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

RBN-GNN Integration

Manfred Jaeger

Aalborg University

Semantics

Directed SRL

Overall SRL semantics

$$V \mapsto P_V \in \Delta\mathcal{G}(V, \mathcal{R})$$

via chain rule and atom independence assumption reduced to functions

$$F_R : (V, Pa(R), \boldsymbol{i}) \mapsto P_V(R(\boldsymbol{i})|Pa(R_h)) \in [0, 1]$$

Inductive GNNs

Node embedding functions:

$$E : (V, \boldsymbol{R}, i) \mapsto \boldsymbol{h}^m(i) \in \mathbb{R}^d$$

Common ground

Abstracting from some non-fundamental differences (vectors vs. scalars, $[0, 1]$ vs. $\mathbb{R}$):

- A GNN is a conditional SRL model for a single probabilistic relation
- Multiple GNNs could define a full generative model (but without support of model checking inference)
- Expressivity depends on SRL framework and GNN architecture to define functions F_R, resp. E.

Expressivity

Remember:

> *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. [Barceló et al.,2020]*
>
> *Every property (node, edge, hyperedge,...) that can be expressed in* **first-order logic** *can be captured by a probability formula [Jaeger, 1997]*

Next up: every ACR-GNN can be expressed by probability formulas.

GNN-2-RBN

From [Barceló et al.,2020]:

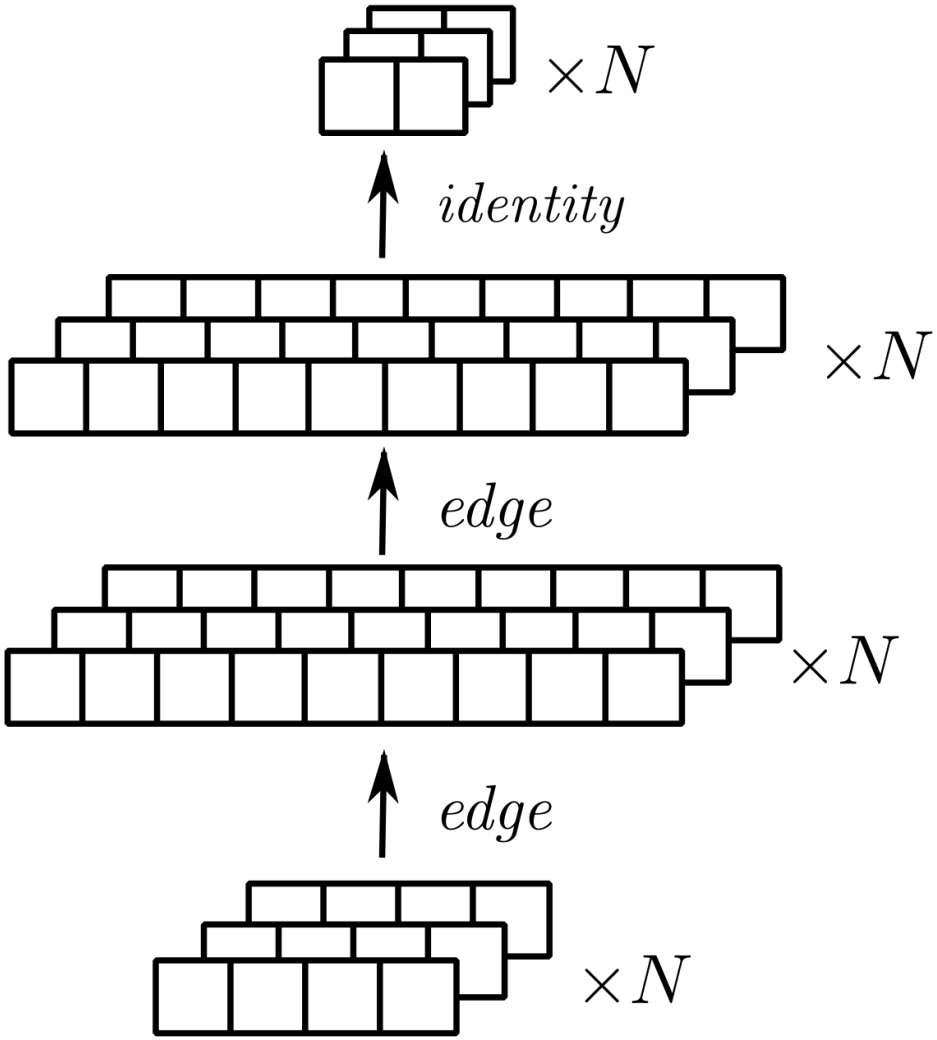
Input graphs defined by signature:

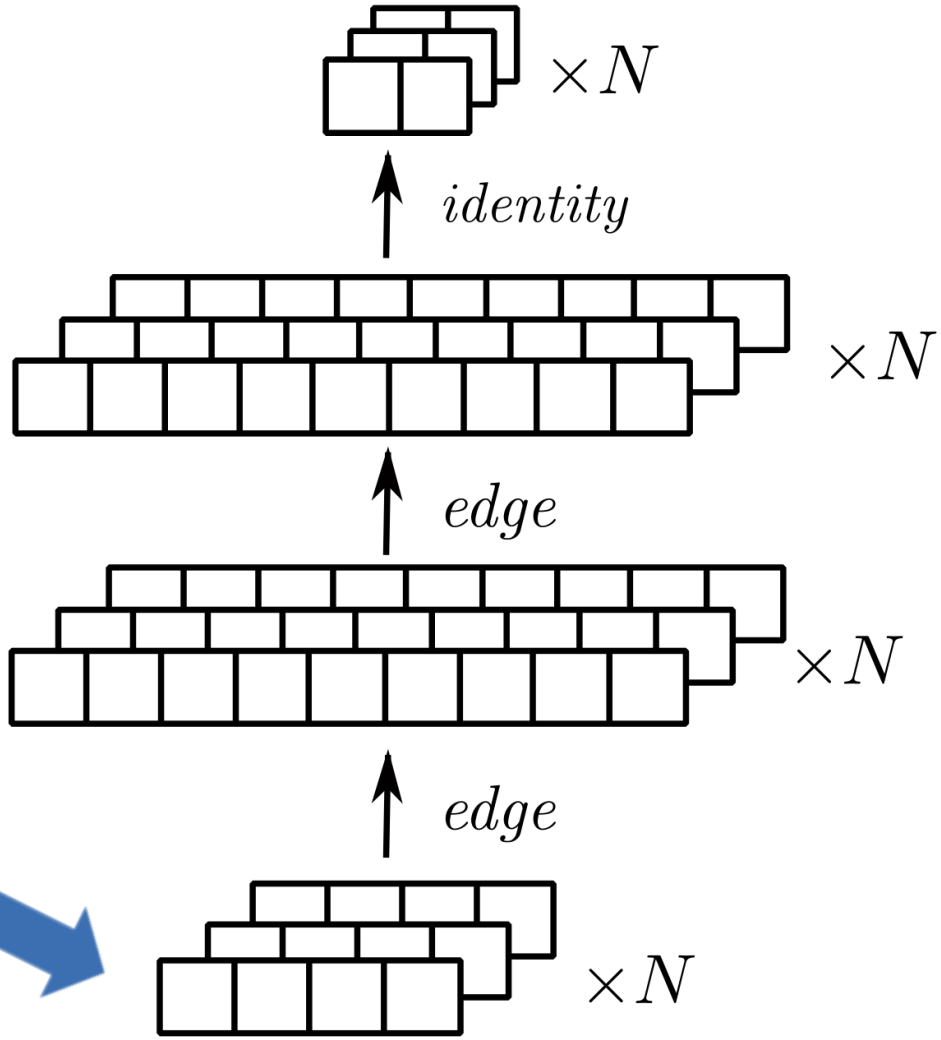
$$\mathcal{R}_{in} = \{blue, green, red, yellow, purple, edge\}$$

Target concept to represent/learn:

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

(in two-variable fragment of first-order logic with counting quantifiers)


$\times N$
$identity$
$\times N$
$edge$
$\times N$
$edge$
$\times N$



Node attributes

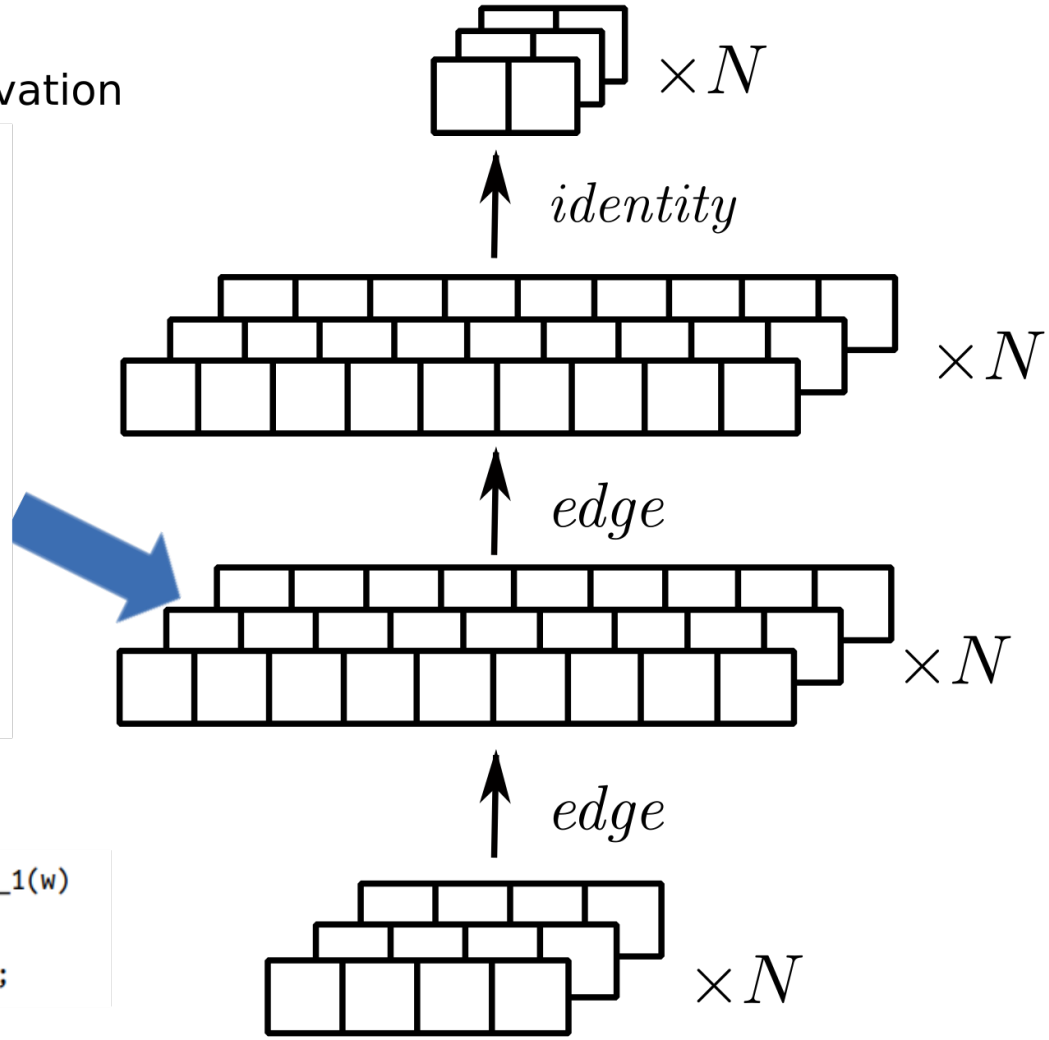
```
@layer_0_0([node]v)= blue(v);
@layer_0_1([node]v)= green(v);
@layer_0_2([node]v)= red(v);
@layer_0_3([node]v)= yellow(v);
@layer_0_4([node]v)= purple(v);
```

Linear combination and activation

```
@layer_1_0([node]v)= COMBINE
          ($c_1_0_0*@layer_0_0(v)),
          ($c_1_0_1*@layer_0_1(v)),
          ($c_1_0_2*@layer_0_2(v)),
          ($c_1_0_3*@layer_0_3(v)),
          ($c_1_0_4*@layer_0_4(v)),
          ($A_1_0_0*@agg_0_0(v)),
          ($A_1_0_1*@agg_0_1(v)),
          ($A_1_0_2*@agg_0_2(v)),
          ($A_1_0_3*@agg_0_3(v)),
          ($A_1_0_4*@agg_0_4(v)),
          ($R_1_0_0*@read_0_0()),
          ($R_1_0_1*@read_0_1()),
          ($R_1_0_2*@read_0_2()),
          ($R_1_0_3*@read_0_3()),
          ($R_1_0_4*@read_0_4()),
          $b_1_0
          WITH l-reg
```

Aggregation

```
@agg_0_1([node]v) = COMBINE @layer_0_1(w)
          WITH sum
          FORALL w
          WHERE (edge(v,w)|edge(w,v));
```

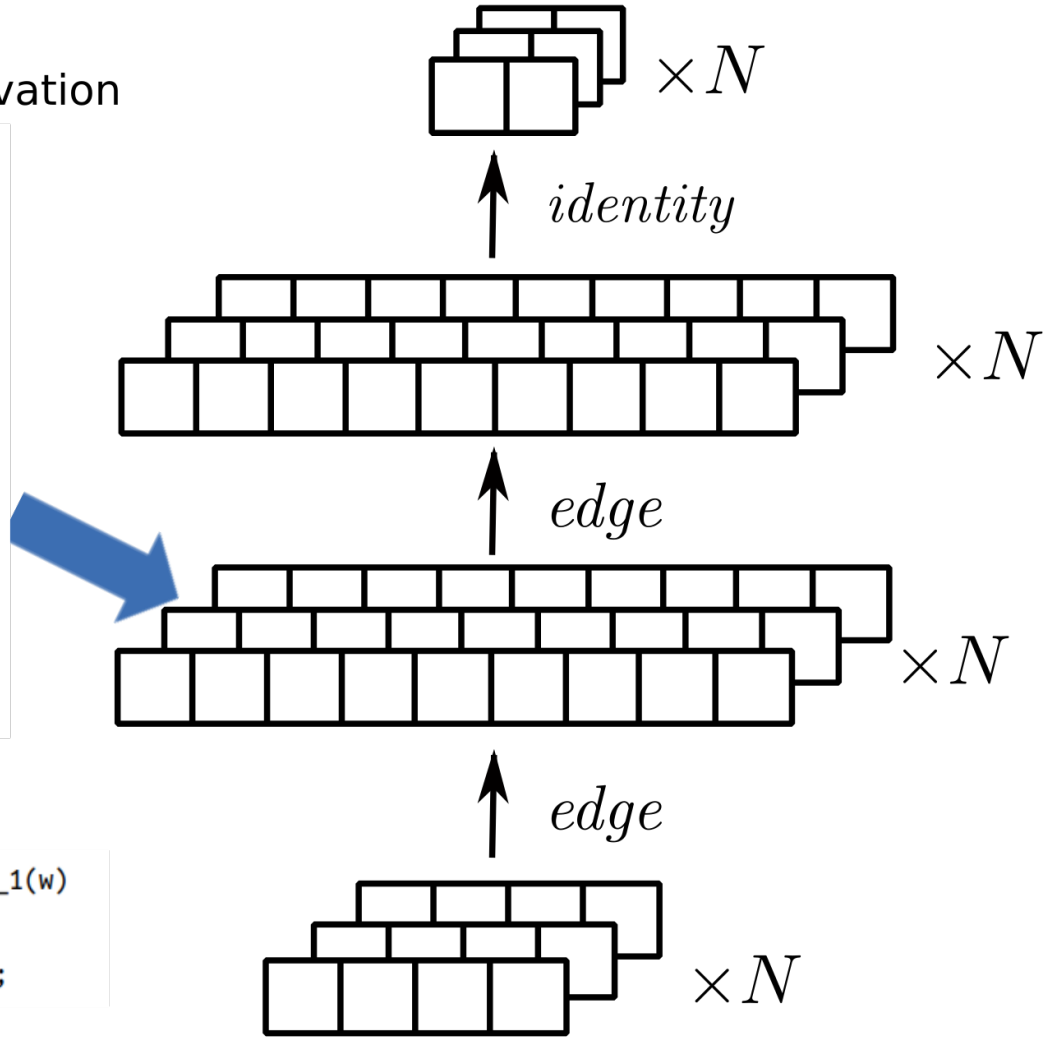


Linear combination and activation

```
@layer_1_0([node]v)= COMBINE
          ($c_1_0_0*@layer_0_0(v)),
          ($c_1_0_1*@layer_0_1(v)),
          ($c_1_0_2*@layer_0_2(v)),
          ($c_1_0_3*@layer_0_3(v)),
          ($c_1_0_4*@layer_0_4(v)),
          ($A_1_0_0*@agg_0_0(v)),
          ($A_1_0_1*@agg_0_1(v)),
          ($A_1_0_2*@agg_0_2(v)),
          ($A_1_0_3*@agg_0_3(v)),
          ($A_1_0_4*@agg_0_4(v)),
          ($R_1_0_0*@read_0_0()),
          ($R_1_0_1*@read_0_1()),
          ($R_1_0_2*@read_0_2()),
          ($R_1_0_3*@read_0_3()),
          ($R_1_0_4*@read_0_4()),
          $b_1_0
          WITH l-reg
```

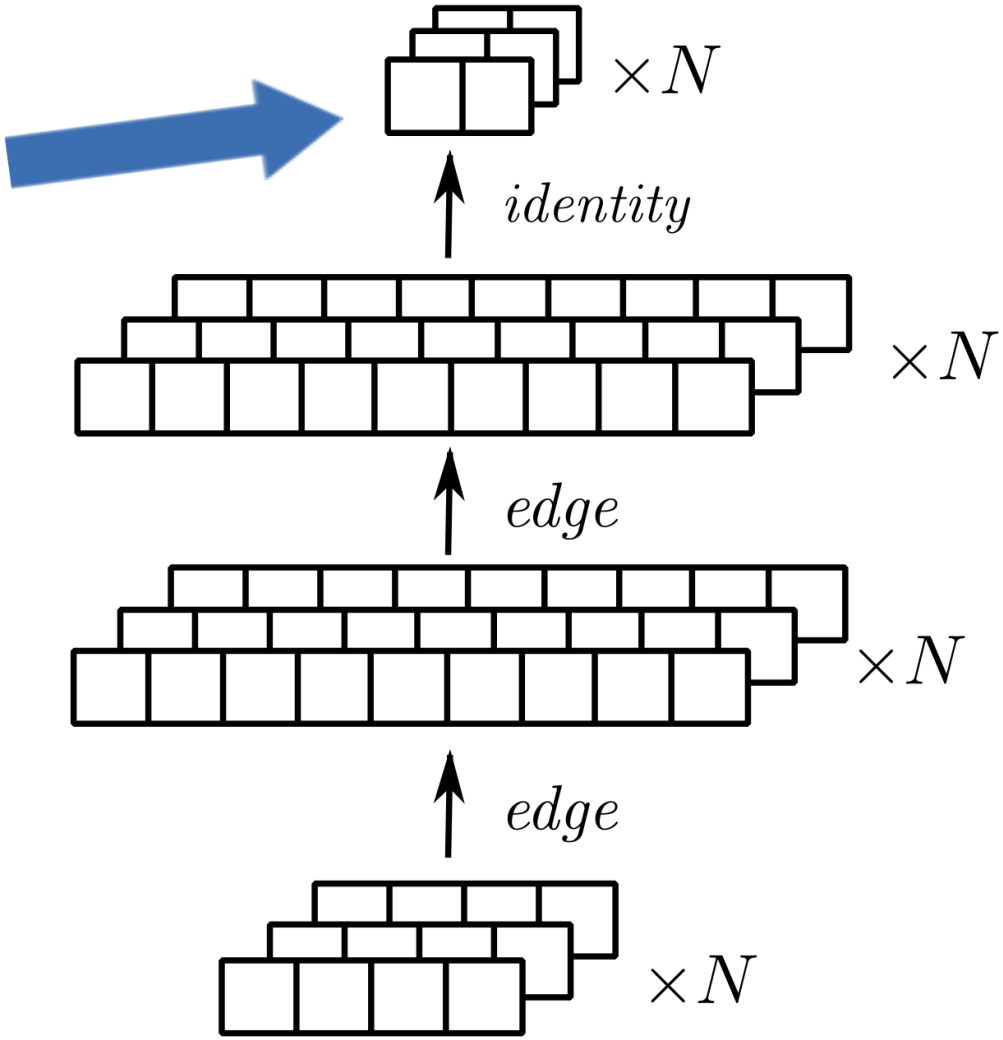
Aggregation

```
@agg_0_1([node]v) = COMBINE @layer_0_1(w)
          WITH sum
          FORALL w
          WHERE (edge(v,w)|edge(w,v));
```

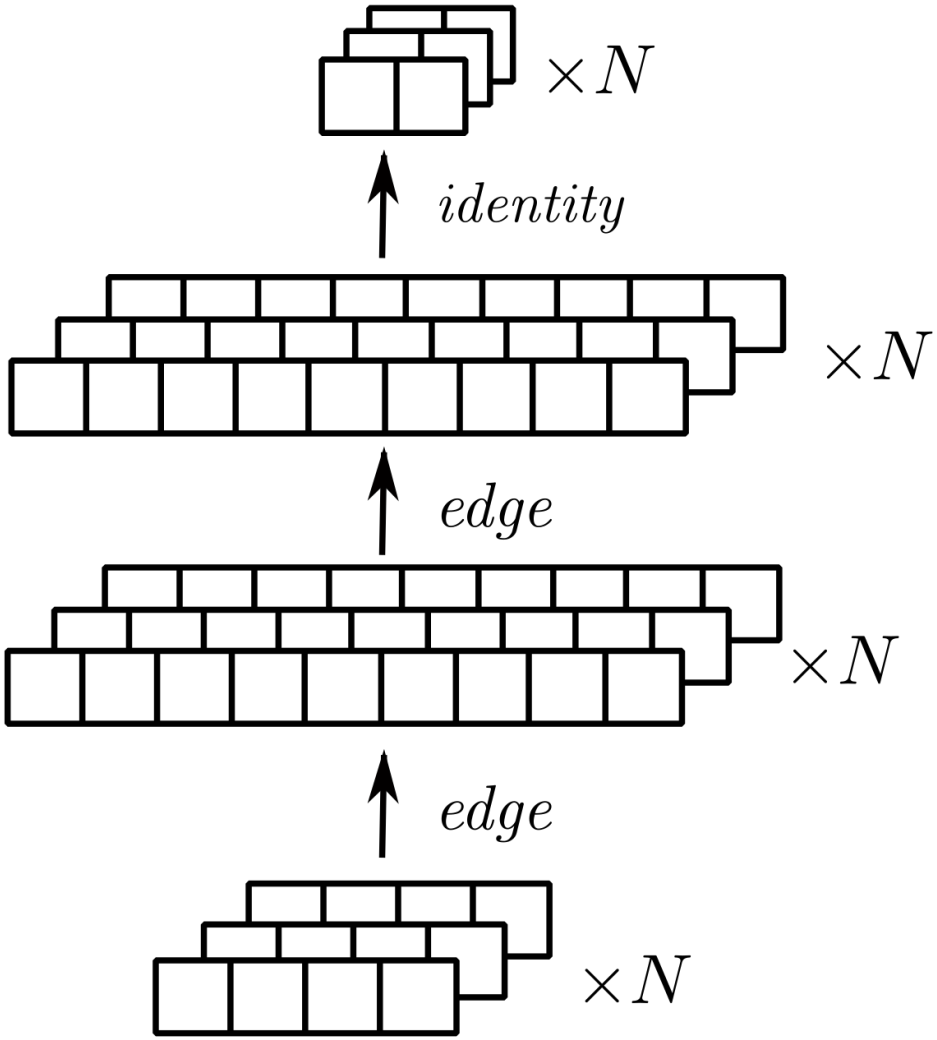


Linear classifier

```
alpha1([node]v)= COMBINE
          ($w_0*@layer_1_0(v)),
          ($w_1*@layer_1_1(v)),
          ($w_2*@layer_1_2(v)),
          ($w_3*@layer_1_3(v)),
          $w_5
          WITH l-reg
```



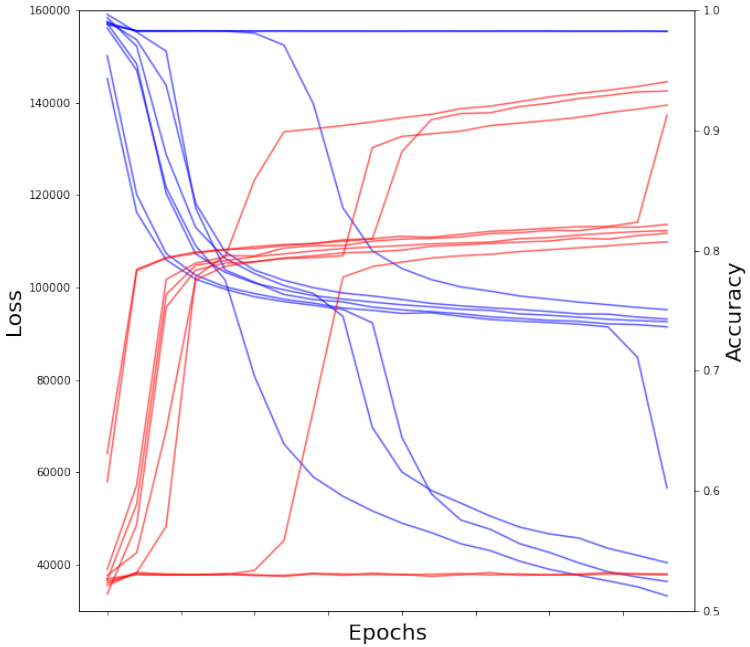
- One-to-one mapping of representation
- Matrix-vector level specifications broken down to the “scalar” level
- GNN training $\sim$ RBN learning (same objective, same gradients, . . .)



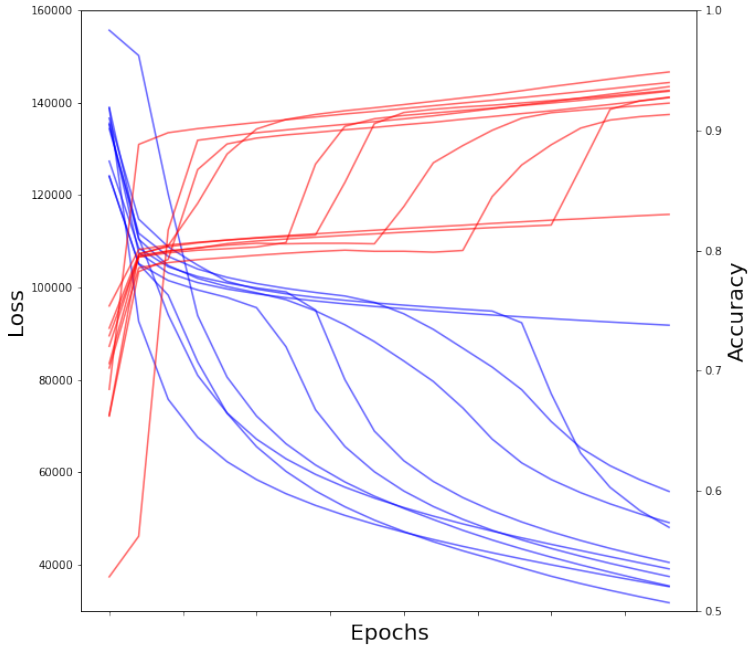
Learning the α_1 target.

Training data: 5000 random graphs of size $N \in 40..50$ (data from [Barceló et al.]).

Pytorch geometric implementation of ACR-GNN:



Primula implementation of RBN encoding:

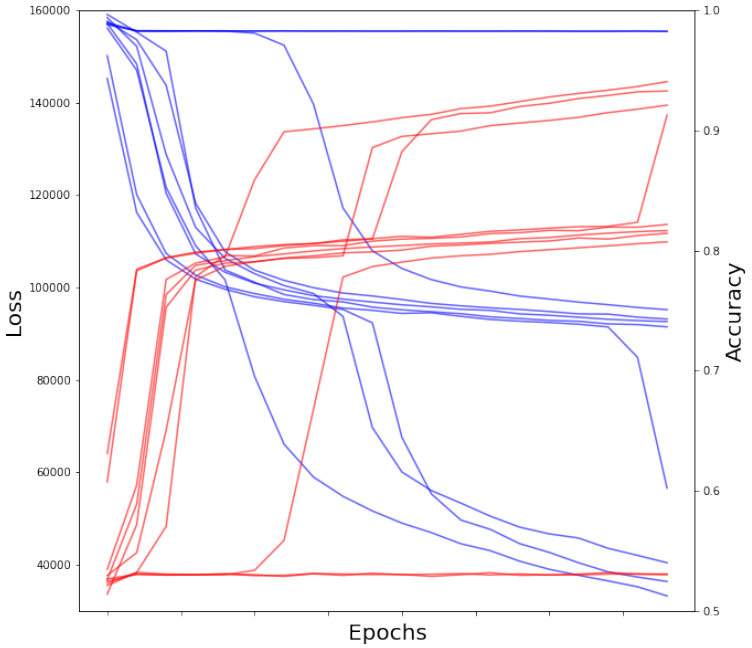


(blue: loss, red: accuracy (on training data); 20 epochs, 10 restarts with random parameter initializations)

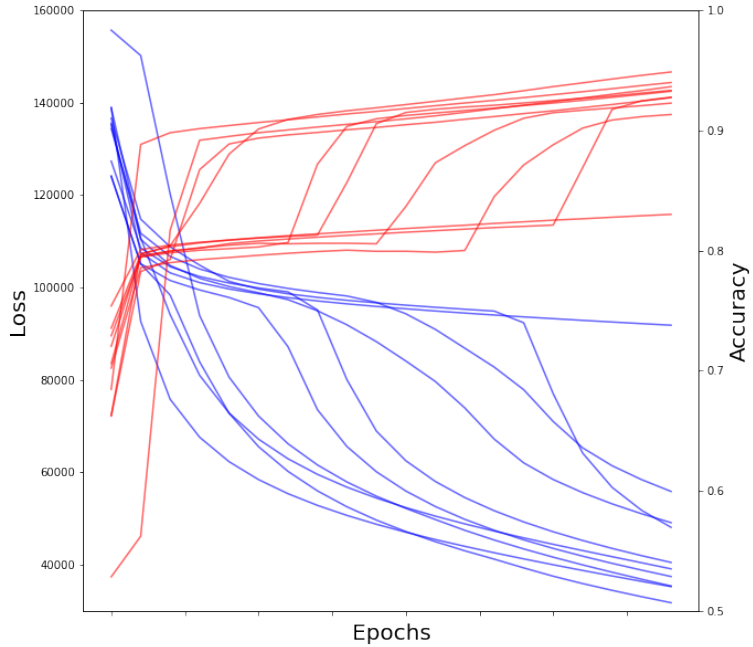
Learning the α_1 target.

Training data: 5000 random graphs of size $N \in 40..50$ (data from [Barceló et al.]).

Pytorch geometric implementation of ACR-GNN:



Primula implementation of RBN encoding:



(blue: loss, red: accuracy (on training data); 20 epochs, 10 restarts with random parameter initializations)

But: Primula takes **much** longer ...

Neuro-Symbolic Integration

Low vs. high level

[R. Manhaeve et al.: DeepProbLog: Neural Probabilistic Logic Programming, 2018]:

- Neural: low-level perception
- Symbolic: high-level reasoning (logic, probabilities)

On the other side:

- High-level reasoning can be implemented via low level, high-dimensional optimization.
  e.g. [Cameron et al.: Predicting Propositional Satisfiability via End-to-End Learning, 2020]

Integration

- Heterogeneous integration; E.g. DeepProbLog: combining a neural and a symbolic component.
- Homogeneous integration: one seamless framework for all levels

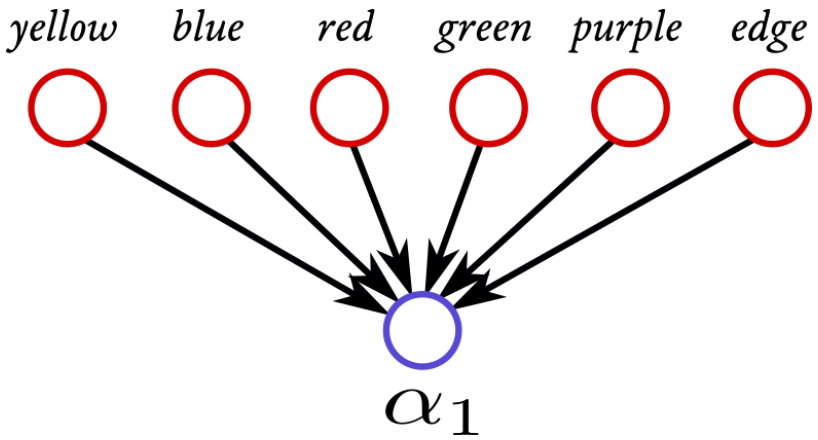
GNN encodings as probability formulas allow construction of models that seamlessly integrate probability formulas:

- that are defined (partly) by expert knowledge:
  - complex “logic” (recall SBM)
  - sparse parameterization
- that are entirely trained from (abundant) data:
  - generic structure (number and dimensions of layers)
  - high-dimensional parameterization

Learning and reasoning:

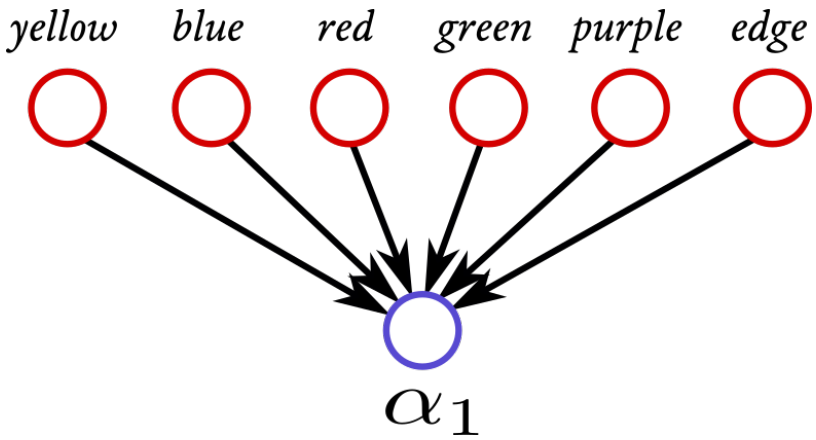
- Training of neural components can be outsourced to powerful GNN tools
- The resulting model is amenable to reasoning in an SRL framework

Making the α_1 model generative:

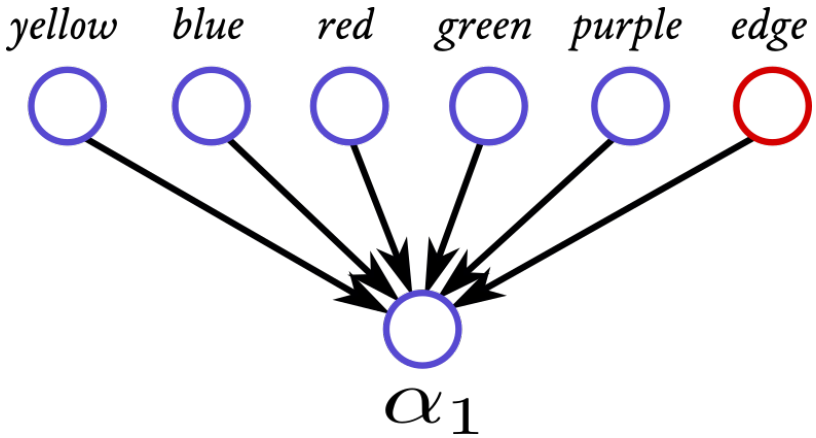


Conditional (prediction) model for α_1 given all other relations as input.

Making the α_1 model generative:



Conditional (prediction) model for α_1 given all other relations as input.



Generative model for node attributes and label, given *edge* as input

$F_{yellow(X)} = 0.18;$

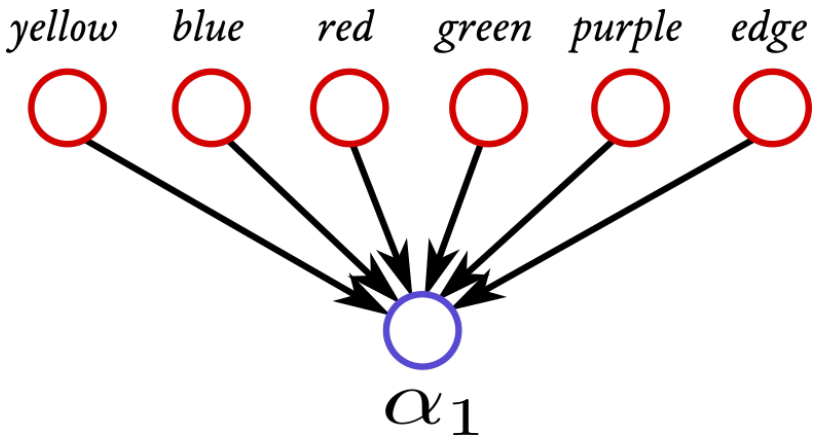
$F_{blue(X)} = 0.26;$

$F_{red(X)} = 0.18;$

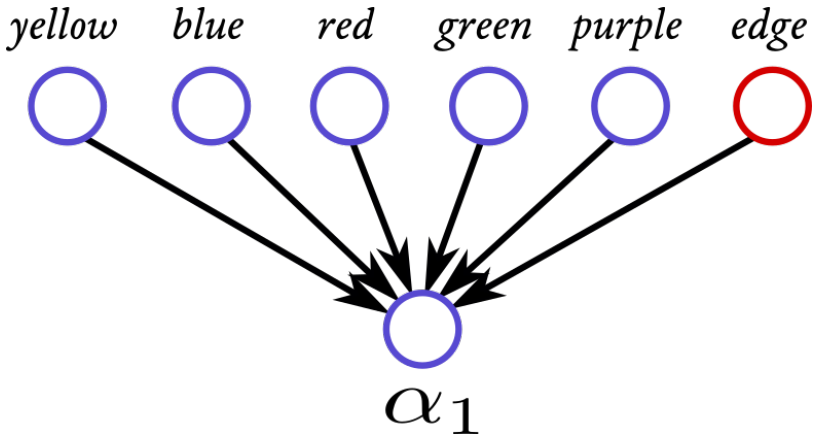
$F_{green(X)} = 0.18;$

$F_{purple(X)} = 0.18;$

Making the α_1 model generative:



Conditional (prediction) model for α_1 given all other relations as input.



Generative model for node attributes and label, given *edge* as input

$F_{yellow(X)} = 0.18;$

$F_{blue(X)} = 0.26;$

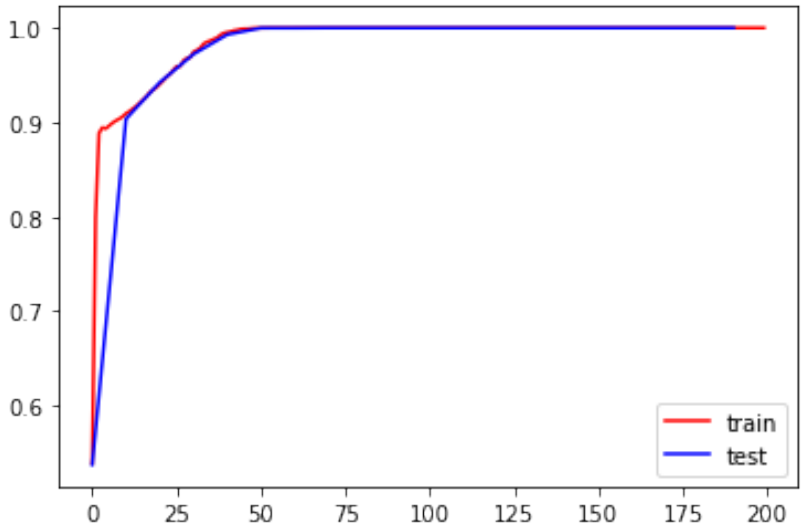
$F_{red(X)} = 0.18;$

$F_{green(X)} = 0.18;$

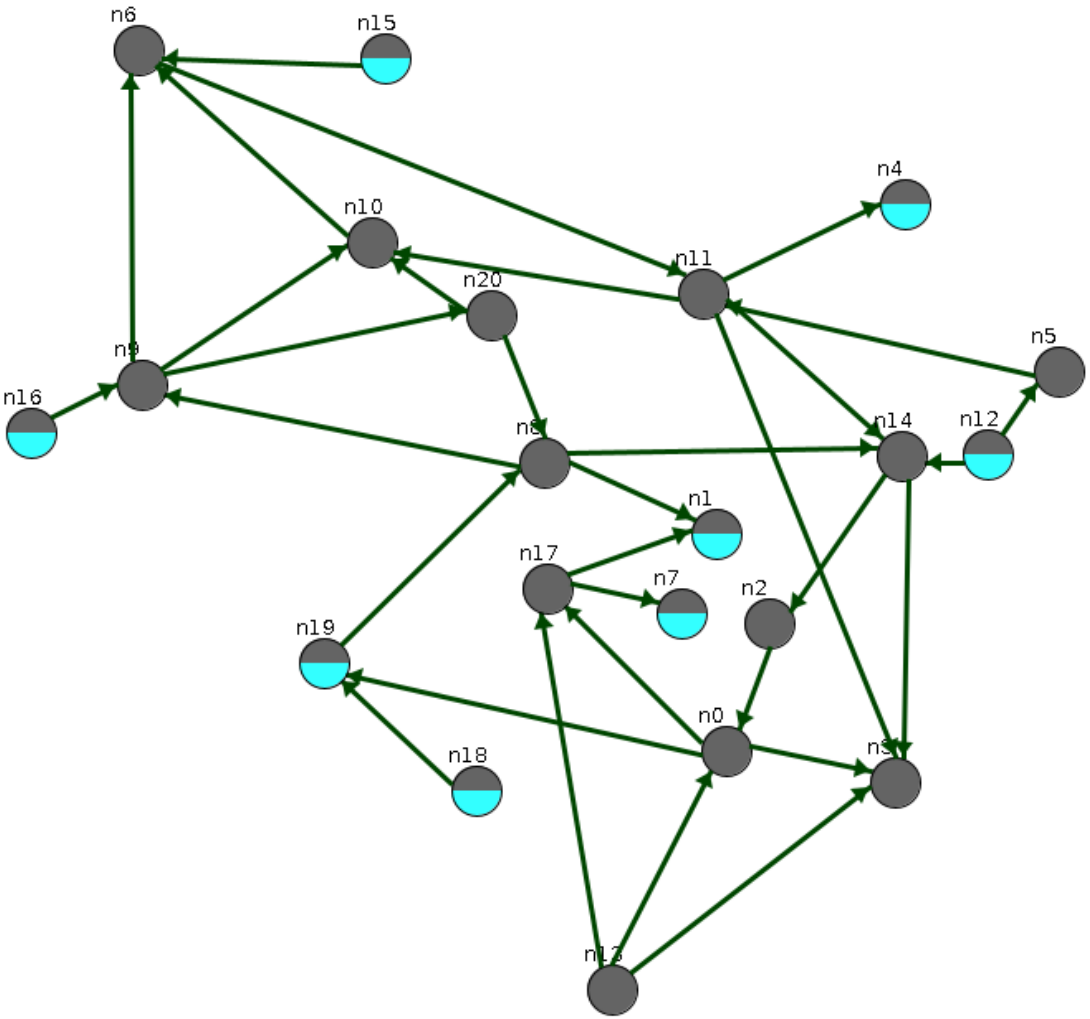
$F_{purple(X)} = 0.18;$

MAP task: given observed α_1 labels, what is the most probable configuration of the *blue* attribute?

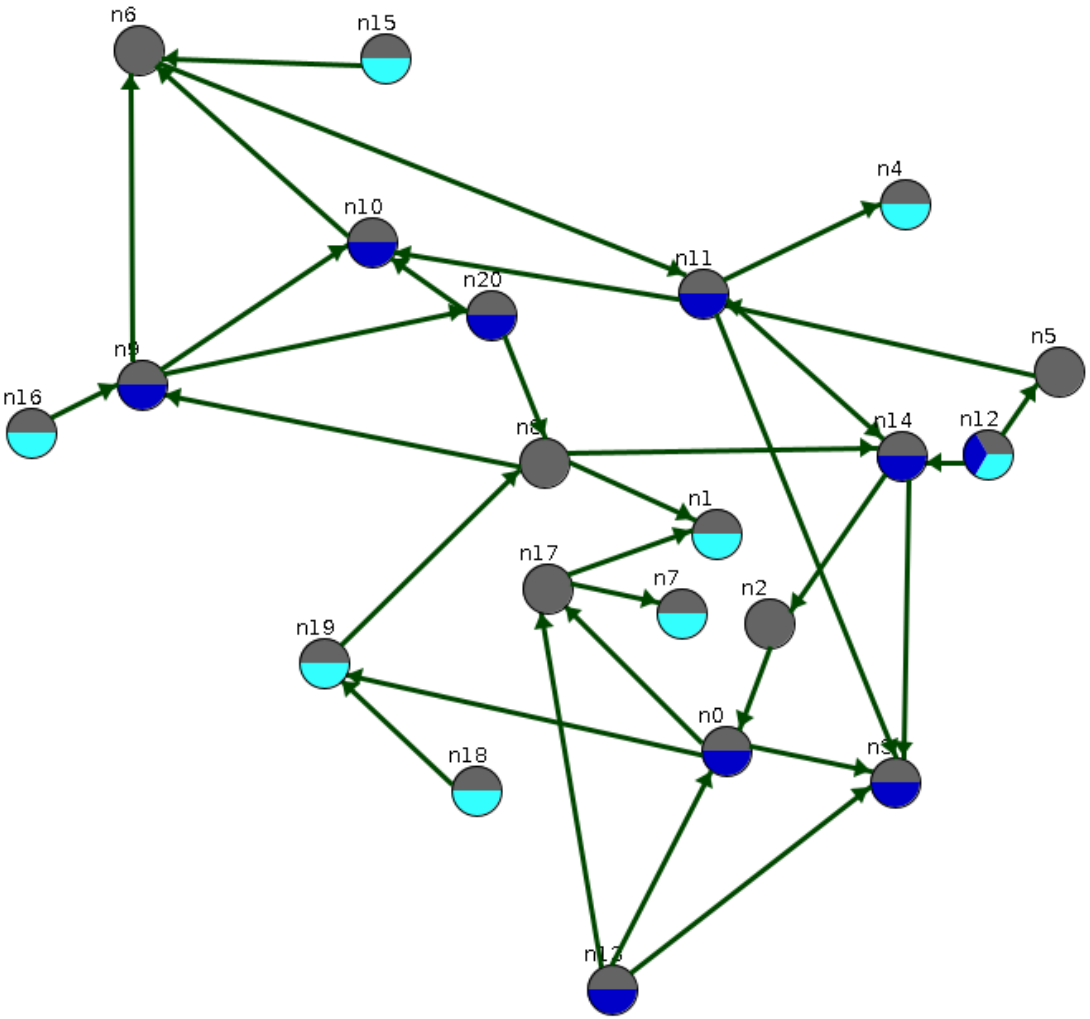
Training predictive model for α_1 given attributes for 200 epochs (PyTorch):



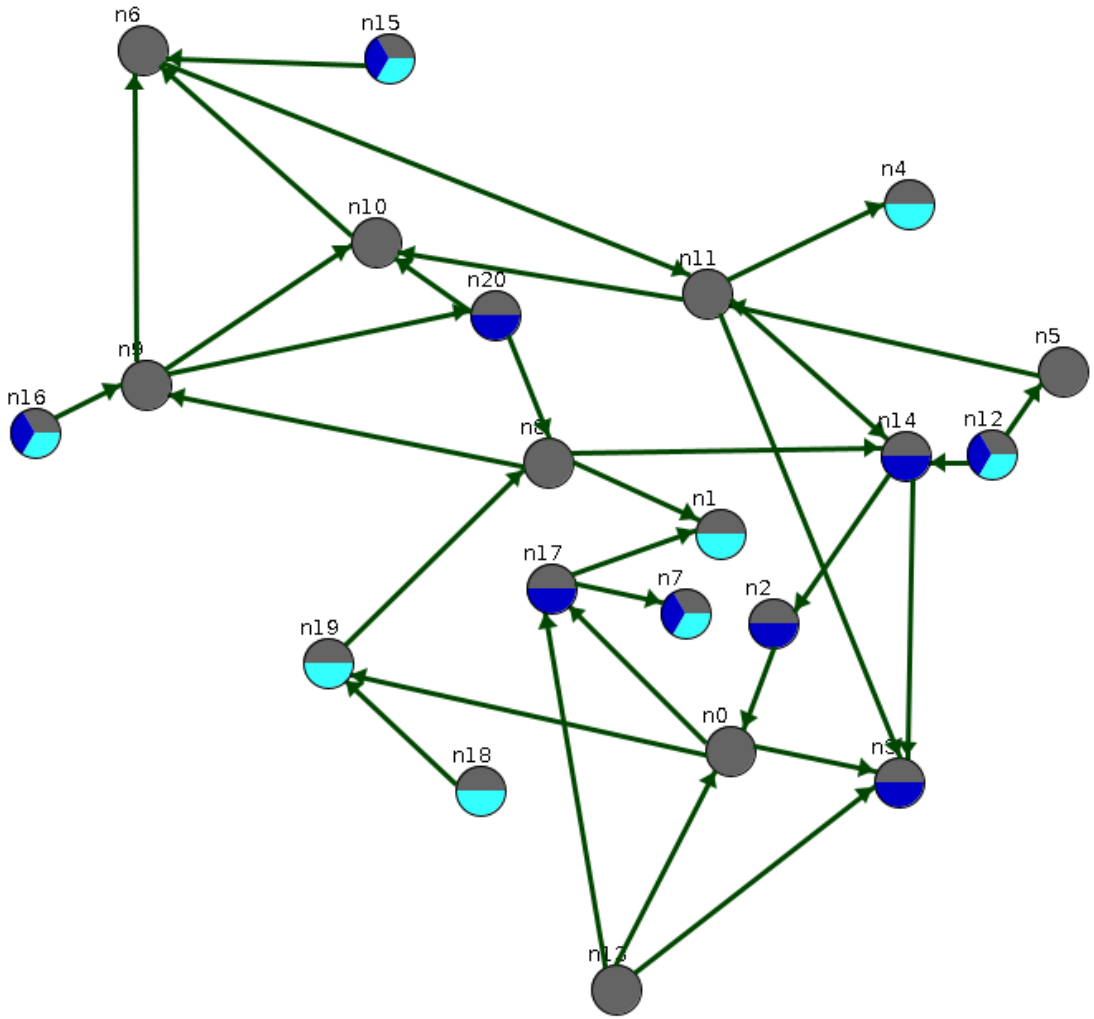
100% accurate from epoch 50.



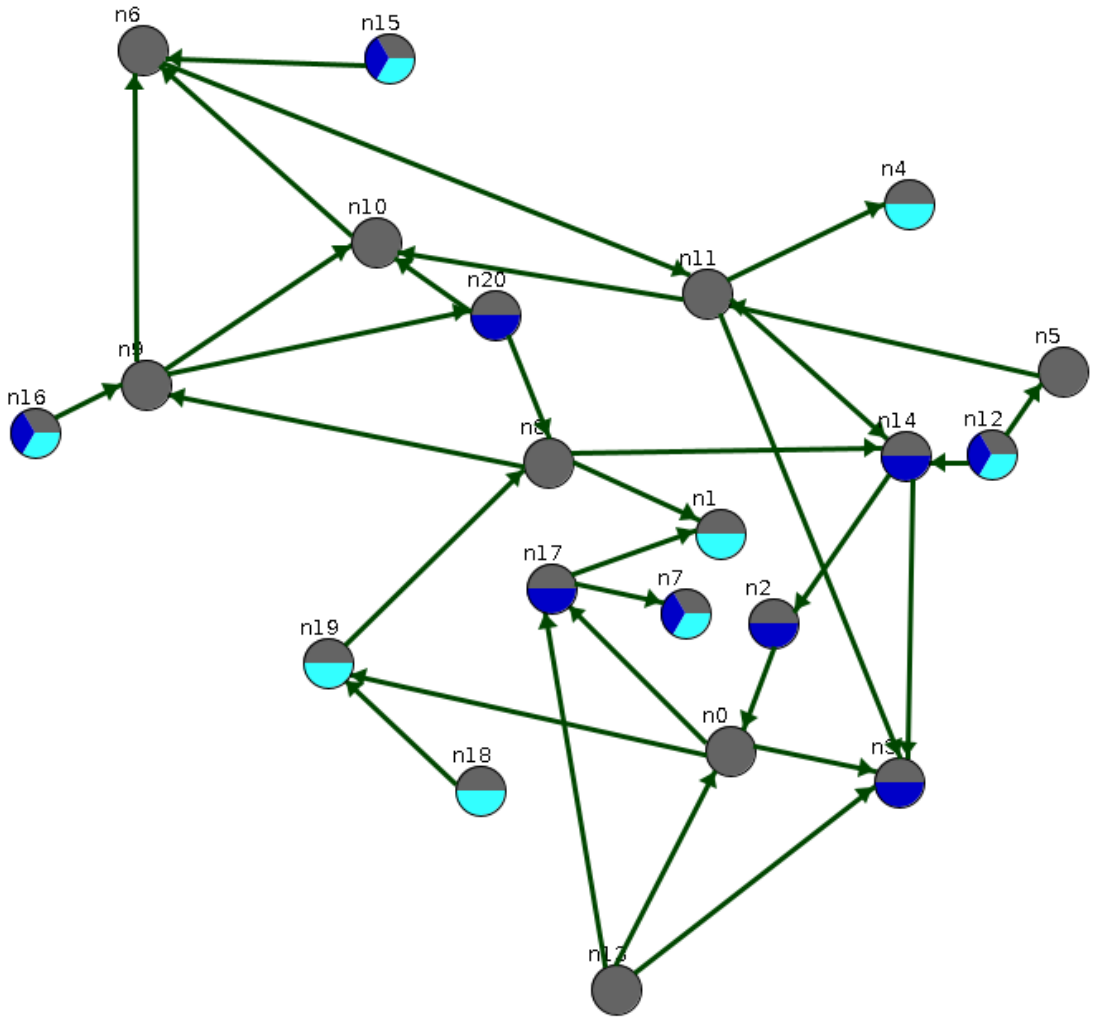
- Graph with observed α_1 relation (21 nodes)



- Graph with observed α_1 relation (21 nodes)
- MAP for *blue* with RBN-GNN manually set parameters (test accuracy: 1.0).



- Graph with observed α_1 relation (21 nodes)
- MAP for *blue* with RBN-GNN manually set parameters (test accuracy: 1.0).
- MAP for *blue* with RBN-GNN learned parameters (test accuracy: 1.0).



- Graph with observed α_1 relation (21 nodes)
- MAP for *blue* with RBN-GNN manually set parameters (test accuracy: 1.0).
- MAP for *blue* with RBN-GNN learned parameters (test accuracy: 1.0).

➥perfect accuracy on primary prediction task does not guarantee high accuracy for other reasoning tasks.

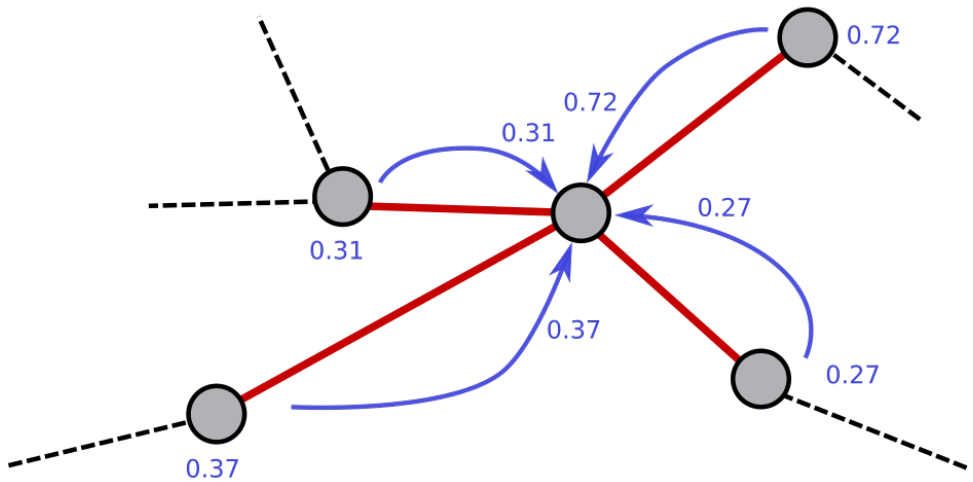
RBN features that enable a tight integration:

- Directed probabilisitc models $\sim$ forward propagation
- Central role of relational neighborhood aggregation
- Nested probability formulas $\sim$ deep neural architectures
- Arithmetic treatment of logical reasoning (Booleans turned into 0/1).

Note: focus here on message-passing GNNs; not e.g. recurrent GNNs.

Aggregation

Central modeling element: aggregating (scalar) feature values from relational neighbors:



Aggregation

Input: multiset $\{\!\{x_1, \ldots, x_m\}\!\}$ of real numbers. In GNN typically: aggregate by *sum*, *mean*, *max*, *min*.

Combination

Input: multiset $\{\!\{p_1, \ldots, p_m\}\!\}$ of probability values. Common in SRL: combine by *noisy-or*:

$$\textit{noisy-or}\{\!\{p_1, \ldots, p_m\}\!\} = 1 - \prod_{i=1}^{m}(1 - p_i)$$

Aggregating feature values $h(j)$ of i's neighbors. Compare:

$$agg\{\!|h(j) \mid j \in N_i\}\!| = sum\{\!|h(j) \mid j \in N_i\}\!|$$

vs.:

$$agg\{\!|h(j) \mid j \in N_i\}\!| = E[i,:](h(1),\ldots,h(n))^T$$

($E[i,:]$: i'th row of adjacency matrix).

- Both definitions equivalent!
- Second version: specification in terms of matrices/vectors opens possibility that function can be dependent on ordering of nodes
- Permutation invariance: aggregation functions defined must not be sensitive to node orderings imposed by vector/matrix expressions
- Never been an issue in SRL: model specifications always at the level of sets/logical theories (no order implied).

$\mathcal{X}$-aggregators

$\mathcal{X}$: set of possible feature values (usually $\mathcal{X} \subseteq \mathbb{R}$). An $\mathcal{X}$-tuple aggregator is any function of the form

$$f : \bigcup_{n \in \mathbb{N}} \mathcal{X}^n \to \mathbb{R}.$$

Universality of Sum

If $\mathcal{X}$ is countable, then every permutation invariant $\mathcal{X}$-tuple aggregator f can be written in the form

$$f(x_1, \ldots, x_n) = \rho(\sum_i \phi(x_i)).$$

for some $\phi : \mathcal{X} \to \mathbb{R}$ and $\rho : \mathbb{R} \to \mathbb{R}$.

➥practical impact limited: the functions ϕ, ρ are not amenable to neural network representation.

[Zaheer, M. et al. Deep sets. 2017]

[Wagstaff, E. et al. On the limitations of representing functions on sets. 2019]

Core element of universality proof: construct ϕ such that

$$(x_1, \ldots, x_n) = \sum_i \phi(x_i) \in \mathbb{R}$$

is *injective*.

Injectivity of aggregation also a major concern for *expressivity*: mapping different feature sets (of neighbors) to a different feature value (of node) means maximal discriminative power.

Fundamental Limitation

There does not exist a continuous injective function $f : \mathbb{R}^n \to \mathbb{R}$ if $n > 1$.

[L.EJ. Brouwer: Beweis der Invarianz des n-dimensionalen Gebiets. 1911]

➥But: this is for $f : \mathbb{R}^n \to \mathbb{R}$. What about $f : \mathcal{X}^n \to \mathbb{R}$?

There exists a function $f : \bigcup_{n\in\mathbb{N}} \mathbb{R}^n \to \mathbb{R}$ such that

- f is continuous and permutation invariant
- f is injective on $\bigcup_{n\in\mathbb{N}} \mathbb{N}^n$, and $f : \bigcup_{n\in\mathbb{N}} \mathbb{N}^n \to \mathbb{N}$

Proof:

(similar strategy as in [Wagstaff, E. et al. 2019])

Let $p_1, p_2, p_3, \ldots = 3, 5, 7, \ldots$ be an enumeration of the prime numbers.

Let $\boldsymbol{x} \in \mathbb{R}^n$

- first sort: $f_{ord} : \boldsymbol{x} \mapsto \boldsymbol{x}^{\leq}$.
  Ex: $(2.7, 1.0, 0.7, 1.0, 5.8, 0.7) \mapsto (0.7, 0.7, 1.0, 1.0, 2.7, 5.8)$
- then encode: $f_{prime} : \boldsymbol{x}^{\leq} \mapsto \prod_{j=1}^{n} p_j^{r_j}$.
  Ex: $(0.7, 0.7, 1.0, 1.0, 2.7, 5.8) \mapsto 3^{0.7}5^{0.7}7^{1.0}11^{1.0}13^{2.7}17^{5.8}$

➥Still not approximable by neural network. Numeric “explosion”.

Homophily

Homophily, also known informally as "birds of a feather", is when a link between individuals (such as friendship or other social connection) is correlated with those individuals being similar in nature. For example, friends often tend to be similar in characteristics like age, social background, and education level.

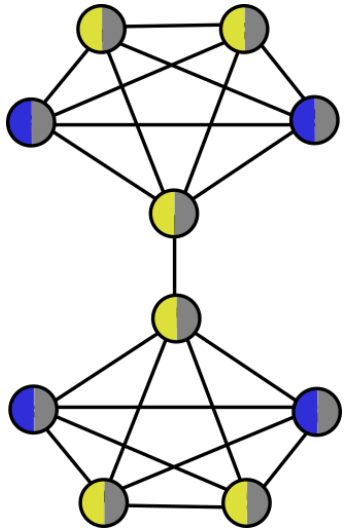
[S. Bhagat et al.: Node Classification in Social Networks, 2011]

➥May apply to any node properties. Main concern: homophily of node class label.

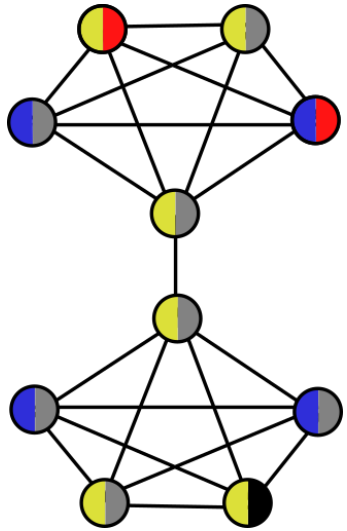
If class label exhibits homophily, should do:

- Collective classification: predict labels jointly for all unlabeled nodes (transductive,inductive)
- Autoregression: use observed labels to predict unobserved labels (transductive)

Illustration: yellow/blue node attribute, red/black node label:



Collective: predict same label for all nodes in a clique



Collective/Autogressive: predict red for top, black for bottom clique.

Homophily a challenge?

> *[Graph neural networks] have been shown to achieve state-of-the-art performance because of their effectiveness in learning object representations on relational data. However, one critical limitation is that the labels of objects are independently predicted based on their representations. In other words, the joint dependency of object labels is ignored.*

[M.Qu, Y. Bengio, J. Tang. "Gmnn: Graph markov neural networks." 2019.]

Heterophily a challenge?

> *Homophily is a key principle of many real-world networks [...] GNNs model the homophily principle by propagating features and aggregating them within various graph neighborhoods via different mechanisms [...] Since many existing GNNs assume strong homophily, they fail to generalize to networks with heterophily*

[J. Zhu et al. "Beyond homophily in graph neural networks: Current limitations and effective designs." 2020]

➥Message-passing feature aggregation leads to “smoothing” of final representations (node’s representation similar to its neighbors). Thus: similar predictions.

➥But: effective tools available to counteract smoothing – cf. ACR-GNN expressivity result!

Markov Logic Networks

Homophily very naturally captured by weighted formulas:

$$\begin{array}{ll} \mathit{friends}(X,Y) \wedge \mathit{republican}(X) \wedge \mathit{republican}(Y) & 2.3 \\ \mathit{friends}(X,Y) \wedge \mathit{republican}(X) \wedge \neg \mathit{republican}(Y) & -0.2 \end{array}$$

➥supports collective classification and autoregression.

ProbLog

"Label propagation" rule:

$$\mathit{republican}(X) \leftarrow \mathit{friends}(X,Y) \wedge \mathit{republican}(Y).$$

(this strict implication needs to be "softened" using suitable ProbLog modeling tricks).

➥supports collective classification and autoregression.

RBNs

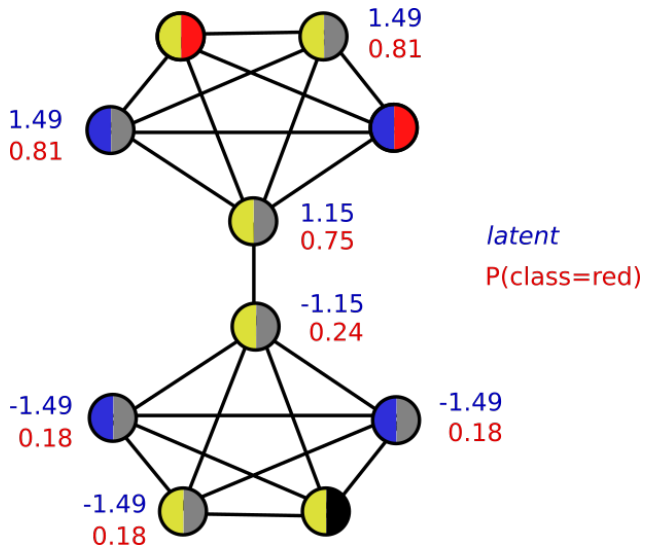
Symmetric dependencies modeled by shared dependence on a *latent* variable. Here: latent numerical relation.

$$F_{class(X)} \equiv \texttt{COMBINE}\ latent(X)\ \texttt{WITH}\ logistic\ regression$$
$$F_{edge(X,Y)} \equiv \texttt{COMBINE}\ latent(X) \cdot latent(Y)\ \texttt{WITH}\ logistic\ regression$$

- Learning values of the *latent* node attribute by gradient descent
- Similar to [P. D. Hoff et al. "Latent space approaches to social network analysis." 2002], [T.N. Kipf, M. Welling. Variational graph auto-encoders. 2016]

Example

Learned latent attribute values and resulting class probabilities.



Learning

		GNN	SRL
Structure	Space	NN architectures	(Logical) model structure
	Manual specification by	NN engineers	SRL experts, domain experts
	Learned by	Optimization/search in combinatorial spaces	
Parameters	Space	High-dimensional	Low-dimensional
	Manual specification	Never	Sometimes possible
	Objective	Loss function (cross-entropy, MSE, ...)	Likelihood (plain, conditional, pseudo, penalized,...)
	Learned by	Gradient descent	Gradient descent, variational inference, expectation maximization, ...

- The structure space for GNNs is less complex than the structure spaces for SRL
- A big part of what is structure learning in SRL becomes parameter learning in GNNs

Example: building a model for a social network domain with *follower*, *influencer* $\in \mathcal{R}$.

Expert knowledge:

> *"whether someone is an influencer depends (among other things) on how many followers he/she has"*

Injecting the expert knowledge into SRL models:

ProbLog: add rule

$$influencer(X) \leftarrow follower(Y, X)$$

- \+ Simple and intuitive
- \+ Modular
- \- Hard to assess/control the quantitative aspect: how does the number of followers affect the probability of being an influencer?

MLN: add formulas

$$\begin{aligned}
&\mathit{influencer}(X) \vee \mathit{follower}(Y, X)\\
&\mathit{influencer}(X) \vee \neg\mathit{follower}(Y, X)\\
&\neg\mathit{influencer}(X) \vee \mathit{follower}(Y, X)\\
&\neg\mathit{influencer}(X) \vee \neg\mathit{follower}(Y, X)
\end{aligned}$$

+ Simple and (still) intuitive
+ Modular
- The quantitative relationship has to be captured by the weights assigned to the formulas

RBN: integrate sub-formula into proability formula $F_{\mathit{influencer}(X)}$:

$$\begin{aligned}
F_{\mathit{influencer}(X)} \equiv \quad & \ldots \\
& \texttt{COMBINE} < \textit{probability formula} > \\
& \texttt{WITH} < \textit{combination function} > \\
& \texttt{FORALL } Y \\
& \texttt{WHERE } \mathit{follower}(Y, X) \\
& \ldots
\end{aligned}$$

+ Faithful representation of expert knowledge
+/- Supports/requires exact control over the quantitative relationship
- Not modular

Data: $(V_1, \boldsymbol{R}_1), \ldots, (V_N, \boldsymbol{R}_N)$ ($N = 1$ in transductive setting).

SRL: Log-likelihood

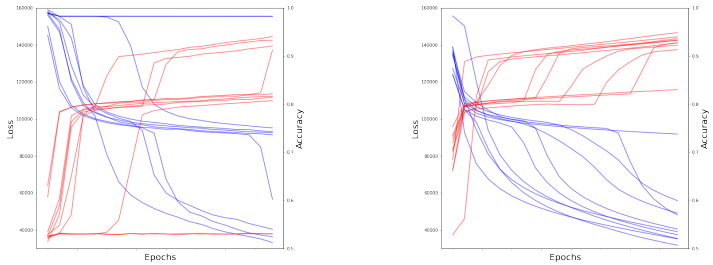
Statistical learning: learn parameters $\boldsymbol{\theta}$ of the model by maximizing *log-likelihood* of the data (maybe penalized):

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log P^{\boldsymbol{\theta}}_{V_i}(\boldsymbol{R}_i) \quad (+penalty(N, \theta))$$

For RBNs log-likelihood term for each $(V_i, \boldsymbol{R}_i)$ decomposes into

$$\log P^{\boldsymbol{\theta}}_{V}(\boldsymbol{R}) = \sum_{R \in \mathcal{R}} \sum_{\boldsymbol{i} \in V^{arity(R)}} logP_V(R(\boldsymbol{i})|Pa(R))$$

➥inner sum equal to negative cross-entropy loss in GNN training (with final softmax layer).



GNN: Most of the model is encoded by the parameter (weight) setting in a high-dimensional parameter space

SRL: Model "equally" encoded by structure, and parameters in low-dimensional parameter space

Advantages of high-dimensional spaces:

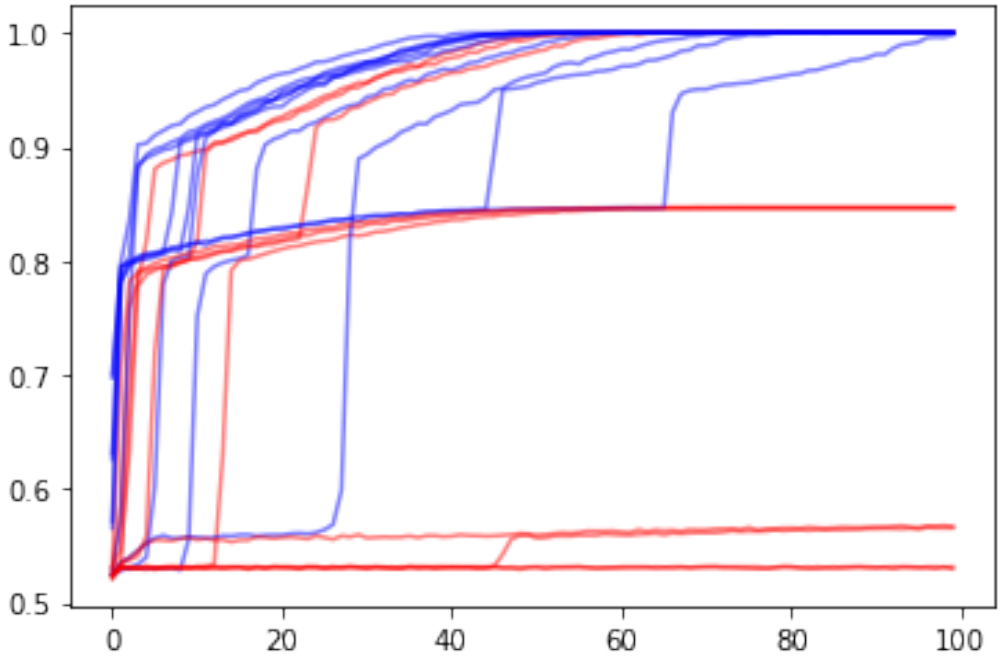
- Model capacity
- Gradient descent more effective

Advantages of low-dimensional spaces:

- Interpretability
- Robustness

Example: For α_1 classification task: 1-layer ACR with hidden dimension 2 is sufficient!

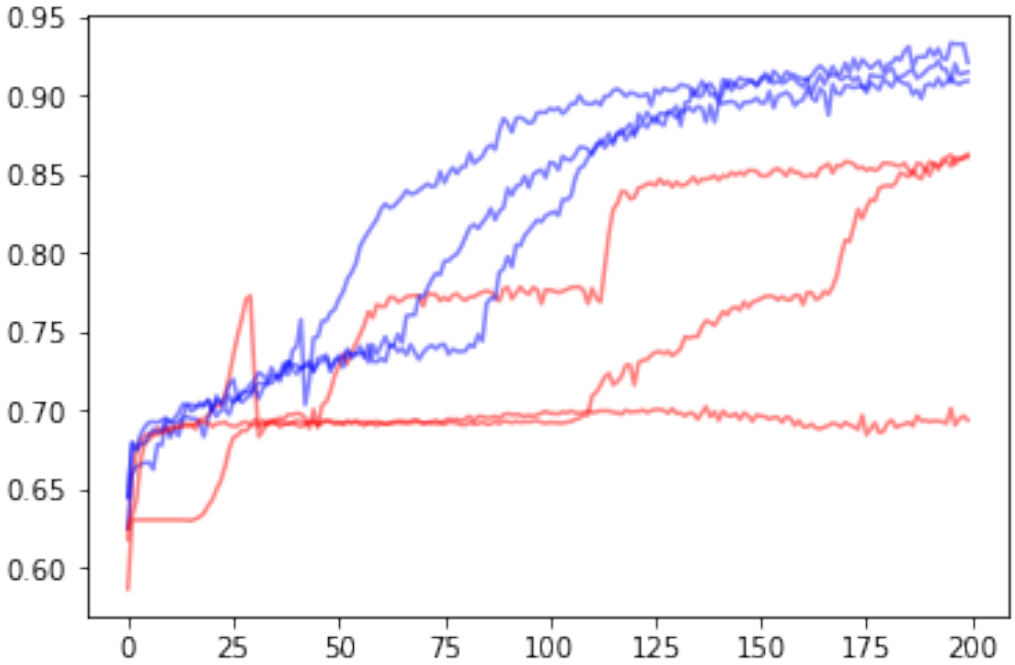
Actual learning curves for hidden dim. = 2 and hidden dim = 8 (10 restarts each):



➥Over-parameterized model converges faster and more consistently to optimal solution.

Example: For α_2 classification task: 2-layer ACR with hidden dimension 2 is sufficient!

Actual learning curves for hidden dim. = 2 and hidden dim = 32 (3 restarts each):



➥Bad news: expert knowledge “dimension 2 is sufficient” can be harmful!

Implementation: Primula 3

Primula tool for RBNs

- first release: 2003
- most recent release: v.2.2: 2009

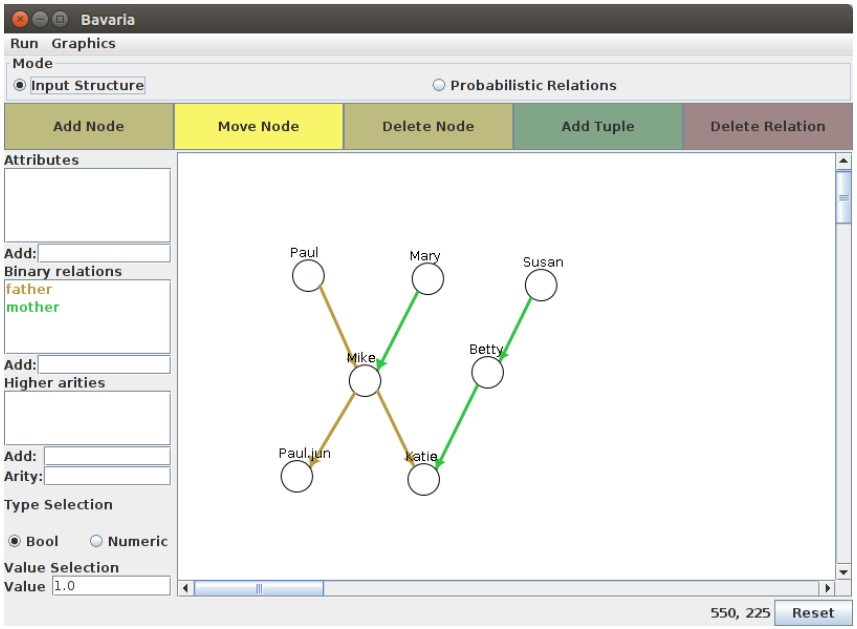
Primula 3

On Github:

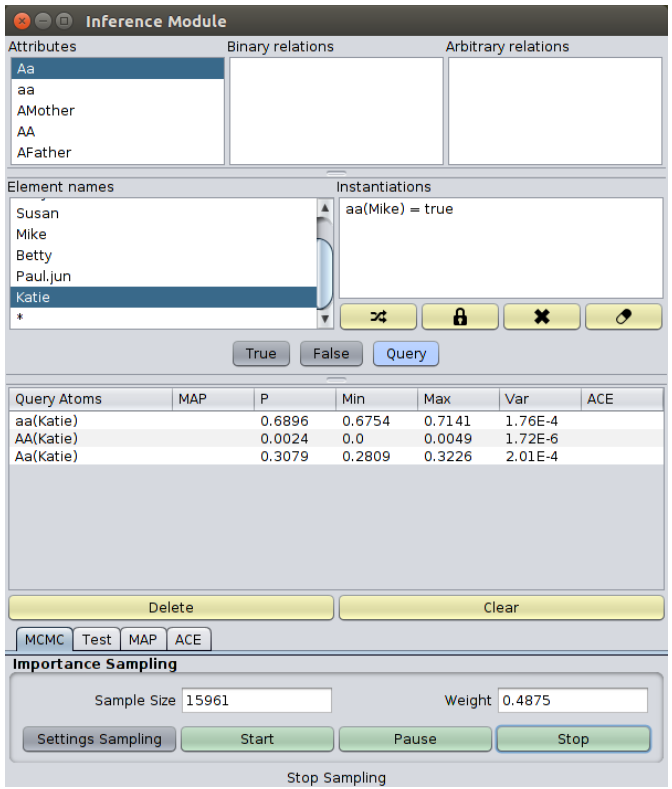
`https://github.com/manfred-jaeger-aalborg/primula3`

- Current tool version
- Example-based documentation: use cases with model/data files.

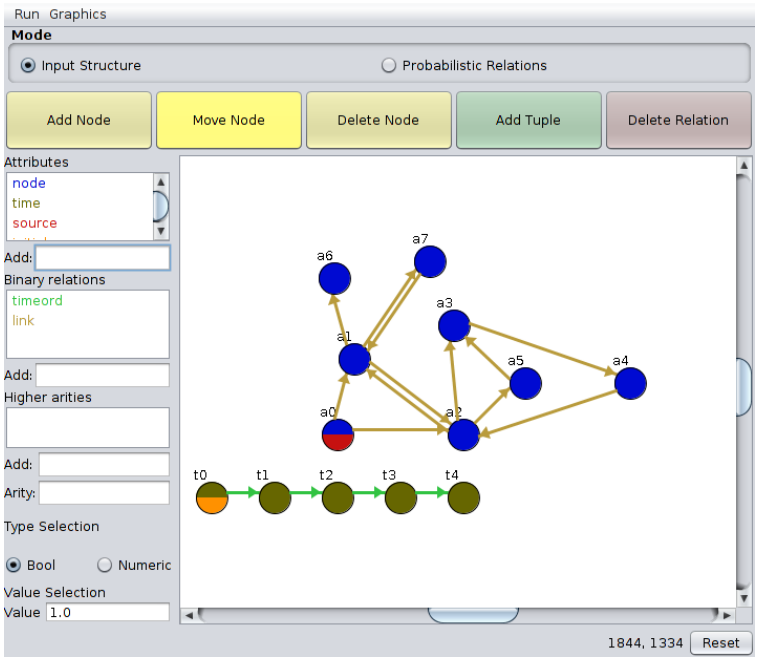
Data: family trees



Reasoning: infer genotypes from partially genotypes of other family members



Data: Social networks

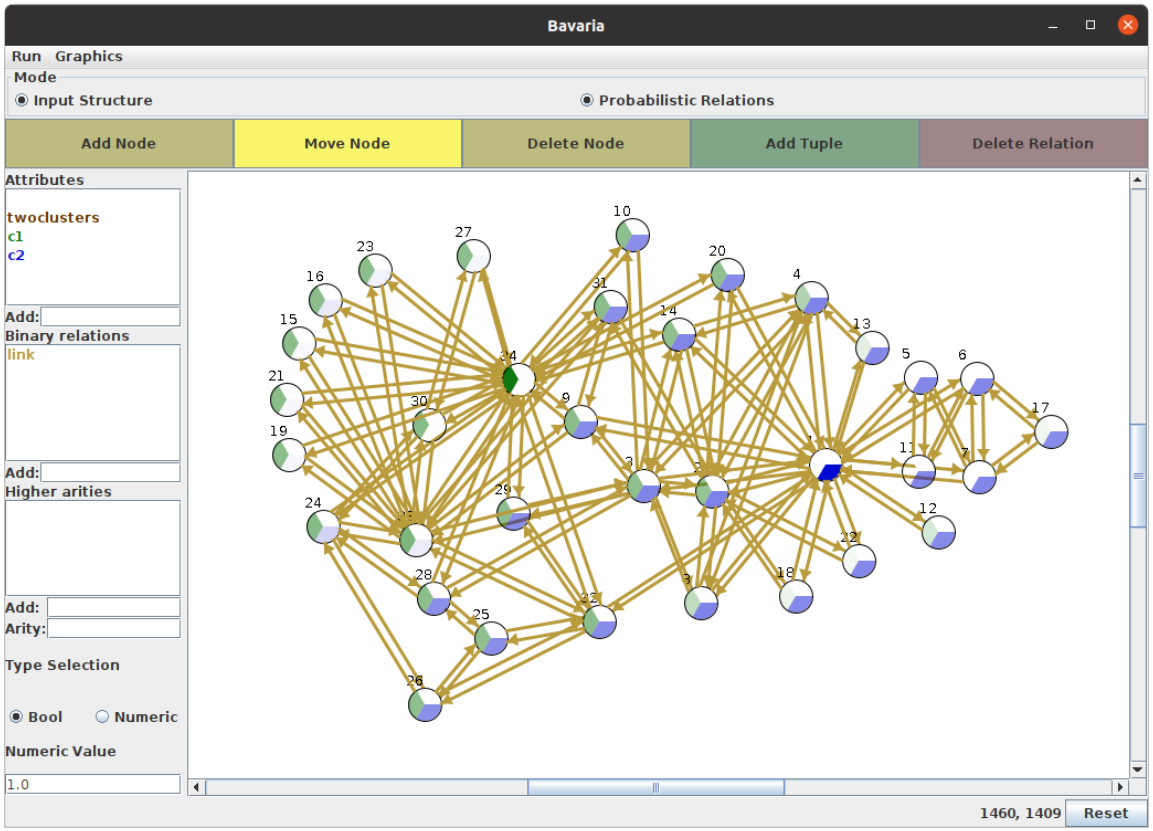


Learning and Reasoning:

- predict information spread according to independent cascade model
- learn information propagation probabilities

Data: Social networks.

Learning: Learn soft community membership degrees for the nodes.



[J. Jiang and M. Jaeger: Numeric Input Relations for Relational Learning with Applications to Community Structure Analysis, ArXiv 1506.05055, 2015.]

GNN and SRL

Complementary strengths:

- GNN: effective training in high-dimensional parameter spaces
- SRL: integration of expert knowledge, interpretability, support for wider range of reasoning tasks

Integration

- RBNs well-suited for homogeneous integration due to GNN-compatible semantics and representation approach

Open Problems

- Understand trade-offs:
  - sparse, constrained parameterization
  - over-parameterization
- From high-dimensional GNN models to sparse and robust models:
  - model distillation?