GPU-accelerated End-to-end Differentiable Planning and Reasoning

Tim Rocktäschel

Whiteson Research Lab, University of Oxford

http://rockt.github.com  Twitter: @_rockt  tim.rocktaschel@cs.ox.ac.uk

Talk at GTC Europe, ID 23372

12th of October 2017
What vegetable is on the plate?  
Neural Net: **broccoli**  
Ground Truth: **broccoli**

What color are the shoes on the person's feet?  
Neural Net: **brown**  
Ground Truth: **brown**

How many school busses are there?  
Neural Net: **2**  
Ground Truth: **2**

What sport is this?  
Neural Net: **baseball**  
Ground Truth: **baseball**

What is on top of the refrigerator?  
Neural Net: **magnets**  
Ground Truth: **cereal**

What uniform is she wearing?  
Neural Net: **shorts**  
Ground Truth: **girl scout**

What is the table number?  
Neural Net: **4**  
Ground Truth: **40**

What are people sitting under in the back?  
Neural Net: **bench**  
Ground Truth: **tent**
a) Chemical Representation of the Synthesis Plan

Target

\[
\begin{align*}
\text{Boc} & \quad \text{N} \quad \text{O} \quad \text{CO}_2\text{Me} \\
\text{Ph} & \quad \text{CO}_2\text{Me}
\end{align*}
\]

\[1\]  \rightarrow  \begin{align*}
\text{MeO}_2\text{C} & \quad \text{\text{CO}_2\text{Me}}} \\
\text{Boc} & \quad \text{N} \quad \text{OH} \\
\text{Ph} & \quad \text{2}
\end{align*}

\[2\]  \rightarrow  \begin{align*}
\text{Boc} & \quad \text{N} \quad \text{OTBS} \\
\text{Ph} & \quad \text{3}
\end{align*}

\[3\]  \rightarrow  \begin{align*}
\text{Boc} & \quad \text{N} \quad \text{OH} \\
\text{Ph} & \quad \text{Br} \\
\text{5}
\end{align*}

\[4\]  \rightarrow  \begin{align*}
\text{Boc} & \quad \text{N} \quad \text{OTBS} \\
\text{Ph} & \quad \text{3}
\end{align*}

\[5\]  \rightarrow  \begin{align*}
\text{Boc} & \quad \text{N} \quad \text{OH} \\
\text{Ph} & \quad \text{2}
\end{align*}

\[6\]  \rightarrow  \begin{align*}
\text{MeO}_2\text{C} & \quad \text{\text{CO}_2\text{Me}}} \\
\text{Boc} & \quad \text{N} \quad \text{OH} \\
\text{Ph} & \quad \text{2}
\end{align*}

\[7\]  \rightarrow  \begin{align*}
\text{HN} & \quad \text{O} \quad \text{CO}_2\text{Me} \\
\text{Ph} & \quad \text{8}
\end{align*}

\[8\]  \rightarrow  \begin{align*}
\text{Boc} & \quad \text{N} \quad \text{OH} \\
\text{Ph} & \quad \text{2}
\end{align*}

\[9\]  \rightarrow  \begin{align*}
\text{HN} & \quad \text{O} \quad \text{CO}_2\text{Me} \\
\text{Ph} & \quad \text{8}
\end{align*}

b) Search Tree Representation

Root (Target): A

A → B → C → D

Terminal solved state:

A = \{1\}  B = \{2,6\}  C = \{3,6\}  D = \{4,5,6\}  E = \{8,9\}  F = \{7,8\}

Mit der Maßnahme soll sichergestellt werden, dass die Polizei die lebensrettende Ausrüstung bekommt, die sie brauche, um ihren Job zu machen, sagte US-Justizminister Jeff Sessions.

The police in the USA are allowed to get heavy equipment and weapons from the military again. This was decided by US President Donald Trump, who overturned an order from his predecessor Barack Obama, according to which the Department of Defense was banned from equipping the police with grenade launchers, armoured vehicles, bayonets, large-calibre weapons and ammunition.

The measure is designed to ensure that the police get the life-saving equipment they need to do their job, US Attorney General Jeff Sessions said.
The measure is designed to ensure that the police get the lifesaving equipment they need to do their job, US Attorney General Jeff Sessions said.
THIS IS YOUR MACHINE LEARNING SYSTEM?

YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.

WHAT IF THE ANSWERS ARE WRONG?

JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.

XKCD, 17th May 2017
Data & Explanations
- Rules
- (Partial) Programs
- Natural Language

XKCD, 17th May 2017
This is your machine learning system?

Yup! You pour the data into this big pile of linear algebra, then collect the answers on the other side.

What if the answers are wrong?

Just stir the pile until they start looking right.

XKCD, 17th May 2017

Data & Explanations
- Rules
- (Partial) Programs
- Natural Language

Answers & Explanations
- Rules
- Programs
- Natural Language
- Plans
XKCD, 17th May 2017

Data & Explanations
- Rules
- (Partial) Programs
- Natural Language

Answers & Explanations
- Rules
- Programs
- Natural Language
- Plans
Data & Explanations
- Rules
- (Partial) Programs
- Natural Language

Answers & Explanations
- Rules
- Programs
- Natural Language
- Plans

Data Efficiency & Model Interpretability

XKCD, 17th May 2017
Joint work with

1. **End-to-end Differentiable Reasoning**, Application: Knowledge Base Inference, *NIPS 2017*

Sebastian Riedel, University College London
Joint work with

1. End-to-end Differentiable Reasoning, Application: Knowledge Base Inference, *NIPS 2017*

Sebastian Riedel, University College London

2. End-to-end Differentiable Planning, Application: Atari, *work-in-progress*

Gregory Farquhar
Maximilian Igl
Shimon Whiteson

University of Oxford
End-to-end Differentiable Reasoning
Lecture Notes

PROLOG AND NATURAL-LANGUAGE ANALYSIS

Fernando C.N. Pereira
and
Stuart M. Shieber
goal problem.

rule 1
   if not turn_over and
   battery_bad
   then problem is battery cf 100.

rule 2
   if lights_weak
   then battery_bad cf 50.

rule 3
   if radio_weak
   then battery_bad cf 50.

rule 4
   if turn_over and
   smell_gas
   then problem is flooded cf 80.

rule 5
   if turn_over and
   gas_gauge is empty
   then problem is out_of_gas cf 90.

rule 6
   if turn_over and
   gas_gauge is low
   then problem is out_of_gas cf 30.
Expert Systems
- No/little training data
- Interpretable

goal problem.

rule 1
if not turn_over and
    battery_bad
then problem is battery cf 100.

rule 2
if lights_weak
then battery_bad cf 50.

rule 3
if radio_weak
then battery_bad cf 50.

rule 4
if turn_over and
    smell_gas
then problem is flooded cf 80.

rule 5
if turn_over and
    gas_gauge is empty
then problem is out_of_gas cf 90.

rule 6
if turn_over and
    gas_gauge is low
then problem is out_of_gas cf 30.

PROLOG AND NATURAL-LANGUAGE ANALYSIS

Fernando C. N. Pereira
and
Stuart M. Shieber
Expert Systems

- No/little training data
- Interpretable
- Behavior manually defined
- No generalization

 goal problem.

rule 1
  if not turn_over and
  battery_bad
  then problem is battery cf 100.

rule 2
  if lights_weak
  then battery_bad cf 50.

rule 3
  if radio_weak
  then battery_bad cf 50.

rule 4
  if turn_over and
  smell_gas
  then problem is flooded cf 80.

rule 5
  if turn_over and
  gas_gauge is empty
  then problem is out_of_gas cf 90.

rule 6
  if turn_over and
  gas_gauge is low
  then problem is out_of_gas cf 30.
goal problem.

rule 1
  if not turn_over and battery_bad
  then problem is battery cf 100.

rule 2
  if lights_weak
  then battery_bad cf 50.

rule 3
  if radio_weak
  then battery_bad cf 50.

rule 4
  if turn_over and smell_gas
  then problem is flooded cf 80.

rule 5
  if turn_over and gas_gauge is empty
  then problem is out_of_gas cf 90.

rule 6
  if turn_over and gas_gauge is low
  then problem is out_of_gas cf 30.

**Expert Systems**

- No/little training data
- Interpretable
- Behavior manually defined
- No generalization
goal problem.

rule 1
if not turn_over and battery_bad
then problem is battery cf 100.

rule 2
if lights_weak
then battery_bad cf 50.

rule 3
if radio_weak
then battery_bad cf 50.

rule 4
if turn_over and smell_gas
then problem is flooded cf 80.

rule 5
if turn_over and gas_gauge is empty
then problem is out_of_gas cf 90.

rule 6
if turn_over and gas_gauge is low
then problem is out_of_gas cf 30.

**Expert Systems**

- No/little training data
- Interpretable
- Behavior manually defined
- No generalization

**Representation Learning**

- Behavior learned
- Strong generalization
Expert Systems
- No/little training data
- Interpretable
- Behavior manually defined
- No generalization

Representation Learning
- Lot of training data needed
- Not interpretable
- Behavior learned
- Strong generalization
goal problem.

rule 1
if not turn_over and
   battery_bad
then problem is battery cf 100.

rule 2
if lights_weak
   then battery_bad cf 50.

rule 3
if radio_weak
   then battery_bad cf 50.

rule 4
if turn_over and
   smell_gas
then problem is flooded cf 80.

rule 5
if turn_over and
   gas_gauge is empty
then problem is out_of_gas cf 90.

rule 6
if turn_over and
   gas_gauge is low
then problem is out_of_gas cf 30.

---

**Expert Systems**
- No/little training data
- Interpretable

**Representation Learning**
- Behavior learned
- Strong generalization
Nando de Freitas @NandoDF · 5 Aug 2016

Neuralise (verb, #neuralize): to implement a known thing with deep nets. Usage:
Let's neuralize warping, neuralize this! And train it!

Yann LeCun
@ylecun

Replying to @NandoDF

sort of like "kernelize" used to be.

10:11 AM - 5 Aug 2016
Aims

- Modular construction of neural networks for end-to-end differentiable reasoning in knowledge bases
Aims

- Modular construction of neural networks for end-to-end differentiable reasoning in knowledge bases
- Incorporate background knowledge in form of rules
  ⇒ Data Efficiency
Aims

- Modular construction of neural networks for end-to-end differentiable reasoning in knowledge bases
- Incorporate background knowledge in form of rules ⇒ Data Efficiency
- Calculate gradient of proof success w.r.t. subsymbolic representations
Aims

- Modular construction of neural networks for end-to-end differentiable reasoning in knowledge bases
- Incorporate background knowledge in form of rules
  ⇒ **Data Efficiency**
- Calculate gradient of proof success w.r.t. subsymbolic representations
- Rule application is explicit, but symbol comparison is neural
Aims

- Modular construction of neural networks for end-to-end differentiable reasoning in knowledge bases
- Incorporate background knowledge in form of rules
  - Data Efficiency
- Calculate gradient of proof success w.r.t. subsymbolic representations
- Rule application is explicit, but symbol comparison is neural
- Use similarity between vector representations of symbols in proofs
Aims

- Modular construction of neural networks for end-to-end differentiable reasoning in knowledge bases
- Incorporate background knowledge in form of rules
  \[ \Rightarrow \textbf{Data Efficiency} \]
- Calculate gradient of proof success w.r.t. subsymbolic representations
- Rule application is explicit, but symbol comparison is neural
- Use similarity between vector representations of symbols in proofs
- Learn vector representations of symbols from data using gradient descent
Aims

- Modular construction of neural networks for end-to-end differentiable reasoning in knowledge bases
- Incorporate background knowledge in form of rules  
  ⇒ Data Efficiency
- Calculate gradient of proof success w.r.t. subsymbolic representations
- Rule application is explicit, but symbol comparison is neural
- Use similarity between vector representations of symbols in proofs
- Learn vector representations of symbols from data using gradient descent
- Induce interpretable logical rules from data by gradient descent  
  ⇒ Model Interpretability
Task: Link Prediction

Real world knowledge bases (like Freebase, DBPedia, YAGO, etc.) are incomplete!
Task: Link Prediction

Real world knowledge bases (like Freebase, DBPedia, YAGO, etc.) are incomplete!

- \texttt{placeOfBirth} attribute is missing for 71% of people!
Real world knowledge bases (like Freebase, DBPedia, YAGO, etc.) are incomplete!

- placeOfBirth attribute is missing for 71% of people!
- Commonsense knowledge often not stated explicitly
Task: Link Prediction

Real world knowledge bases (like Freebase, DBPedia, YAGO, etc.) 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
Task: Link Prediction

Real world knowledge bases (like Freebase, DBPedia, YAGO, etc.) 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

Das et al. (2017)
Real world knowledge bases (like Freebase, DBPedia, YAGO, etc.) 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

Das et al. (2017)
Real world knowledge bases (like Freebase, DBPedia, YAGO, etc.) 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

Das et al. (2017)
Real world knowledge bases (like Freebase, DBPedia, YAGO, etc.) 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

Das et al. (2017)
Notation

- **Constant**: HOMER, BART, LISA etc. (lowercase)
Notation

- **Constant**: HOMER, BART, LISA etc. (lowercase)
- **Variable**: X, Y etc. (uppercase, universally quantified)
**Notation**

- **Constant**: HOMER, BART, LISA etc. (lowercase)
- **Variable**: X, Y etc. (uppercase, universally quantified)
- **Term**: constant or variable
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
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)
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)
- **Rule**: head :- body.
  head: atom
  body: (possibly empty) list of atoms representing conjunction
  grandfatherOf(X, Y) :- fatherOf(X, Z), parentOf(Z, Y).
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)
- **Rule**: head :- body.
  head: atom
  body: (possibly empty) list of atoms representing conjunction
  grandfatherOf(X, Y) :- fatherOf(X, Z), parentOf(Z, Y).
- **Fact**: ground rule (no free variables) with empty body, e.g.,
  parentOf(HOMER, BART).
Example Knowledge Base

1. \texttt{fatherOf(ABE, HOMER)}.
2. \texttt{parentOf(HOMER, LISA)}.
3. \texttt{parentOf(HOMER, BART)}.
4. \texttt{grandpaOf(ABE, LISA)}.
5. \texttt{grandfatherOf(ABE, MAGGIE)}.
Example Knowledge Base

1. \texttt{fatherOf(ABE, HOMER)}.  
2. \texttt{parentOf(HOMER, LISA)}.  
3. \texttt{parentOf(HOMER, BART)}.  
4. \texttt{grandpaOf(ABE, LISA)}.  
5. \texttt{grandfatherOf(ABE, MAGGIE)}.  
6. \texttt{grandfatherOf(X_1, Y_1) :-} 
   \begin{align*} 
   & \texttt{fatherOf(X_1, Z_1)}, \\
   & \texttt{parentOf(Z_1, Y_1)}. 
   \end{align*}  
7. \texttt{grandparentOf(X_2, Y_2) :-} 
   \begin{align*} 
   & \texttt{grandfatherOf(X_2, Y_2)}. 
   \end{align*}
Prolog Backward Chaining Example

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

What about grandfatherOf(ABE, BART)?

failure

failure

success

{X/ABE, Y/BART, Z/HOMER}
Prolog Backward Chaining Example

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

grandfatherOf(ABE, BART)?
Prolog Backward Chaining Example

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

grandfatherOf(ABE, BART)?

1

failure

3

success

{X/ABE, Y/BART}

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

grandfatherOf(ABE, BART)?

1. failure
2. failure
3. success
   \{X/ABE, Y/BART\}

3.1 fatherOf(ABE, Z)?
1. success
   \{X/ABE, Y/BART, Z/HOMER\}
2. failure
3. failure

3.2 parentOf(HOMER, BART)?
Prolog Backward Chaining Example

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

grandfatherOf(ABE, BART)?

failure

fatherOf(ABE, Z)?

success

{X/ABE, Y/BART, Z/HOMER}

parentOf(HOMER, BART)?

failure

success

{X/ABE, Y/BART, Z/HOMER}

What about grandfatherOf(ABE, BART)?
Prolog Backward Chaining Example

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

What about \( \text{grandfatherOf}(\text{ABE}, \text{BART})? \)?
Symbolic Representations

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

- Symbols (constants and predicates) do not share any information:
  \[ \text{grandpaOf} \neq \text{grandfatherOf} \]
- No notion of similarity: \( \text{APPLE} \sim \text{ORANGE}, \text{professorAt} \sim \text{lecturerAt} \)
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:
  \( \text{isFruit(APPLE)}, \text{APPLE} \sim \text{ORANGE}, \text{isFruit(ORANGE)}? \)
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:
  \(\text{isFruit(APPLE)}, \text{APPLE} \sim \text{ORGANGE}, \text{isFruit(ORANGE)}?\)

- Hard to work with language, vision and other modalities
  ‘‘is a film based on the novel of the same name by’’(\(X, Y\))
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:
\( \text{isFruit(APPLE)}, \ \text{APPLE} \sim \text{ORGANIZE}, \ \text{isFruit(ORANGE)}? \)

Hard to work with language, vision and other modalities
\(''is a film based on the novel of the same name by’’(X, Y)"

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

- Symbols (constants and predicates) do not share any information:
  \( \text{grandpa0f} \neq \text{grandfather0f} \)

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

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

- Hard to work with language, vision and other modalities
  ‘‘is a film based on the novel of the same name by’’(X, Y)

- But... leads to powerful inference mechanisms and proofs for predictions:
  \( \text{father0f(ABE, HOMER)}, \text{parent0f(HOMER, LISA)}, \text{parent0f(HOMER, BART)} \).
  \( \text{grandfather0f(X, Y)} \leftarrow \text{father0f(X, Z)}, \text{parent0f(Z, Y)} \).
  \( \text{grandfather0f(ABE, Q)}? \{Q/LISA\}, \{Q/BART\} \)

- Fairly easy to debug and trivial to incorporate domain knowledge:
  Show to domain expert and let her change/add rules and facts
Neural Representations

- Lower-dimensional fixed-length vector representations of symbols (predicates and constants):
  \[ v_{\text{APPLE}}, v_{\text{ORANGE}}, v_{\text{fatherOf}}, \ldots \in \mathbb{R}^k \]
Neural Representations

- Lower-dimensional fixed-length vector representations of symbols (predicates and constants):
  \[ \mathbf{v}_{\text{APPLE}}, \mathbf{v}_{\text{ORANGE}}, \mathbf{v}_{\text{father0f}}, \ldots \in \mathbb{R}^k \]

- Can capture similarity and even semantic hierarchy of symbols:
  \[ \mathbf{v}_{\text{grandpa0f}} = \mathbf{v}_{\text{grandfather0f}}, \]
  \[ \mathbf{v}_{\text{APPLE}} \sim \mathbf{v}_{\text{ORANGE}}, \mathbf{v}_{\text{APPLE}} < \mathbf{v}_{\text{FRUIT}} \]
Neural Representations

- Lower-dimensional fixed-length vector representations of symbols (predicates and constants):
  \[ \mathbf{v}_{\text{APPLE}}, \mathbf{v}_{\text{ORANGE}}, \mathbf{v}_{\text{father0f}}, \ldots \in \mathbb{R}^k \]

- Can capture similarity and even semantic hierarchy of symbols:
  \[ \mathbf{v}_{\text{grandpa0f}} = \mathbf{v}_{\text{grandfather0f}}, \]
  \[ \mathbf{v}_{\text{APPLE}} \sim \mathbf{v}_{\text{ORANGE}}, \mathbf{v}_{\text{APPLE}} < \mathbf{v}_{\text{FRUIT}} \]

- Can be trained from raw task data (e.g. facts in a knowledge base)
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 in a knowledge base)

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

- Lower-dimensional fixed-length vector representations of symbols (predicates and constants):
  \[ \mathbf{v}_{\text{APPLE}}, \mathbf{v}_{\text{ORANGE}}, \mathbf{v}_{\text{father0f}}, \ldots \in \mathbb{R}^k \]

- Can capture similarity and even semantic hierarchy of symbols:
  \[ \mathbf{v}_{\text{grandpa0f}} = \mathbf{v}_{\text{grandfather0f}}, \quad \mathbf{v}_{\text{APPLE}} \sim \mathbf{v}_{\text{ORANGE}}, \mathbf{v}_{\text{APPLE}} < \mathbf{v}_{\text{FRUIT}} \]

- Can be trained from raw task data (e.g. facts in a knowledge base)

- 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
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 in a knowledge base)

- 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
  \( \mathbf{v}_{\text{grandfatherOf}}(X, Y) : \leftarrow \mathbf{v}_{\text{fatherOf}}(X, Z), \mathbf{v}_{\text{parentOf}}(Z, Y). \)
Machine Learning & Logic

- Fuzzy Logic (Zadeh, 1965)
Machine Learning & Logic

- Fuzzy Logic (Zadeh, 1965)
- Probabilistic Logic Programming, e.g.,
Machine Learning & Logic

- **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) . . .
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.,
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.,
Machine Learning & Logic

- 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)
Machine Learning & Logic

- 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
Machine Learning & Logic

- 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)
Machine Learning & Logic

- 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)
State-of-the-art Neural Link Prediction

\[
livesIn(\text{MELINDA, SEATTLE})? = f(v_{livesIn}, v_{MELINDA}, v_{SEATTLE})
\]
State-of-the-art Neural Link Prediction

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

**DistMult** (Yang et al., 2015)

\[
\mathbf{v}_s, \mathbf{v}_i, \mathbf{v}_j \in \mathbb{R}^k
\]
State-of-the-art Neural Link Prediction

\[
livesIn(MELINDA, SEATTLE)? = f(v_{livesIn}, v_{MELINDA}, v_{SEATTLE})
\]

DistMult \textit{(Yang et al., 2015)}

\[
v_s, v_i, v_j \in \mathbb{R}^k
\]

\[
f(v_s, v_i, v_j) = v_s^T (v_i \odot v_j)
= \sum_k v_{sk} v_{ik} v_{jk}
\]
State-of-the-art Neural Link Prediction

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

**DistMult** (Yang et al., 2015)

\[ \mathbf{v}_s, \mathbf{v}_i, \mathbf{v}_j \in \mathbb{R}^k \]

\[ f(\mathbf{v}_s, \mathbf{v}_i, \mathbf{v}_j) = \mathbf{v}_s^\top (\mathbf{v}_i \odot \mathbf{v}_j) \]

\[ = \sum_k \mathbf{v}_{sk} \mathbf{v}_{ik} \mathbf{v}_{jk} \]

**ComplEx** (Trouillon et al., 2016)

\[ \mathbf{v}_s, \mathbf{v}_i, \mathbf{v}_j \in \mathbb{C}^k \]
State-of-the-art Neural Link Prediction

\[ \text{livesIn(}\text{MELINDA, SEATTLE})? = f(\text{livesIn}, \text{MELINDA}, \text{SEATTLE}) \]

**DistMult** (Yang et al., 2015)

\[ \mathbf{v}_s, \mathbf{v}_i, \mathbf{v}_j \in \mathbb{R}^k \]

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

**ComplEx** (Trouillon et al., 2016)

\[ \mathbf{v}_s, \mathbf{v}_i, \mathbf{v}_j \in \mathbb{C}^k \]

\[
f(\mathbf{v}_s, \mathbf{v}_i, \mathbf{v}_j) = \]
\[
= \operatorname{real}(\mathbf{v}_s)^\top(\operatorname{real}(\mathbf{v}_i) \odot \operatorname{real}(\mathbf{v}_j)) + \operatorname{real}(\mathbf{v}_s)^\top(\operatorname{imag}(\mathbf{v}_i) \odot \operatorname{imag}(\mathbf{v}_j)) + \operatorname{imag}(\mathbf{v}_s)^\top(\operatorname{real}(\mathbf{v}_i) \odot \operatorname{imag}(\mathbf{v}_j)) - \operatorname{imag}(\mathbf{v}_s)^\top(\operatorname{imag}(\mathbf{v}_i) \odot \operatorname{real}(\mathbf{v}_j))\]
State-of-the-art Neural Link Prediction

\[
livesIn(MELINDA, SEATTLE)? = f(v_{livesIn}, v_{MELINDA}, v_{SEATTLE})
\]

**DistMult** (Yang et al., 2015)
\[
v_s, v_i, v_j \in \mathbb{R}^k
\]
\[
f(v_s, v_i, v_j) = v_s^\top (v_i \odot v_j) = \sum_k v_{sk} v_{ik} v_{jk}
\]

**ComplEx** (Trouillon et al., 2016)
\[
v_s, v_i, v_j \in \mathbb{C}^k
\]
\[
f(v_s, v_i, v_j) =
\]
\[
\begin{align*}
&\text{real}(v_s)^\top (\text{real}(v_i) \odot \text{real}(v_j)) \\
+ &\text{real}(v_s)^\top (\text{imag}(v_i) \odot \text{imag}(v_j)) \\
+ &\text{imag}(v_s)^\top (\text{real}(v_i) \odot \text{imag}(v_j)) \\
- &\text{imag}(v_s)^\top (\text{imag}(v_i) \odot \text{real}(v_j))
\end{align*}
\]

**Training Loss**
\[
\mathcal{L} = \sum_{r_s(e_i, e_j), y \in \mathcal{T}} -y \log (\sigma(f(v_s, v_i, v_j))) - (1 - y) \log (1 - \sigma(f(v_s, v_i, v_j)))
\]
Example Knowledge Base:
1. fatherOf(ABE, HOMER).
2. parentOf(HOMER, BART).
3. grandfatherOf(X, Y) :-
   fatherOf(X, Z),
   parentOf(Z, Y).
Differentiable Proving in a Nutshell

Example Knowledge Base:
1. `fatherOf(ABE, HOMER).`
2. `parentOf(HOMER, BART).`
3. `grandfatherOf(X, Y) :- fatherOf(X, Z), parentOf(Z, Y).`
Differentiable Proving in a Nutshell

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

\[
\begin{align*}
\text{grandfatherOf} & \quad \text{ABE} \quad \text{BART} \\
1. \text{fatherOf}(\text{ABE}, \text{HOMER})
\end{align*}
\]
Differentiable Proving in a Nutshell

Example Knowledge Base:
1. fatherOf(ABE, HOMER).
2. parentOf(HOMER, BART).
3. grandfatherOf(X, Y) :- fatherOf(X, Z), parentOf(Z, Y).
Differentiable Proving in a Nutshell

Example Knowledge Base:
1. fatherOf(ABE, HOMER).
2. parentOf(HOMER, BART).
3. grandfatherOf(X, Y) :- fatherOf(X, Z), parentOf(Z, Y).
Differentiable Proving in a Nutshell

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

Tim Rocktäschel  GPU-accelerated End-to-end Differentiable Planning and Reasoning 16/39
Differentiable Proving in a Nutshell

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

\[
\begin{align*}
1. & \text{ fatherOf}(ABE, HOMER) \\
2. & \text{ parentOf}(HOMER, BART) \\
3. & \text{ grandfatherOf}(X, Y) :\neg \text{ fatherOf}(X, Z), \text{ parentOf}(Z, Y)
\end{align*}
\]
Differentiable Proving in a Nutshell

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

1. fatherOf(ABE, HOMER)
2. parentOf(HOMER, BART)
3. grandfatherOf(X, Y) :- fatherOf(X, Z), parentOf(Z, Y)
Differentiable Proving in a Nutshell

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

Tim Rocktäschel  GPU-accelerated End-to-end Differentiable Planning and Reasoning 16/39
Example Knowledge Base:
1. fatherOf(ABE, HOMER).
2. parentOf(HOMER, BART).
3. grandfatherOf(X, Y) :- fatherOf(X, Z), parentOf(Z, Y).

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

fatherOf

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

parentOf

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

grandfatherOf

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

\[ S = (\Psi, \rho) \]

- Substitution set \( \Psi \) constructed in the proof so far
Proof States

\[ S = (\Psi, \rho) \]

- Substitution set \( \Psi \) constructed in the proof so far
- Neural network \( \rho \) that outputs a real-valued proof success score
Proof States

$S = (\Psi, \rho)$

- Substitution set $\Psi$ constructed in the proof so far
- Neural network $\rho$ that outputs a real-valued proof success score
Proof States

\[ S = (\Psi, \rho) \]

- Substitution set \( \Psi \) constructed in the proof so far
- Neural network \( \rho \) that outputs a real-valued proof success score
Proof Modules

\[ \text{unify}_\theta, \text{or}_\theta, \text{and}_\theta \]

- Modular construction of differentiable prover
Proof Modules

\[ \text{unify}_\theta, \text{or}_\theta, \text{and}_\theta \]

- Modular construction of differentiable prover
- Discrete objects (rules, atoms etc.) are used to instantiate proof modules
Proof Modules

\[ \text{unify}_\theta, \text{or}_\theta, \text{and}_\theta \]

- Modular construction of differentiable prover
- Discrete objects (rules, atoms etc.) are used to instantiate proof modules
- **Modules** transform proof states into new proof states
Proof Modules

unify_θ, or_θ, and_θ

- Modular construction of differentiable prover
- Discrete objects (rules, atoms etc.) are used to instantiate proof modules
- Modules transform proof states into new proof states
Proof Modules

\[ \text{unify}_\theta, \text{or}_\theta, \text{and}_\theta \]

- Modular construction of differentiable prover
- Discrete objects (rules, atoms etc.) are used to instantiate proof modules
- **Modules** transform proof states into new proof states

![Diagram of proof modules](image-url)
Proof Modules

unify_θ, or_θ, and_θ

- Modular construction of differentiable prover
- Discrete objects (rules, atoms etc.) are used to instantiate proof modules
- **Modules** transform proof states into new proof states
Unification Module

`unify` takes two atoms represented as lists of terms and an upstream proof state, and maps these to a new proof state (substitution set and proof success)
Unification Module

\texttt{unify} takes two atoms represented as lists of terms and an upstream proof state, and maps these to a new proof state (substitution set and proof success)

1. \( \text{unify}_\theta([ ],[ ],S) = S \)
Unification Module

`unify` takes two atoms represented as lists of terms and an upstream proof state, and maps these to a new proof state (substitution set and proof success)

1. $\text{unify}_\theta([],[],S) = S$
2. $\text{unify}_\theta([],G,S) = \text{FAIL}$
Unification Module

**unify** takes two atoms represented as lists of terms and an upstream proof state, and maps these to a new proof state (substitution set and proof success)

1. \(\text{unify}_\theta([],[],S) = S\)
2. \(\text{unify}_\theta([],G,S) = \text{FAIL}\)
3. \(\text{unify}_\theta(H,[],S) = \text{FAIL}\)
Unification Module

**unify** takes two atoms represented as lists of terms and an upstream proof state, and maps these to a new proof state (substitution set and proof success)

1. $\text{unify}_\theta([\ ],[\ ],S) = S$
2. $\text{unify}_\theta([\ ],G,S) = \text{FAIL}$
3. $\text{unify}_\theta(H,[\ ],S) = \text{FAIL}$
4. $\text{unify}_\theta(h :: H, g :: G, S) = \text{unify}_\theta(H, G, S')$
**Unification Module**

`unify` takes two atoms represented as lists of terms and an upstream proof state, and maps these to a new proof state (substitution set and proof success)

1. \( \text{unify}_\theta([\ ],[\ ],S) = S \)
2. \( \text{unify}_\theta([\ ],G,S) = \text{FAIL} \)
3. \( \text{unify}_\theta(H,[\ ],S) = \text{FAIL} \)
4. \( \text{unify}_\theta(h :: H,g :: G,S) = \text{unify}_\theta(H,G,S') \)

\[
S'_\psi = S_\psi \cup \begin{cases} 
\{h/g\} & \text{if } h \in \mathcal{V} \\
\{g/h\} & \text{if } g \in \mathcal{V}, h \not\in \mathcal{V} \\
\emptyset & \text{otherwise} 
\end{cases}
\]
**Unification Module**

`unify` takes two atoms represented as lists of terms and an upstream proof state, and maps these to a new proof state (substitution set and proof success)

1. $\text{unify}_\theta([ ], [ ], S) = S$
2. $\text{unify}_\theta([ ], G, S) = \text{FAIL}$
3. $\text{unify}_\theta(H, [ ], S) = \text{FAIL}$
4. $\text{unify}_\theta(h :: H, g :: G, S) = \text{unify}_\theta(H, G, S')$

$$S'_\psi = S_\psi \cup \begin{cases} 
\{ h/g \} & \text{if } h \in \mathcal{V} \\
\{ g/h \} & \text{if } g \in \mathcal{V}, h \notin \mathcal{V} \\
\emptyset & \text{otherwise}
\end{cases}$$

$$S'_\rho = \min \left( S_\rho, \begin{cases} 
\exp(-\|\theta_h - \theta_g\|_2) & \text{if } h \notin \mathcal{V}, g \notin \mathcal{V} \\
1 & \text{otherwise}
\end{cases} \right)$$
**Unification Module**

`unify` takes two atoms represented as lists of terms and an upstream proof state, and maps these to a new proof state (substitution set and proof success)

1. `unify_\theta([], [], S) = S`
2. `unify_\theta([], G, S) = FAIL`
3. `unify_\theta(H, [], S) = FAIL`
4. `unify_\theta(h :: H, g :: G, S) = unify_\theta(H, G, S')`

\[
S'_\psi = S_\psi \cup \begin{cases} 
\{h/g\} & \text{if } h \in \mathcal{V} \\
\{g/h\} & \text{if } g \in \mathcal{V}, h \notin \mathcal{V} \\
\emptyset & \text{otherwise}
\end{cases}
\]

\[
S'_\rho = \min\left(S_\rho, \begin{cases} 
\exp\left(-\|\theta_h: - \theta_g:\|_2\right) & \text{if } h \notin \mathcal{V}, g \notin \mathcal{V} \\
1 & \text{otherwise}
\end{cases}\right)
\]

Example:

\[
unify_\theta([\text{grandpaOf}, \text{ABE, BART}], [s, Q, i], (\emptyset, 0.7)) = \]

\[
\left(\{Q/\text{ABE}\}, \min(0.7, \exp(-\|\theta_{\text{grandpaOf}}: - \theta_s:\|_2), \exp(-\|\theta_{\text{BART}}: - \theta_i:\|_2))\right)
\]
OR Module

1. $\text{or}_\theta^R(G, d, S) = [S' \mid S' \in \text{and}_\theta^R(B, d, \text{unify}_\theta(H, G, S)), H \leftarrow B \in \mathcal{R}]$

   - $G$ is a goal atom, $d$ is the maximum proof depth, and $H \leftarrow B$ is a rule
OR Module

1. \( \text{or}^\mathcal{R}(G, d, S) = \{ S' \mid S' \in \text{and}^\mathcal{R}(B, d, \text{unify}_\theta(H, G, S)), H \leftarrow B \in \mathcal{R} \} \)

- \( G \) is a goal atom, \( d \) is the maximum proof depth, and \( H \leftarrow B \) is a rule
- \( \text{or} \) iterates through all rules (including rules with an empty body, \( i.e., \) facts) and unifies the goal with the respective rule head
OR Module

1. \( \text{or}_\theta^R(G, d, S) = \{ S' \mid S' \in \text{and}_\theta^R(B, d, \text{unify}_\theta(H, G, S)), H :\!-\! B \in \mathcal{R} \} \)

- \( G \) is a goal atom, \( d \) is the maximum proof depth, and \( H :\!-\! B \) is a rule
- \text{or} iterates through all rules (including rules with an empty body, i.e., facts) and unifies the goal with the respective rule head
- If unification succeeds, it instantiates an \text{and} module to prove all atoms in the body of the rule.
OR Module

1. \( \text{or}_\theta^R(G, d, S) = [S' \mid S' \in \text{and}_\theta^R(B, d, \text{unify}_\theta(H, G, S)), H \leftarrow B \in \mathcal{R}] \)

- \( G \) is a goal atom, \( d \) is the maximum proof depth, and \( H \leftarrow B \) is a rule
- \text{or} iterates through all rules (including rules with an empty body, \( i.e. \), facts) and unifies the goal with the respective rule head
- If unification succeeds, it instantiates an \text{and} module to prove all atoms in the body of the rule.
- In other words, it is translating goals into subgoals using rules, \( e.g. \), \text{grandfatherOf}(Q, \text{BART}) \) is translated into subgoals \text{fatherOf}(Q, Z) \) and \text{parentOf}(Z, \text{BART}) \) using the rule

\( \text{grandfatherOf}(X, Y) \leftarrow \text{fatherOf}(X, Z), \text{parentOf}(Z, Y) \)
OR Module

1. \( \text{or}_\theta^R(G, d, S) = [S' \mid S' \in \text{and}_\theta^R(B, d, \text{unify}_\theta(H, G, S)), H :\neg B \in \mathcal{R}] \)

- \( G \) is a goal atom, \( d \) is the maximum proof depth, and \( H :\neg B \) is a rule
- \text{or} iterates through all rules (including rules with an empty body, \textit{i.e.}, facts) and unifies the goal with the respective rule head
- If unification succeeds, it instantiates an \text{and} module to prove all atoms in the body of the rule.
- In other words, it is translating goals into subgoals using rules, \textit{e.g.},
  \text{grandfatherOf}(Q, \text{BART}) is translated into subgoals \text{fatherOf}(Q, Z) and \text{parentOf}(Z, \text{BART}) using the rule
  \text{grandfatherOf}(X, Y) :\neg \text{fatherOf}(X, Z), \text{parentOf}(Z, Y)

Example:
\[
\text{or}_\theta^R([\text{grandfatherOf}, Q, \text{BART}], d, S) = \\
[S' \mid S' \in \text{and}_\theta^R([\text{fatherOf}, X, Z], [\text{parentOf}, Z, Y]), d, \{X/Q, Y/BART\}, \hat{S}_\rho), \ldots]
\]
AND Module

1. \( \text{and}_\theta^\mathcal{G}(G, d, \text{FAIL}) = \text{FAIL} \)
AND Module

1. $\text{and}_\theta^ \tilde{g}(G, d, \text{FAIL}) = \text{FAIL}$
2. $\text{and}_\theta^ \tilde{g}(G, 0, S) = \text{FAIL}$
1. \( \text{and}_{\theta}^{\bar{r}}(G, d, \text{FAIL}) = \text{FAIL} \)
2. \( \text{and}_{\theta}^{\bar{r}}(G, 0, S) = \text{FAIL} \)
3. \( \text{and}_{\theta}^{\bar{r}}([ ], d, S) = S \)
1. \( \text{and}^\theta(G, d, \text{FAIL}) = \text{FAIL} \)
2. \( \text{and}^\theta(G, 0, S) = \text{FAIL} \)
3. \( \text{and}^\theta([ ], d, S) = S \)
4. \( \text{and}^\theta(G :: G, d, S) = [S'' \mid S'' \in \text{and}^\theta(G, d, S'), S' \in \text{or}^\theta(\text{substitute}(G, S), d - 1, S)] \)
AND Module

1. \( \text{and}^\mathcal{R}_\theta(G, d, \text{FAIL}) = \text{FAIL} \)
2. \( \text{and}^\mathcal{R}_\theta(G, 0, S) = \text{FAIL} \)
3. \( \text{and}^\mathcal{R}_\theta([], d, S) = S \)
4. \( \text{and}^\mathcal{R}_\theta(G :: G, d, S) = [S'' \mid S'' \in \text{and}^\mathcal{R}_\theta(G, d, S'), S' \in \text{or}^\mathcal{R}_\theta(\text{substitute}(G, S), d - 1, S)] \)

Example:

\[
\text{and}^\mathcal{R}_\theta([[\text{father0f}, X, Z], [\text{parent0f}, Z, Y]], d, \{X/Q, Y/BART\}, \hat{S}_\rho)) = \text{result of unify}_\theta \quad \text{in or}^\mathcal{R}_\theta
\]

\[
[S'' | S'' \in \text{and}^\mathcal{R}_\theta([[\text{parent0f}, Z, Y]], d, S'), S' \in \text{or}^\mathcal{R}_\theta([[\text{father0f}, Q, Z]], d - 1, S)] \quad \text{result of substitute}
\]
Proof Aggregation

- Goal $G = [s, i, j]$ where $s$ is the index of a predicate symbol and $i, j$ are indices of constant symbols
- $d$ maximum proof depth and proof start state $(\emptyset, 1)$

$$ntp_{\theta}^R(G, d) = \max_{S \in \text{or}_{\theta}^R(G, d, (\emptyset, 1))} S_p$$

$S_p \neq \text{FAIL}$
Proof Aggregation

- Goal \( G = [s, i, j] \) where \( s \) is the index of a predicate symbol and \( i, j \) are indices of constant symbols
- \( d \) maximum proof depth and proof start state \( (\emptyset, 1) \)

\[
\text{ntp}_{\theta}^{\vec{r}}(G, d) = \max_{S \in \text{or}_{\theta}^{\vec{r}}(G,d,(\emptyset,1))} S_{\rho}
\]

\( S \neq \text{FAIL} \)

Training Loss

\[
\mathcal{L}_{\text{ntp}_{\theta}^{\vec{r}}} = \sum_{(G,y) \in T} -y \log(\text{ntp}_{\theta}^{\vec{r}}(G, d)) - (1 - y) \log(1 - \text{ntp}_{\theta}^{\vec{r}}(G, d))
\]
Example Knowledge Base:
1. fatherOf(ABE, HOMER).
2. parentOf(HOMER, BART).
3. grandfatherOf(X, Y) :-
   fatherOf(X, Z),
   parentOf(Z, Y).

\( or_\theta([s, i, j], 2, (\emptyset, 1)) \)
Example Knowledge Base:
1. \texttt{fatherOf(ABE, HOMER)}.
2. \texttt{parentOf(HOMER, BART)}.
3. \texttt{grandfatherOf(X, Y) :- fatherOf(X, Z), parentOf(Z, Y)}.

\begin{align*}
\text{unify}_\theta([\texttt{fatherOf, ABE, HOMER}], [s, i, j], (\emptyset, 1)) & \quad\text{1.} \\
\text{or}_\theta([s, i, j], 2, (\emptyset, 1)) & \quad\text{2.} \\
\text{unify}_\theta([\texttt{grandfatherOf, X, Y}], [s, i, j], (\emptyset, 1)) & \quad\text{3.}
\end{align*}

\begin{align*}
S_1 = (\emptyset, \rho_1) \\
S_2 = (\emptyset, \rho_2)
\end{align*}
Example Knowledge Base:
1. \texttt{fatherOf}(\texttt{ABE, HOMER}).
2. \texttt{parentOf}(\texttt{HOMER, BART}).
3. \texttt{grandfatherOf}(X, Y) :-
   \texttt{fatherOf}(X, Z),
   \texttt{parentOf}(Z, Y).

\begin{align*}
\text{unify}_\theta([\texttt{fatherOf}, \texttt{ABE, HOMER}], [s, i, j], (\emptyset, 1)) \\
S_1 = (\emptyset, \rho_1) \\
\text{unify}_\theta([\texttt{grandfatherOf}, X, Y], [s, i, j], (\emptyset, 1)) \\
S_2 = (\emptyset, \rho_2) \\
\text{and}_\theta([[\texttt{fatherOf}, X, Z], [\texttt{parentOf}, Z, Y]], 2, S_3) \\
S_3 = (\{X/i, Y/j\}, \rho_3)
\end{align*}
Example Knowledge Base:
1. fatherOf(ABE, HOMER).
2. parentOf(HOMER, BART).
3. grandfatherOf(X, Y) :-
   fatherOf(X, Z),
   parentOf(Z, Y).

or$^k_{\theta}([s, i, j], 2, (\emptyset, 1))$

unify$^\theta([\text{fatherOf}, ABE, HOMER], [s, i, j], (\emptyset, 1))$

S_1 = (\emptyset, \rho_1)

unify$^\theta([\text{grandfatherOf}, X, Y], [s, i, j], (\emptyset, 1))$

S_2 = (\emptyset, \rho_2)

and$^k_{\theta}([[\text{fatherOf}, X, Z], [\text{parentOf}, Z, Y]], 2, S_3)$

S_3 = (\{X/i, Y/j\}, \rho_3)

or$^k_{\theta}([[\text{fatherOf}, i, Z], 1, S_3])$
Example Knowledge Base:
1. fatherOf(ABE, HOMER).
2. parentOf(HOMER, BART).
3. grandfatherOf(X, Y) :-
   fatherOf(X, Z),
   parentOf(Z, Y).

S_1 = (\emptyset, \rho_1)
S_2 = (\emptyset, \rho_2)
S_3 = \{X/i, Y/j\}, \rho_3

and_\theta([fatherOf, X, Z], [parentOf, Z, Y], 2, S_3)

substitute

unify(\theta, [fatherOf, ABE, HOMER], [fatherOf, i, Z], S_3)
unify(\theta, [parentOf, HOMER, BART], [fatherOf, i, Z], S_3)

S_{33} = \text{FAIL}
Example Knowledge Base:
1. fatherOf(ABE, HOMER).
2. parentOf(HOMER, BART).
3. grandfatherOf(X, Y) :-
   fatherOf(X, Z),
   parentOf(Z, Y).

\[ \text{unify}_\theta([\text{fatherOf}, \text{ABE, HOMER}], [s, i, j], (\emptyset, 1)) \]

\[ S_1 = (\emptyset, \rho_1) \]

\[ \text{or}_\theta([s, i, j], 2, (\emptyset, 1)) \]

\[ S_2 = (\emptyset, \rho_2) \]

\[ \text{unify}_\theta([\text{grandfatherOf}, X, Y], [s, i, j], (\emptyset, 1)) \]

\[ S_3 = (\{X/i, Y/j\}, \rho_3) \]

\[ \text{and}_\theta([\text{fatherOf}, X, Z], [\text{parentOf}, Z, Y], 2, S_3) \]

\[ \text{substitute} \]

\[ \text{unify}_\theta([\text{fatherOf}, ABE, HOMER], [\text{fatherOf}, i, Z], S_3) \]

\[ S_{31} = (\{X/i, Y/j, Z/HOMER\}, \rho_{31}) \]

\[ S_{33} = \text{FAIL} \]

\[ \text{and}_\theta([\text{parentOf}, Z, Y], 2, S_{31}) \]
Example Knowledge Base:
1. fatherOf(ABE, HOMER).
2. parentOf(HOMER, BART).
3. grandfatherOf(X, Y) :-
   fatherOf(X, Z),
   parentOf(Z, Y).

\[
\begin{align*}
K_\theta & (\text{fatherOf}(\text{abe}, \text{homer}), [s, i, j], [\emptyset, 1]) \\
K_\theta & (\text{parentOf}(\text{homer}, \text{bart}), [s, i, j], [\emptyset, 1]) \\
K_\theta & (\text{grandfatherOf}(X, Y), [s, i, j], [\emptyset, 1])
\end{align*}
\]
Example Knowledge Base:
1. father0f(ABE, HOMER).
2. parent0f(HOMER, BART).
3. grandfather0f(X, Y) :- father0f(X, Z),
   parent0f(Z, Y).

\[
\begin{align*}
\text{unify}_\theta([\text{father0f}, \text{ABE, HOMER}], [s, i, j], (\emptyset, 1)) & \quad S_1 = (\emptyset, \rho_1) \\
\text{unify}_\theta([\text{grandfather0f}, X, Y], [s, i, j], (\emptyset, 1)) & \quad S_2 = (\emptyset, \rho_2) \\
\text{unify}_\theta([\text{parent0f}, Z, Y], [s, i, j], (\emptyset, 1)) & \quad S_3 = ([X/i, Y/j], \rho_3)
\end{align*}
\]

and_\theta([\text{father0f}, X, Z], [\text{parent0f}, Z, Y], 2, S_3)

\[
\begin{align*}
\text{unify}_\theta([\text{father0f}, \text{ABE, HOMER}], [\text{father0f}, i, Z], S_3) & \quad S_{31} = ([X/i, Y/j, Z/HOMER], \rho_{31}) \\
\text{unify}_\theta([\text{parent0f}, \text{HOMER, BART}], [\text{father0f}, i, Z], S_3) & \quad S_{33} = \text{FAIL}
\end{align*}
\]

and_\theta([\text{parent0f}, Z, Y], 2, S_{31})

or_\theta([\text{parent0f}, \text{HOMER, j}], 1, S_{31})

\[
\begin{align*}
S_{311} = ([X/i, Y/j, Z/HOMER], \rho_{311}) & \quad S_{313} = \text{FAIL} \\
S_{312} = ([X/i, Y/j, Z/HOMER], \rho_{312})
\end{align*}
\]
Example Knowledge Base:
1. \(\text{fatherOf}(\text{ABE, HOMER})\).
2. \(\text{parentOf}(\text{HOMER, BART})\).
3. \(\text{grandfatherOf}(X, Y) :- \)
   \(\text{fatherOf}(X, Z), \text{parentOf}(Z, Y)\).

\[
\begin{align*}
\text{unify}_\theta([\text{fatherOf}, \text{ABE, HOMER}],[s, i, j], (\emptyset, 1)) &= S_1 = (\emptyset, \rho_1) \\
\text{unify}_\theta([\text{parentOf}, \text{HOMER, BART}],[\text{fatherOf}, i, Z], S_3) &= \text{FAIL} \\
\text{unify}_\theta([\text{parentOf}, Z, Y], S_3) &= S_3 = (\{X/i, Y/j, Z/homer\}, \rho_3) \\
\text{unify}_\theta([\text{grandfatherOf}, X, Y], S_3) &= S_3 = (\{X/i, Y/j, (\emptyset, 1)\}) \\
\text{unify}_\theta([\text{parentOf}, Z, Y], S_3) &= S_3 = (\{X/i, Y/j, (\emptyset, 1)\}) \\
\text{unify}_\theta([\text{fatherOf}, i, Z], S_3) &= S_31 = (\{X/i, Y/j, Z/homer\}, \rho_{31}) \\
\text{unify}_\theta([\text{fatherOf}, i, Z], S_3) &= S_33 = \text{FAIL} \\
\text{unify}_\theta([\text{parentOf}, Z, Y], S_3) &= S_32 = (\{X/i, Y/j, Z/bart\}, \rho_{32}) \\
\text{unify}_\theta([\text{parentOf}, Z, Y], S_3) &= S_33 = \text{FAIL} \\
\text{unify}_\theta([\text{fatherOf}, i, Z], S_3) &= S_311 = (\{X/i, Y/j, Z/homer\}, \rho_{311}) \\
\text{unify}_\theta([\text{parentOf}, Z, Y], S_3) &= S_313 = \text{FAIL} \\
\text{unify}_\theta([\text{fatherOf}, i, Z], S_3) &= S_312 = (\{X/i, Y/j, Z/homer\}, \rho_{312})
\end{align*}
\]
Example Knowledge Base:
1. fatherOf(ABE, HOMER).
2. parentOf(HOMER, BART).
3. grandfatherOf(X, Y) :-
   fatherOf(X, Z),
   parentOf(Z, Y).

\[
\begin{align*}
&\text{or}_\theta([s, i, j], 2, (\emptyset, 1)) \quad 1. \\
&\text{unify}_\theta([\text{fatherOf}, \text{ABE, HOMER}], [s, i, j], (\emptyset, 1)) \quad \cdots \\
&S_1 = (\emptyset, \rho_1) \quad \cdots \\
&\text{unify}_\theta([\text{grandfatherOf}, X, Y], [s, i, j], (\emptyset, 1)) \\
&S_3 = ([X/i, Y/j], \rho_3)
\end{align*}
\]

\[
\begin{align*}
&\text{and}_\theta([\text{fatherOf}, X, Z], [\text{parentOf}, Z, Y], 2, S_3) \\
&\text{substitute}
\end{align*}
\]

\[
\begin{align*}
&\text{or}_\theta([\text{fatherOf}, i, Z], 1, S_3) \quad 1. \\
&\text{unify}_\theta([\text{fatherOf}, \text{ABE, HOMER}], [\text{fatherOf}, i, Z], S_3) \quad \cdots \\
&S_31 = ([X/i, Y/j, Z/HOMER], \rho_{31}) \quad \cdots \\
&S_33 = \text{FAIL} \quad \cdots \\
&\text{unify}_\theta([\text{parentOf}, \text{HOMER, BART}], [\text{fatherOf}, i, Z], S_3) \\
&S_32 = ([X/i, Y/j, Z/BART], \rho_{32})
\end{align*}
\]

\[
\begin{align*}
&\text{and}_\theta([\text{parentOf}, Z, Y], 2, S_31) \\
&\text{substitute}
\end{align*}
\]

\[
\begin{align*}
&\text{or}_\theta([\text{parentOf}, \text{HOMER}, j], 1, S_31) \quad 1. \\
&\cdots \cdots \\
&S_311 = ([X/i, Y/j, Z/HOMER], \rho_{311}) \quad \cdots \cdots \\
&S_313 = \text{FAIL} \\
&\text{substitute}
\end{align*}
\]

\[
\begin{align*}
&\text{or}_\theta([\text{parentOf}, \text{BART}, j], 1, S_32) \\
&\text{substitute}
\end{align*}
\]
Example Knowledge Base:
1. fatherOf(ABE, HOMER).
2. parentOf(HOMER, BART).
3. grandfatherOf(X, Y) :=
   fatherOf(X, Z),
   parentOf(Z, Y).

\[
\text{unify}_\vartheta([\text{fatherOf}, \text{ABE}, \text{HOMER}], [s, i, j], (\varnothing, 1)) \\
S_1 = (\varnothing, \rho_1)
\]

\[
\text{unify}_\vartheta([\text{grandfatherOf}, X, Y], [s, i, j], (\varnothing, 1)) \\
S_2 = (\varnothing, \rho_2)
\]

\[
\text{unify}_\vartheta([\text{fatherOf}, \text{ABE}, \text{HOMER}], [s, i, j], (\varnothing, 1)) \\
S_3 = ([X/i, Y/j], \rho_3)
\]

\[
\text{and}_\vartheta([\text{fatherOf}, X, Z], [\text{parentOf}, Z, Y]), 2, S_3)
\]

\[
\text{or}_\vartheta([\text{fatherOf}, i, Z], 1, S_3)
\]

\[
\text{unify}_\vartheta([\text{fatherOf}, \text{ABE}, \text{HOMER}], [\text{fatherOf}, i, Z], S_3) \\
S_{31} = ([X/i, Y/j, Z/\text{HOMER}], \rho_{31})
\]

\[
\text{unify}_\vartheta([\text{parentOf}, \text{HOMER}, \text{BART}], [\text{fatherOf}, i, Z], S_3) \\
S_{33} = \text{FAIL}
\]

\[
\text{and}_\vartheta([\text{parentOf}, Z, Y], 2, S_{31})
\]

\[
\text{or}_\vartheta([\text{parentOf}, \text{HOMER}, j], 1, S_{31})
\]

\[
S_{311} = ([X/i, Y/j, Z/\text{HOMER}], \rho_{311}) \\
S_{312} = ([X/i, Y/j, Z/\text{HOMER}], \rho_{312}) \\S_{313} = \text{FAIL}
\]

\[
\text{or}_\vartheta([\text{parentOf}, \text{BART}, j], 1, S_{32})
\]

\[
S_{322} = ([X/i, Y/j, Z/\text{BART}], \rho_{322})
\]

\[
\text{and}_\vartheta([\text{parentOf}, \text{Z, Y}], 2, S_{32})
\]

\[
\text{or}_\vartheta([\text{parentOf}, \text{Z, Y}], 2, S_{32})
\]

\[
S_{323} = \text{FAIL} \\
S_{321} = ([X/i, Y/j, Z/\text{BART}], \rho_{321})
\]
Example Knowledge Base:
1. fatherOf(ABE, HOMER).
2. parentOf(HOMER, BART).
3. grandfatherOf(X, Y) :-
   fatherOf(X, Z),
   parentOf(Z, Y).

unify\(\theta([\text{fatherOf, ABE, HOMER}], [s, i, j], (\emptyset, 1))\)  
\(S_1 = (\emptyset, \rho_1)\)  
\(S_2 = (\emptyset, \rho_2)\)  
\(S_3 = ([X/i, Y/j], \rho_3)\)

\(\text{and}_\theta([\text{fatherOf, X, Z}], [\text{parentOf, Z, Y}], 2, S_3)\)

unify\(\theta([\text{fatherOf, ABE, HOMER}], [\text{fatherOf, i, Z}], S_3)\)
\(S_{31} = ([X/i, Y/j, Z/HOMER], \rho_{31})\)
\(S_{33} = \text{FAIL}\)
\(S_{32} = ([X/i, Y/j, Z/BART], \rho_{32})\)

\(\text{and}_\theta([\text{parentOf, Z, Y}], 2, S_{31})\)

\(\text{or}_\theta([\text{parentOf, HOMER, BART}], [\text{fatherOf, i, Z}, S_3])\)

unify\(\theta([\text{parentOf, HOMER, BART}], [\text{parentOf, Z, Y}], 2, S_{32})\)

\(\text{or}_\theta([\text{parentOf, BART, j}], 1, S_{32})\)

\(S_{311} = ([X/i, Y/j, Z/HOMER], \rho_{311})\)
\(S_{313} = \text{FAIL}\)
\(S_{312} = ([X/i, Y/j, Z/HOMER], \rho_{312})\)

\(S_{321} = ([X/i, Y/j, Z/BART], \rho_{321})\)
\(S_{322} = ([X/i, Y/j, Z/BART], \rho_{322})\)
\(S_{323} = \text{FAIL}\)
Batch Proving

- \( A \in \mathbb{R}^{N \times k} \) matrix of \( N \) subsymbolic representations
- \( B \in \mathbb{R}^{M \times k} \) matrix of \( M \) other subsymbolic representations

\[
\exp \left( -\sqrt{\left( \sum_{i=1}^{k} A_{1i}^2 \right) 1^T_M} + \left( 1_N \left[ \sum_{i=1}^{k} B_{1i}^2 \right] ^T \right) - 2AB^T \right) \in \mathbb{R}^{N \times M}
\]

where \( 1_N \) and \( 1_M \) are vectors of \( N \) and \( M \) ones respectively, and the square root is taken element-wise.
Calculation on GPU

\[ Q \]

\[ \text{dad0f} \]

\[ \text{parent0f} \]

\[ \text{ABE} \]

\[ \text{HOMER} \]

\[ \text{Q} \]
Calculation on GPU
Calculation on GPU

Q

dad0f
parent0f

ABE
HOMER

unify

father0f
parent0f
grandma0f

unify

ABE
HOMER
MONA

HOMER
BART
LISA
Calculation on GPU

father0f
parent0f
grandma0f

father0f
parent0f
grandma0f

unify
unify

HOMER
ABE
Mona

HOMER
ABE
Mona

HOMER
BART
LISA

HOMER
BART
LISA

Q

Q /

unify (symbolic)
Calculation on GPU

fatherOf
parentOf
grandmaOf
dad0f
parent0f

unify

ABE
HOMER

unify

HOMER
BART
LISA

unify (symbolic)

Q /
Neural Inductive Logic Programming

1. \( \text{\textit{vfatherOf}}(\text{\textit{v}_{\text{ABE}}, \text{\textit{v}_{\text{HOMER}}}}). \)
2. \( \text{\textit{vparentOf}}(\text{\textit{v}_{\text{HOMER}}, \text{\textit{v}_{\text{LISA}}}}). \)
3. \( \text{\textit{vparentOf}}(\text{\textit{v}_{\text{HOMER}}, \text{\textit{v}_{\text{BART}}}}). \)
4. \( \text{\textit{vgrandpaOf}}(\text{\textit{v}_{\text{ABE}}, \text{\textit{v}_{\text{LISA}}}}). \)
5. \( \text{\textit{vgrandfatherOf}}(\text{\textit{v}_{\text{ABE}}, \text{\textit{v}_{\text{MAGGIE}}}}). \)
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. $\theta_1(X_1, Y_1) : \neg \theta_2(X_1, Z_1), \theta_3(Z_1, Y_1)$.
7. $\theta_4(X_2, Y_2) : \neg \theta_5(X_2, Y_2)$. 
Neural Inductive Logic Programming

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

6. $\theta_1(X_1, Y_1) :\neg \theta_2(X_1, Z_1), \theta_3(Z_1, Y_1)$.
7. $\theta_4(X_2, Y_2) :\neg \theta_5(X_2, Y_2)$.

Decoding Induced Rules

- Find closest representations of known predicate
Neural Inductive Logic Programming

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

6. $\theta_1(X_1, Y_1) : \leftarrow \theta_2(X_1, Z_1), \theta_3(Z_1, Y_1)$.
7. $\theta_4(X_2, Y_2) : \leftarrow \theta_5(X_2, Y_2)$.

Decoding Induced Rules

- Find closest representations of known predicate
- Take minimum RBF similarity as rule confidence
Neural Inductive Logic Programming

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. $\theta_1(X_1, Y_1) :- \theta_2(X_1, Z_1), \theta_3(Z_1, Y_1)$.
7. $\theta_4(X_2, Y_2) :- \theta_5(X_2, Y_2)$.

Decoding Induced Rules

- Find closest representations of known predicate
- Take minimum RBF similarity as rule confidence
- Rule confidence is an upper bound on the proof success that can be achieved when applying the rule
Experiments

Benchmark Knowledge Bases: **Kinship**, **Nations**, **UMLS** (Kok and Domingos, 2007), and **Countries** (Bouchard et al., 2015)
Experiments

Benchmark Knowledge Bases: **Kinship, Nations, UMLS** (Kok and Domingos, 2007), and **Countries** (Bouchard et al., 2015)
Experiments

Benchmark Knowledge Bases: **Kinship**, **Nations**, **UMLS** (Kok and Domingos, 2007), and **Countries** (Bouchard et al., 2015)
Experiments

Benchmark Knowledge Bases: **Kinship, Nations, UMLS** (Kok and Domingos, 2007), and **Countries** (Bouchard et al., 2015)
Details

- Models implemented in TensorFlow
Details

- Models implemented in TensorFlow
  - **ComplEx** Neural link prediction model by Trouillon et al. (2016)
Details

- Models implemented in TensorFlow
  - **ComplEx**: Neural link prediction model by Trouillon et al. (2016)
  - **NTP**: End-to-end differentiable prover
Details

- Models implemented in TensorFlow
  - **ComplEx** Neural link prediction model by Trouillon et al. (2016)
  - **NTP** End-to-end differentiable prover
  - **NTPλ** Prover trained with ComplEx as auxiliary loss
- Models implemented in TensorFlow
  
  **ComplEx** Neural link prediction model by Trouillon et al. (2016)
  
  **NTP** End-to-end differentiable prover
  
  **NTPλ** Prover trained with ComplEx as auxiliary loss

- Rule Templates:
  
  **Kinship, Nations & UMLS**
  
  20 \( \#1(X, Y) :\sim \#2(X, Y). \)
  
  20 \( \#1(X, Y) :\sim \#2(Y, X). \)
  
  20 \( \#1(X, Y) :\sim \#2(X, Z), \#3(Z, Y). \)

  **Countries S1**
  
  3 \( \#1(X, Y) :\sim \#1(Y, X). \)
  
  3 \( \#1(X, Y) :\sim \#2(X, Z), \#2(Z, Y). \)

  **Countries S2**
  
  3 \( \#1(X, Y) :\sim \#2(X, Z), \#3(Z, Y). \)

  **Countries S3**
  
  3 \( \#1(X, Y) :\sim \#2(X, Z), \#3(Z, W), \#4(W, Y). \)
## Results

<table>
<thead>
<tr>
<th>Corpus</th>
<th>Metric</th>
<th>Model</th>
<th>ComplEx</th>
<th>NTP</th>
<th>NTPλ</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Countries</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td>AUC-PR</td>
<td>99.37 ± 0.4</td>
<td>90.83 ± 15.4</td>
<td><strong>100.00 ± 0.0</strong></td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>AUC-PR</td>
<td>87.95 ± 2.8</td>
<td>87.40 ± 11.7</td>
<td><strong>93.04 ± 0.4</strong></td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>AUC-PR</td>
<td>48.44 ± 6.3</td>
<td>56.68 ± 17.6</td>
<td><strong>77.26 ± 17.0</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Kinship</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MRR</td>
<td>0.46</td>
<td>0.36</td>
<td><strong>0.48</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HITS@1</td>
<td>0.34</td>
<td>0.24</td>
<td><strong>0.39</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HITS@3</td>
<td>0.49</td>
<td>0.40</td>
<td><strong>0.47</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HITS@10</td>
<td>0.74</td>
<td>0.60</td>
<td><strong>0.71</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Nations</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MRR</td>
<td>0.60</td>
<td><strong>0.63</strong></td>
<td>0.62</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HITS@1</td>
<td>0.46</td>
<td><strong>0.48</strong></td>
<td>0.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HITS@3</td>
<td>0.67</td>
<td>0.69</td>
<td><strong>0.72</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HITS@10</td>
<td>0.97</td>
<td>0.98</td>
<td><strong>0.99</strong></td>
<td></td>
</tr>
<tr>
<td><strong>UMLS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MRR</td>
<td>0.58</td>
<td>0.57</td>
<td><strong>0.60</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HITS@1</td>
<td>0.47</td>
<td>0.47</td>
<td><strong>0.51</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HITS@3</td>
<td>0.63</td>
<td>0.60</td>
<td><strong>0.64</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HITS@10</td>
<td>0.80</td>
<td>0.79</td>
<td><strong>0.81</strong></td>
<td></td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th>Corpus</th>
<th>Examples of induced rules and their confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Countries</td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td>0.90 <code>locatedIn(X,Y) :- locatedIn(X,Z), locatedIn(Z,Y)</code>.</td>
</tr>
<tr>
<td>S2</td>
<td>0.63 <code>locatedIn(X,Y) :- neighborOf(X,Z), locatedIn(Z,Y)</code>.</td>
</tr>
<tr>
<td>S3</td>
<td>0.32 <code>locatedIn(X,Y) :- neighborOf(X,Z), neighborOf(Z,W), locatedIn(W,Y)</code>.</td>
</tr>
<tr>
<td>Nations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.68 <code>blockpositionindex(X,Y) :- blockpositionindex(Y,X)</code>.</td>
</tr>
<tr>
<td></td>
<td>0.46 <code>expeldiplomats(X,Y) :- negativebehavior(X,Y)</code>.</td>
</tr>
<tr>
<td></td>
<td>0.38 <code>negativecomm(X,Y) :- commonbloc0(X,Y)</code>.</td>
</tr>
<tr>
<td></td>
<td>0.38 <code>intergovorgs3(X,Y) :- intergovorgs(Y,X)</code>.</td>
</tr>
<tr>
<td>UMLS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.88 <code>interacts_with(X,Y) :- interacts_with(X,Z), interacts_with(Z,Y)</code>.</td>
</tr>
<tr>
<td></td>
<td>0.77 <code>isa(X,Y) :- isa(X,Z), isa(Z,Y)</code>.</td>
</tr>
<tr>
<td></td>
<td>0.71 <code>derivative_of(X,Y) :- derivative_of(X,Z), derivative_of(Z,Y)</code>.</td>
</tr>
</tbody>
</table>
End-to-end Differentiable Planning

*work-in-progress*
DQN

\[ \text{DQN} = \theta = \text{\text{target}} \left( r + \gamma \text{discount} \max_{a'} Q(s', a', \theta - ) - Q(s, a, \theta) \right) \]

Mnih et al. (2015)
$O_t$ \quad \text{encode} \quad Z_t \quad \text{evaluate} \quad Q$

$L(\theta) = \left( \text{target} \leftarrow \text{reward} + \gamma \max \limits_{a'} \text{discount} \leftarrow \text{Q}(s', a', \theta^-) - \text{Q}(s, a, \theta) \right)^2$

Mnih et al. (2015)
DQN

\[ L(\theta) = \left( r + \gamma \max_{a'} Q(s', a', \theta^-) - Q(s, a, \theta) \right)^2 \]

Mnih et al. (2015)
Tree Planning

\[ Q(o_t, a_3) \]
\[ Q(o_t, a_2) \]
\[ Q(o_t, a_1) \]
Tree Planning

\[ Q(o_t, a_3) \]
\[ Q(o_t, a_2) \]
\[ Q(o_t, a_1) \]

Tim Rocktäschel
GPU-accelerated End-to-end Differentiable Planning and Reasoning
Tree Planning

Tree Transitioning

\[ Q(o_t, a_3) \]

\[ \max \]

\[ Q(o_t, a_2) \]

\[ \max \]

\[ Q(o_t, a_1) \]

\[ \max \]

Tim Rocktäschel  
GPU-accelerated End-to-end Differentiable Planning and Reasoning 36/39
Tree Planning

Tree Transitioning

\[ Q(o_t, a_3) \]

\[ Q(o_t, a_2) \]

\[ Q(o_t, a_1) \]

Tim Rocktäschel  GPU-accelerated End-to-end Differentiable Planning and Reasoning 36/39
Tree Planning

Tree Transitioning

\[ Q(o_t, a_1) \]
\[ Q(o_t, a_2) \]
\[ Q(o_t, a_3) \]
Tree Planning

Tree Transitioning

Value Prediction

\[ Q(o_t, a_1) \]

\[ Q(o_t, a_2) \]

\[ Q(o_t, a_3) \]
Results

Enduro

Average Reward over 100 Episodes

Steps

DQN     938
TreeQN 1028

Tim Rocktäschel
GPU-accelerated End-to-end Differentiable Planning and Reasoning
Results

Average Reward over 100 Episodes

Steps

DQN 2497
TreeQN 3467
Results

MsPacman

Average Reward over 100 Episodes

DQN    3854
TreeQN 4670

Steps

Tim Rocktäschel  GPU-accelerated End-to-end Differentiable Planning and Reasoning
Summary

- Prolog’s backward chaining as recipe for recursively constructing a neural network to prove facts in a knowledge base

Proof success differentiable w.r.t. subsymbolic representations

Can learn vector representations of symbols and induce interpretable rules of predefined structure

Various GPU optimizations: batch proving, tree pruning etc.

Outperform neural link prediction model on benchmark knowledge bases

Future research:
- Scale to larger knowledge bases
- Connect to RNNs for natural language statements
- Proving of mathematical theorems
- Visual reasoning

Encouraging preliminary results using tree planning for Atari
Summary

- Prolog’s backward chaining as recipe for recursively constructing a neural network to prove facts in a knowledge base
- Proof success differentiable w.r.t. subsymbolic representations

Future research:
- Scale to larger knowledge bases
- Connect to RNNs for natural language statements
- Proving of mathematical theorems
- Visual reasoning

Encouraging preliminary results using tree planning for Atari
Summary

- Prolog’s backward chaining as recipe for recursively constructing a neural network to prove facts in a knowledge base
- Proof success differentiable w.r.t. subsymbolic representations
- Can learn vector representations of symbols and induce interpretable rules of predefined structure

Various GPU optimizations: batch proving, tree pruning etc.

Outperform neural link prediction model on benchmark knowledge bases

Future research:
- Scale to larger knowledge bases
- Connect to RNNs for natural language statements
- Proving of mathematical theorems
- Visual reasoning

Encouraging preliminary results using tree planning for Atari
Summary

- Prolog’s backward chaining as recipe for recursively constructing a neural network to prove facts in a knowledge base
- Proof success differentiable w.r.t. subsymbolic representations
- Can learn vector representations of symbols and induce interpretable rules of predefined structure
- Various GPU optimizations: batch proving, tree pruning etc.
Summary

- Prolog’s backward chaining as recipe for recursively constructing a neural network to prove facts in a knowledge base
- Proof success differentiable w.r.t. subsymbolic representations
- Can learn vector representations of symbols and induce interpretable rules of predefined structure
- Various GPU optimizations: batch proving, tree pruning etc.
- Outperform neural link prediction model on benchmark knowledge bases
Summary

- Prolog’s backward chaining as recipe for recursively constructing a neural network to prove facts in a knowledge base
- Proof success differentiable w.r.t. subsymbolic representations
- Can learn vector representations of symbols and induce interpretable rules of predefined structure
- Various GPU optimizations: batch proving, tree pruning etc.
- Outperform neural link prediction model on benchmark knowledge bases
- Future research:
Summary

- Prolog’s backward chaining as recipe for recursively constructing a neural network to prove facts in a knowledge base
- Proof success differentiable w.r.t. subsymbolic representations
- Can learn vector representations of symbols and induce interpretable rules of predefined structure
- Various GPU optimizations: batch proving, tree pruning etc.
- Outperform neural link prediction model on benchmark knowledge bases
- Future research:
  - Scale to larger knowledge bases
Summary

- Prolog’s backward chaining as recipe for recursively constructing a neural network to prove facts in a knowledge base
- Proof success differentiable w.r.t. subsymbolic representations
- Can learn vector representations of symbols and induce interpretable rules of predefined structure
- Various GPU optimizations: batch proving, tree pruning etc.
- Outperform neural link prediction model on benchmark knowledge bases
- Future research:
  - **Scale** to larger knowledge bases
  - Connect to RNNs for natural language statements
Summary

- Prolog’s backward chaining as recipe for recursively constructing a neural network to prove facts in a knowledge base
- Proof success differentiable w.r.t. subsymbolic representations
- Can learn vector representations of symbols and induce interpretable rules of predefined structure
- Various GPU optimizations: batch proving, tree pruning etc.
- Outperform neural link prediction model on benchmark knowledge bases
- Future research:
  - **Scale** to larger knowledge bases
  - Connect to RNNs for natural language statements
  - Proving of mathematical theorems
Summary

- Prolog’s backward chaining as recipe for recursively constructing a neural network to prove facts in a knowledge base
- Proof success differentiable w.r.t. subsymbolic representations
- Can learn vector representations of symbols and induce interpretable rules of predefined structure
- Various GPU optimizations: batch proving, tree pruning etc.
- Outperform neural link prediction model on benchmark knowledge bases
- Future research:
  - **Scale** to larger knowledge bases
  - Connect to RNNs for natural language statements
  - Proving of mathematical theorems
  - Visual reasoning
Summary

- Prolog’s backward chaining as recipe for recursively constructing a neural network to prove facts in a knowledge base
- Proof success differentiable w.r.t. subsymbolic representations
- Can learn vector representations of symbols and induce interpretable rules of predefined structure
- Various GPU optimizations: batch proving, tree pruning etc.
- Outperform neural link prediction model on benchmark knowledge bases
- Future research:
  - **Scale** to larger knowledge bases
  - Connect to RNNs for natural language statements
  - Proving of mathematical theorems
  - Visual reasoning
- Encouraging preliminary results using tree planning for Atari
Thank you!

http://rockt.github.com
tim.rocktaschel@cs.ox.ac.uk
Twitter: @_rockt
References


