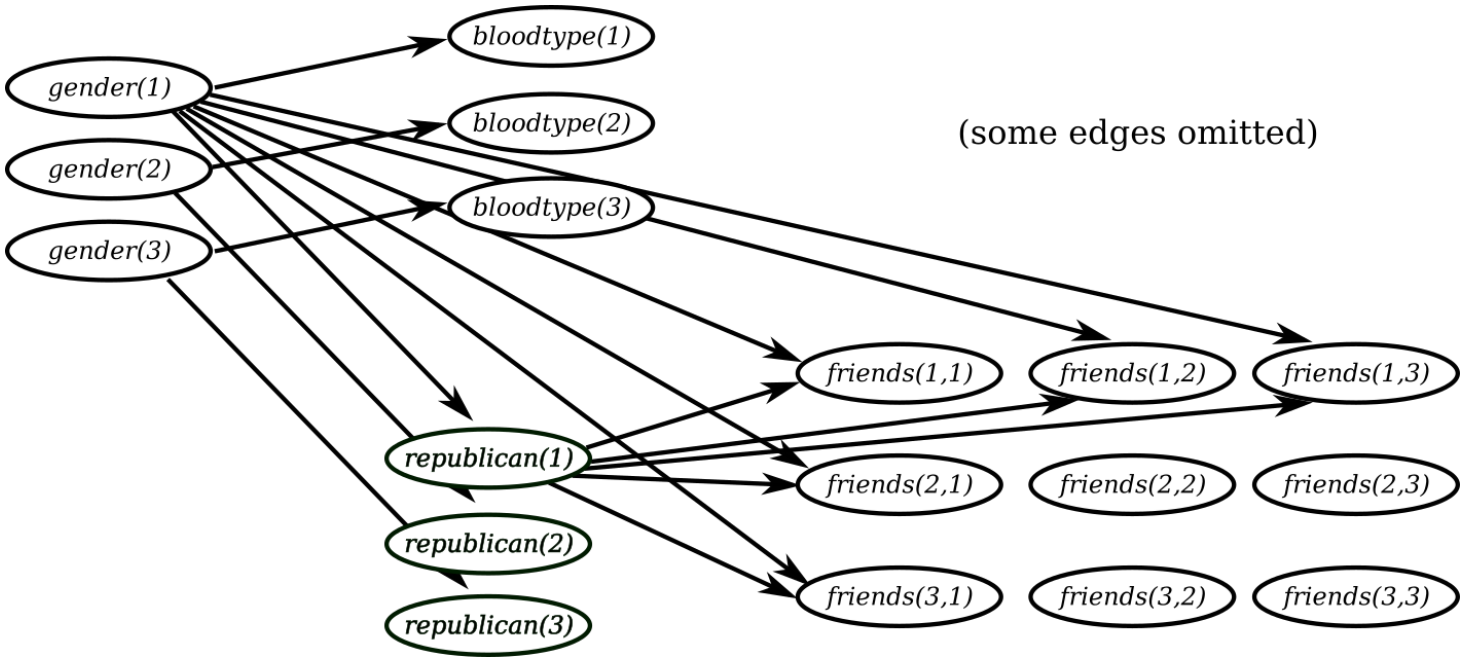
➥All SRL frameworks support divisions $\mathcal{R} = \mathcal{R}_{prob} \cup \mathcal{R}_{in}$

Atom independence

Assume atoms of one relation are mutually independent, given the parent relations:

$$P_V(R_h|Pa(R_h)) := \prod_{\boldsymbol{i} \in V^{arity(R_h)}} P_V(R_h(\boldsymbol{i})|Pa(R_h))$$

As a Bayesian network:



➥Leads to limitations for modeling e.g. symmetry constaints $friends(1,2) \Leftrightarrow friends(2,1)$, or homophily (exist modeling tricks to circumvent this!).

A relational Bayesian network for signature $\mathcal{R}$ consists of

- a directed acyclic graph whose nodes are the relations $R \in \mathcal{R}$,
- for each $R \in \mathcal{R}$ a *probability formula F_R in the signature $Pa(R)$* that defines the conditional probabilities

$$P_V(R(\boldsymbol{i})|Pa(R))$$

Probability formulas: semantics

A probability formula F maps tuples of entities $\boldsymbol{i}$ in a graph $G = (V, \boldsymbol{R})$ to a real number:

$$((V, \boldsymbol{R}), \boldsymbol{i}) \quad \mapsto \quad eval(F, \boldsymbol{i}, G) \in [0, 1]$$

[M. Jaeger: Relational Bayesian Networks. UAI 1997]

Constants

For any $q \in [0, 1]$,

$$F \equiv q$$

is a probability formula with

$$eval(F, \boldsymbol{i}, G) = q$$

for all $\boldsymbol{i}, G$.

Example

Let $\mathcal{R} = \{edge\}$. Then

$$F_{edge(X,Y)} \equiv 0.5$$

defines the classic Erdős-Rényi random graph model.

Atoms

For any $R \in \mathcal{R}$, and variables $Y_1, \ldots, Y_{arity(R)}$

$$F \equiv R(Y_1, \ldots, Y_{arity(R)})$$

is a probability formula with

$$eval(F, \boldsymbol{i}, G) = \begin{cases} 1 & \text{if } R(\boldsymbol{i}) \text{ is true in } G \\ 0 & \text{if } R(\boldsymbol{i}) \text{ is false in } G \end{cases}$$

WIF-THEN-ELSE

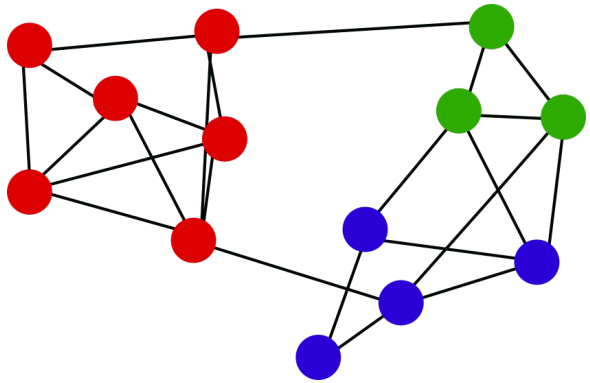
If F_1, F_2, F_3 are probability formulas, then

$$F \equiv \texttt{WIF} \quad F_1 \quad \texttt{THEN} \quad F_2 \quad \texttt{ELSE} \quad F_3$$

is a probability formula with

$$eval(F, \boldsymbol{i}, G) = eval(F_1, \boldsymbol{i}, G)eval(F_2, \boldsymbol{i}, G) + (1 - eval(F_1, \boldsymbol{i}, G))eval(F_3, \boldsymbol{i}, G)$$

➥Generalization of Boolean operations ($F_i \in \{0, 1\}$)



- Nodes partitioned into *blocks*
- Probability of edges depends on block memberships

With the constructs introduced so far:

A. partitioning into red, green, blue nodes:

$$
\begin{aligned}
F_{red(X)} &\equiv 0.5 \\
F_{blue(X)} &\equiv \texttt{WIF } red(X) \texttt{ THEN } 0 \texttt{ ELSE } 0.7 \\
F_{green(X)} &\equiv \texttt{WIF } red(X) \vee blue(X) \texttt{ THEN } 0 \texttt{ ELSE } 1.0
\end{aligned}
$$

B. generating edges:

$$F_{edge(X,Y)} \equiv \texttt{WIF } red(X) \wedge red(Y) \texttt{ THEN } 0.6 \texttt{ ELSEIF } \ldots$$

(In standard SBM: block membership not given by observable attribute, but by latent variable)

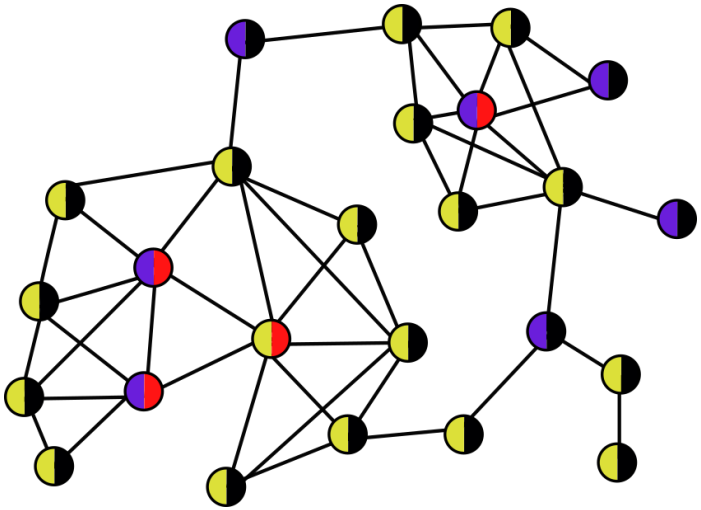
Combination Function

(related to first-order quantifiers $\forall, \exists$, GNN message passing aggregation, ...)

If $F_1, \ldots, F_t$ are probability formulas, then

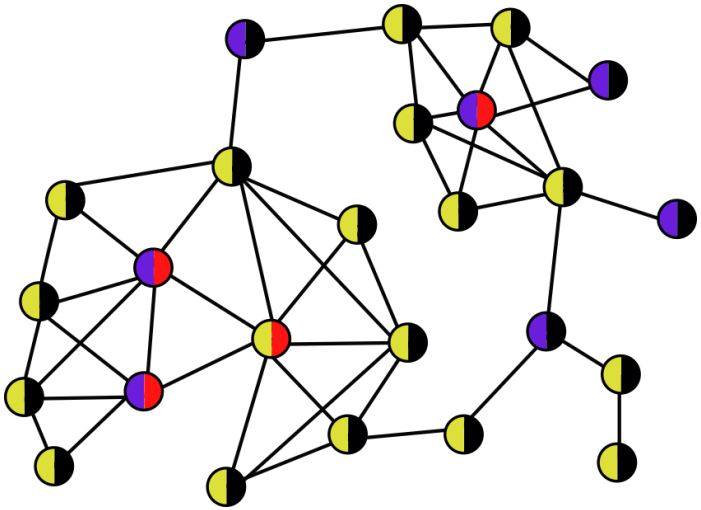
$F \equiv$ `COMBINE` $F_1, \ldots, F_t$
`WITH` $<$ *combination function* $>$
`FORALL` $<$ *variables* $>$
`WHERE` $<$ *Boolean* $\mathcal{R}_{in}$ *condition* $>$

is a probability formula.



$P_V(red(i))$ higher if

- i is *blue*
- i is part of many triangles

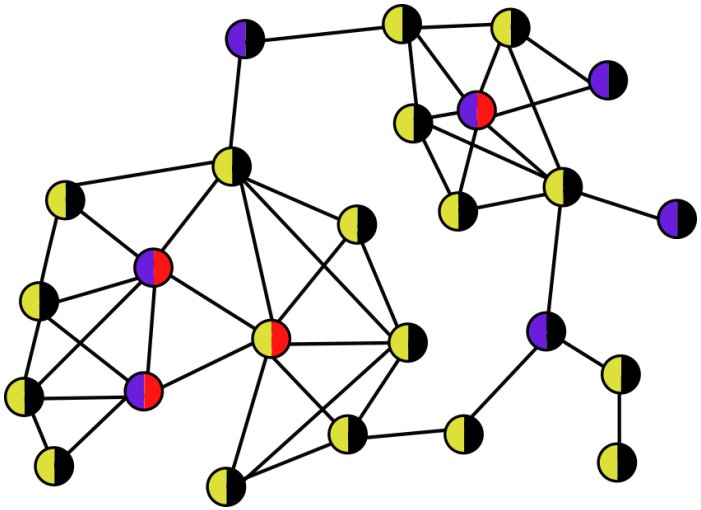


$P_V(red(i))$ higher if

- i is *blue*
- i is part of many triangles

Defining triangles:

$$F_{triangle(X,Y,Z)} \equiv$$
$$edge(X,Y) \wedge edge(X,Z) \wedge edge(Y,Z)$$



$P_V(red(i))$ higher if

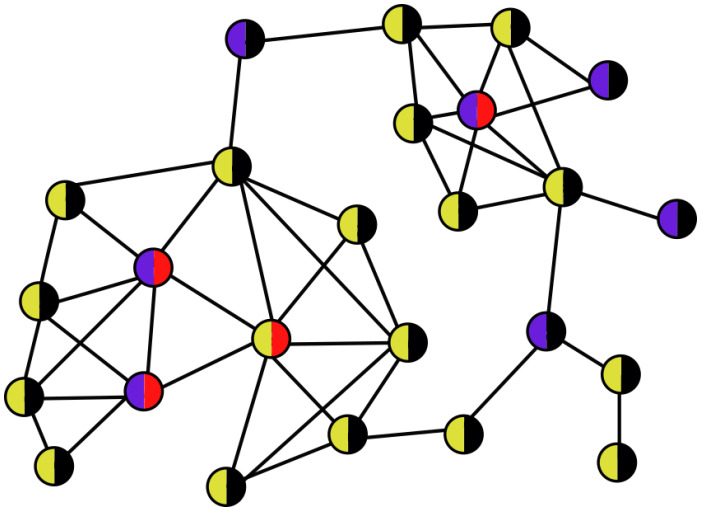
- i is *blue*
- i is part of many triangles

Defining triangles:

$$F_{triangle(X,Y,Z)} \equiv$$
$$edge(X, Y) \wedge edge(X, Z) \wedge edge(Y, Z)$$

Counting triangles:

$$F_{triangle\_count(X)} \equiv$$
$$\texttt{COMBINE}\ 1.0$$
$$\texttt{WITH}\ sum$$
$$\texttt{FORALL}\ Y, Z$$
$$\texttt{WHERE}\ F_{triangle(X,Y,Z)}(X, Y, Z)$$



$P_V(red(i))$ higher if

- i is *blue*
- i is part of many triangles

Defining triangles:

$$F_{triangle(X,Y,Z)} \equiv$$
$$edge(X,Y) \wedge edge(X,Z) \wedge edge(Y,Z)$$

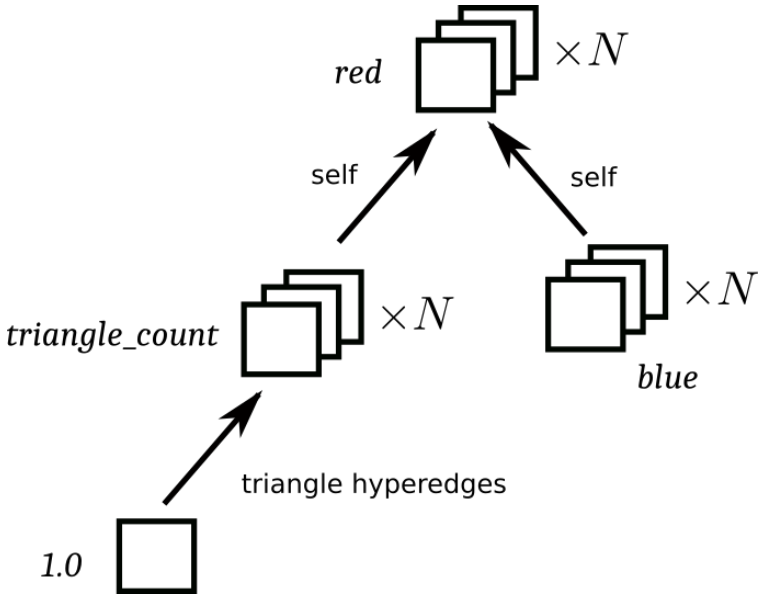
Counting triangles:

$$F_{triangle\_count(X)} \equiv$$
$$\texttt{COMBINE}\ 1.0$$
$$\texttt{WITH}\ sum$$
$$\texttt{FORALL}\ Y, Z$$
$$\texttt{WHERE}\ F_{triangle(X,Y,Z)}(X,Y,Z)$$

Logistic regression of *triangle_count* and *blue* feature:

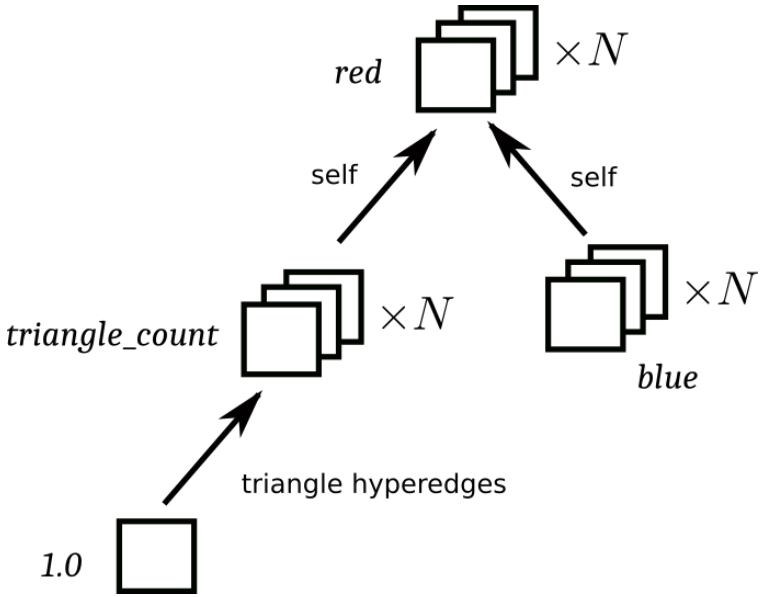
$$F_{red(X)} \equiv$$
$$\texttt{COMBINE}\ \ 0.6 \cdot F_{triangle\_count(X)}(X),$$
$$0.3 \cdot blue(X),$$
$$-3.0$$
$$\texttt{WITH}\ logistic\ regression$$

The computation graph of a probability formula:



- Each probability (sub-)formula defines a feature of $0, 1, 2, \ldots$-tuples of entities
- Nested formulas give “deep” models
- Aggregation along “Boolean hyperedges”

The computation graph of a probability formula:



- Each probability (sub-)formula defines a feature of $0, 1, 2, \ldots$-tuples of entities
- Nested formulas give "deep" models
- Aggregation along "Boolean hyperedges"
- A single probability formula defines a scalar feature
- Use multiple formulas in parallel for vector features

RBNs: Pros and Cons

+ Natural procedural, causal, temporal, ... models
+ Often: statistically interpretable model parameters
+ Can use expert knowledge or machine learning to construct model
- Limited structure learning capabilities
- Awkward for modeling undirected dependencies