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

GNN and SRL

Manfred Jaeger

Aalborg University
Graph Neural Networks: Basics

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

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

- $H_1$: aggregate neighbors' $H_0$ features (e.g., sum) and apply some (activation) functions

- $H_2$: aggregate neighbors' $H_1$ features and apply some (activation) functions
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
GNN basics

Message Passing

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
A basic form of message passing updates:

\[ h^0(i) = \text{initial node feature vector of node } i \]

\[ h^{k+1}(i) = f \left( W^k h^k(i) + U^k \sum_{j \in N_i} h^k(j) \right) \]

with ingredients:

- \( W^k, U^k \): weight matrices (dimensions: \( d^{k+1} \times d^k \))
- \( f \): (nonlinear) activation function (component-wise)
$h^k(i)$: $d^k$-dimensional vector representation of node $i$ at $k$th iteration (layer).

A basic form of message passing updates:

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

with ingredients:

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

In full matrix notation:

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

with ingredients:

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

GNN basic architecture

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 \( \leq 3 \) to node 26.

\[\text{this only works in transductive settings.}\]
Initial features: node attributes (e.g. $\text{color} \in \{\text{yellow}, \text{blue}\}$)

Can represent/learn classification rule: node is $\text{red}$, if it has distance $\leq 2$ 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.: $h^0(i) = 1$ for all $i$).

Can represent/learn classification rule: node is red, if it has distance $\leq 2$ to a node with degree $\geq 5$.

This works in inductive settings: rule can be applied to new graphs.
GNN reasons: node classification

- On top of the final embedding $H^K$: add neural network layers for classifying nodes based on their feature vectors $h^K$
- 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 $H_K$: 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 $h(i)$).

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 basics

GNN reasoning: link prediction

- GNN only computes embeddings
- Trained using reconstruction loss, such as
  \[
  \sum_{i,j \in V} (h(i) \cdot h(j) - E[i,j])^2
  \]
- Use the score \( h(i) \cdot h(j) \) to predict whether there is a (not yet observed) edge between \( i \) and \( j \).
Graph Neural Networks: Extensions
GNN Architecture: Multi-Relational

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

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

GNN Architecture: Multi-Relational

- 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 $\mathbf{v}$ of nodes.

Neighbors for $k$-tuples defined by $\text{edge}^*$-relation:

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

[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?

\[ a, b \text{ indistinguishable by any GNN.} \]
\[ a, c \text{ indistinguishable by 2-layer GNNs, distinguishable by 3-layer GNNs.} \]

\[ \Rightarrow l\text{-layer GNNs can only access information in the } l - 1 \text{ hop node neighborhood.} \]
\[ \Rightarrow \text{(standard) GNNs can not access “global” graph properties for node classification.} \]
Aggregate-Combine-Readout GNN:

- **Output**: \( \times N \)
- **Identity**: \( \uparrow \)
- **Edge**: \( \uparrow \)
- **Full**: \( \uparrow \)

**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**

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

Counting in FOL without variable restrictions:

\[ \exists^{\geq 3} Y \phi(X,Y) \rightsquigarrow \exists Y_1, Y_2, Y_3 (Y_1 \neq Y_2 \land Y_1 \neq Y_3 \land Y_2 \neq Y_3 \land \phi(X,Y_1) \land \phi(X,Y_2) \land \phi(X,Y_3)) \]
Result from [Jaeger, Relational Bayesian Networks, 1997]:

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

Predicate logic (relational)
\[
\forall x (r(x) \rightarrow \exists y (e(x, y) \land b(y)))
\]
\[
\exists z, x, y \neg (e(x, y) \land e(x, z) \land (e(y, z))
\]
\ldots
Some SRL Frameworks

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:

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBN</td>
<td>Directed probabilistic graphical models</td>
</tr>
<tr>
<td>MLN</td>
<td>Undirected probabilistic graphical models</td>
</tr>
<tr>
<td>ProbLog</td>
<td>(Inductive) logic programming</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>( F )</th>
<th>( w )</th>
</tr>
</thead>
<tbody>
<tr>
<td>friends(X, Y)</td>
<td>-0.2</td>
</tr>
<tr>
<td>republican(X)</td>
<td>0.1</td>
</tr>
<tr>
<td>friends(X, Y) ( \Rightarrow ) (republican(X) ( \Rightarrow ) republican(Y))</td>
<td>0.8</td>
</tr>
</tbody>
</table>

\( \mathcal{R} = \{ \text{friends, republican} \} \).

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

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

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

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**

\[
G:\quad \begin{array}{ccc}
\text{friends}(X,Y) & -0.2 \\
\text{republican}(X) & 0.1 \\
\text{friends}(X, Y) \Rightarrow (\text{republican}(X) \Rightarrow \text{republican}(Y)) & 0.8 \\
\end{array}
\]

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

$$P(G) = \text{???} \ (\text{need to first compute } Z, \text{ i.e., the weight of all graphs})$$
The distribution defined by an MLN over 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
+ 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**

\[
\begin{align*}
\text{edge}(a, b) & : 0.3 \\
\text{edge}(b, c) & : 0.3 \\
\text{path}(X, Y) & \leftarrow \text{edge}(X, Y) \\
\text{path}(X, Y) & \leftarrow \text{edge}(X, Z), \text{path}(Z, Y)
\end{align*}
\]

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**

\[
P \begin{pmatrix}
    \text{edge}(a, b) \\
    \text{path}(X, Y) \leftarrow \text{edge}(X, Y) \\
    \text{path}(X, Y) \leftarrow \text{edge}(X, Z), \text{path}(Z, Y)
\end{pmatrix} = 0.3 \cdot 0.7 = 0.21
\]

\( LP \) 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 
\[ P_V(A|B) \text{ for some } A, B \subseteq \mathcal{G}(V, \mathcal{R}) \]
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**

\( V = \{ \text{mary}, \text{tom}, \text{carl}, \text{sue} \}; \quad P_V(\text{republican(mary)}|\text{friends(mary, carl)}, \neg \text{republican(carl)}) = ? \)

\( V = \{ a, b, c, d, e \}; \quad P_V(\text{path}(a, d)|\text{edge}(a, c), \text{path}(e, d)) = ? \)

**Special case: prediction**

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

Node classification then consists of queries

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

where \( B \) is a complete specification of \( \{ \text{edge} \} \cup \mathcal{A} \).
Model checking: checking properties (conditional probabilities) of a particular model (probability distribution $P_V$).

**Chain Rule**

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

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

**Conditional independence of relations**

Conditional independencies lead to simplifications:

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

$\Rightarrow$ directed acyclic graph over relations (relation DAG).

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

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

$$P_V(\text{gender}) P_V(\text{republican}|\text{gender}) P_V(\text{bloodtype}|\text{gender}) P_V(\text{friends}|\text{republican, gender})$$
(Conditional) generative models

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

$\mathbf{RBNs}$

Sometimes: assume some relations $\mathcal{R} \in \mathcal{R}$ are predefined input relations: $\mathcal{R} = \mathcal{R}_{\text{prob}} \cup \mathcal{R}_{\text{in}}$

- Make these relations roots in the relation DAG
- Do not define a distribution $P_{\mathcal{V}}(\mathcal{R}_h)$ for these relations
- Defines a conditional distribution $P_{\mathcal{V}}(\mathcal{R}_{\text{prob}} | \mathcal{R}_{\text{in}})$
RBNs

(Conditional) generative models

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

▶ Sometimes: assume some relations $R \in \mathcal{R}$ are predefined input relations:

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

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

$$P_V(R_{\text{prob}} | R_{\text{in}})$$
Definitions for full generative probabilistic model for graphs in signature $\mathcal{R}$.

Sometimes, assume some relations $R \in \mathcal{R}$ are predefined input relations:

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

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

$$P_V(R_{\text{prob}} | R_{\text{in}})$$

All SRL frameworks support divisions $\mathcal{R} = \mathcal{R}_{\text{prob}} \cup \mathcal{R}_{\text{in}}$.
Atom independence

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

\[ P_V(R_h|Pa(R_h)) := \prod_{i \in \text{arity}(R_h)} P_V(R_h(i)|Pa(R_h)) \]

As a Bayesian network:

leads to limitations for modeling e.g. symmetry constraints \texttt{friends}(1, 2) \Leftrightarrow \texttt{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 $\text{Pa}(R)$ that defines the conditional probabilities

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

**Probability formulas: semantics**

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

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

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

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

$$F \equiv q$$

is a probability formula with 

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

for all $i, G$.

**Example**

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

$$F_{\text{edge}}(x, y) \equiv 0.5$$

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

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

WIF-THEN-ELSE

If $F_1, F_2, F_3$ are probability formulas, then

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

is a probability formula with

$$eval(F, i, G) = eval(F_1, i, G) eval(F_2, i, G) + (1 - eval(F_1, i, G)) eval(F_3, i, G)$$

Generalization of Boolean operations ($F_i \in \{0, 1\}$)
Example: Stochastic block model

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

With the constructs introduced so far:

**A. partitioning into red, green, blue nodes:**

\[
F_{\text{red}}(x) \equiv 0.5 \\
F_{\text{blue}}(x) \equiv \text{WIF} \, \text{red}(x) \text{ THEN } 0 \text{ ELSE } 0.7 \\
F_{\text{green}}(x) \equiv \text{WIF} \, \text{red}(x) \lor \text{blue}(x) \text{ THEN } 0 \text{ ELSE } 1.0
\]

**B. generating edges:**

\[
F_{\text{edge}}(x, y) \equiv \text{WIF} \, \text{red}(x) \wedge \text{red}(y) \text{ THEN } 0.6 \text{ ELSEIF } ... 
\]

(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

$$F \equiv \text{COMBINE } F_1, \ldots, F_t$$

WITH $<\text{combination function}>$

FORALL $<\text{variables}>$

WHERE $<\text{Boolean } \mathcal{R}_{\text{in condition}}>$

is a probability formula.
$P_V(red(i))$ higher if

- $i$ is blue
- $i$ is part of many triangles
Defining triangles:

\[ F_{triangle}(x, y, z) \equiv edge(x, y) \land edge(x, z) \land edge(y, z) \]

\[ P_V(red(i)) \] higher if

- \( i \) is blue
- \( i \) is part of many triangles
Defining triangles:

\[
F_{\text{triangle}}(x, y, z) \equiv \\
\text{edge}(x, y) \land \text{edge}(x, z) \land \text{edge}(y, z)
\]

Counting triangles:

\[
F_{\text{triangle\_count}}(x) \equiv \\
\text{COMBINE} \ 1.0 \\
\text{WITH} \ \text{sum} \\
\text{FORALL} \ y, z \\
\text{WHERE} \ F_{\text{triangle}}(x, y, z)(x, y, z)
\]

\[P_v(\text{red}(i)) \text{ higher if} \]

\[\text{▶ } i \text{ is } \text{blue} \]

\[\text{▶ } i \text{ is part of many triangles} \]
### Defining triangles:

\[ F_{\text{triangle}}(x, y, z) \equiv \edge(x, y) \land \edge(x, z) \land \edge(y, z) \]

### Counting triangles:

\[ F_{\text{triangle\_count}}(x) \equiv \text{COMBINE } 1.0 \text{ WITH } \text{sum} \text{ FORALL } y, z \text{ WHERE } F_{\text{triangle}}(x, y, z)(x, y, z) \]

\[ P_V(\text{red}(i)) \text{ higher if} \]

- \text{i is blue}
- \text{i is part of many triangles}

### Logistic regression of \text{triangle\_count} and \text{blue} feature:

\[ F_{\text{red}}(x) \equiv \text{COMBINE } 0.6 \cdot F_{\text{triangle\_count}}(x)(x), 0.3 \cdot \text{blue}(x), -3.0 \text{ WITH logistic regression} \]
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, ...-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