Deep Probabilistic Logic Programming

Arnaud Nguembang Fadja Evelina Lamma Fabrizio Riguzzi

Dipartimento di Ingegneria – University of Ferrara

Dipartimento di Matematica e Informatica – University of Ferrara
[arnaud.nguembangfadja,evelina.lamma,fabrizio.riguzzi]@unife.it

Università
degli Studi
di Ferrara

Introduction

- Probabilistic logic programming is a powerful tool for reasoning with uncertain relational models
- Learning probabilistic logic programs is expensive due to the high cost of inference.
- We consider a restriction of the language of Logic Programs with Annotated Disjunctions called hierarchical PLP in which clauses and predicates are hierarchically organized.
- Hierarchical PLP is truth-functional and equivalent to the product fuzzy logic.
- Inference then is much cheaper as a simple dynamic programming algorithm is sufficient

Probabilistic Logic Programming

- Distribution Semantics [Sato ICLP95]
- A probabilistic logic program defines a probability distribution over normal logic programs (called instances or possible worlds or simply worlds)
- The distribution is extended to a joint distribution over worlds and interpretations (or queries)
- The probability of a query is obtained from this distribution

PLP under the Distribution Semantics

- A PLP language under the distribution semantics with a general syntax is Logic Programs with Annotated Disjunctions (LPADs)
- Heads of clauses are disjunctions in which each atom is annotated with a probability.
- LPAD T with n clauses: $T = \{C_1, \ldots, C_n\}$.
- Each clause C_i takes the form:

$$h_{i1} : \pi_{i1}; \ldots; h_{iv_i} : \pi_{iv_i} :- b_{i1}, \ldots, b_{iu_i}$$

,

- Each grounding $C_i\theta_j$ of a clause C_i corresponds to a random variable X_{ij} with values $\{1, \ldots, v_i\}$
- The random variables X_{ij} are independent of each other.

Example

- UW-CSE domain:

$$advisedby(A, B) : 0.3 :-$$
$$\quad student(A), professor(B), project(C, A), project(C, B).$$
$$advisedby(A, B) : 0.6 :-$$
$$\quad student(A), professor(B), ta(C, A), taughtby(C, B).$$

$$student(harry). \quad student(john). \quad \ldots$$
$$professor(ben). \quad professor(stephen). \quad \ldots$$
$$project(p1, harry). \quad project(p1, ben). \quad \ldots$$
$$ta(c1, harry). \quad ta(c2, john). \quad \ldots$$
$$taughtby(c1, ben). \quad taughtby(c2, ben). \quad \ldots$$

Distribution Semantics

- Case of no function symbols: finite Herbrand universe, finite set of groundings of each clause
- Atomic choice: selection of the k-th atom for grounding $C_i\theta_j$ of clause C_i
- Represented with the triple (C_i, θ_j, k)
- Example $C_1 = advisedby(A, B)$:
  $0.3 :- student(A), professor(B), project(C, A), project(C, B).$,
  $(C_1, \{A/harry, B/ben, C/p1\}, 1)$
- Composite choice κ: consistent set of atomic choices
- The probability of composite choice κ is

$$P(\kappa) = \prod_{(C_i,\theta_j,k)\in\kappa} \pi_{ik}$$

Distribution Semantics

- Selection σ: a total composite choice (one atomic choice for every grounding of each clause)
- A selection σ identifies a logic program w_σ called world
- The probability of w_σ is $P(w_\sigma) = P(\sigma) = \prod_{(C_i,\theta_j,k)\in\sigma} \pi_{ik}$
- Finite set of worlds: $W_T = \{w_1, \ldots, w_m\}$
- $P(w)$ distribution over worlds: $\sum_{w\in W_T} P(w) = 1$

Example

- Suppose the program is:

$$
\begin{array}{l}
advisedby(harry, ben) : 0.3 :- \\
\quad student(harry), professor(ben), \\
\quad project(p1, harry), project(p1, ben). \\
advisedby(harry, ben) : 0.6 :- \\
\quad student(harry), professor(ben), \\
\quad ta(c1, harry), taughtby(c1, ben).
\end{array}
\qquad B = \quad
\begin{array}{l}
student(harry). \quad student(john). \quad \ldots \\
professor(ben). \quad professor(stephen). \quad \ldots \\
project(p1, harry). \quad project(p1, ben). \quad \ldots \\
ta(c1, harry). \quad ta(c2, john). \quad \ldots \\
taughtby(c1, ben). \quad taughtby(c2, ben). \quad \ldots
\end{array}
$$

- 4 worlds

$$
\begin{array}{l}
advisedby(harry, ben) :- \\
\quad student(harry), professor(ben), \\
\quad project(p1, harry), project(p1, ben). \\
advisedby(harry, ben) :- \\
\quad student(harry), professor(ben), \\
\quad ta(c1, harry), taughtby(c1, ben).
\end{array}
\qquad P = 0.3 \cdot 0.6 = 0.18
$$

$$
\begin{array}{l}
advisedby(harry, ben) :- \\
\quad student(harry), professor(ben), \\
\quad project(p1, harry), project(p1, ben). \\
null :- \\
\quad student(harry), professor(ben), \\
\quad ta(c1, harry), taughtby(c1, ben).
\end{array}
\qquad P = 0.3 \cdot 0.4 = 0.12
$$

$$
\begin{array}{l}
null :- \\
\quad student(harry), professor(ben), \\
\quad project(p1, harry), project(p1, ben). \\
advisedby(harry, ben) :- \\
\quad student(harry), professor(ben), \\
\quad ta(c1, harry), taughtby(c1, ben).
\end{array}
\qquad P = 0.7 \cdot 0.6 = 0.42
$$

$$
\begin{array}{l}
null :- \\
\quad student(harry), professor(ben), \\
\quad project(p1, harry), project(p1, ben). \\
null :- \\
\quad student(harry), professor(ben), \\
\quad ta(c1, harry), taughtby(c1, ben).
\end{array}
\qquad P = 0.7 \cdot 0.4 = 0.28
$$

Università degli Studi di Ferrara

Distribution Semantics

- Ground query q
- We consider only *sound* LPADs, where each possible world has a total well-founded model, so $w \models q$ means that the query q is true in the well-founded model of the program w.
- $P(q|w) = 1$ if q is true in w and 0 otherwise
- $P(q) = \sum_w P(q, w) = \sum_w P(q|w)P(w) = \sum_{w \models q} P(w)$

Example

- $q = advisedby(harry, ben)$, 4 worlds

$advisedby(harry, ben) :-$
$\quad student(harry), professor(ben),$
$\quad project(p1, harry), project(p1, ben).$
$advisedby(harry, ben) :-$
$\quad student(harry), professor(ben),$
$\quad ta(c1, harry), taughtby(c1, ben).$
$P = 0.3 \cdot 0.6 = 0.18$

$advisedby(harry, ben) :-$
$\quad student(harry), professor(ben),$
$\quad project(p1, harry), project(p1, ben).$
$null :-$
$\quad student(harry), professor(ben),$
$\quad ta(c1, harry), taughtby(c1, ben).$
$P = 0.3 \cdot 0.4 = 0.12$

$null :-$
$\quad student(harry), professor(ben),$
$\quad project(p1, harry), project(p1, ben).$
$advisedby(harry, ben) :-$
$\quad student(harry), professor(ben),$
$\quad ta(c1, harry), taughtby(c1, ben).$
$P = 0.7 \cdot 0.6 = 0.42$

$null :-$
$\quad student(harry), professor(ben),$
$\quad project(p1, harry), project(p1, ben).$
$null :-$
$\quad student(harry), professor(ben),$
$\quad ta(c1, harry), taughtby(c1, ben).$
$P = 0.7 \cdot 0.4 = 0.28$

- $P(advisedby(harry, ben)) = 0.3 \cdot 0.6 + 0.3 \cdot 0.4 + 0.7 \cdot 0.6 = 0.18 + 0.12 + 0.42 = 0.72$
- In this case $P(advisedby(harry, ben)) = P(X_1 \vee X_2) = 1 - (1 - 0.3)(1 - 0.6) = 0.72$

Example

- In general simple formulas cannot be applied

$$
\begin{aligned}
&epidemic : 0.6 :- flu(david), cold.\\
&epidemic : 0.6 :- flu(robert), cold.\\
&cold : 0.7.\\
&flu(david).\\
&flu(robert).
\end{aligned}
$$

- 8 worlds, *epidemic* is true in three of them: $P(epidemic) = 0.588$
- $P(epidemic) \neq 1 - (1 - 0.6 \cdot 0.7)(1 - 0.6 \cdot 0.7) = 0.6636$

Inference

- Generation of the worlds: infeasible
- Knowledge Compilation+Weighted Model Counting:
  - the truth of the query is represented using a Boolean formula over independent Boolean random variables
  - the formula is compiled to a language where WMC is efficient
- Languages: Binary Decision Diagrams, deterministic-decomposable Negation Normal Form, Sentential Decision Diagrams
- Systems: ProbLog, PITA
- `http://cplint.eu`

Hierarchical PLP

- We want to compute the probability of atoms for a predicate r: $r(\boldsymbol{t})$, where $\boldsymbol{t}$ is a vector of constants.
- $r(\boldsymbol{t})$ can be an example in a learning problem and r a target predicate.
- A specific form of an LPADs defining r in terms of the input predicates.
- The program defined r using a number of input and hidden predicates disjoint from input and target predicates.
- Each rule in the program has a single head atom annotated with a probability.
- The program is hierarchically defined so that it can be divided into layers.

Hierarchical PLP

- Each layer contains a set of hidden predicates that are defined in terms of predicates of the layer immediately below or in terms of input predicates.
- Extreme form of program stratification: stronger than acyclicity [Apt NGC91] because it is imposed on the predicate dependency graph, and is also stronger than stratification [Chandra, Harel JLP85] that allows clauses with positive literals built on predicates in the same layer.
- It prevents inductive definitions and recursion in general, thus making the language not Turing-complete.

Hierarchical PLP

- Generic clause C:

$$C = p(\boldsymbol{X}) : \pi :- \phi(\boldsymbol{X}, \boldsymbol{Y}), b_1(\boldsymbol{X}, \boldsymbol{Y}), \ldots, b_m(\boldsymbol{X}, \boldsymbol{Y})$$

  where $\phi(\boldsymbol{X}, \boldsymbol{Y})$ is a conjunction of literals for the input predicates using variables $\boldsymbol{X}, \boldsymbol{Y}$.
- $b_i(\boldsymbol{X}, \boldsymbol{Y})$ for $i = 1, \ldots, m$ is a literal built on a hidden predicate.
- $\boldsymbol{Y}$ is a possibly empty vector of variables existentially quantified with scope the body.
- Literals for hidden predicates must use the whole set of variables $\boldsymbol{X}, \boldsymbol{Y}$.
- The predicate of each $b_i(\boldsymbol{X}, \boldsymbol{Y})$ does not appear elsewhere in the body of C or in the body of any other clause.

Hierarchical PLP

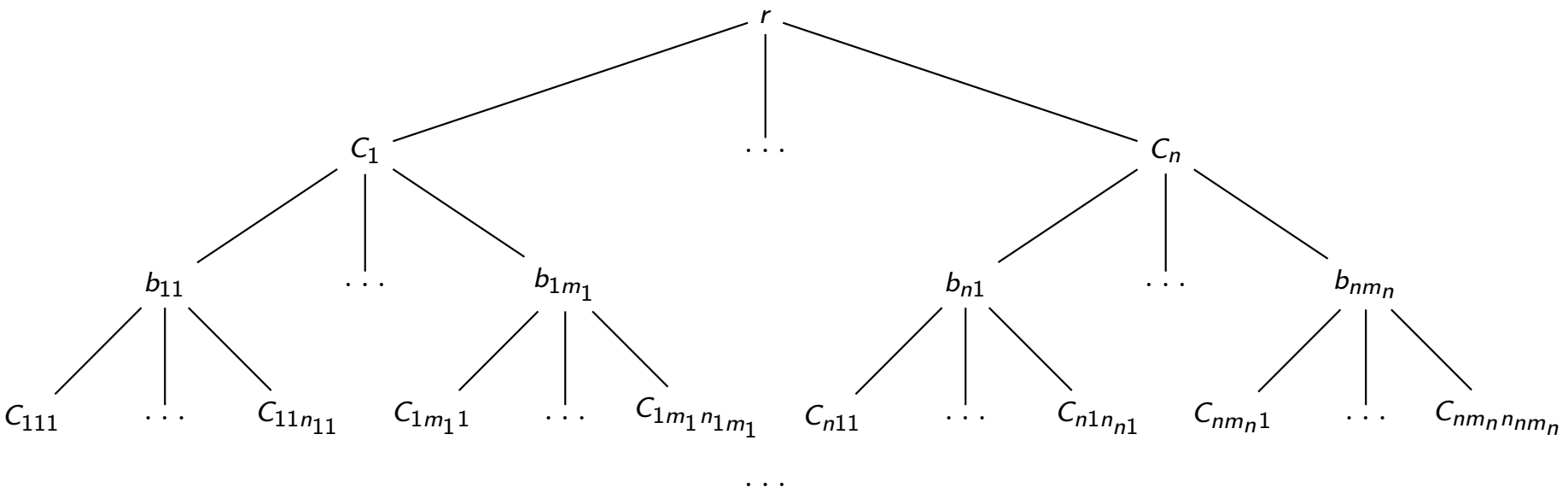
- A generic program defining r is thus:

$$
\begin{aligned}
C_1 = r(\boldsymbol{X}) : \pi_1 \quad &:- \quad \phi_1, b_{11}, \ldots, b_{1m_1} \\
&\ldots \\
C_n = r(\boldsymbol{X}) : \pi_n \quad &:- \quad \phi_n, b_{n1}, \ldots, b_{nm_n} \\
C_{111} = r_{11}(\boldsymbol{X}) : \pi_{111} \quad &:- \quad \phi_{111}, b_{1111}, \ldots, b_{111m_{111}} \\
&\ldots \\
C_{11n_{11}} = r_{11}(\boldsymbol{X}) : \pi_{11n_{11}} \quad &:- \quad \phi_{11n_{11}}, b_{11n_{11}1}, \ldots, b_{11n_{11}m_{11n_{11}}} \\
&\ldots \\
C_{n11} = r_{n1}(\boldsymbol{X}) : \pi_{n11} \quad &:- \quad \phi_{n11}, b_{n111}, \ldots, b_{n11m_{n11}} \\
&\ldots \\
C_{n1n_{n1}} = r_{n1}(\boldsymbol{X}) : \pi_{n1n_{n1}} \quad &:- \quad \phi_{n1n_{n1}}, b_{n1n_{n1}1}, \ldots, b_{n1n_{n1}m_{n1n_{n1}}} \\
&\ldots
\end{aligned}
$$

Example

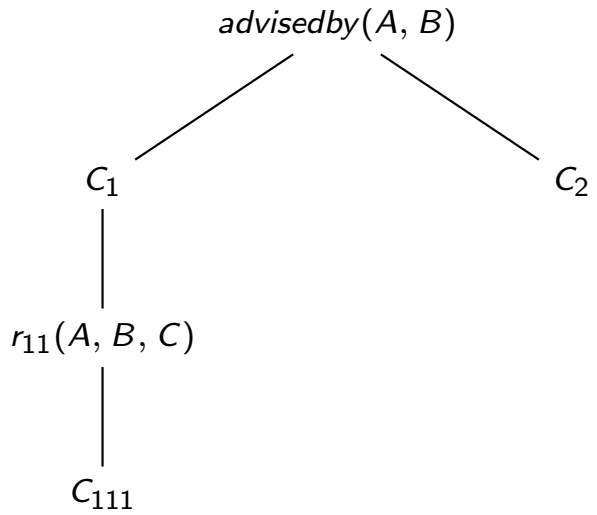
$$
\begin{aligned}
C_1 &= advisedby(A,B) : 0.3 :- \\
&\quad student(A), professor(B), project(C,A), project(C,B), \\
&\quad r_{11}(A,B,C). \\
C_2 &= advisedby(A,B) : 0.6 :- \\
&\quad student(A), professor(B), ta(C,A), taughtby(C,B). \\
C_{111} &= r_{11}(A,B,C) : 0.2 :- \\
&\quad publication(D,A,C), publication(D,B,C).
\end{aligned}
$$

Program Tree



Example

$$
\begin{aligned}
C_1 \quad &= \quad advisedby(A, B) : 0.3 :- \\
&\quad\quad student(A), professor(B), project(C, A), project(C, B), \\
&\quad\quad r_{11}(A, B, C). \\
C_2 \quad &= \quad advisedby(A, B) : 0.6 :- \\
&\quad\quad student(A), professor(B), ta(C, A), taughtby(C, B). \\
C_{111} \quad &= \quad r_{11}(A, B, C) : 0.2 :- \\
&\quad\quad publication(D, A, C), publication(D, B, C).
\end{aligned}
$$



Hierarchical PLP

- Writing programs in hierarchical PLP may be unintuitive for humans because of the need of satisfying the constraints and because the hidden predicates may not have a clear meaning.
- The structure of the program should be learned by means of a specialized algorithm
- Hidden predicates generated by a form of predicate invention.

Inference

- Generate the grounding.
- Each ground probabilistic clause is associated with a random variable whose probability of being true is given by the parameter of the clause and that is independent of all the other clause random variables.
- Ground clause $C_{\boldsymbol{p}i} = a_{\boldsymbol{p}} : \pi_{\boldsymbol{p}i} :- b_{\boldsymbol{p}i1}, \ldots, b_{\boldsymbol{p}im_{\boldsymbol{p}}}$. where $\boldsymbol{p}$ is a path in the program tree
- $P(b_{\boldsymbol{p}i1}, \ldots, b_{\boldsymbol{p}im_{\boldsymbol{p}}}) = \prod_{i=k}^{m_{\boldsymbol{p}}} P(b_{\boldsymbol{p}ik})$ and $P(b_{\boldsymbol{p}ik}) = 1 - P(a_{\boldsymbol{p}ik})$ if $b_{\boldsymbol{p}ik} = \textbf{not}\ a_{\boldsymbol{p}ik}$.
- If a is a literal for an input predicate, then $P(a) = 1$ if a belongs to the example interpretation and $P(a) = 0$ otherwise.

Inference

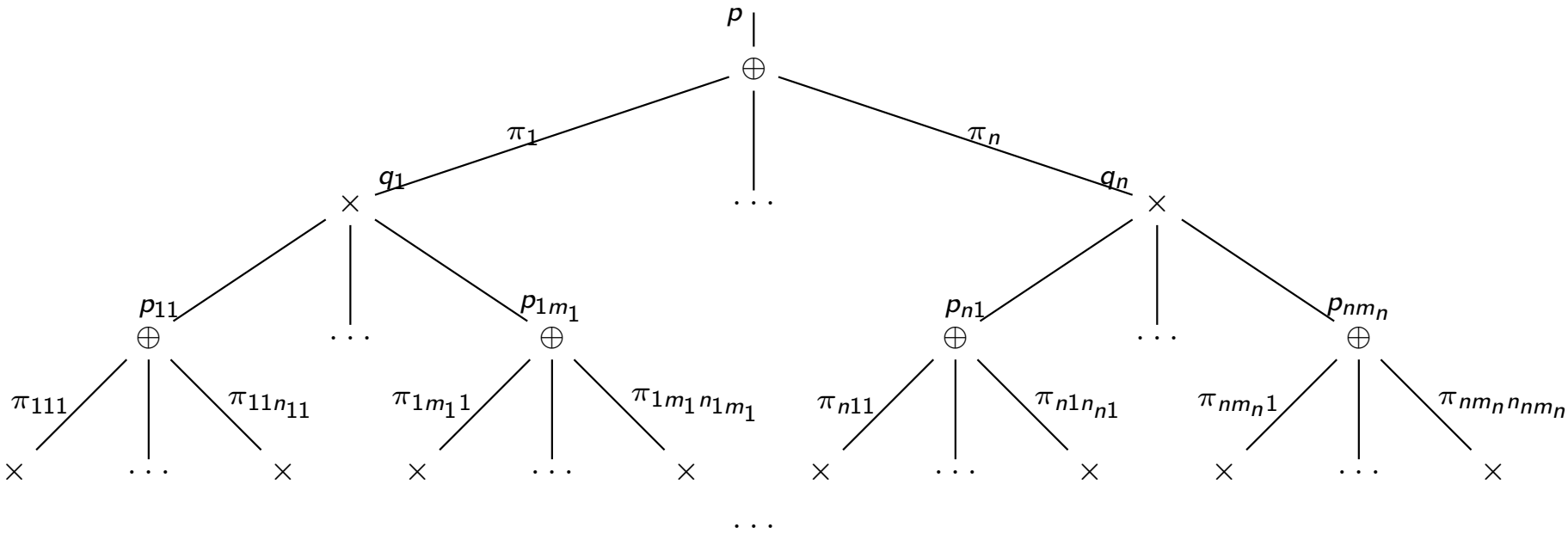
- Hidden predicates: to compute $P(a_{\boldsymbol{p}})$ we need to take into account the contribution of every ground clause for the predicate of $a_{\boldsymbol{p}}$.
- Suppose these clauses are $\{C_{\boldsymbol{p}1}, \ldots, C_{\boldsymbol{p}o_{\boldsymbol{p}}}\}$.
- If we have two clauses,
  $P(a_{\boldsymbol{p}i}) = 1 - (1 - \pi_{\boldsymbol{p}1} \cdot P(body(C_{\boldsymbol{p}1})) \cdot (1 - \pi_{\boldsymbol{p}2} \cdot P(body(C_{\boldsymbol{p}2})))$
- $p \oplus q \triangleq 1 - (1 - p) \cdot (1 - q)$.
- This operator is commutative and associative:

$$\bigoplus_i p_i = 1 - \prod_i (1 - p_i)$$

- The operators $\times$ and $\oplus$ are respectively the t-norm and t-conorm of the product fuzzy logic [Hajek 98]: product t-norm and probabilistic sum.

Inference

- If the probabilistic program is ground, the probability of the example atom can be computed with the arithmetic circuit:



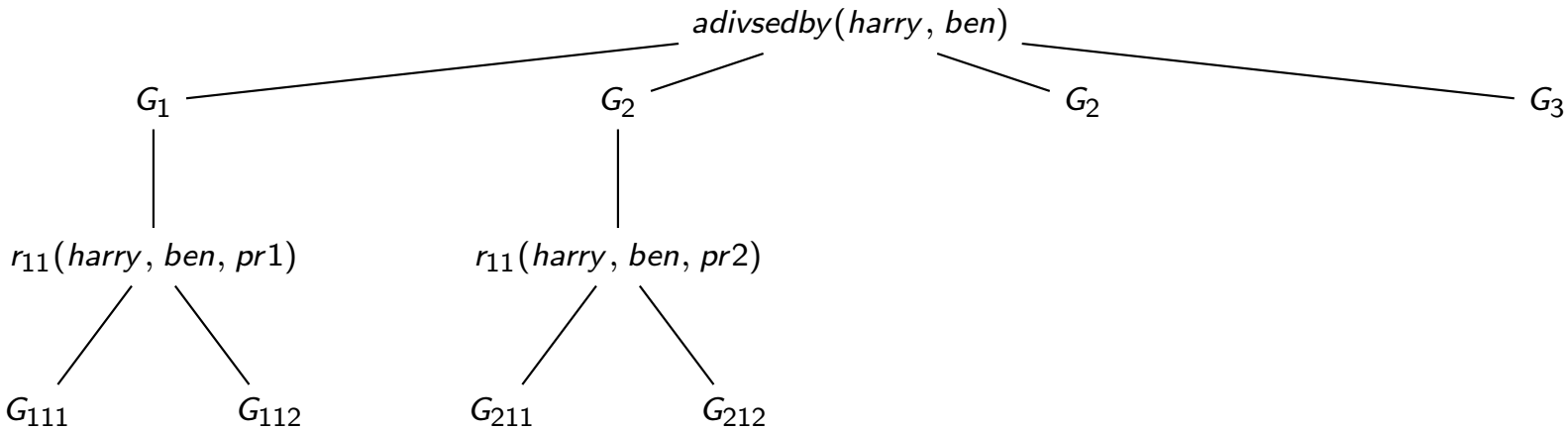
- The arithmetic circuit can be interpreted as a deep neural network where nodes have the activation functions $\times$ and $\oplus$

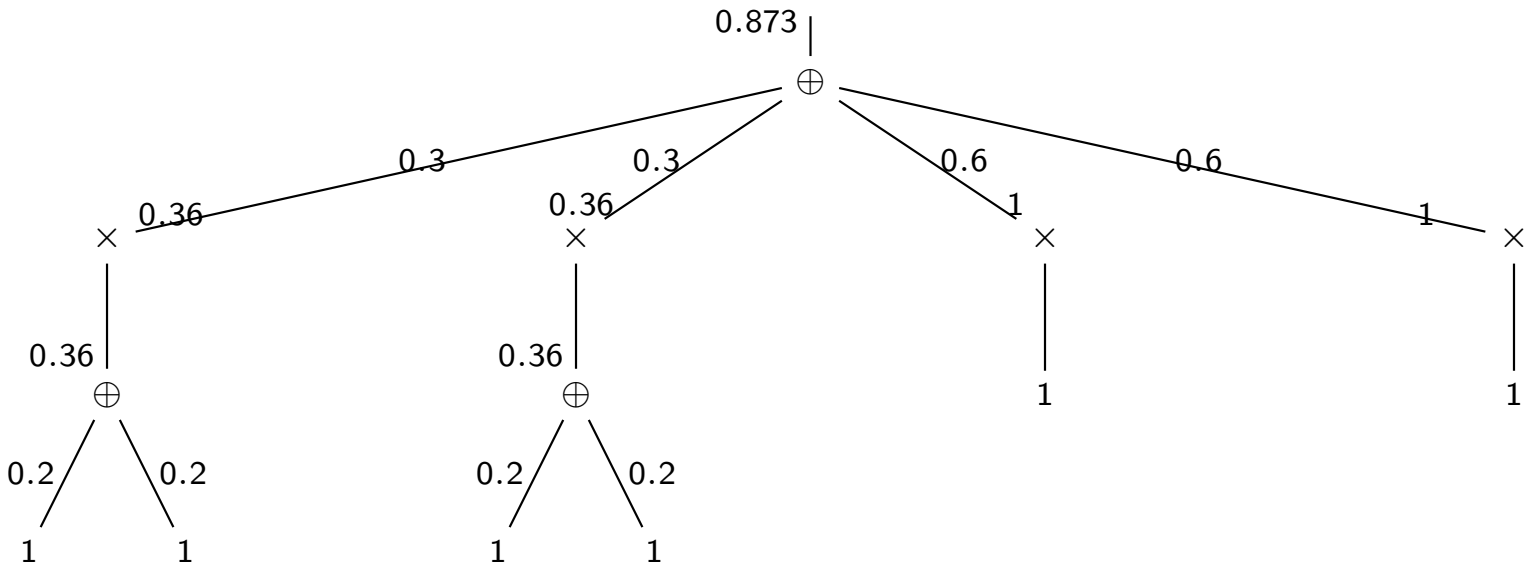
Example

$$
\begin{aligned}
G_1 &= advisedby(harry, ben) : 0.3 :- \\
&\quad student(harry), professor(ben), project(pr1, harry), \\
&\quad project(pr1, ben), r_{11}(harry, ben, pr1). \\
G_2 &= advisedby(harry, ben) : 0.3 :- \\
&\quad student(harry), professor(ben), project(pr2, harry), \\
&\quad project(pr2, ben), r_{11}(harry, ben, pr2). \\
G_3 &= advisedby(harry, ben) : 0.6 :- \\
&\quad student(harry), professor(ben), ta(c1, harry), taughtby(c1, ben). \\
G_4 &= advisedby(harry, ben) : 0.6 :- \\
&\quad student(harry), professor(ben), ta(c2, harry), taughtby(c2, ben). \\
G_{111} &= r_{11}(harry, ben, pr1) : 0.2 :- \\
&\quad publication(p1, harry, pr1), publication(p1, ben, pr1). \\
G_{112} &= r_{11}(harry, ben, pr1) : 0.2 :- \\
&\quad publication(p2, harry, pr1), publication(p2, ben, pr1). \\
G_{211} &= r_{11}(harry, ben, pr2) : 0.2 :- \\
&\quad publication(p3, harry, pr2), publication(p3, ben, pr2). \\
G_{212} &= r_{11}(harry, ben, pr2) : 0.2 :- \\
&\quad publication(p4, harry, pr2), publication(p4, ben, pr2).
\end{aligned}
$$

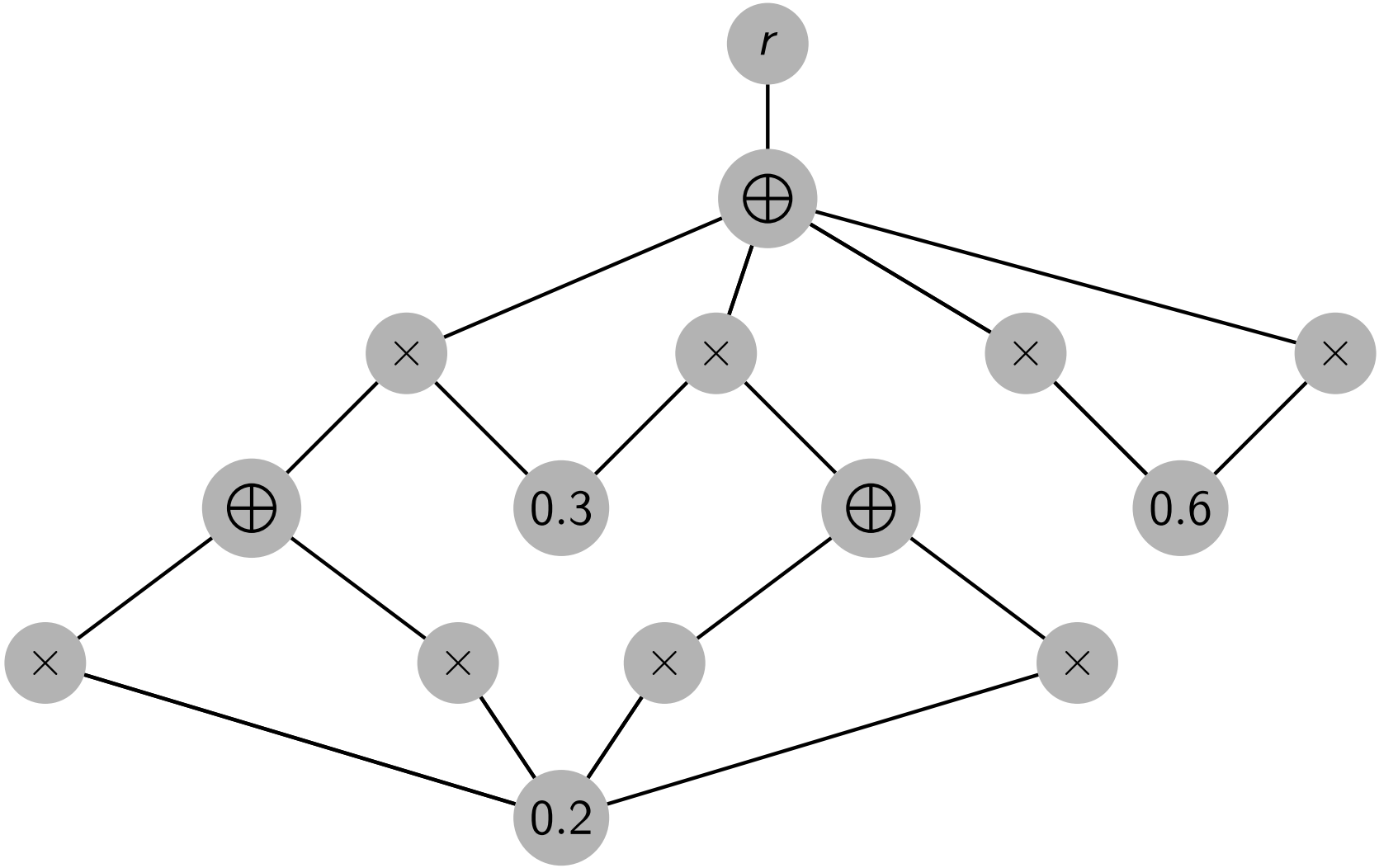
Università degli Studi di Ferrara

Example


adivsedby(harry, ben)
G1
G2
G2
G3
r11(harry, ben, pr1)
r11(harry, ben, pr2)
G111
G112
G211
G212


0.873
0.3
0.3
0.6
0.6
0.36
0.36
1
1
0.36
0.36
0.2
0.2
0.2
0.2
1
1
1
1
1
1

Arithmetic Circuit of the example



Building the Network

- The network can be built by performing inference using Logic Programming technology (tabling), e.g. PITA(IND,IND) [Riguzzi CJ14]

Parameter Learning

- Parameter learning by EM or backpropagation.
- Inference has to be performed repeatedly on the same program with different values of the parameters.
- PITA(IND,IND) can build a representation of the arithmetic circuit, instead of just computing the probability.
- Implementing EM would adapt the algorithm of [Bellodi and Riguzz IDA13] for hierarchical PLP.

Parameter Learning

- Given a Hierarchical PLP T with parameters Π, an interpretation I defining input predicates and a training set $E = \{e_1, \ldots, e_M, \textbf{not } e_{M+1}, \ldots, \textbf{not } e_N\}$ find the values of Π that maximize the log likelihood:

$$\arg\max_{\Pi} \sum_{i=1}^{M} \log P(e_i) + \sum_{i=M+1}^{N} \log(1 - P(e_i)) \tag{1}$$

  where $P(e_i)$ is the probability assigned to e_i by $T \cup I$.

- Maximizing the log likelihood can be equivalently seen as minimizing the sum of *cross entropy errors* err_i for all the examples

$$err_i = -y_i \log(P(e_i)) - (1 - y_i) \log(1 - P(e_i)) \tag{2}$$

  where $y_i = 1$ for positive example, $y_i = 0$ otherwise

Gradient Descent

- $v(n)$: value of the node n, $d(n) = \frac{\partial v(r)}{\partial v(n)}$
- The partial derivative of the error with respect to each node $v(n)$ is:

$$\frac{\partial err}{\partial v(n)} = \begin{cases} -\frac{1}{v(r)} d(n) & \text{if } e \text{ is positive,} \\ \frac{1}{1-v(r)} d(n) & \text{if } e \text{ negative.} \end{cases}$$

where

$$d(n) = \begin{cases} d(p_n)\frac{v(p_n)}{v(n)} & \text{if } n \text{ is a } \bigoplus node, \\ d(p_n)\frac{1-v(p_n)}{1-v(n)} & \text{if } n \text{ is a } \times \text{ node} \\ \sum_{p_n} d(p_n)v(p_n)(1-\Pi_i) & \text{if } n \text{ is a leaf node } \Pi_i \\ -d(p_n) & \text{if } p_n = not(n) \end{cases} \tag{3}$$

and p_n is a parent of n.

- $v(n)$ are computed in the forward pass and $d(n)$ in the backward pass of the algorithm

Structure Learning

- Writing programs in hierarchical PLP unintuitive
- The structure of the program should be learned by means of a specialized algorithm.
- Hidden predicates generated by a form of predicate invention.
- Future work

Related Work

- [Giannini et al. ECML17]
- Lukasiewicz fuzzy logic
- Continuos features
- Convex optimization problem
- Quadratic programming

Related Work

- [Sourek et al NIPS15]: build deep neural networks using a template expressed as a set of weighted rules.
- Nodes for ground atoms and ground rules
- Values of ground rule nodes aggregated to compute the value of atom nodes.
- Aggregation in two steps, first the contributions of different groundings of the same rule sharing the same head and then the contributions of groundings for different rules.
- Proposal parametric in the activation functions of ground rule nodes.
- Example: two families of activation functions that are inspired by Lukasiewicz fuzzy logic.
- We build a neural network whose output is the probability of the example according to the distribution semantics.

Related Work

- Edward [Tran et al. ICLR17]: Turing-complete probabilistic programming language
- Programs in Edward define computational graphs and inference is performed by stochastic graph optimization using TensorFlow.
- Hierarchical PLP is not Turing-complete as Edward but ensures fast inference by circuit evaluation.
- Being based on logic it handles well domains with multiple entities connected by relationships.
- Similarly to Edward, hierarchical PLP can be compiled to TensorFlow

Related Work

- Probabilistic Soft Logic (PSL) [Bach et al. arXiv15]: Markov Logic with atom random variables taking continuous values in $[0, 1]$ and logic formulas interpreted using Lukasiewicz fuzzy logic.
- PSL defines a joint probability distribution over fuzzy variables, while the random variables in hierarchical PLP are still Boolean and the fuzzy values are the probabilities that are combined with the product fuzzy logic.
- The main inference problem in PSL is MAP rather than MARG as in hierarchical PLP.

Related Work

- Sum-product networks [Poon, Domingos UAI11]: hierarchical PLP circuits can be seen as sum-product networks where children of sum nodes are not mutually exclusive but independent and each product node has a leaf child that is associated to a hidden random variable.
- Sum-product networks represent a distribution over input data while programs in hierarchical PLP describe only a distribution over the truth values of the query.
- Inference in hierarchical PLP is in a way "lifted": the probability of the ground atoms can be computed knowing only the sizes of the populations of individuals that can instantiate the existentially quantified variables

Related Work

- *Neural Logic Programming* [Yang et al NIPS17]
- Embedding: given the set of n entities and the set of r binary relations
  - each entity is a vector $\{0,1\}^n$ with all 0 except for the position corresponding to the index associated to the entity;
  - each predicate is a matrix $\{0,1\}^{n \times n}$ with all 0 except for the positions i, j where the two associated entities are linked by the predicate.

Neural Logic Programming

- Inference is done by means of matrix multiplications.
- Given the rule $\alpha : R(Y, X) \leftarrow P(Y, Z) \wedge Q(Z, X)$ and entity X,
  - inference is performed by multiplying matrices of P and Q with vector for X. The resulting vector has value 1 in correspondence of the values taken by Y.
- The confidence of each result is the value computed by summing the confidences of each rule that implies the query.

Neural Logic Programming

- As the distribution semantics but rules are considered as exclusive
- Learning the form of rules and the weights by means of a neural controller

Related Work

- End-to-end Differentiable Proving [Rocktaschel and Riedel NIPS2017]
- Constants and predicates represented by real vectors
- Unification replaced by approximate matching by similarity
- Logical operations implemented by differentiable operations
- Prolog backward chaining for building neural nets
- Learning by means of gradient descent: rules with fixed structure, tuning of the embedding

Open Problems

- Web data: Semantic Web, knowledge graphs, Wikidata, semantically annotated Web pages, text, images, videos, multimedia.
- Uncertain, incomplete, or inconsistent information, complex relationships among individuals, mixed discrete and continuous unstructured data and extremely large size.
- Uncertainty $\rightarrow$ graphical models, entities connected by relations $\rightarrow$ logic, mixed discrete and continuous data $\rightarrow$ kernel machines/deep learning
- Up to now combinations of pairs of techniques: probability and logic $\rightarrow$ Statistical Relational Artificial Intelligence, graphical models and kernel machines/deep learning $\rightarrow$ Sum-Product networks, and logic and kernel machines/deep learning $\rightarrow$ neuro-symbolic systems.
- We need a combination of the three approaches.

THANKS FOR LISTENING AND ANY QUESTIONS ?

Università degli Studi di Ferrara