Deep Prolog: End-to-end Differentiable Proving in Knowledge Bases

Tim Rocktäschel

University College London
Computer Science

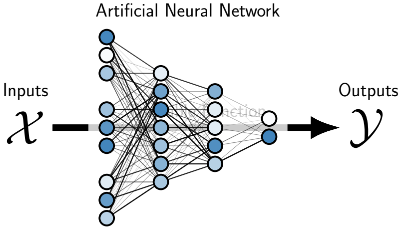
2nd Conference on Artificial Intelligence and Theorem Proving

26th of March 2017

Overview

Machine Learning

Deep Learning



- Behavior learned automatically
- Strong generalization
- Needs a lot of training data
- Behavior not interpretable

First-order Logic

"Every father of a parent is a grandfather."

```
grandfatherOf(X, Y) :–
    fatherOf(X, Z),
    parentOf(Z, Y).
```

- Behavior defined manually
- No generalisation
- Needs no training data
- Behavior interpretable

Outline

Outline

Notation

- **Constant**: HOMER, BART, LISA etc. (lowercase)
- **Variable**: X, Y etc. (uppercase, universally quantified)
- **Term**: constant or variable
- **Predicate**: `fatherOf`, `parentOf` etc.
  function from terms to a Boolean
- **Atom**: predicate and terms, e.g., `parentOf`(X, BART)
- **Literal**: negated or non-negated atom, e.g.,
  not `parentOf`(BART, LISA)
- **Rule**: `head` :– `body`.
  `head`: literal
  `body`: (possibly empty) list of literals representing conjunction
- **Fact**: ground rule (no free variables) with empty body, e.g.,
  `parentOf`(HOMER, BART).

Example Knowledge Base

1 fatherOf(ABE, HOMER).
2 parentOf(HOMER, LISA).
3 parentOf(HOMER, BART).
4 grandpaOf(ABE, LISA).
5 grandfatherOf(ABE, MAGGIE).
6 grandfatherOf(X_1, Y_1) :–
    fatherOf(X_1, Z_1),
    parentOf(Z_1, Y_1).
7 grandparentOf(X_2, Y_2) :–
    grandfatherOf(X_2, Y_2).

Backward Chaining

```
def or(KB, goal, Ψ):
    for rule head :– body in KB do
        Ψ′ ← unify(head, goal, Ψ)
        if Ψ′ ≠ failure then
            for Ψ″ in and(KB, body, Ψ′) do
                yield Ψ″

def and(KB, subgoals, Ψ):
    if subgoals is empty then return Ψ;
    else
        subgoal ← substitute(head(subgoals), Ψ)
        for Ψ′ in or(KB, subgoal, Ψ) do
            for Ψ″ in and(KB, tail(subgoals), Ψ′) do  yield Ψ″ ;
```

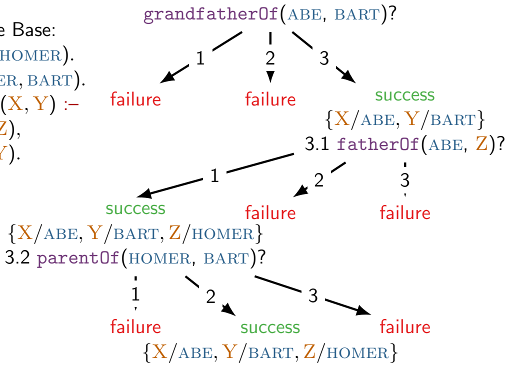
Unification

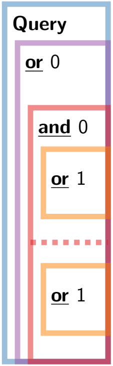
```
def unify(A, B, Ψ):
    if Ψ = failure then return failure;
    else if A is variable then
        return unifyvar(A, B, Ψ)
    else if B is variable then
        return unifyvar(B, A, Ψ)
    else if A = [a_1, ..., a_N] and B = [b_1, ..., b_N] are atoms then
        Ψ' ← unify([a_2, ..., a_N], [b_2, ..., b_N], Ψ)
        return unify(a_1, b_1, Ψ')
    else if A = B then return Ψ;
    else return failure;
```

Example

Example Knowledge Base:

1. fatherOf(ABE, HOMER).
2. parentOf(HOMER, BART).
3. grandfatherOf(X, Y) :–
   fatherOf(X, Z),
   parentOf(Z, Y).





Outline

Symbolic Representations

- Symbols (constants and predicates) do not share any information:
  grandpaOf $\neq$ grandfatherOf
- No notion of similarity:
  APPLE $\sim$ ORANGE, professorAt $\sim$ lecturerAt
- No generalization beyond what can be symbolically inferred:
  isFruit(APPLE), APPLE $\sim$ ORGANGE, isFruit(ORANGE)?
- But... leads to powerful inference mechanisms and proofs for predictions: fatherOf(ABE, HOMER). parentOf(HOMER, LISA).
  parentOf(HOMER, BART).
  grandfatherOf(X, Y) :– fatherOf(X, Z), parentOf(Z, Y).
  grandfatherOf(ABE, Q)? {Q/LISA}, {Q/BART}
- Fairly easy to debug and trivial to incorporate domain knowledge: just change/add rules
- Hard to work with language, vision and other modalities
  ''is a film based on the novel of the same name by''(X, Y)

Neural Representations

- Lower-dimensional fixed-length vector representations of symbols (predicates and constants):
  $\boldsymbol{v}_{\text{APPLE}}, \boldsymbol{v}_{\text{ORANGE}}, \boldsymbol{v}_{\texttt{fatherOf}}, \ldots \in \mathbb{R}^k$
- Can capture similarity and even semantic hierarchy of symbols: $\boldsymbol{v}_{\texttt{grandpaOf}} = \boldsymbol{v}_{\texttt{grandfatherOf}}$,
  $\boldsymbol{v}_{\text{APPLE}} \sim \boldsymbol{v}_{\text{ORANGE}}, \boldsymbol{v}_{\text{APPLE}} < \boldsymbol{v}_{\text{FRUIT}}$
- Can be trained from raw task data (e.g. facts)
- Can be compositional
  $\boldsymbol{v}_{\texttt{``is the father of''}} = \text{RNN}_\theta(\boldsymbol{v}_{\texttt{is}}, \boldsymbol{v}_{\texttt{the}}, \boldsymbol{v}_{\texttt{father}}, \boldsymbol{v}_{\texttt{of}})$
- But... need large amount of training data
- No direct way of incorporating prior knowledge
  $\boldsymbol{v}_{\texttt{grandfatherOf}}(\mathrm{X}, \mathrm{Y}) \text{ :- } \boldsymbol{v}_{\texttt{fatherOf}}(\mathrm{X}, \mathrm{Z}), \boldsymbol{v}_{\texttt{parentOf}}(\mathrm{Z}, \mathrm{Y}).$

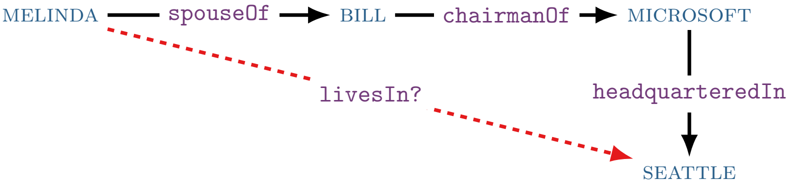
Related Work

- Fuzzy Logic (Zadeh, 1965)
- Probabilistic Logic Programming, e.g.,
  - IBAL (Pfeffer, 2001), BLOG (Milch et al., 2005), **Markov Logic Networks** (Richardson and Domingos, 2006), ProbLog (De Raedt et al., 2007) ...
- Inductive Logic Programming, e.g.,
  - Plotkin (1970), Shapiro (1991), Muggleton (1991), De Raedt (1999) ...
  - **Statistical Predicate Invention** (Kok and Domingos, 2007)
- Neural-symbolic Connectionism
  - Propositional rules: EBL-ANN (Shavlik and Towell, 1989), KBANN (Towell and Shavlik, 1994), C-LIP (Garcez and Zaverucha, 1999)
  - First-order inference (no training of symbol representations): **Unification Neural Networks** (Holldöbler, 1990; Komendantskaya 2011), SHRUTI (Shastri, 1992), **Neural Prolog** (Ding, 1995), CLIP++ (Franca et al. 2014), Lifted Relational Networks (Sourek et al. 2015)

Neural Link Prediction

Real world knowledge bases (like Freebase) are incomplete!

- placeOfBirth attribute is missing for 71% of people!
- Commonsense knowledge often not stated explicitly
- Weak logical relationships that can be used for inferring facts



Predict livesIn(MELINDA, SEATTLE) using local scoring function

$$f(\boldsymbol{v}_{\texttt{livesIn}}, \boldsymbol{v}_{\text{MELINDA}}, \boldsymbol{v}_{\text{SEATTLE}})$$

Das et al. (2016)

State-of-the-art Neural Link Prediction

$$f(\boldsymbol{v}_{\texttt{livesIn}}, \boldsymbol{v}_{\text{MELINDA}}, \boldsymbol{v}_{\text{SEATTLE}})$$

DistMult (Yang et al., 2014)

$\boldsymbol{v}_s, \boldsymbol{v}_i, \boldsymbol{v}_j \in \mathbb{R}^k$

$$f(\boldsymbol{v}_s, \boldsymbol{v}_i, \boldsymbol{v}_j) = \boldsymbol{v}_s^\top (\boldsymbol{v}_i \odot \boldsymbol{v}_j) = \sum_k \boldsymbol{v}_{sk} \boldsymbol{v}_{ik} \boldsymbol{v}_{jk}$$

ComplEx (Trouillon et al., 2016)

$\boldsymbol{v}_s, \boldsymbol{v}_i, \boldsymbol{v}_j \in \mathbb{C}^k$

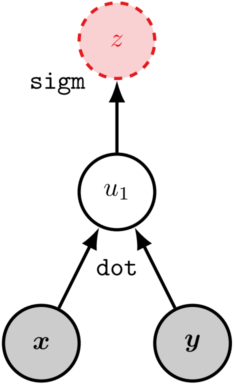
$$\begin{aligned} f(\boldsymbol{v}_s, \boldsymbol{v}_i, \boldsymbol{v}_j) = \; & \text{real}(\boldsymbol{v}_s)^\top (\text{real}(\boldsymbol{v}_i) \odot \text{real}(\boldsymbol{v}_j)) \\ & + \text{real}(\boldsymbol{v}_s)^\top (\text{imag}(\boldsymbol{v}_i) \odot \text{imag}(\boldsymbol{v}_j)) \\ & + \text{imag}(\boldsymbol{v}_s)^\top (\text{real}(\boldsymbol{v}_i) \odot \text{imag}(\boldsymbol{v}_j)) \\ & - \text{imag}(\boldsymbol{v}_s)^\top (\text{imag}(\boldsymbol{v}_i) \odot \text{real}(\boldsymbol{v}_j)) \end{aligned}$$

Training Loss

$$\mathfrak{L} = \sum_{r_s(e_i, e_j), y \;\in\; \mathcal{T}} -y \log\left(\sigma(f(\boldsymbol{v}_s, \boldsymbol{v}_i, \boldsymbol{v}_j))\right) - (1-y) \log\left(1 - \sigma(f(\boldsymbol{v}_s, \boldsymbol{v}_i, \boldsymbol{v}_j))\right)$$

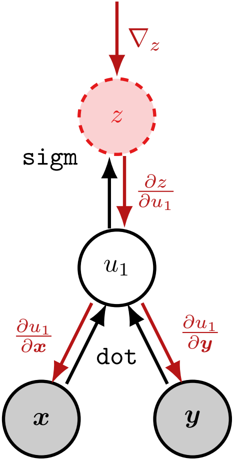
- Gradient-based optimization for learning $\boldsymbol{v}_s, \boldsymbol{v}_i, \boldsymbol{v}_j$ from data
- How do we calculate gradients $\nabla_{\boldsymbol{v}_s}\mathfrak{L}$, $\nabla_{\boldsymbol{v}_i}\mathfrak{L}$, $\nabla_{\boldsymbol{v}_j}\mathfrak{L}$?

Computation Graphs



- Example: $z = f(\boldsymbol{x}, \boldsymbol{y}) = \sigma(\boldsymbol{x}^\top \boldsymbol{y})$
- Nodes represent variables (inputs or parameters)
- Directed edges to a node correspond to a differentiable operation

Backpropagation



- Chain Rule of Calculus:
  Given function $\boldsymbol{z} = f(\boldsymbol{a}) = f(g(\boldsymbol{b}))$
  $\nabla_{\boldsymbol{a}} \boldsymbol{z} = \left(\frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}}\right)^\top \nabla_{\boldsymbol{b}} \boldsymbol{z}$
- Backpropagation is efficient recursive application of the Chain Rule
- Gradient of $z = \sigma(\boldsymbol{x}^\top \boldsymbol{y})$ w.r.t. $\boldsymbol{x}$
  $\nabla_{\boldsymbol{x}} z = \frac{\partial z}{\partial \boldsymbol{x}} = \frac{\partial z}{\partial u_1} \frac{\partial u_1}{\partial \boldsymbol{x}} = \sigma(u_1)(1 - \sigma(u_1))\boldsymbol{y}$
- Given upstream supervision on z, we can learn $\boldsymbol{x}$ and $\boldsymbol{y}$!

Deep Learning = "Large" differentiable computation graphs

Outline

Aims

"We are attempting to replace symbols by vectors so we can replace logic by algebra." — *Yann LeCun*

- End-to-end-differentiable proving
- Calculate gradient of proof success w.r.t. symbol representations
- Train symbol representations from facts and rules in a knowledge base via gradient descent
- Use similarity of symbol representations during proofs
- Induce rules of predefined structure via gradient descent

Neural Knowledge Base

Symbolic Representation

1 $\texttt{fatherOf}(\text{ABE}, \text{HOMER})$.
2 $\texttt{parentOf}(\text{HOMER}, \text{LISA})$.
3 $\texttt{parentOf}(\text{HOMER}, \text{BART})$.
4 $\texttt{grandpaOf}(\text{ABE}, \text{LISA})$.
5 $\texttt{grandfatherOf}(\text{ABE}, \text{MAGGIE})$.
6 $\texttt{grandfatherOf}(X_1, Y_1)$:–
   $\texttt{fatherOf}(X_1, Z_1)$,
   $\texttt{parentOf}(Z_1, Y_1)$.
7 $\texttt{grandparentOf}(X_2, Y_2)$:–
   $\texttt{grandfatherOf}(X_2, Y_2)$.

Neural-Symbolic Representation

1 $\boldsymbol{v}_{\texttt{fatherOf}}(\boldsymbol{v}_{\text{ABE}}, \boldsymbol{v}_{\text{HOMER}})$.
2 $\boldsymbol{v}_{\texttt{parentOf}}(\boldsymbol{v}_{\text{HOMER}}, \boldsymbol{v}_{\text{LISA}})$.
3 $\boldsymbol{v}_{\texttt{parentOf}}(\boldsymbol{v}_{\text{HOMER}}, \boldsymbol{v}_{\text{BART}})$.
4 $\boldsymbol{v}_{\texttt{grandpaOf}}(\boldsymbol{v}_{\text{ABE}}, \boldsymbol{v}_{\text{LISA}})$.
5 $\boldsymbol{v}_{\texttt{grandfatherOf}}(\boldsymbol{v}_{\text{ABE}}, \boldsymbol{v}_{\text{MAGGIE}})$.
6 $\boldsymbol{v}_{\texttt{grandfatherOf}}(X_1, Y_1)$:–
   $\boldsymbol{v}_{\texttt{fatherOf}}(X_1, Z_1)$,
   $\boldsymbol{v}_{\texttt{parentOf}}(Z_1, Y_1)$.
7 $\boldsymbol{v}_{\texttt{grandparentOf}}(X_2, Y_2)$:–
   $\boldsymbol{v}_{\texttt{grandfatherOf}}(X_2, Y_2)$.

Neural Unification

Soft-matching: $\tau_{A,B} = e^{-\|\mathbf{v}_A - \mathbf{v}_B\|_2} \in [0, 1]$

```
def unify(A, B, Ψ, τ):
    if Ψ = failure then return failure, 0;
    else if A is variable then
        return unifyvar(A, B, Ψ), τ
    else if B is variable then
        return unifyvar(B, A, Ψ), τ
    else if A = [a_1, ..., a_N] and B = [b_1, ..., b_N] are atoms then
        Ψ', τ' ← unify([a_2, ..., a_N], [b_2, ..., b_N], Ψ, τ)
        return unify(a_1, b_1, Ψ', τ')
    else if A and B are symbol representations then return Ψ, min(τ, τ_{A,B});
    else return failure, 0;
```

Example: unify $\mathbf{v}_{\texttt{grandfatherOf}}(\mathrm{X}, \mathbf{v}_{\mathrm{BART}})$ with $\mathbf{v}_{\texttt{grandpaOf}}(\mathbf{v}_{\mathrm{ABE}}, \mathbf{v}_{\mathrm{BART}})$

$$\Psi = \{\mathrm{X}/\mathbf{v}_{\mathrm{ABE}}\}, \quad \tau = \min(e^{-\|\mathbf{v}_{\texttt{grandfatherOf}} - \mathbf{v}_{\texttt{grandpaOf}}\|_2}, e^{-\|\mathbf{v}_{\mathrm{BART}} - \mathbf{v}_{\mathrm{BART}}\|_2})$$

Compiling a Computation Graph using Backward Chaining

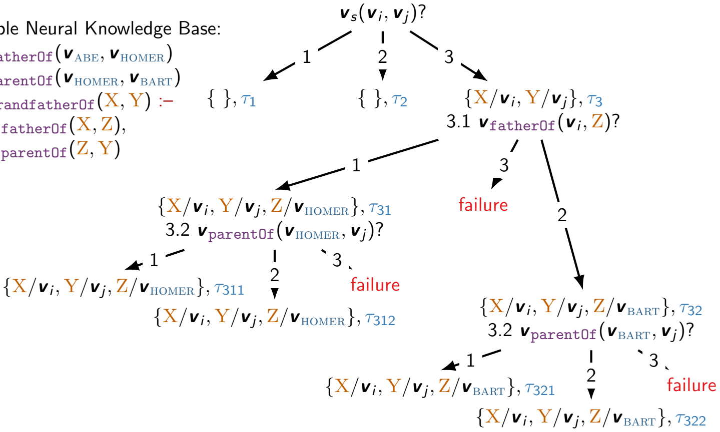
```
def or(KB, goal, Ψ, τ, D):
    for rule head :– body in KB do
        Ψ′, τ′ ← unify(head, goal, Ψ, τ)
        if Ψ′ ≠ failure then
            for Ψ″, τ″ in and(KB, body, Ψ′, τ′, D) do
                yield Ψ″, τ″

def and(KB, subgoals, Ψ, τ, D):
    if subgoals is empty then return Ψ, τ;
    else if D = 0 then return failure;
    else
        subgoal ← substitute(head(subgoals), Ψ)
        for Ψ′, τ′ in or(KB, subgoal, Ψ, τ, D − 1) do
            for Ψ″, τ″ in and(KB, tail(subgoals), Ψ′, τ′, D) do  yield
            Ψ″, τ″ ;
```

Example

Example Neural Knowledge Base:

1. $\boldsymbol{v}_{\texttt{fatherOf}}(\boldsymbol{v}_{\text{ABE}}, \boldsymbol{v}_{\text{HOMER}})$
2. $\boldsymbol{v}_{\texttt{parentOf}}(\boldsymbol{v}_{\text{HOMER}}, \boldsymbol{v}_{\text{BART}})$
3. $\boldsymbol{v}_{\texttt{grandfatherOf}}(\mathrm{X}, \mathrm{Y})$:–
   $\boldsymbol{v}_{\texttt{fatherOf}}(\mathrm{X}, \mathrm{Z})$,
   $\boldsymbol{v}_{\texttt{parentOf}}(\mathrm{Z}, \mathrm{Y})$



Training

Proof Aggregation

$$\Psi, \tau = \underline{\mathsf{or}}(KB, \mathrm{Q}, \{\,\}, 1, D)$$
$$\tau_{\mathrm{Q}} = \max \tau$$

Supervision Signal

$$y_{\mathrm{Q}} = \begin{cases} 1.0 & \text{if } \mathrm{Q} \in \mathcal{F} \\ 0.0 & \text{otherwise} \end{cases}$$

Masking Unification for Training Facts

$$\tilde{\tau}_{\mathrm{Q},B} = \begin{cases} 0.0 & \text{if } \mathrm{Q} \in \mathcal{F} \text{ and } \mathrm{Q} = B \\ \tau_{\mathrm{Q},B} & \text{otherwise} \end{cases}$$

Loss

$$\mathfrak{L} = \sum_{\mathrm{Q} \in \mathcal{T}} -y_{\mathrm{Q}} \log(\tau_{\mathrm{Q}}) - (1 - y_{\mathrm{Q}}) \log(1 - \tau_{\mathrm{Q}})$$

Neural Inductive Logic Programming

1 $\boldsymbol{v}_{\texttt{fatherOf}}(\boldsymbol{v}_{\text{ABE}}, \boldsymbol{v}_{\text{HOMER}}).$

2 $\boldsymbol{v}_{\texttt{parentOf}}(\boldsymbol{v}_{\text{HOMER}}, \boldsymbol{v}_{\text{LISA}}).$

3 $\boldsymbol{v}_{\texttt{parentOf}}(\boldsymbol{v}_{\text{HOMER}}, \boldsymbol{v}_{\text{BART}}).$

4 $\boldsymbol{v}_{\texttt{grandpaOf}}(\boldsymbol{v}_{\text{ABE}}, \boldsymbol{v}_{\text{LISA}}).$

5 $\boldsymbol{v}_{\texttt{grandfatherOf}}(\boldsymbol{v}_{\text{ABE}}, \boldsymbol{v}_{\text{MAGGIE}}).$

6 $\theta_1(\text{X}_1, \text{Y}_1) :-$
$\quad \theta_2(\text{X}_1, \text{Z}_1),$
$\quad \theta_3(\text{Z}_1, \text{Y}_1).$

7 $\theta_4(\text{X}_2, \text{Y}_2) :-$
$\quad \theta_5(\text{X}_2, \text{Y}_2).$

Outline

Batch Proving: Utilizing GPUs

Let $\boldsymbol{A} \in \mathbb{R}^{N \times k}$ be a matrix of N symbol representations that are to be unified with M other symbol representations $\boldsymbol{B} \in \mathbb{R}^{M \times k}$

$$\tau_{\boldsymbol{A},\boldsymbol{B}} = e^{-\sqrt{\boldsymbol{A}^{sq} + \boldsymbol{B}^{sq} - 2\boldsymbol{A}\boldsymbol{B}^\top} + \epsilon} \qquad \in \mathbb{R}^{N \times M}$$

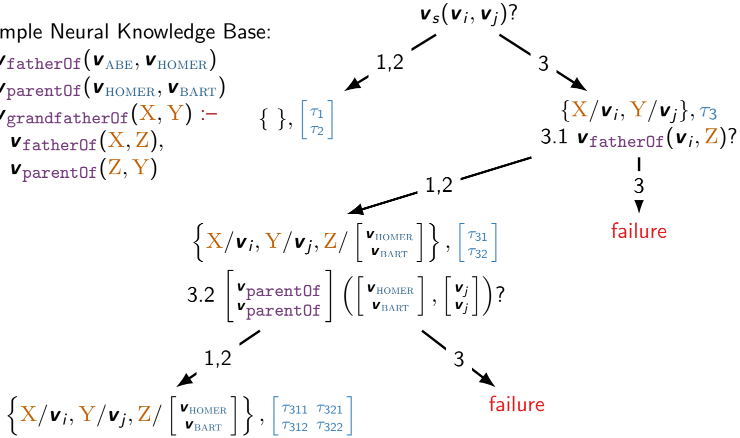
$$\boldsymbol{A}^{sq} = \begin{bmatrix} \sum_{i=1}^{k} \boldsymbol{A}_{1i}^2 \\ \vdots \\ \sum_{i=1}^{k} \boldsymbol{A}_{Ni}^2 \end{bmatrix} \mathbf{1}_M^\top \qquad \in \mathbb{R}^{N \times M}$$

$$\boldsymbol{B}^{sq} = \mathbf{1}_N \begin{bmatrix} \sum_{i=1}^{k} \boldsymbol{B}_{1i}^2 \\ \vdots \\ \sum_{i=1}^{k} \boldsymbol{B}_{Mi}^2 \end{bmatrix}^\top \qquad \in \mathbb{R}^{N \times M}$$

Batch Proving Example

Example Neural Knowledge Base:

1. $\boldsymbol{v}_{\texttt{fatherOf}}(\boldsymbol{v}_{\text{ABE}}, \boldsymbol{v}_{\text{HOMER}})$
2. $\boldsymbol{v}_{\texttt{parentOf}}(\boldsymbol{v}_{\text{HOMER}}, \boldsymbol{v}_{\text{BART}})$
3. $\boldsymbol{v}_{\texttt{grandfatherOf}}(\mathrm{X}, \mathrm{Y})$:–
   $\boldsymbol{v}_{\texttt{fatherOf}}(\mathrm{X}, \mathrm{Z})$,
   $\boldsymbol{v}_{\texttt{parentOf}}(\mathrm{Z}, \mathrm{Y})$



Gradient Approximation with Kmax Proofs

Example Neural Knowledge Base:

1. $\boldsymbol{v}_{\texttt{fatherOf}}(\boldsymbol{v}_{\text{ABE}}, \boldsymbol{v}_{\text{HOMER}})$
2. $\boldsymbol{v}_{\texttt{parentOf}}(\boldsymbol{v}_{\text{HOMER}}, \boldsymbol{v}_{\text{BART}})$
3. $\boldsymbol{v}_{\texttt{grandfatherOf}}(\mathrm{X}, \mathrm{Y})$:–
   $\boldsymbol{v}_{\texttt{fatherOf}}(\mathrm{X}, \mathrm{Z})$,
   $\boldsymbol{v}_{\texttt{parentOf}}(\mathrm{Z}, \mathrm{Y})$

$\boldsymbol{v}_s(\boldsymbol{v}_i, \boldsymbol{v}_j)$?

1,2

$\{\ \}, \begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix}$

3

$\{\mathrm{X}/\boldsymbol{v}_i, \mathrm{Y}/\boldsymbol{v}_j\}, \tau_3$

3.1 $\boldsymbol{v}_{\texttt{fatherOf}}(\boldsymbol{v}_i, \mathrm{Z})$?

1,2

$\{\mathrm{X}/\boldsymbol{v}_i, \mathrm{Y}/\boldsymbol{v}_j, \mathrm{Z}/[\boldsymbol{v}_K]\}, [\tau_{3K}]$

3.2 $[\boldsymbol{v}_{\texttt{parentOf}}]([\boldsymbol{v}_K], [\boldsymbol{v}_j])$?

1,2

$\{\mathrm{X}/\boldsymbol{v}_i, \mathrm{Y}/\boldsymbol{v}_j, \mathrm{Z}/[\boldsymbol{v}_K]\}, \begin{bmatrix}\tau_{3K1}\\ \tau_{3K2}\end{bmatrix}$

3

failure

3

failure

Regularization by Neural Link Predictor

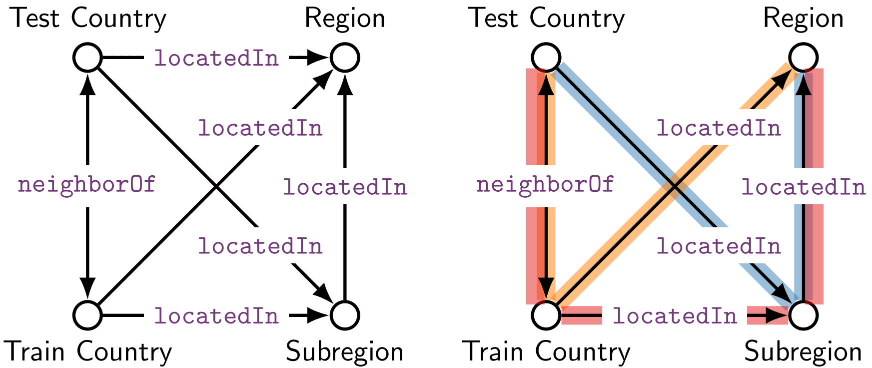
- Train jointly with neural link prediction method
- Share symbol representations
- Neural link prediction model quickly learns similarities between symbols
- Let p_{Q} be score by neural link prediction model (DistMult or ComplEx), and τ_{Q} be the proof success
- Multi-task training loss:

$$\mathfrak{L} = \sum_{\mathrm{Q} \in \mathcal{T}} -y_{\mathrm{Q}}(\log(\tau_{\mathrm{Q}}) + \log(p_{\mathrm{Q}})) - (1 - y_{\mathrm{Q}})(\log(1 - \tau_{\mathrm{Q}}) + \log(1 - p_{\mathrm{Q}}))$$

Outline

Experiments

Countries Knowledge Base (Bouchard et al., 2015)



Models

NTP: prover is trained alone

DistMult: neural link prediction model by Yang et al. (2014)

NTP DistMult: jointly training prover and DistMult, and use maximum prediction at test time

NTP DistMult λ: only prover is used at test time; DistMult acts as a regularizer

ComplEx: neural link prediction model by Trouillon et al. (2016)

NTP ComplEx: jointly training prover and ComplEx, and use the maximum prediction at test time

NTP ComplEx λ: only prover is used at test time; ComplEx acts as a regularizer

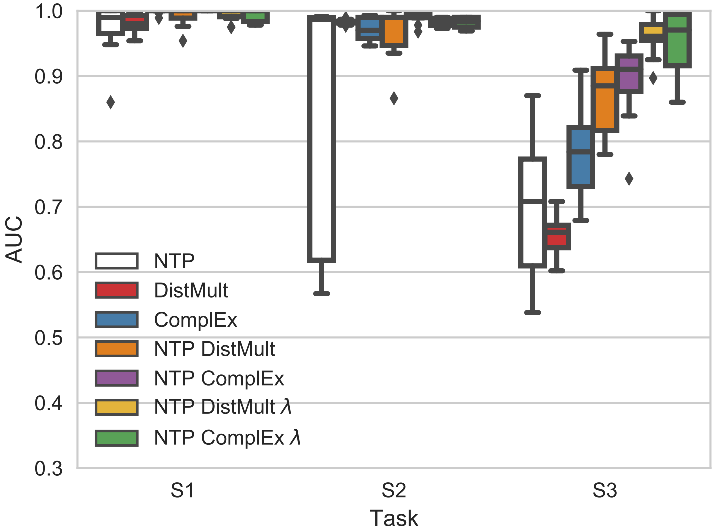
Rule Templates

$$
\begin{array}{ll}
\textbf{S1} & \theta_1(X, Y) :- \theta_2(Y, Z). \\
 & \theta_1(X, Y) :- \theta_2(X, Z), \theta_2(Z, Y). \\
\textbf{S2} & \theta_1(X, Y) :- \theta_2(X, Z), \theta_3(Z, Y). \\
\textbf{S3} & \theta_1(X, Y) :- \theta_2(X, Z), \theta_3(Z, W), \theta_4(W, Y).
\end{array}
$$

Results

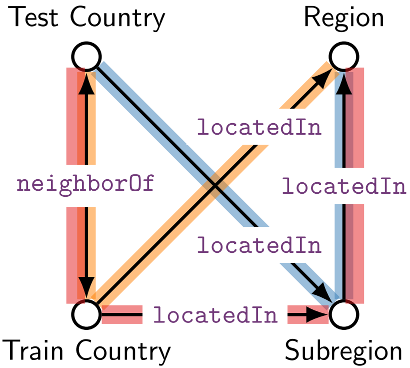
Model	**S1**	**S2**	**S3**
Random	32.3	32.3	32.3
Frequency	32.3	32.3	30.8
ER-MLP (Dong et al., 2014)	96.0	74.5	65.0
Rescal (Nickel et al., 2012)	99.7	74.5	65.0
HolE (Nickel et al., 2015)	99.7	77.2	69.7
TARE (Wang et al., 2017)	99.4	90.6	89.0
NTP	97.3	83.7	70.0
DistMult (Yang et al., 2014)	98.1	98.3	65.5
NTP DistMult	99.2	96.7	87.0
NTP DistMult λ	99.4	98.3	**95.9**
ComplEx (Trouillon et al., 2016)	99.9	97.1	78.6
NTP ComplEx	**100.0**	**98.9**	89.1
NTP ComplEx λ	99.3	98.2	95.1

Results


1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
AUC
S1
S2
S3
Task
NTP
DistMult
ComplEx
NTP DistMult
NTP ComplEx
NTP DistMult λ
NTP ComplEx λ

Induced Logic Programs

Task	Confidence	Rule
S1	0.999	neighborOf(X, Y) :– neighborOf(Y, X).
	0.767	locatedIn(X, Y) :– locatedIn(X, Z), locatedIn(Z, Y).
S2	0.998	neighborOf(X, Y) :– neighborOf(Y, X).
	0.995	locatedIn(X, Y) :– locatedIn(X, Z), locatedIn(Z, Y).
	0.705	locatedIn(X, Y) :– neighborOf(X, Z), locatedIn(Z, Y).
S3	0.891	neighborOf(X, Y) :– neighborOf(Y, X).
	0.750	locatedIn(X, Y) :– neighborOf(X, Z), neighborOf(Z, W), locatedIn(W, Y).



Summary

- Prolog's backward chaining can be used as a recipe for recursively constructing a neural network
- Proof success differentiable w.r.t. symbol representations
- Can learn vector representations of symbols and rules of predefined structure
- Various optimizations: batch proving, gradient approximation
- Outperforms neural link prediction models on a medium-sized knowledge base
- Induces interpretable rules

Thank you!

http://rockt.github.com
tim [dot] rocktaeschel [at] gmail [dot] com
Twitter: @_rockt

References

Guillaume Bouchard, Sameer Singh, and Theo Trouillon. 2015. On approximate reasoning capabilities of low-rank vector spaces. *AAAI Spring Syposium on Knowledge Representation and Reasoning (KRR): Integrating Symbolic and Neural Approaches*.

Rajarshi Das, Arvind Neelakantan, David Belanger, and Andrew McCallum. 2016. Chains of reasoning over entities, relations, and text using recurrent neural networks. *arXiv preprint arXiv:1607.01426*.

Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. 2014. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610. ACM.

Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. 2015. Holographic embeddings of knowledge graphs. *arXiv preprint arXiv:1510.04935*.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2012. Factorizing yago: scalable machine learning for linked data. In *Proc. of International Conference on World Wide Web (WWW)*, pages 271–280.

Tim Rocktäschel. 2017. *Combining Representation Learning with Logic for Language Processing*. Ph.D. thesis, University College London, Gower Street, London WC1E 6BT, United Kingdom.

Tim Rocktäschel and Sebastian Riedel. 2016. Learning knowledge base inference with neural theorem provers. In *NAACL Workshop on Automated Knowledge Base Construction (AKBC)*.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction.

Mengya Wang, Hankui Zhuo, and Huiling Zhu. 2017. Embedding knowledge graphs based on transitivity and antisymmetry of rules. *arXiv preprint arXiv:1702.07543*.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2014. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*.