Neuro-Symbolic Integration (NeSy)

Andrea Passerini
andrea.passerini@unitn.it

Advanced Topics in Machine Learning and Optimization

Deep Learning

PROs

- Efficient processing of high-dimensional data
- Robust to noise and ambiguity
- Does not require extensive background knowledge and feature engineering

CONs

- Data hungry (large training sets needed)
- Non-interpretable models and predictions
- Hard to incorporate complex domain knowledge

Symbolic Reasoning

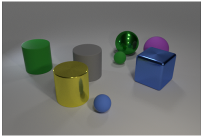
PROs

- Expressive, can formalize complex domain knowledge
- Interpretable, inference can be explained in terms reasoning steps (proofs)
- Can generalize from few examples

CONs

- Inefficient, inference is typically expensive
- No support for noise or ambiguity
- Difficult to deal with high-dimensional data

Neuro-Symbolic Integration (NeSy)



Q: How many objects are both right of the green cylinder and have the same material as the small blue ball?
A: 3

Best of both worlds

- Deep networks for low-level data processing and “atomic” predictions
- Symbolic approaches for reasoning on top of atomic predictions
- Probabilities (or scores) for dealing with uncertainty

Image from Mao et al. 2019

Dimensions: directed vs undirected models

Directed models

- Generalize Bayesian Networks to deal with (first-order) logic
- Generalize Logic Programs to deal with probabilities
- Incorporare Neural "primitives" (e.g., predicates)

Undirected models

- Generalize Markov Networks to deal with (first-order) logic
- Enforce logical constraints over neural predictions
- Relax logical constraints to deal with uncertainty

Dimensions: integration vs regularization

Integration

- Neural primitives inside reasoning framework (typically logic program)
- Differentiability via probability of worlds or proof score.

Regularization

- Logical Constraints are used as regularizers for neural network training
- Differentiability by relaxed constraints or consistency in expectation

Dimensions: semantics

Probabilistic semantics

- Extends Boolean logic with probabilities
- Defines a probability distribution over possible worlds
- Allows to perform inference under uncertainty (expensive)

Fuzzy semantics

- Relax Boolean variables in [0,1] interval
- Relies on t-norms for relaxing Boolean connectives
- Efficient inference, Boolean semantics not preserved

Semantic-based Regularization

Setting

- Model problems with multiple related predictions
- Incorporate knowledge as constraints over related predictions

Solution

- Model each prediction task with a statistical learner (kernel machine, neural network)
- Represent constraints over predictions in fuzzy logic
- Combine regularization with loss on fuzzy constraint satisfaction (including label supervision)

Semantic-based Regularization: Fuzzy logic

Boolean	Gödel	Product	Łukasiewicz
$X \wedge Y$	$\min(X, Y)$	$X\,Y$	$\max(0, X + Y - 1)$
$X \vee Y$	$\max(X, Y)$	$1 - (1 - X)(1 - Y)$	$\min(1, X + Y)$
$\neg X$	$1 - X$	$1 - X$	$1 - X$

Fuzzy logic

- Boolean variables relaxed into real variables in $[0, 1]$.
- Conjunction relaxed using **t-norm**
- Disjunction relaxed using **t-conorm**
- Existential quantifier relaxed as maximum (over dataset)
- Universal quantifier relaxed as minimum (over dataset, usually replaced by average)

Semantic-based Regularization: formulation

$$\mathcal{L}(\boldsymbol{f}, \Phi) = \sum_{k=1}^{|\boldsymbol{f}|} ||f_k||^2 + \sum_{h=1}^{|\Phi|} \lambda_h (1 - \hat{\Phi}_h(\boldsymbol{f}))$$

Objective function

- $\boldsymbol{f}$ is a vector of parameterized predictors (one per task)
- Φ is a set of logic formulas (the constraints)
- $||f_k||$ is the norm of f_k (e.g. norm of the weights for kernel machines)
- λ_h is a weight associated to constraint h
- $\hat{\Phi}_h$ is the fuzzy version of formula Φ_h

Semantic-based Regularization: example

positive supervision

manifold regularisation

$$\begin{aligned} F &:= \forall d\ P_A(d) \Rightarrow A(d) \\ F_R &:= \forall d\ \forall d'\ R(d,d') \Rightarrow \big((A(d) \wedge A(d')) \vee (\neg A(d) \wedge \neg A(d'))\big) \\ C &= \{d_1, d_2\} \end{aligned}$$

Evidence Predicate Groundings

$P_A(d_1) = 1$
$R(d_1, d_2) = 1$

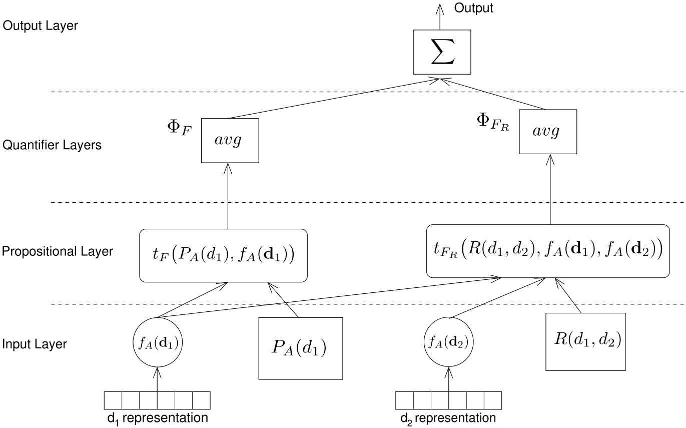


Image adapted from Diligenti et al., 2017

Semantic-based Regularization: learning

$$\frac{\partial \mathcal{L}(\boldsymbol{f}, \Phi)}{\partial w_{k,j}} = \frac{\partial ||f_k||^2}{\partial w_{k,j}} + \sum_{h=1}^{|\Phi|} \lambda_h \frac{\partial (1 - \hat{\Phi}_h)}{\partial \hat{\Phi}_h} \cdot \left(\sum_{t_{\Phi_h}} \frac{\partial t_{\Phi_h}}{\partial f_k} \cdot \frac{\partial f_k}{\partial w_{k,j}} \right)$$

Gradient-based learning

- $w_{k,j}$ is a parameter of a predictor f_k
- t_{Φ_h} is a grounding of formula Φ_h

Note

Learning problem is convex if:

- f_k are kernel machines (or similar)
- A convex fragment of the Łukasiewicz logic is used

Semantic-based Regularization: MAP inference

$$\mathcal{L}(\bar{\boldsymbol{f}}(\mathcal{X}), \boldsymbol{f}(\mathcal{X})) = \frac{1}{2}||\bar{\boldsymbol{f}}(\mathcal{X}) - \boldsymbol{f}(\mathcal{X})||^2 + \sum_h \lambda_h \left(1 - \hat{\Phi}_h(\bar{\boldsymbol{f}}(\mathcal{X}))\right)$$

Gradient-based MAP inference

- $\mathcal{X}$ set of (related) test examples
- $\boldsymbol{f}(\mathcal{X})$ set of independent predictions over test examples
- $\bar{\boldsymbol{f}}(\mathcal{X})$ set of collective predictions over test examples (accounting for constraints)
- Inference of $\bar{\boldsymbol{f}}(\mathcal{X})$ is performed by gradient descent:

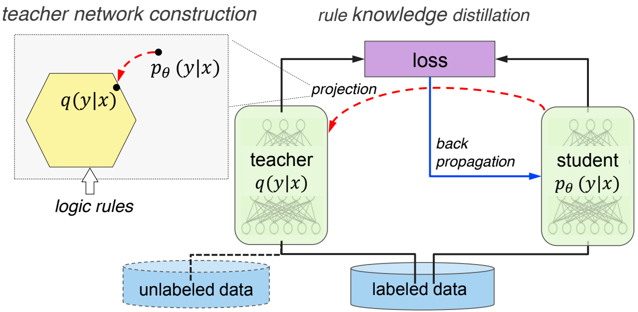
$$\frac{\mathcal{L}(\bar{\boldsymbol{f}}(\mathcal{X}), \boldsymbol{f}(\mathcal{X}))}{\partial \bar{\boldsymbol{f}}_k(\mathcal{X}_i)} = \bar{\boldsymbol{f}}_k(\mathcal{X}_i) - \boldsymbol{f}_k(\mathcal{X}_i) + \sum_h \lambda_h \left(\frac{\partial 1 - \hat{\Phi}_h(\bar{\boldsymbol{f}}(\mathcal{X}))}{\partial \bar{\boldsymbol{f}}_k(\mathcal{X}_i)} \right)$$

Semantic-based Regularization: dimensions

dimensions

- **Undirected model**: constraints as set of FOL formulas (probabilistc variant as deep Markov Logic Network exists)
- **Regularization approach**: soft consistency is a regularization term in training loss
- **Fuzzy semantics**: fuzzy logic is employed as relaxation

Knowledge distillation



Teacher-student distillation

- Student learns to fit data and satisfy rules
- Teacher "shows" student how to change predictions to satisfy rules (projection in feasible space)
- Student should learn to implicitly satisfy rules (no rule enforcement at prediction time)

Knowledge distillation: learning

$$\mathcal{L}(\mathcal{D};\Phi) = \sum_{(\boldsymbol{x}_n,\boldsymbol{y}_n)\in\mathcal{D}} (1-\pi)\ell(\boldsymbol{y}_n, f_p(\boldsymbol{x}_n)) + \pi\ell(f_q(\boldsymbol{x}_n), f_p(\boldsymbol{x}_n))$$

Iterative procedure

- $f_p(\boldsymbol{x}_n)$ are the student predictions for $\boldsymbol{x}_n$ (i.e., according to $p_\theta(\boldsymbol{y}|\boldsymbol{x}_n)$)
- $f_q(\boldsymbol{x}_n)$ is the teacher projection of those predictions in the feasible space Φ (i.e., according to $q(\boldsymbol{y}|\boldsymbol{x}_n)$)
- π is a parameter trading-off data fitting and constraint satisfaction (possibly on unlabelled data too)
- At each iteration θ is updated minimizing the loss

Knowledge distillation: teacher projection

$$\min_{q,\xi} \quad KL(q(Y|X)||p_\theta(Y|X)) + C\sum_h \sum_g \xi_{h,g}$$

$$\text{s.t.} \quad \lambda_h(1 - E_q[\hat{\Phi}_{h,g}(X,Y)]) \leq \xi_{h,g}$$

Projection as constrained optimization

- KL divergence between student and teacher predictions
- $\hat{\Phi}_{h,g}(X,Y)$ is the g-th grounding of a fuzzy version of formula Φ_h on (X,Y).
- $E_q[\hat{\Phi}_{h,g}(X,Y)]$ is satisfaction of $\hat{\Phi}_{h,g}(X,Y)$ in expectation over $q(Y|X)$.
- λ_h is the weight of formula Φ_h
- $\xi_{h,g}$ is a slack variable to penalize unsatisfied constraints
- C is a parameter trading-off divergence with student prediction and satisfaction of formulas

Knowledge distillation: teacher projection

$$q^*(Y|X) \propto p_\theta(Y|X) \cdot \exp\left(-\sum_h \sum_g C\lambda_h(1 - \hat{\Phi}_{h,g}(X,Y))\right)$$

Closed form solution

- The constrained otimization problem has a closed form solution.
- The normalization term is computed by dynamic programming if relationship between constraints allows for it, or approximated with sampling approaches otherwise.

Knowledge distillation: dimensions

dimensions

- **Undirected model**: constraints as set of FOL formulas
- **Regularization approach**: projection on consistent predictions is a regularization term in training loss
- **Fuzzy semantics**: fuzzy logic is employed as relaxation

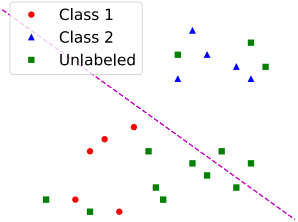
Semantic Loss Regularization

Semantic Loss

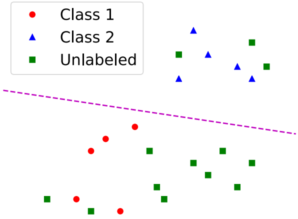
$$\mathcal{L}_s(\phi, \boldsymbol{p}) \propto -\log \sum_{\mathbf{y} \models \phi} \prod_{\mathbf{y} \models Y_i} p_i \prod_{\mathbf{y} \models \neg Y_i} (1 - p_i)$$

- ϕ is a propositional formula (a constraint that should hold)
- $\boldsymbol{p}$ is a vector of probabilities associated to $\boldsymbol{Y}$ variables (e.g. outputs of a neural network)
- The semantic loss is proportional to the negative logarithm of the probability that sampling $\boldsymbol{Y}$ according to $\boldsymbol{p}$ produces a value $\mathbf{y}$ satisfying the constraint ϕ.

Semantic Loss Regularization



$\phi = \top$



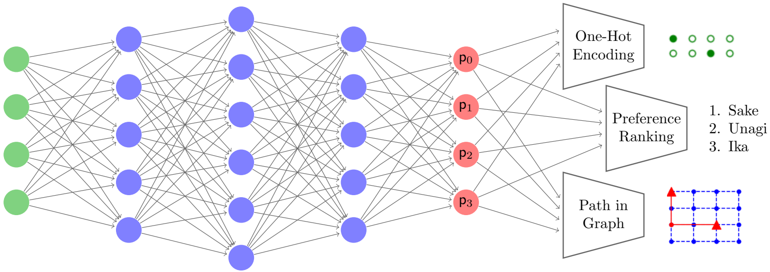
$\phi = \neg Y_1 \vee \neg Y_2$

Regularizing with semantic Loss

$$\mathcal{L}_{reg} = \textit{traning\_loss} + \lambda\ \textit{semantic\_loss}$$

- Semantic loss as regularizer of training loss (encourages predictions satisfying constraints)

Semantic Loss Regularization



End-to-end training with semantic Loss

- Semantic loss can be compiled into an arithmetic circuit
- Partial derivatives can be computed on the circuit (see e.g. Deep ProbLog)

Semantic Loss Regularization: dimensions

dimensions

- **Undirected model**: constraints as set of propositional formulas
- **Regularization approach**: semantic loss is additional term to training loss
- **Probabilistic semantics**: constraints are enforced in expectation over probabilities of possible worlds

Deep ProbLog

```
nn(m_digit, [X], Y, [0, . . . , 9]) :: digit(X, Y).
```

From ProbLog to Deep ProbLog

- Introduce neural networks to process low-level data (softmax output layer)
- **neural annotated disjunction** (nAD) maps inputs to distributions over candidate outputs
- `nn` is a reserved word (stands for neural network)
- `m_digit` is the identifier of a neural network (CNN classifying digit images)
- `digit` is a **neural predicate** evaluated via `m_digit`.

Deep ProbLog: nAD example

nn(m_digit, [X], Y, [0, ..., 9]) :: digit(X, Y).

ground on 

nn(m_digit, [], 0) :: digit(, 0) ; ... ; nn(m_digit, [], 9) :: digit(, 9).

evaluate m_digit on 

p_0 :: digit(, 0) ; ... ; p_9 :: digit(, 9).

Deep ProbLog: inference

Inference by knowledge compilation

1. Ground relevant part of the program to answer query (including nADs).
2. Run forward step in neural nets to turn ground nAD into ground AD.
3. Compile resulting formula (same as ProbLog)
4. convert into AC (same as ProbLog)
5. evaluate AC (same as ProbLog)

Deep ProbLog: grounding example

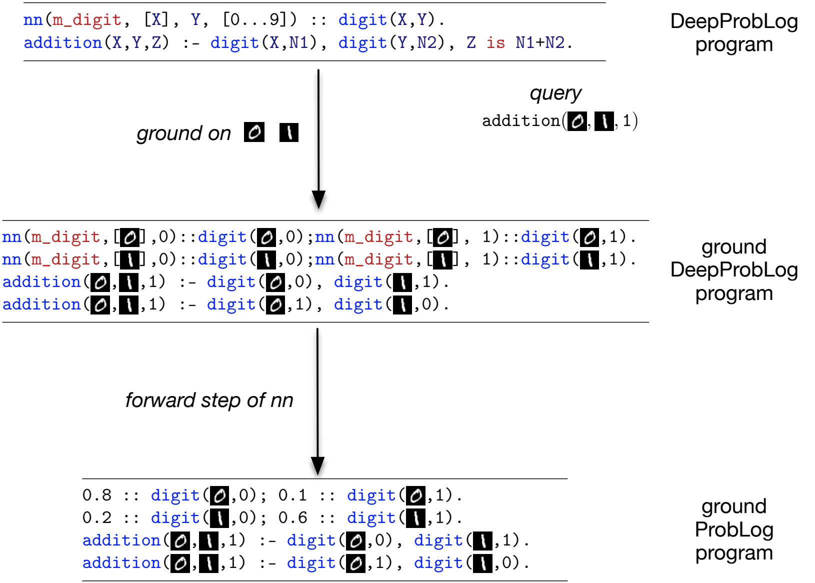


Image adapted from Manhaeve et al., 2019

Deep ProbLog: learning

Learning by gradient descent in ProbLog

- Gradient computation can be done over arithmetic circuit used for inference.
- Need to replace probability semiring used for inference with **gradient semiring** (algebraic Problog)
- Gradient update followed by normalization to get valid probabilities

Deep ProbLog: probability vs gradient semiring

probability	gradient
$a \oplus b = a + b$	$(a, \boldsymbol{a}_\nabla) \oplus (b, \boldsymbol{b}_\nabla) = (a + b, \boldsymbol{a}_\nabla + \boldsymbol{b}_\nabla)$
$a \otimes b = a\,b$	$(a, \boldsymbol{a}_\nabla) \otimes (b, \boldsymbol{b}_\nabla) = (a\,b, a\,\boldsymbol{b}_\nabla + b\,\boldsymbol{a}_\nabla)$
$e^\oplus = 0$	$e^\oplus = (0, 0_\nabla)$
$e^\otimes = 1$	$e^\oplus = (1, 0_\nabla)$
$L(f) = p$	$L(f) = (p, 0_\nabla)$ (fixed p)
$L(f_i) = p_i$	$L(f_i) = (p_i, \boldsymbol{e}_i)$ (learnable p_i)
$L(\neg f) = 1 - p$	$L(\neg f) = (1 - p, -\nabla p)$ (with $L(f) = (p, \nabla p)$)

ProbLog: gradient semiring example

```
0.2::earthquake.
0.1::burglary.
0.5::hears_alarm(mary).
0.4::hears_alarm(john).
alarm :- earthquake.
alarm :- burglary.
calls(X):-alarm,hears_alarm(X).
```

grounding for

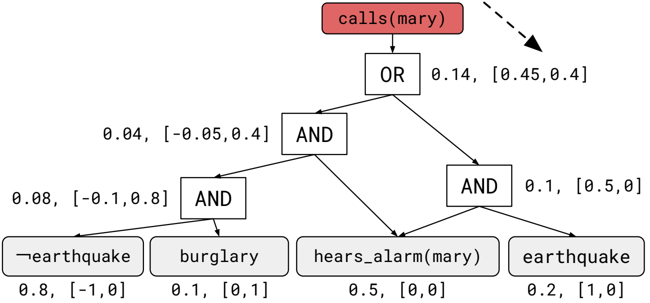
calls(mary)

```
0.2::earthquake.
0.1::burglary.
0.5::hears_alarm(mary).

alarm :- earthquake.
alarm :- burglary.
calls(mary):-alarm,hears_alarm(mary).
```

learnable parameters

compilation using gradient semiring



Deep ProbLog: learning

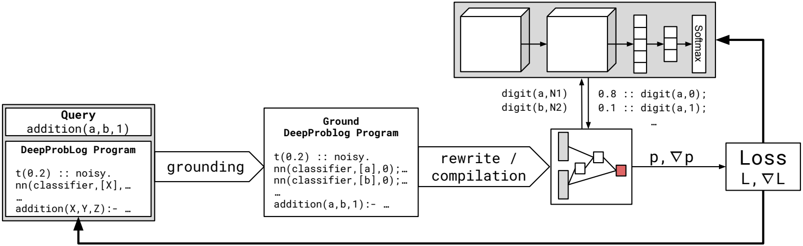
Learning by gradient descent in DeepProbLog

- Use gradient semiring as for ProbLog (considering outputs of neural predicates as abstract parameters).
- Backpropagate gradient from abstract parameters into the corresponding neural network

$$\frac{d\mathcal{L}}{d\theta_k} = \frac{d\mathcal{L}}{dP(q)} \sum_{i=1}^{m} \frac{dP(q)}{d\hat{p}_i} \frac{d\hat{p}_i}{d\theta_k}$$

- $\mathcal{L}$ is a loss function
- $P(q)$ is the probability of a traning example q (query)
- m is the number of outputs of a neural network (alternatives)
- $\hat{p}_i$ is the i-th output of the network for example q.
- θ_k is the k-th parameter of a neural network

Deep ProbLog: learning pipeline



Deep ProbLog: dimensions

dimensions

- **Directed model**: probabilistic logic program (definite clauses)
- **Integration approach**: probabilistic logic program enriched with neural predicates
- **Probabilistic semantics**: constraints are enforced in expectation over probabilities of possible worlds

Neural Theorem Proving

Motivation

- Theorem proving allows to infer novel facts entailed by a KB, but fails with noisy or ambiguous knowledge (e.g. slightly different names for the same relation)
- Neural models are robust to noise and ambiguity but have limited reasoning capabilities
- Neural theorem proving aims at combining the best of both worlds

Neural Theorem Proving

In a nutshell

- End-to-end differentiable deductive reasoner
- Use Prolog backward-chaining algorithm for proving goals
- Replace symbolic unification between atoms with a differantiable similarity between their embeddings
- Collect the highest scoring proof as the goal proof
- Embeddings are learned by gradient descent over goal proofs for true (positive) and false (negative) facts.

Neural Theorem Proving: Prolog backward chaining

```
grandfatherOf(X, Y) :- fatherOf(X, Z), fatherOf(Z, Y).
grandfatherOf(X, Y) :- fatherOf(X, Z), motherOf(Z, Y).
fatherOf(tom, ann).
motherOf(ann, bart).
```

OR / AND search

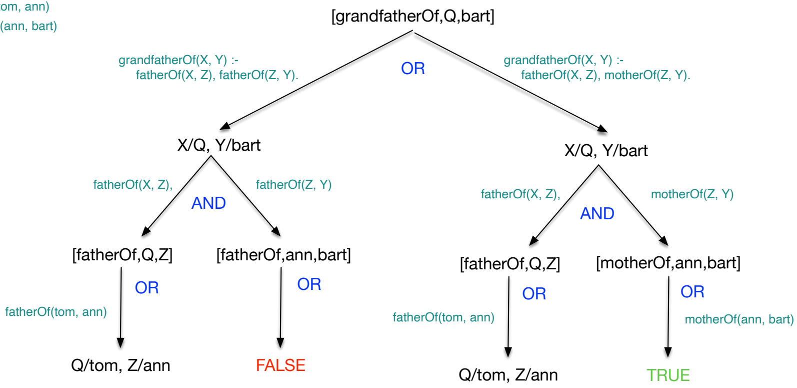
- OR iterates over all rules and unifies the rule head with the goal (one rule suffice)
- AND iterates over all atoms in the body of the rule (all atoms should be proved)
- OR is recursively applied to each atom in the body

Prolog backward chaining: example

grandfatherOf(X, Y) :- fatherOf(X, Z), fatherOf(Z, Y).
grandfatherOf(X, Y) :- fatherOf(X, Z), motherOf(Z, Y).

fatherOf(tom, ann)

motherOf(ann, bart)



Neural Theorem Proving: unification

previous variable unifications

previous soft unifications (neural)

$$\texttt{unify}_{\boldsymbol{\theta}}([\texttt{grandpaOf}, \text{ABE}, \text{BART}], [s, \text{Q}, i], (\varnothing, \rho)) = (S'_{\psi}, S'_{\rho}) =$$
$$(\{\text{Q}/\text{ABE}\}, \min\left(\rho, \exp(-\|\boldsymbol{\theta}_{\texttt{grandpaOf}:} - \boldsymbol{\theta}_{s:}\|_2), \exp(-\|\boldsymbol{\theta}_{\text{BART}:} - \boldsymbol{\theta}_{i:}\|_2)\right))$$

variable unification

soft unifications

embeddings

Soft unification

- Variables unify with variables or symbols as in Prolog
- Constants and predicates unify softly via similarity of their embeddings

Neural Theorem Proving: OR

grandfatherOf(X, Y) :- fatherOf(X, Z), parentOf(Z, Y).

current state
(possible unifications)

body matching

head unification

$$\mathtt{or}_{\boldsymbol{\theta}}^{\mathfrak{K}}([s, \mathrm{Q}, i], d, S) = [S' | S' \in \mathtt{and}_{\boldsymbol{\theta}}^{\mathfrak{K}}([[\mathtt{fatherOf}, \mathrm{X}, \mathrm{Z}], [\mathtt{parentOf}, \mathrm{Z}, \mathrm{Y}]], d, \underbrace{(\{\mathrm{X}/\mathrm{Q}, \mathrm{Y}/i\}, \hat{S}_\rho)}_{\text{result of } \mathtt{unify}}), \dots]$$

max proof depth

other rules
matching the goal

OR module

- The goal is (soft) unified with the head of a rule (for all possible rules that soft unify)
- The AND module is called for all atoms in the body

Neural Theorem Proving: AND

$$\texttt{and}_{\boldsymbol{\theta}}^{\mathfrak{K}}([[\texttt{fatherOf}, \mathrm{X}, \mathrm{Z}], [\texttt{parentOf}, \mathrm{Z}, \mathrm{Y}]], d, \underbrace{(\{\mathrm{X}/\mathrm{Q}, \mathrm{Y}/i\}, \hat{S}_{\rho})}_{\text{result of } \texttt{unify} \text{ in } \texttt{or}}) =$$

$$[S''|S'' \in \texttt{and}_{\boldsymbol{\theta}}^{\mathfrak{K}}([[\texttt{parentOf}, \mathrm{Z}, \mathrm{Y}]], d, S') \text{ for } S' \in \texttt{or}_{\boldsymbol{\theta}}^{\mathfrak{K}}(\underbrace{[\texttt{fatherOf}, \mathrm{Q}, \mathrm{Z}]}_{\text{result of substitute}}, d-1, \underbrace{(\{\mathrm{X}/\mathrm{Q}, \mathrm{Y}/i\}, \hat{S}_{\rho})}_{\text{result of } \texttt{unify} \text{ in } \texttt{or}})]$$

AND called
on remaining atoms

OR called
on first atom

max depth is
reduced

AND module

- The AND module fails if the maximum depth is reached (or the upstream OR failed)
- The AND module succeeds if it reaches the end of the list of atoms
- Otherwise it recurs over the atoms substituting variables wherever possible and calling OR

Neural Theorem Proving: Proof

$$\mathtt{ntp}_{\boldsymbol{\theta}}^{\mathfrak{K}}(\mathrm{G}, d) = \mathop{\arg\max}_{\substack{S \,\in\, \mathtt{or}_{\boldsymbol{\theta}}^{\mathfrak{K}}(\mathrm{G}, d, (\varnothing, 1)) \\ S \neq \mathtt{FAIL}}} S_{\rho}$$

Proof with maximal score

- The search is initialized with an empty substitution set and a score of 1
- The maximization is over all possible goal proofs
- The score of a proof is the minimal score of all soft unifications in the proof

Neural Theorem Proving: proof example

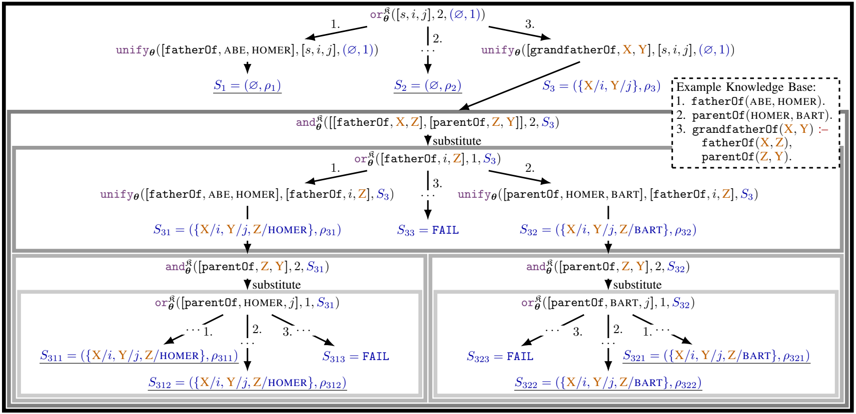


Image from Rocktäschel and Riedel, 2017

Neural Theorem Proving: prediction examples

QUERY: `part_of(CONGO.N.03,AFRICA.N.01)`

Score	Proofs
0.995	`part_of(X, Y) :- has_part(Y, X)` `has_part(AFRICA.N.01, CONGO.N.03)`
0.787	`part_of(X, Y) :- instance_hyponym(Y, X)` `instance_hyponym(AFRICAN COUNTRY.N.01, CONGO.N.03)`

QUERY: `hyponym(EXTINGUISH.V.04, DECOUPLE.V.03)`

Score	Proofs
0.987	`hyponym(X, Y) :- hypernym(Y, X)` `hypernym(DECOUPLE.V.03, EXTINGUISH.V.04)`

Neural Theorem Proving: dimensions

dimensions

- **Directed model**: logic program (definite clauses)
- **Integration approach**: logic program enriched with neural similarity in place of symbolic unification
- **“Fuzzy” semantics**: a score is associated to a proof, no explicit probabilistic interpretation

References

Bibliography

- Luc de Raedt, Sebastijan Dumancic, Robin Manhaeve, Giuseppe Marra, *From Statistical Relational to Neuro-Symbolic Artificial Intelligence*, In IJCAI 2020.
- Michelangelo Diligenti, Marco Gori, Claudio Saccà, *Semantic-based regularization for learning and inference*, Artificial Intelligence, Volume 244, 2017.
- Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, Eric Xing, *Harnessing Deep Neural Networks with Logic Rules*, Proc. of ACL, 2016.
- Xu, J., Zhang, Z., Friedman, T., Liang, Y. and Van den Broeck, G., *A Semantic Loss Function for Deep Learning with Symbolic Knowledge*. In ICML 2018.
- Manhaeve R., Dumancic S., Kimmig A., Demeester T. and De Raedt, Luc., *DeepProbLog: Neural Probabilistic Logic Programming*. In NeurIPS 2018.
- T. Rocktäschel and S. Riedel, *End-to-End Differentiable Proving*, Proc. of NIPS 2017.

References

Software Libraries

- Semantic-based regularization (SBR) [https://sites.google.com/site/semanticbasedregularization/home/software]
- Deep ProbLog [https://bitbucket.org/problog/deepproblog/src/master/]
- Greedy Neural Theorem Provers (GNTP) [https://github.com/uclnlp/gntp]