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

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

First-order Logic

“Every father of a parent is a grandfather.”

\[
\text{grandfatherOf}(X, Y) :\neg \\
\text{fatherOf}(X, Z), \\
\text{parentOf}(Z, Y).
\]
Outline

1.Reasoning with Symbols
   - Knowledge Bases
   - Prolog: Backward Chaining

2. Reasoning with Neural Representations
   - Symbolic vs. Neural Representations
   - Neural Link Prediction
   - Computation Graphs

3. Deep Prolog: Neural Backward Chaining

4. Optimizations
   - Batch Proving
   - Gradient Approximation
   - Regularization by Neural Link Predictor

5. Experiments

6. Summary
Outline

1 Reasoning with Symbols
   ■ Knowledge Bases
   ■ Prolog: Backward Chaining

2 Reasoning with Neural Representations
   ■ Symbolic vs. Neural Representations
   ■ Neural Link Prediction
   ■ Computation Graphs

3 Deep Prolog: Neural Backward Chaining

4 Optimizations
   ■ Batch Proving
   ■ Gradient Approximation
   ■ Regularization by Neural Link Predictor

5 Experiments

6 Summary
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 \texttt{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).}
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

```python
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 = \[a_1, \ldots, a_N\] and B = \[b_1, \ldots, b_N\] are atoms then
        ψ′ ← unify([a_2, \ldots, a_N], [b_2, \ldots, b_N], ψ)
        return unify(a_1, b_1, ψ′)
    else if A = B then return ψ;
    else return failure;
```
Example Knowledge Base:
1. fatherOf(ABE, HOMER).
2. parentOf(HOMER, BART).
3. grandfatherOf(X, Y) :-
   fatherOf(X, Z),
   parentOf(Z, Y).

Example Knowledge Base:
1. fatherOf(ABE, HOMER).
2. parentOf(HOMER, BART).
3. grandfatherOf(X, Y) :-
   fatherOf(X, Z),
   parentOf(Z, Y).
Outline

1. Reasoning with Symbols
   - Knowledge Bases
   - Prolog: Backward Chaining

2. Reasoning with Neural Representations
   - Symbolic vs. Neural Representations
   - Neural Link Prediction
   - Computation Graphs

3. Deep Prolog: Neural Backward Chaining

4. Optimizations
   - Batch Proving
   - Gradient Approximation
   - Regularization by Neural Link Predictor

5. Experiments

6. Summary
Symbolic Representations

- Symbols (constants and predicates) do not share any information:
  \( \text{grandpaOf} \neq \text{grandfatherOf} \)

- No notion of similarity:
  \( \text{APPLE} \sim \text{ORANGE}, \text{professorAt} \sim \text{lecturerAt} \)

- No generalization beyond what can be symbolically inferred:
  \( \text{isFruit(APPLE)}, \text{APPLE} \sim \text{ORGANGE}, \text{isFruit(ORANGE)}? \)

- But... leads to powerful inference mechanisms and proofs for predictions:
  \( \text{fatherOf(ABE, HOMER)}, \text{parentOf(HOMER, LISA)}, \text{parentOf(HOMER, BART)} \)
  \( \text{grandfatherOf(X, Y)} :– \text{fatherOf(X, Z)}, \text{parentOf(Z, Y)} \)
  \( \text{grandfatherOf(ABE, Q)}? \quad \{Q/LISA\}, \{Q/BART\} \)

- 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):
  \[ \mathbf{v}_{\text{APPLE}}, \mathbf{v}_{\text{ORANGE}}, \mathbf{v}_{\text{fatherOf}}, \ldots \in \mathbb{R}^k \]

- Can capture similarity and even semantic hierarchy of symbols:
  \[ \mathbf{v}_{\text{grandpaOf}} = \mathbf{v}_{\text{grandfatherOf}}, \]
  \[ \mathbf{v}_{\text{APPLE}} \sim \mathbf{v}_{\text{ORANGE}}, \mathbf{v}_{\text{APPLE}} < \mathbf{v}_{\text{FRUIT}} \]

- Can be trained from raw task data (e.g. facts)

- Can be compositional
  \[ \mathbf{v}^{\text{‘is the father of’}} = \text{RNN}_\theta(\mathbf{v}_{\text{is}}, \mathbf{v}_{\text{the}}, \mathbf{v}_{\text{father}}, \mathbf{v}_{\text{of}}) \]

- But... need large amount of training data

- No direct way of incorporating prior knowledge
  \[ \mathbf{v}_{\text{grandfatherOf}}(X, Y) \leftarrow \mathbf{v}_{\text{fatherOf}}(X, Z), \mathbf{v}_{\text{parentOf}}(Z, Y). \]
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.,
- 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 \( \text{livesIn}(\text{MELINDA}, \text{SEATTLE}) \) using local scoring function

\[
f(\mathbf{v}_{\text{livesIn}}, \mathbf{v}_{\text{MELINDA}}, \mathbf{v}_{\text{SEATTLE}})
\]
State-of-the-art Neural Link Prediction

\[ f(\mathbf{v}_{\text{livesIn}}, \mathbf{v}_{\text{MELINDA}}, \mathbf{v}_{\text{SEATTLE}}) \]

**DistMult** (Yang et al., 2014)
\[ \mathbf{v}_s, \mathbf{v}_i, \mathbf{v}_j \in \mathbb{R}^k \]

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

**ComplEx** (Trouillon et al., 2016)
\[ \mathbf{v}_s, \mathbf{v}_i, \mathbf{v}_j \in \mathbb{C}^k \]

\[
f(\mathbf{v}_s, \mathbf{v}_i, \mathbf{v}_j) = \text{real}(\mathbf{v}_s)^\top (\text{real}(\mathbf{v}_i) \odot \text{real}(\mathbf{v}_j)) + \text{real}(\mathbf{v}_s)^\top (\text{imag}(\mathbf{v}_i) \odot \text{imag}(\mathbf{v}_j)) + \text{imag}(\mathbf{v}_s)^\top (\text{real}(\mathbf{v}_i) \odot \text{imag}(\mathbf{v}_j)) - \text{imag}(\mathbf{v}_s)^\top (\text{imag}(\mathbf{v}_i) \odot \text{real}(\mathbf{v}_j))
\]

**Training Loss**
\[
\mathcal{L} = \sum_{r_s(e_i, e_j), y \in \mathcal{T}} -y \log (\sigma(f(\mathbf{v}_s, \mathbf{v}_i, \mathbf{v}_j))) - (1 - y) \log (1 - \sigma(f(\mathbf{v}_s, \mathbf{v}_i, \mathbf{v}_j)))
\]

- Gradient-based optimization for learning \( \mathbf{v}_s, \mathbf{v}_i, \mathbf{v}_j \) from data
- How do we calculate gradients \( \nabla_{\mathbf{v}_s} \mathcal{L}, \nabla_{\mathbf{v}_i} \mathcal{L}, \nabla_{\mathbf{v}_j} \mathcal{L} \)?
Computation Graphs

Example: $z = f(x, y) = \sigma(x^\top y)$

- Nodes represent variables (inputs or parameters)
- Directed edges to a node correspond to a differentiable operation
Backpropagation

- Chain Rule of Calculus:
  Given function $z = f(a) = f(g(b))$
  $$\nabla_a z = \left( \frac{\partial b}{\partial a} \right)^\top \nabla_b z$$

- Backpropagation is efficient recursive application of the Chain Rule

- Gradient of $z = \sigma(x^\top y)$ w.r.t. $x$
  $$\nabla_x z = \frac{\partial z}{\partial x} = \frac{\partial z}{\partial u_1} \frac{\partial u_1}{\partial x} = \sigma(u_1)(1 - \sigma(u_1))y$$

- Given upstream supervision on $z$, we can learn $x$ and $y$!

Deep Learning = “Large” differentiable computation graphs
Outline

1. Reasoning with Symbols
   - Knowledge Bases
   - Prolog: Backward Chaining

2. Reasoning with Neural Representations
   - Symbolic vs. Neural Representations
   - Neural Link Prediction
   - Computation Graphs

3. Deep Prolog: Neural Backward Chaining

4. Optimizations
   - Batch Proving
   - Gradient Approximation
   - Regularization by Neural Link Predictor

5. Experiments

6. Summary
“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. fatherOf(ABE, HOMER).
2. parentOf(HOMER, LISA).
3. parentOf(HOMER, BART).
4. grandpaOf(ABE, LISA).
5. grandfatherOf(ABE, MAGGIE).
6. grandfatherOf(X₁, Y₁) :-
   fatherOf(X₁, Z₁),
   parentOf(Z₁, Y₁).
7. grandparentOf(X₂, Y₂) :-
   grandfatherOf(X₂, Y₂).

Neural-Symbolic Representation

1. v_fatherOf(v_ABE, v_HOMER).
2. v_parentOf(v_HOMER, v_LISA).
3. v_parentOf(v_HOMER, v_BART).
4. v_grandpaOf(v_ABE, v_LISA).
5. v_grandfatherOf(v_ABE, v_MAGGIE).
6. v_grandfatherOf(v_X₁, v_Y₁) :-
   v_fatherOf(v_X₁, v_Z₁),
   v_parentOf(v_Z₁, v_Y₁).
7. v_grandparentOf(v_X₂, v_Y₂) :-
   v_grandfatherOf(v_X₂, v_Y₂).
Neural Unification

Soft-matching: \( \tau_{A,B} = e^{-\|v_A - v_B\|^2} \in [0, 1] \)

```python
def unify(A, B, \Psi, \tau):
    if \( \Psi = \text{failure} \) then return failure, 0;
    else if A is variable then
        return unifyvar(A, B, \Psi), \tau
    else if B is variable then
        return unifyvar(B, A, \Psi), \tau
    else if A = [a_1, \ldots, a_N] and B = [b_1, \ldots, b_N] are atoms then
        \( \Psi', \tau' \leftarrow \text{unify}([a_2, \ldots, a_N], [b_2, \ldots, b_N], \Psi, \tau) \)
        return unify(a_1, b_1, \Psi', \tau')
    else if A and B are symbol representations then return \( \Psi, \min(\tau, \tau_{A,B}) \);
    else return failure, 0;
```

Example: unify \( v_{\text{grandfatherOf}}(X, v_{\text{BART}}) \) with \( v_{\text{grandpaOf}}(v_{\text{ABE}}, v_{\text{BART}}) \)

\( \Psi = \{X/v_{\text{ABE}}\}, \quad \tau = \min(e^{-\|v_{\text{grandfatherOf}} - v_{\text{grandpaOf}}\|^2}, e^{-\|v_{\text{BART}} - v_{\text{BART}}\|^2}) \)
Compiling a **Computation Graph** using Backward Chaining

```python
    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'' ;
```

Tim Rocktäschel  Deep Prolog: End-to-end Differentiable Proving in Knowledge Bases  21/37
Example Neural Knowledge Base:

1. \( v_{fatherOf}(v_{ABE}, v_{HOMER}) \)
2. \( v_{parentOf}(v_{HOMER}, v_{BART}) \)
3. \( v_{grandfatherOf}(X, Y) \) :- \( \{ \}, \tau_1 \)
   \( v_{fatherOf}(X, Z), \)
   \( v_{parentOf}(Z, Y) \)
   \( \{ X/v_i, Y/v_j, Z/v_homer \}, \tau_3 \)
   \( 3.1 v_{fatherOf}(v_i, Z)? \)
   \( \{ X/v_i, Y/v_j \}, \tau_2 \)
   \( 3.2 v_{parentOf}(v_{HOMER}, v_j)? \)
   \( \{ X/v_i, Y/v_j, Z/v_homer \}, \tau_31 \)
   \( \{ X/v_i, Y/v_j, Z/v_homer \}, \tau_311 \)
   \( failure \)
   \( \{ X/v_i, Y/v_j, Z/v_homer \}, \tau_312 \)
   \( failure \)
   \( \{ X/v_i, Y/v_j, Z/v_homer \}, \tau_31 \)

3. \( v_{fatherOf}(v_i, Z)? \)
   \( \{ X/v_i, Y/v_j \}, \tau_2 \)
   \( 3.2 v_{parentOf}(v_{HOMER}, v_j)? \)
   \( \{ X/v_i, Y/v_j, Z/v_homer \}, \tau_32 \)
   \( \{ X/v_i, Y/v_j, Z/v_homer \}, \tau_321 \)
   \( failure \)
   \( \{ X/v_i, Y/v_j, Z/v_homer \}, \tau_322 \)
Training

Proof Aggregation

$$\Psi, \tau = \text{or}(KB, Q, \{\}, 1, D)$$

$$\tau_Q = \max \tau$$

Supervision Signal

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

Masking Unification for Training Facts

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

Loss

$$\mathcal{L} = \sum_{Q \in \mathcal{T}} -y_Q \log(\tau_Q) - (1 - y_Q) \log(1 - \tau_Q)$$
Neural Inductive Logic Programming

1  \( \nu \text{fatherOf}(\nu_{\text{ABE}}, \nu_{\text{HOMER}}). \)
2  \( \nu \text{parentOf}(\nu_{\text{HOMER}}, \nu_{\text{LISA}}). \)
3  \( \nu \text{parentOf}(\nu_{\text{HOMER}}, \nu_{\text{BART}}). \)
4  \( \nu \text{grandpaOf}(\nu_{\text{ABE}}, \nu_{\text{LISA}}). \)
5  \( \nu \text{grandfatherOf}(\nu_{\text{ABE}}, \nu_{\text{MAGGIE}}). \)

6  \( \theta_1(X_1, Y_1) :– \)
    \( \theta_2(X_1, Z_1), \)
    \( \theta_3(Z_1, Y_1). \)

7  \( \theta_4(X_2, Y_2) :– \)
    \( \theta_5(X_2, Y_2). \)
Outline

1. Reasoning with Symbols
   - Knowledge Bases
   - Prolog: Backward Chaining

2. Reasoning with Neural Representations
   - Symbolic vs. Neural Representations
   - Neural Link Prediction
   - Computation Graphs

3. Deep Prolog: Neural Backward Chaining

4. Optimizations
   - Batch Proving
   - Gradient Approximation
   - Regularization by Neural Link Predictor

5. Experiments

6. Summary
Batch Proving: Utilizing GPUs

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

$$
\tau_{A,B} = e^{-\sqrt{A^{sq} + B^{sq} - 2AB^\top} + \epsilon}
$$

$$
A^{sq} = \begin{bmatrix}
\sum_{i=1}^{k} A_{1i}^2 \\
\vdots \\
\sum_{i=1}^{k} A_{Ni}^2
\end{bmatrix} 1_M^\top 
$$

$$
B^{sq} = 1_N \begin{bmatrix}
\sum_{i=1}^{k} B_{1i}^2 \\
\vdots \\
\sum_{i=1}^{k} B_{Mi}^2
\end{bmatrix}^\top
$$
Batch Proving Example

Example Neural Knowledge Base:

1. \( \text{fatherOf}(v_{\text{ABE}}, v_{\text{HOMER}}) \)
2. \( \text{parentOf}(v_{\text{HOMER}}, v_{\text{BART}}) \)
3. \( \text{grandfatherOf}(X, Y) : \neg \) \( \text{fatherOf}(X, Z), \) \( \text{parentOf}(Z, Y) \)

\[ \begin{align*}
&\{ X / v_i, Y / v_j, Z / \begin{bmatrix} v_{\text{HOMER}} \\ v_{\text{BART}} \end{bmatrix} \}, \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} \\
&\{ X / v_i, Y / v_j \}, \begin{bmatrix} \tau_3 \end{bmatrix} \\
&\text{failure} \\
&\text{failure}
\end{align*} \]
Gradient Approximation with $K_{\text{max}}$ Proofs

Example Neural Knowledge Base:
1. $v_{\text{fatherOf}}(v_{\text{ABE}}, v_{\text{HOMER}})$
2. $v_{\text{parentOf}}(v_{\text{HOMER}}, v_{\text{BART}})$
3. $v_{\text{grandfatherOf}}(X, Y) :=$
   - $v_{\text{fatherOf}}(X, Z)$,
   - $v_{\text{parentOf}}(Z, Y)$

$\Gamma_{s}(v_{i}, v_{j})$?

1.2 $\{X/v_{i}, Y/v_{j}, Z/[v_{K}]\}, [\tau_{3K}]$
3 $\{X/v_{i}, Y/v_{j}\}, \tau_{3}$

3.1 $v_{\text{fatherOf}}(v_{i}, Z) ?$
   - $1,2$
   - $3$

$\Gamma_{s}(v_{i}, v_{j})$?

1.2 $\{X/v_{i}, Y/v_{j}, Z/[v_{K}]\}, [\tau_{3K}]$
3 $\{X/v_{i}, Y/v_{j}\}, \tau_{3}$

failure

$\Gamma_{s}(v_{i}, v_{j})$?

1.2 $\{X/v_{i}, Y/v_{j}, Z/[v_{K}]\}, [\tau_{3K}]$
3 $\{X/v_{i}, Y/v_{j}\}, \tau_{3}$

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 $\tau_Q$ be the proof success
- Multi-task training loss:

$$\mathcal{L} = \sum_{Q \in \mathcal{T}} -y_Q(\log(\tau_Q) + \log(p_Q)) - (1 - y_Q)(\log(1 - \tau_Q) + \log(1 - p_Q))$$
Outline

1. Reasoning with Symbols
   - Knowledge Bases
   - Prolog: Backward Chaining

2. Reasoning with Neural Representations
   - Symbolic vs. Neural Representations
   - Neural Link Prediction
   - Computation Graphs

3. Deep Prolog: Neural Backward Chaining

4. Optimizations
   - Batch Proving
   - Gradient Approximation
   - Regularization by Neural Link Predictor

5. Experiments

6. Summary
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 $\lambda$**: 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 $\lambda$**: only prover is used at test time; ComplEx acts as a regularizer
Rule Templates

S1 \( \theta_1(X, Y) :- \theta_2(Y, Z). \)
\( \theta_1(X, Y) :- \theta_2(X, Z), \theta_2(Z, Y). \)

S2 \( \theta_1(X, Y) :- \theta_2(X, Z), \theta_3(Z, Y). \)

S3 \( \theta_1(X, Y) :- \theta_2(X, Z), \theta_3(Z, W), \theta_4(W, Y). \)
<table>
<thead>
<tr>
<th>Model</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>32.3</td>
<td>32.3</td>
<td>32.3</td>
</tr>
<tr>
<td>Frequency</td>
<td>32.3</td>
<td>32.3</td>
<td>30.8</td>
</tr>
<tr>
<td>ER-MLP (Dong et al., 2014)</td>
<td>96.0</td>
<td>74.5</td>
<td>65.0</td>
</tr>
<tr>
<td>Rescal (Nickel et al., 2012)</td>
<td>99.7</td>
<td>74.5</td>
<td>65.0</td>
</tr>
<tr>
<td>HolE (Nickel et al., 2015)</td>
<td>99.7</td>
<td>77.2</td>
<td>69.7</td>
</tr>
<tr>
<td>TARE (Wang et al., 2017)</td>
<td>99.4</td>
<td>90.6</td>
<td>89.0</td>
</tr>
<tr>
<td>NTP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DistMult (Yang et al., 2014)</td>
<td>98.1</td>
<td>98.3</td>
<td>65.5</td>
</tr>
<tr>
<td>NTP DistMult</td>
<td>99.2</td>
<td>96.7</td>
<td>87.0</td>
</tr>
<tr>
<td>NTP DistMult λ</td>
<td>99.4</td>
<td>98.3</td>
<td><strong>95.9</strong></td>
</tr>
<tr>
<td>ComplEx (Trouillon et al., 2016)</td>
<td>99.9</td>
<td>97.1</td>
<td>78.6</td>
</tr>
<tr>
<td>NTP ComplEx</td>
<td><strong>100.0</strong></td>
<td><strong>98.9</strong></td>
<td>89.1</td>
</tr>
<tr>
<td>NTP ComplEx λ</td>
<td>99.3</td>
<td>98.2</td>
<td>95.1</td>
</tr>
</tbody>
</table>
Results

![Boxplot showing AUC results for different tasks and models. The x-axis represents different tasks (S1, S2, S3), and the y-axis represents the AUC score. The boxplots compare various models, including NTP, DistMult, and ComplEx, with different variations like NTP DistMult and NTP ComplEx.]
### Induced Logic Programs

<table>
<thead>
<tr>
<th>Task</th>
<th>Confidence</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>0.999</td>
<td><code>neighborOf(X, Y) :- neighborOf(Y, X).</code></td>
</tr>
<tr>
<td></td>
<td>0.767</td>
<td><code>locatedIn(X, Y) :- locatedIn(X, Z), locatedIn(Z, Y).</code></td>
</tr>
<tr>
<td>S2</td>
<td>0.998</td>
<td><code>neighborOf(X, Y) :- neighborOf(Y, X).</code></td>
</tr>
<tr>
<td></td>
<td>0.995</td>
<td><code>locatedIn(X, Y) :- locatedIn(X, Z), locatedIn(Z, Y).</code></td>
</tr>
<tr>
<td></td>
<td>0.705</td>
<td><code>locatedIn(X, Y) :- neighborOf(X, Z), locatedIn(Z, Y).</code></td>
</tr>
<tr>
<td>S3</td>
<td>0.891</td>
<td><code>neighborOf(X, Y) :- neighborOf(Y, X).</code></td>
</tr>
<tr>
<td></td>
<td>0.750</td>
<td><code>locatedIn(X, Y) :- neighborOf(X, Z), neighborOf(Z, W), locatedIn(W, Y).</code></td>
</tr>
</tbody>
</table>
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


