Neuro-Symbolic Models in Deep Learning

2020.6.10

Seung-Hoon Na
Jeonbuk National University

Contents

Reference

- The symbol grounding problem [Harnad ’90]
- Knowledge-based artificial neural networks [Towell & Shavlik ‘94]
- Harnessing Deep Neural Networks with Logic Rules [Hu et al ‘16]
- TensorLog: Deep Learning Meets Probabilistic DBs [Cohen et al ’16]
- Neural Programmer-Interpreters [Reed & Freitas ‘16]
- Neural theorem prover [Rocktäschel & Riedel ‘16]
- Neural Symbolic Machines [Liang et al ‘16]
- Harnessing Deep Neural Networks with Logic Rules [Hu et al ‘16]
- Towards Deep Symbolic Reinforcement Learning [Garnelo et al ‘16]
- Learning Explanatory Rules from Noisy Data [Evans & Grefenstette ‘17]
- Logic Tensor Networks for Semantic Image Interpretation [Denodello et al ‘17]
- The consciousness prior [Bengio ‘17]
- An Empirical Evaluation of Rule Extraction from Recurrent Neural Networks [Wang et al ’18]
- Neural-Symbolic Computing: An Effective Methodology for Principled Integration of Machine Learning and Reasoning [Garcez et a’ 19]
- Scalable Neural Methods for Reasoning With a Symbolic Knowledge Base [Cohen et al ’20]
- Neural Query Language: A Knowledge Base Query Language for Tensorflow [Cohen et al ‘19]
- Greedy NTPs [Minervini et al ’19]

Neural-Symbolic Models

- Integrating two most fundamental cognitive abilities: Learning & reasoning [Valiant]
  - 1) Learning: The ability to learn from the environment
  - 2) Reasoning: The ability to reason from what has been learned
- Neural-symbolic computing [Garcez et al ‘19]
  - Aims at reconciling the dominating symbolic and connectionist paradigms of AI under a principled foundation
  - Knowledge is represented in symbolic form
  - Learning and reasoning are computed by a neural network

➔ Make Interpretability & explainability of AI systems enriched

https://arxiv.org/pdf/1905.06088.pdf

Neural-Symbolic Models

- Neural-symbolic computing [Garcez et al ‘19]
  - Neural learning and inference under uncertainty
    - may address the brittleness of symbolic systems
  - Symbolism provides additional knowledge for learning
    - May ameliorate neural network’s well-known catastrophic forgetting or difficulty with extrapolating
  - The integration of neural models with logic-based symbolic models provides an AI system
    - Capable of bridging lower-level information processing (for perception and pattern recognition)
    - Higher-level abstract knowledge (for reasoning and explanation)

Neural-symbolic models

- Neural symbolic computing: Issues [Garcez et al '19]
  - Representation
    - Knowledge representation in NN
      - Propositional logic, first-order logic, tensorisation
  - Learning
    - Inductive logic programming
      - ∂ILP [Evans and Grenfenstette '18]
    - Horizontal hybrid learning
      - Combining logic rules/formula with data during learning, also using the data to fine-tune knowledge
      - E.g.) Self-transfer with symbolic knowledge distillation [Hu et al '16]
    - Vertical hybrid learning
      - Low-level: neural model
      - High-level: symbolic knowledge
      - E.g.) logic tensor network [Donadello et al '17]

Neural-symbolic models

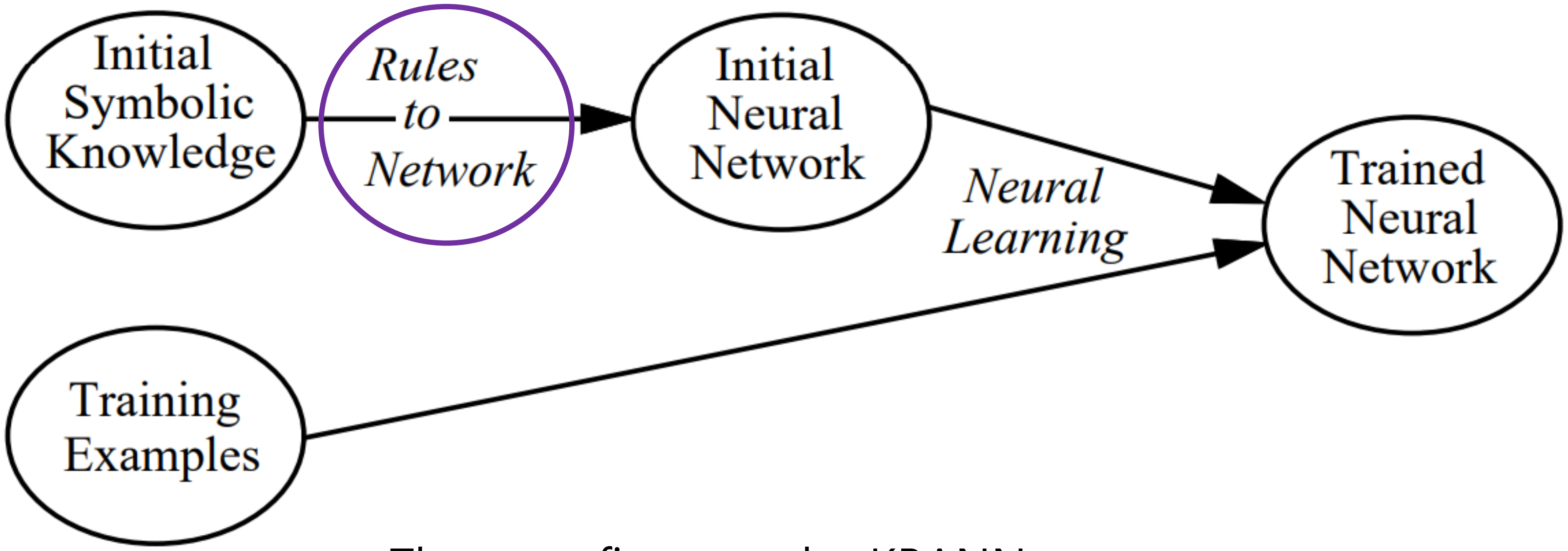
- Neural symbolic computing: Issues [Garcez et al ‘19]
  - Reasoning
    - Forward /backward chaining
    - Neural theorem prover [Rocktaschel et al ’16]
    - Neural logic machine [Dong et al ‘19]
  - Extraction
    - Explainable AI
- Neural program induction
  - Neural Programmer-Interpreters [Reed & Freitas ‘16]

KBANN [Towell & Shavlik ‘94]

- The need for hybrid systems
  - 1. Hand-built classifiers: Non-learning systems
    - Assume that domain theory is complete and correct; but, for most real-world tasks, completeness and correctness are extremely difficult
    - Domain theories can be intractable to use and difficult to modify
  - 2. Empirical learning
    - An unbounded number of features can be used to describe any object
    - Feature construction is a difficult, error-prone, enterprise
    - Uncommon cases may be very difficult to correctly handle
  - 3. ANN
    - Training times are lengthy
    - The initial parameters of the network can greatly affect how well concepts are learned
    - There is not yet a problem-independent way to choose a good network topology
    - The lack interpretability

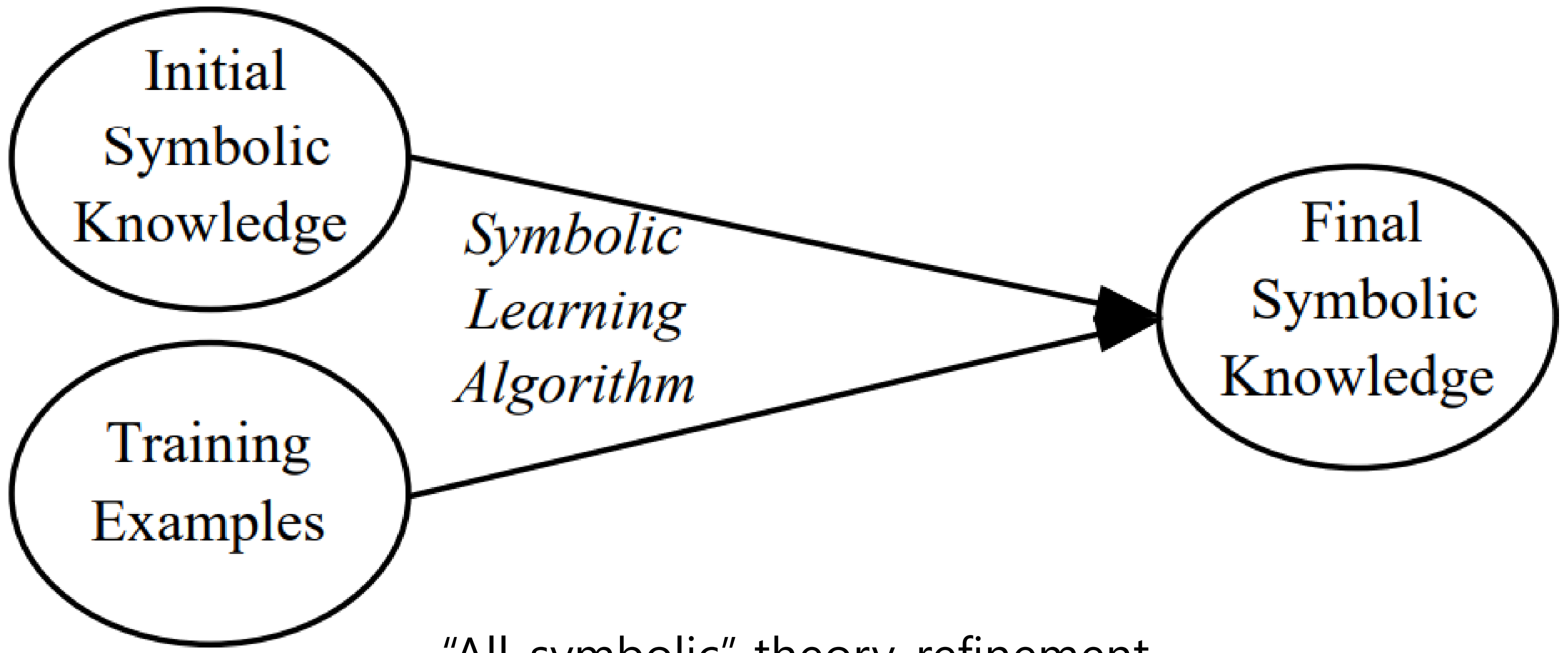
KBANN [Towell & Shavlik '94]

- Hybrid Learning systems
  - Use both hand-constructed rules and classified examples during learning



Theory-refinement by KBANN

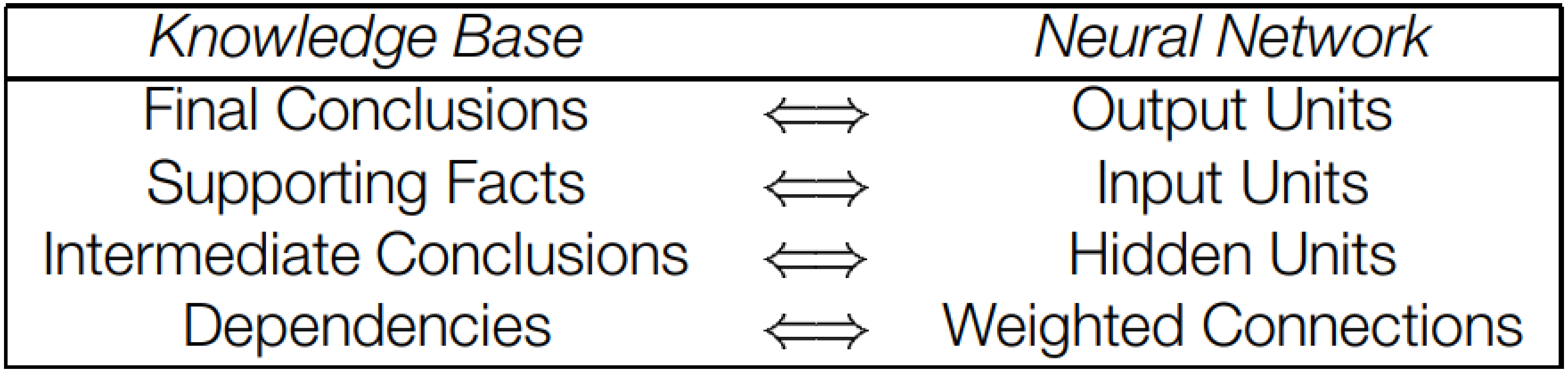
KBANN [Towell & Shavlik '94]



"All-symbolic" theory-refinement

KBANN [Towell & Shavlik ‘94]

- Correspondences between knowledge-bases and neural networks

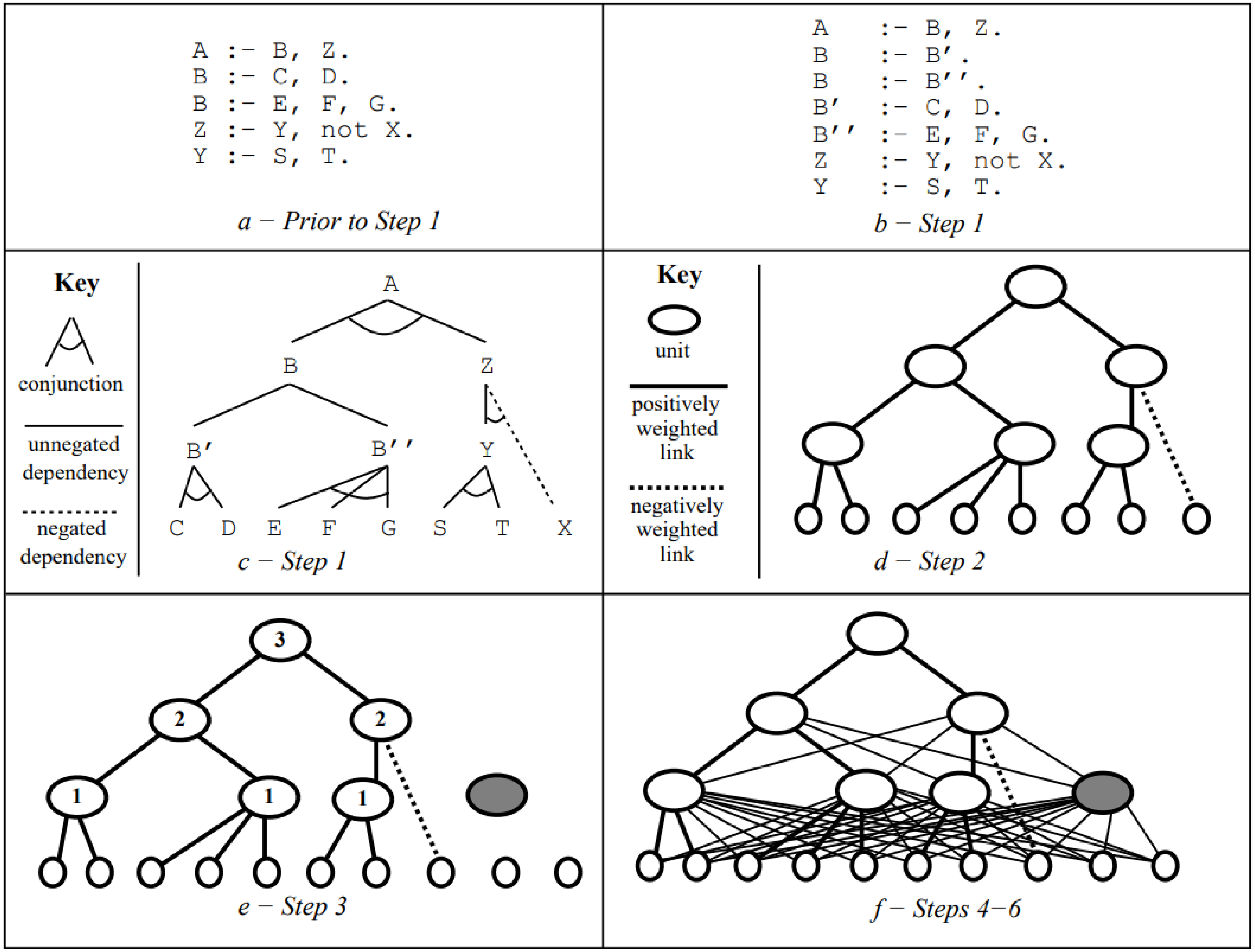


Knowledge Base		*Neural Network*
Final Conclusions	$\Longleftrightarrow$	Output Units
Supporting Facts	$\Longleftrightarrow$	Input Units
Intermediate Conclusions	$\Longleftrightarrow$	Hidden Units
Dependencies	$\Longleftrightarrow$	Weighted Connections

KBANN [Towell & Shavlik ‘94]

- The rules-to-networks algorithm of KBANN
  - 1. **Rewriting**: Rewrite rules so that disjuncts are expressed as a set of rules that each have only one antecedent.
  - 2. **Mapping**: Directly map the rule structure into a neural network.
  - 3. **Numbering**: Label units in the KBANN-net according to their “level.”
  - 4. **Adding hidden units**: Add hidden units to the network at user-specified levels (optional).
  - 5. **Adding input units**: Add units for known input features that are not referenced in the rules.
  - 6. **Adding links**: Add links not specified by translation between all units in topologically-contiguous levels.
  - 7. **Perturbing**: Perturb the network by adding near-zero random numbers to all link weights and biases.

KBANN [Towell & Shavlik '94]



```
A :- B, Z.
B :- C, D.
B :- E, F, G.
Z :- Y, not X.
Y :- S, T.
```

a – Prior to Step 1

```
A    :- B, Z.
B    :- B′.
B    :- B′′.
B′   :- C, D.
B′′  :- E, F, G.
Z    :- Y, not X.
Y    :- S, T.
```

b – Step 1

c – Step 1

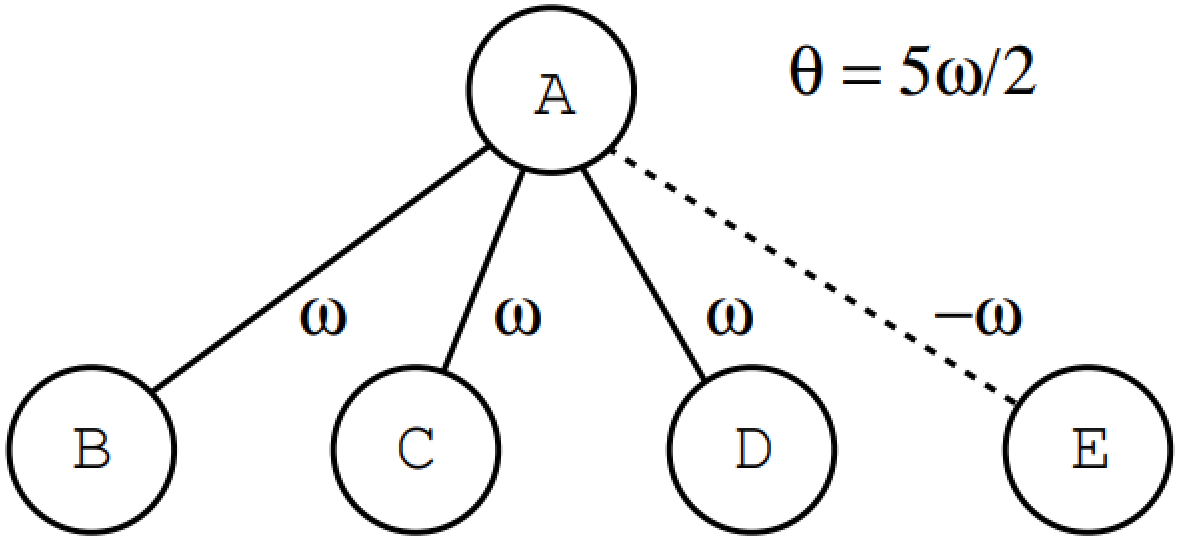
d – Step 2

e – Step 3

f – Steps 4–6

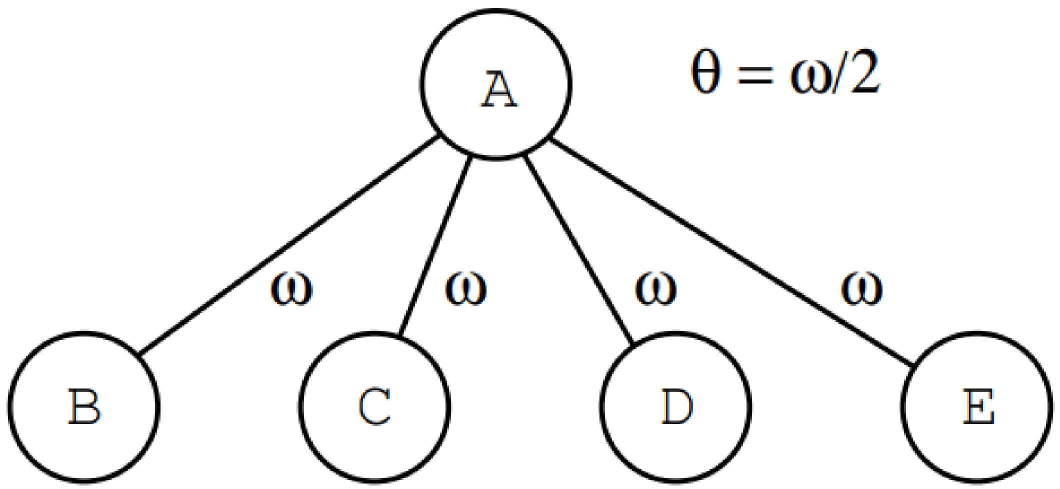
- Translation of a conjunctive rule into a KBANN-net
  - Weights of all links to positive (i.e., unnegated) antecedents: ω
  - Weights of all links to negated antecedents: $-\omega$
  - Weight of the bias: $(P - \frac{1}{2})\omega$

```
A :- B, C, D, not(E).
```



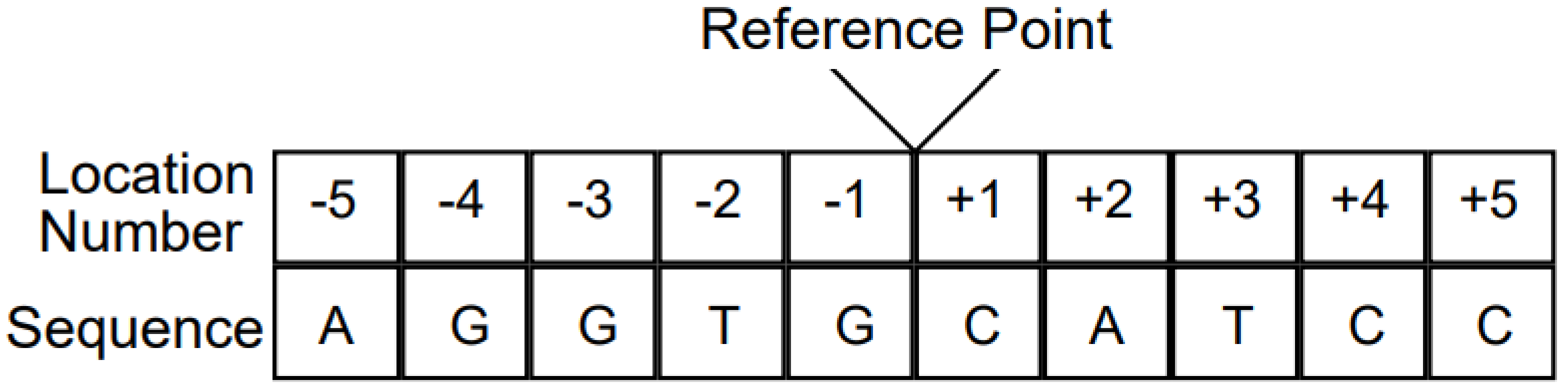
- Translation of a conjunctive rule into a KBANN-net
  - The same as the case of conjunctive rules, with two exceptions:
  - 1) KBANN rewrites disjuncts as multiple rules with the same consequent in step 1 of the rules-to-network algorithm ➔ as independent rules
  - 2) The bias in the unit encoding the consequent: $\omega/2$

```
A:-B.   A:-C.   A:-D.   A:-E.
```



KBANN [Towell & Shavlik ‘94]

- Task1: Promoter recognition
    - Promoters: short DNA sequences that precede the beginnings of genes
    - Task: Given a sequence of 57 consecutive DNA nucleotides, the reference point for promoter recognition is the site at which gene transcription begins (if the example is a promoter). The reference point is located seven nucleotides from the right of the 57-long sequence



Location Number	-5	-4	-3	-2	-1	+1	+2	+3	+4	+5
Sequence	A	G	G	T	G	C	A	T	C	C

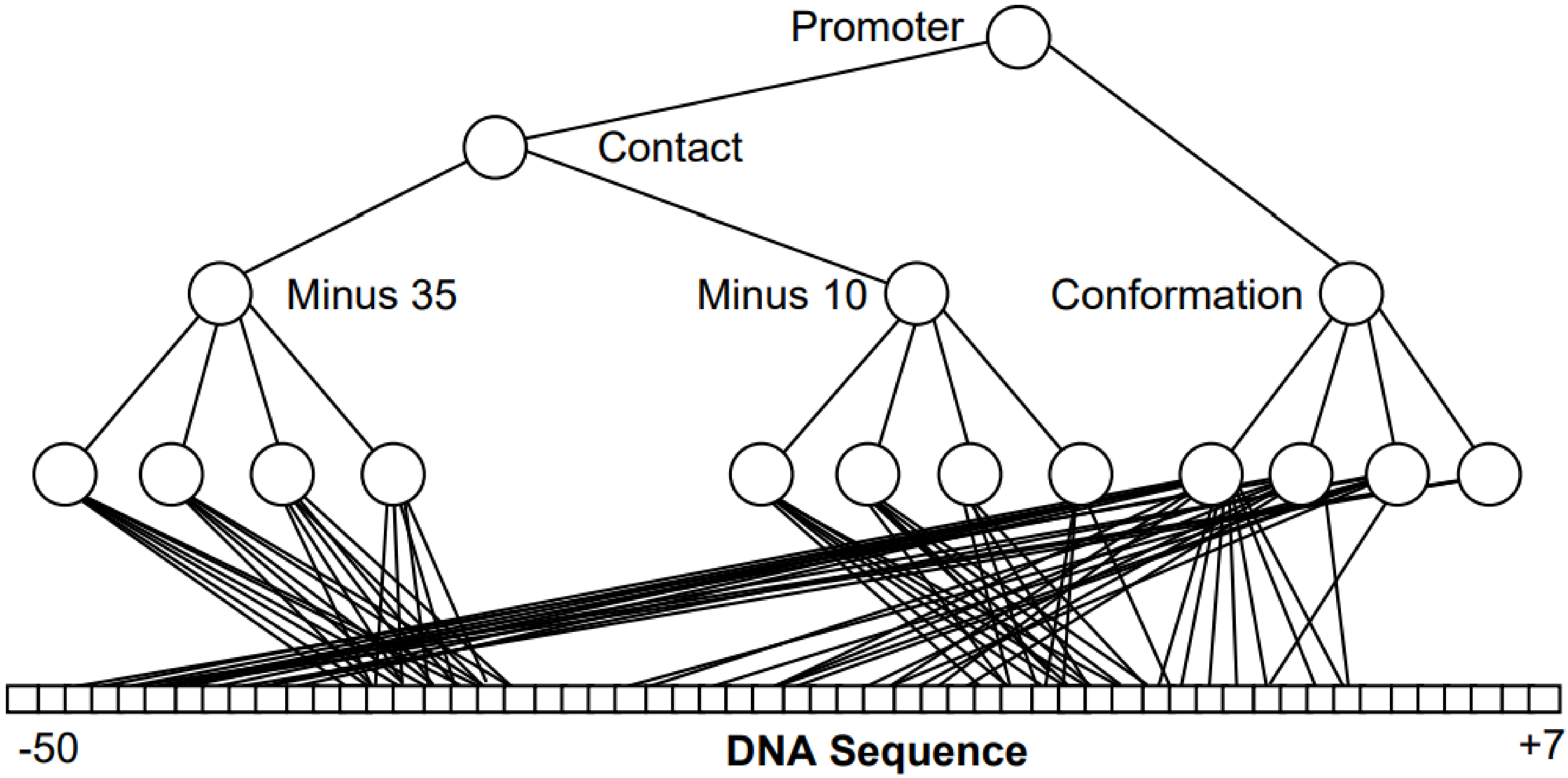
KBANN [Towell & Shavlik ‘94]

- Initial rules for promoter recognition

```
promoter   :- contact, conformation.
contact    :- minus-35, minus-10.

minus-35 :- @-37 ‘CTTGAC’.                     minus-10 :- @-13  ‘TA*A*T’.
minus-35 :- @-36  ‘TTGACA’.                    minus-10 :- @-12   ‘TA***T’.
minus-35 :- @-36  ‘TTG*CA’.                    minus-10 :- @-14 ‘TATAAT’.
minus-35 :- @-36  ‘TTGAC’.                     minus-10 :- @-13  ‘TATAAT’.

conformation  :-  @-45      ‘AA**A’.
conformation  :-  @-45      ‘A***A’, @-28 ‘T***T*AA**T’, @-04 ‘T’.
conformation  :-  @-49 ‘A****T’,     @-27  ‘T****A**T*TG’, @-01 ‘A’.
conformation  :-  @-47   ‘CAA*TT*AC’, @-22 ‘G***T*C’, @-08 ‘GCGCC*CC’
```

KBANN [Towell & Shavlik '94]

- The initial KBANN-net for promoter recognition



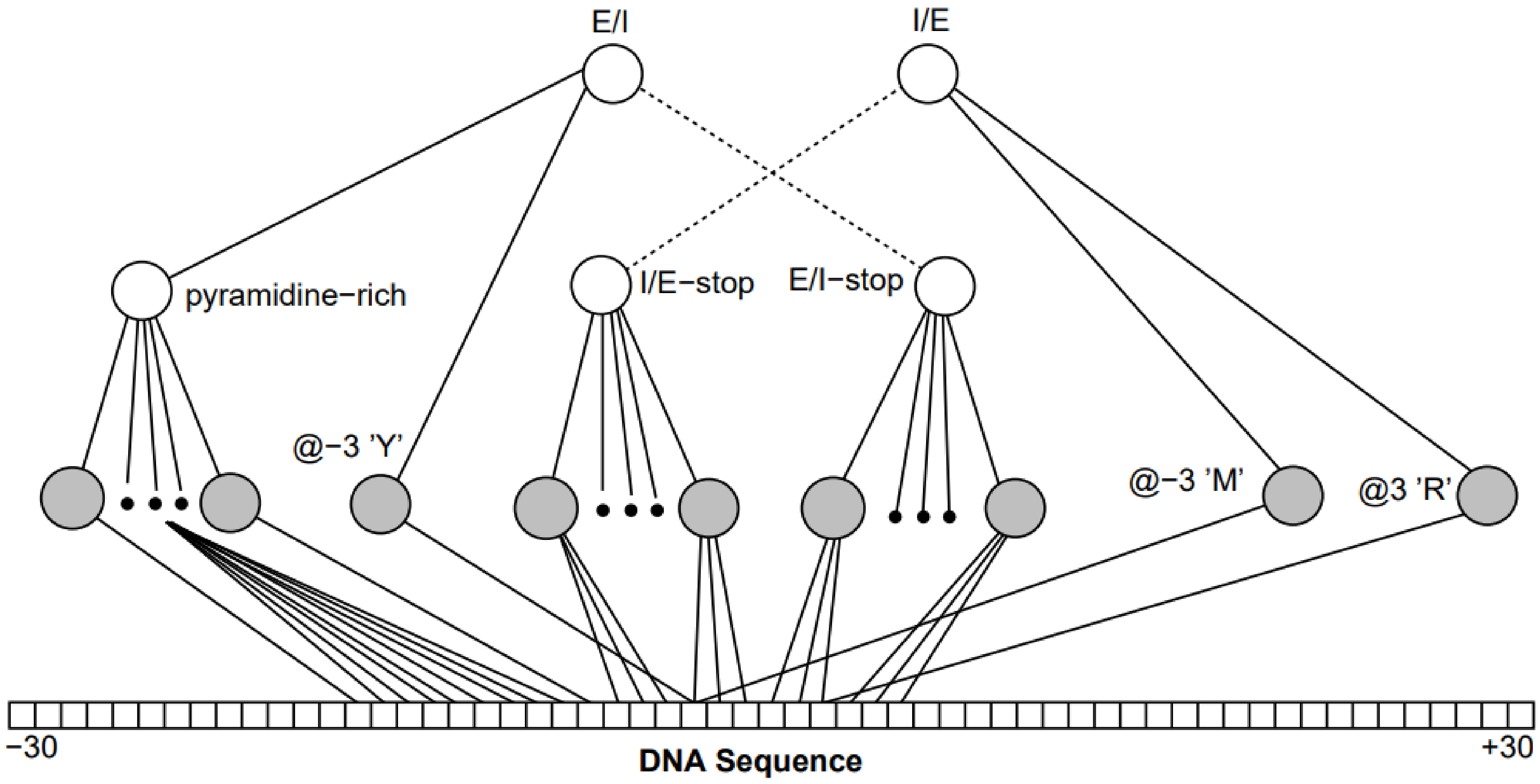
KBANN [Towell & Shavlik '94]

- Task2: Splice-junction determination
  - Splice junctions: points on a DNA sequence at which the cell removes superfluous DNA during the process of protein creation

```
E/I :- @-3 'MAGGTRAGT', not(E/I-stop).

E/I-stop:-@-3 'TAA'. E/I-stop:-@-4 'TAA'. E/I-stop:-@-5 'TAA'.
E/I-stop:-@-3 'TAG'. E/I-stop:-@-4 'TAG'. E/I-stop:-@-5 'TAG'.
E/I-stop:-@-3 'TGA'. E/I-stop:-@-4 'TGA'. E/I-stop:-@-5 'TGA'.

I/E :- pyramidine-rich, @-3 'YAGG', not(I/E-stop).
pyramidine-rich :- 6 of (@-15 'YYYYYYYYYY').
For i from (-30 to +30 skipping 0)
           {@<i> 'Y' :- @<i> 'C'.  @<i> 'Y' :- @<i> 'T'.}

I/E-stop:-@1 'TAA'.  I/E-stop:-@2 'TAA'.  I/E-stop:-@3 'TAA'.
I/E-stop:-@1 'TAG'.  I/E-stop:-@2 'TAG'.  I/E-stop:-@3 'TAG'.
I/E-stop:-@1 'TGA'.  I/E-stop:-@2 'TGA'.  I/E-stop:-@3 'TGA'.
```

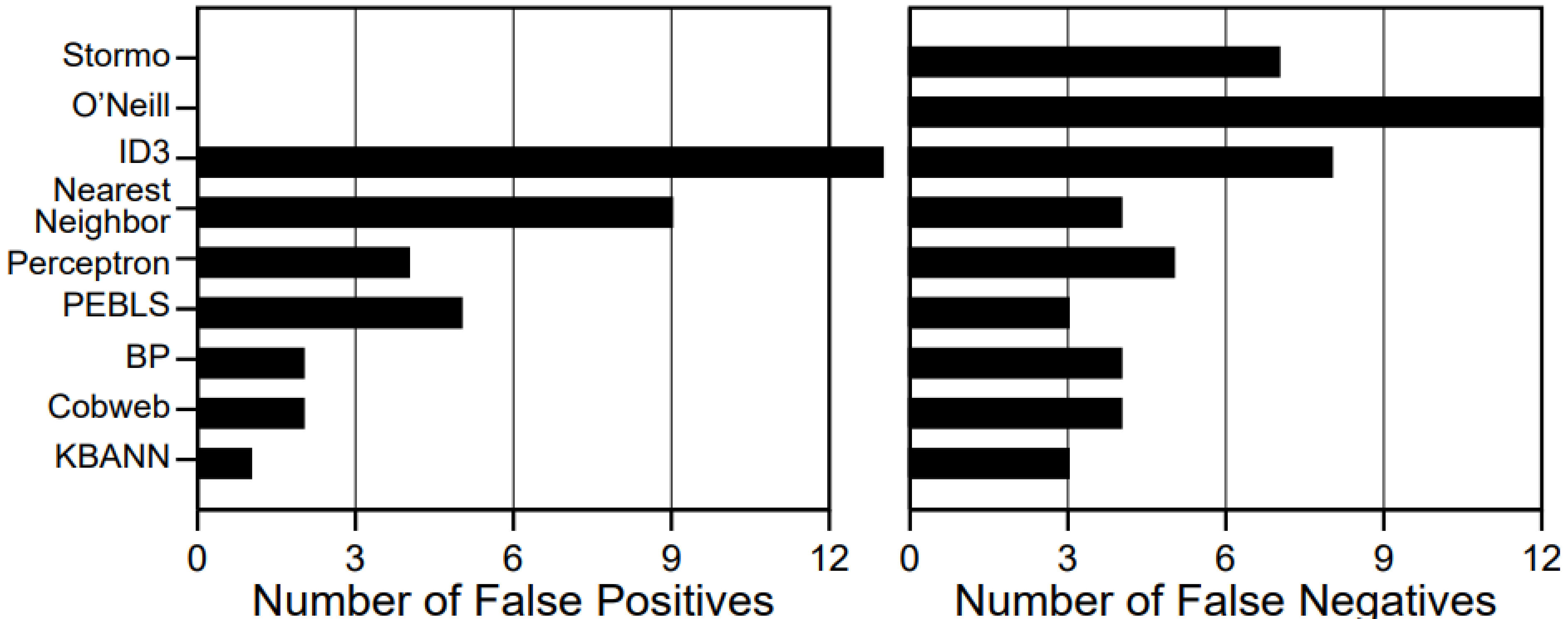
KBANN [Towell & Shavlik '94]

- The initial splice-junction KBANN-net



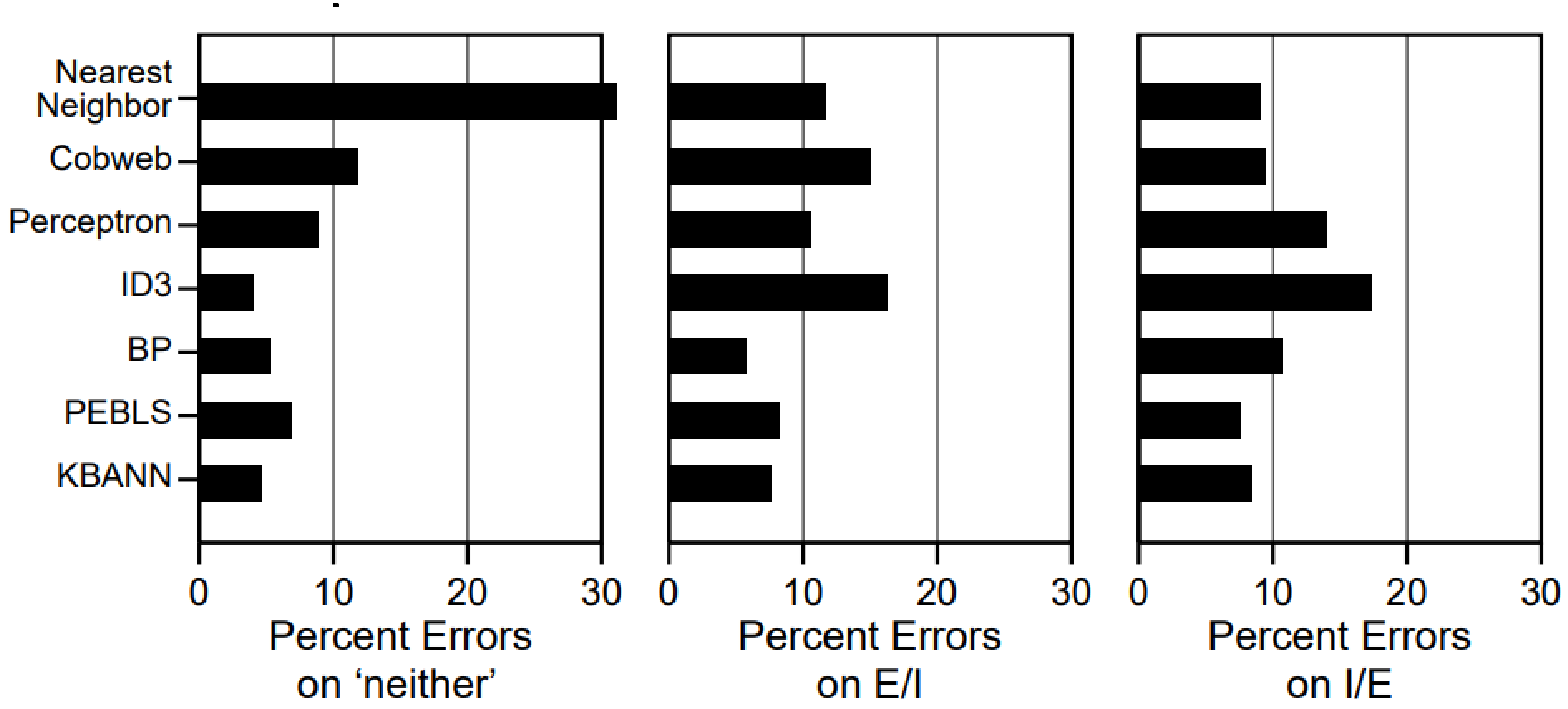
KBANN [Towell & Shavlik ‘94]

- Test-set performance on the promoter recognition task
  - leaving-one-out cross-validation

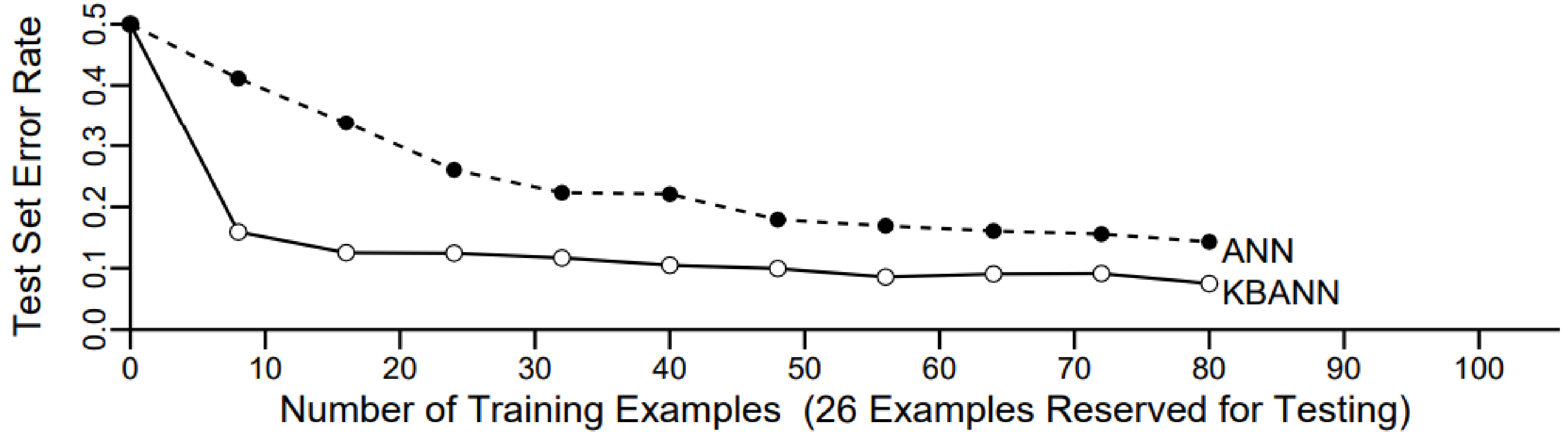


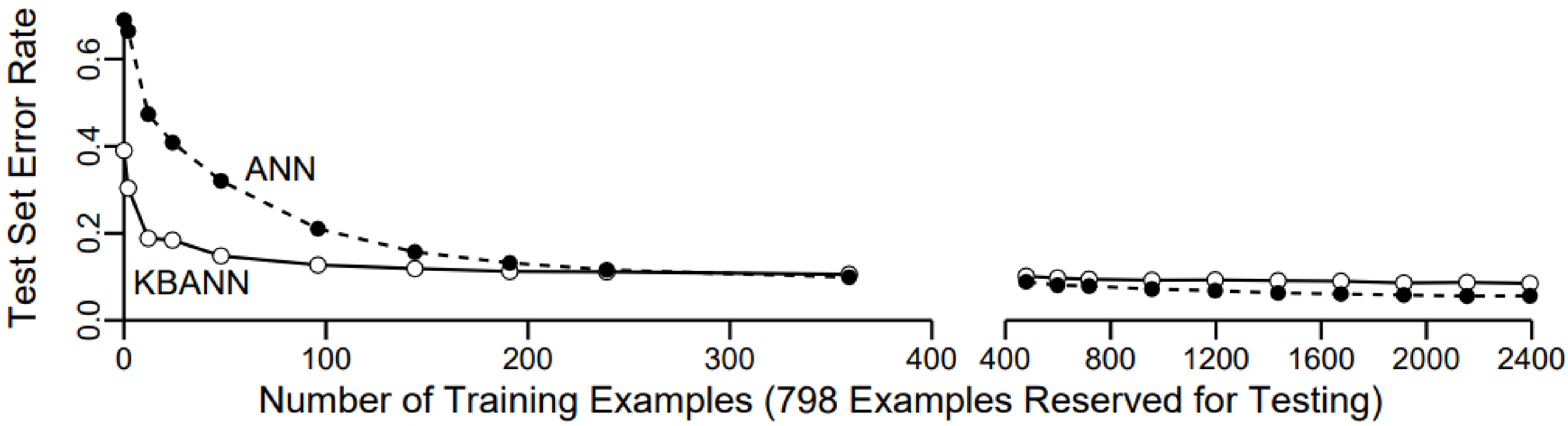
KBANN [Towell & Shavlik '94]

- Test-set performance, assessed using 10-fold cross-validation, on the splice-junction determination task with 1000 training



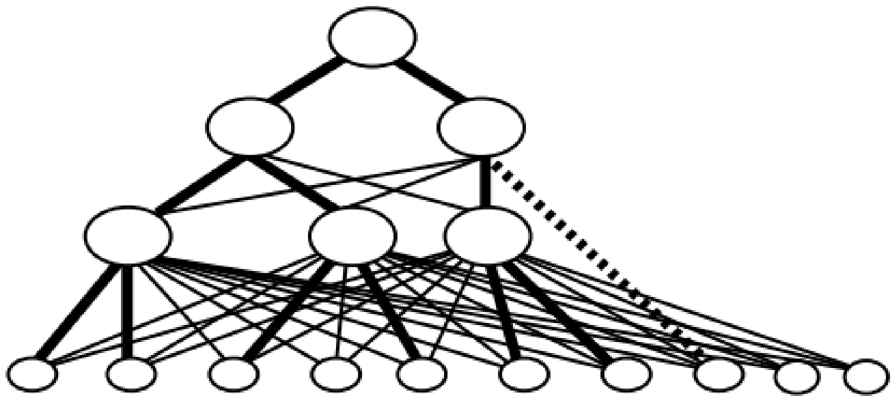
KBANN [Towell & Shavlik '94]

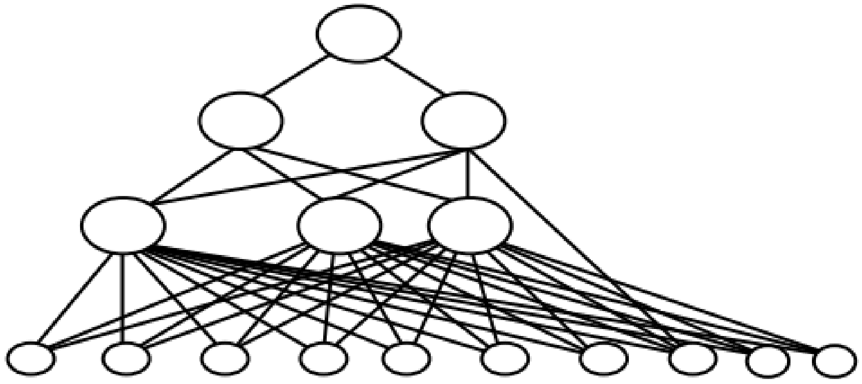




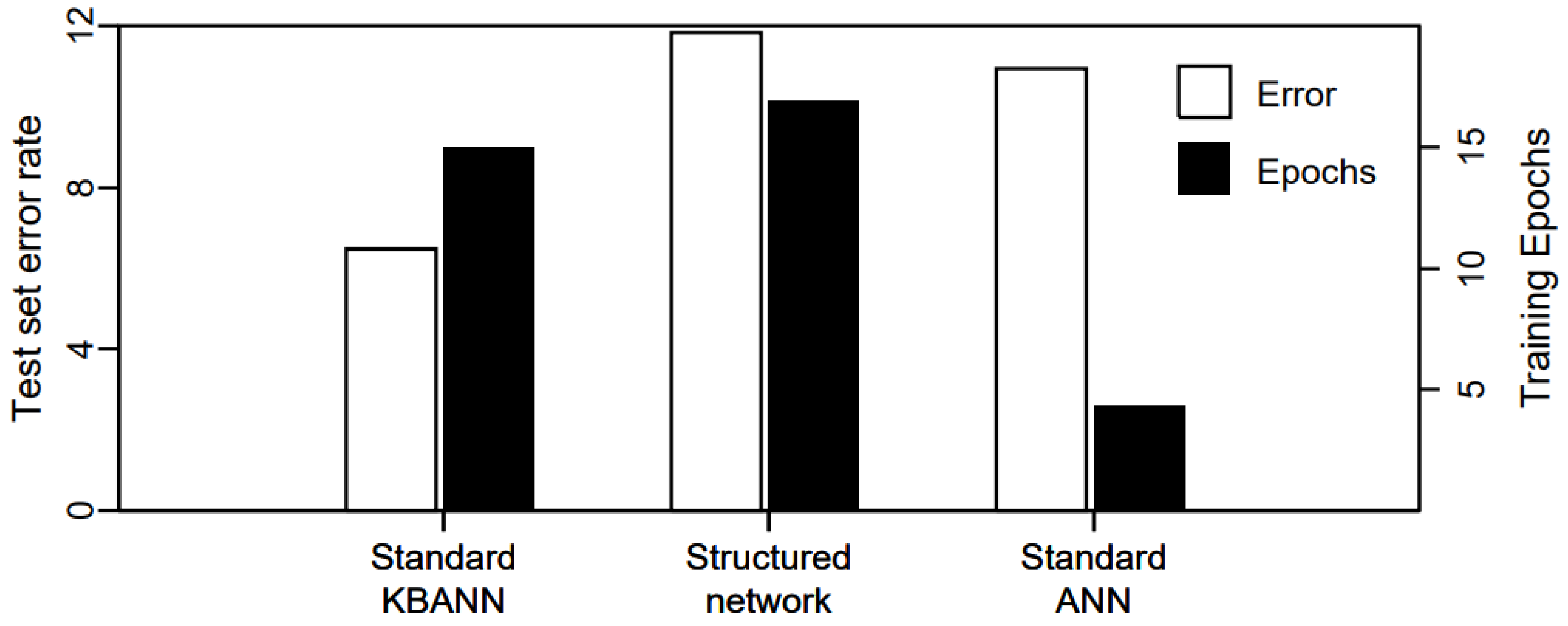
KBANN [Towell & Shavlik '94]

- Comparing KBANN to standard backpropagation
  - Two hypotheses
  - 1) Structure is responsible for KBANN's strength
  - 2) Initial weights are responsible for KBANN's strength

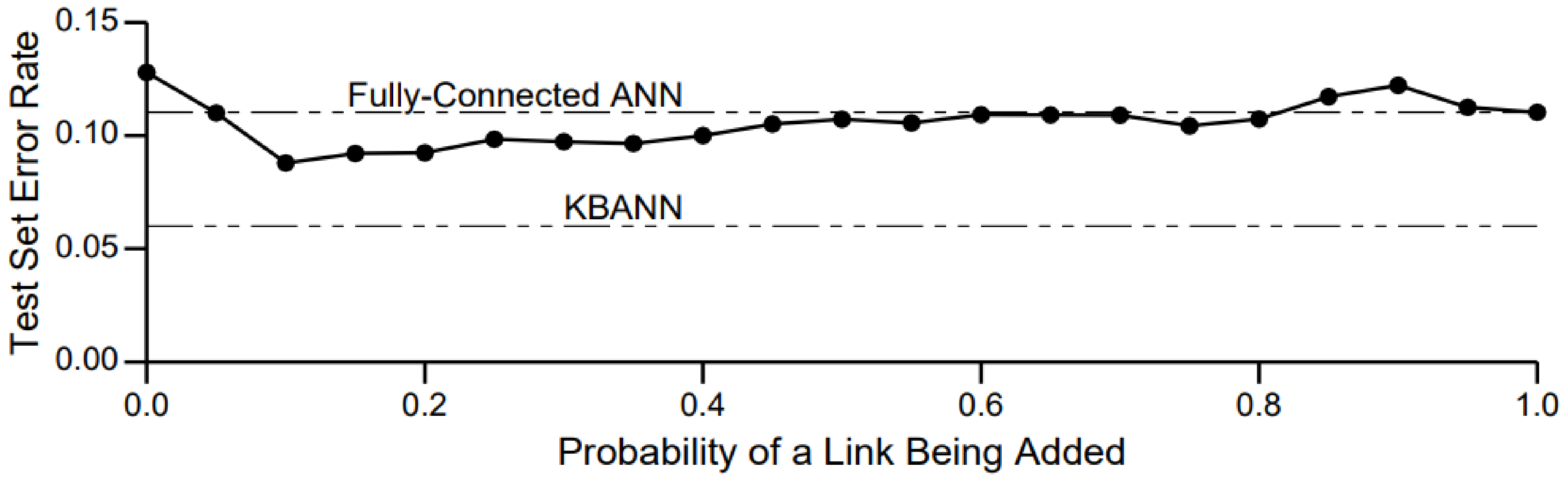

Standard KNN


"structure–only" network

KBANN [Towell & Shavlik '94]



KBANN [Towell & Shavlik '94]



The classification performance of standard ANNs that initially have links only to those features specified by the promoter domain theory.

KBANN [Towell & Shavlik ‘94]

- Discussion
  - These tests indicate that alone neither structure nor weight (focusing) account for the superiority of KBANN-nets over standard ANNs.

  - Therefore, the third hypothesis, that it is a combination the structure and the focusing weights that give KBANN its advantage over backpropagation, is likely true.

KBANN [Towell & Shavlik ‘94]

- Limitations
  - The concepts KBANN learns are incomprehensible to humans.
  - KBANN’s rule syntax is limited
  - There is no mechanism for handling uncertainty in rules
  - Neural learning in KBANN ignores the symbolic meaning of the initial network
  - There is no mechanism for changing the topology of the network

Learning explanatory rules from noisy data [Evans and Grefenstette '18]

$\text{edge}(a, b)$ $\quad$ $\text{connected}(X, Y) \leftarrow \text{edge}(X, Y)$

$\text{edge}(b, c)$ $\quad$ $\text{connected}(X, Y) \leftarrow \text{edge}(X, Z), \text{connected}(Z, Y)$

$\text{edge}(c, a)$

- An extensional predicate: a predicate that is wholly defined by a set of ground atoms
  - E.g.) , **edge** is an extensional predicate

$$\{\text{edge}(a, b), \text{edge}(b, c), \text{edge}(c, a)\}$$

- An intensional predicate: defined by a set of clauses.
  - E.g.) **connected** is an intensional predicate defined by the clauses:

$$\begin{aligned} \text{connected}(X, Y) &\leftarrow \text{edge}(X, Y) \\ \text{connected}(X, Y) &\leftarrow \text{edge}(X, Z), \text{connected}(Z, Y) \end{aligned}$$

Learning explanatory rules from noisy data [Evans and Grefenstette ‘18]

- Inductive Logic Programming (ILP)
  - A tuple $(\mathcal{B}, \mathcal{P}, \mathcal{N})$ of ground atoms
  - $\mathcal{B}$ is a set of background assumptions, a set of ground atoms
  - $\mathcal{P}$ is a set of positive instances - examples taken from the extension of the target predicate to be learned
  - $\mathcal{N}$ is a set of negative instances - examples taken outside the extension of the target predicate

Learning explanatory rules from noisy data [Evans and Grefenstette '18]

- Given an ILP problem (B, P, N), a solution is a set R of definite clauses such that

$$\mathcal{B}, R \models \gamma \text{ for all } \gamma \in \mathcal{P}$$

$$\mathcal{B}, R \not\models \gamma \text{ for all } \gamma \in \mathcal{N}$$

- Examples:

$$\mathcal{B} = \{zero(0), succ(0,1), succ(1,2), succ(2,3), ...\}$$

$$\begin{aligned} \mathcal{P} &= \{\text{even}(0), \text{even}(2), \text{even}(4), \text{even}(6), ...\} \\ \mathcal{N} &= \{\text{even}(1), \text{even}(3), \text{even}(5), \text{even}(7), ...\} \end{aligned}$$

Learning explanatory rules from noisy data [Evans and Grefenstette ‘18]

- one solution is the set R:

$$\begin{array}{rcl} even(X) & \leftarrow & zero(X) \\ even(X) & \leftarrow & even(Y), succ2(Y,X) \\ succ2(X,Y) & \leftarrow & succ(X,Z), succ(Z,Y) \end{array}$$

Learning explanatory rules from noisy data [Evans and Grefenstette ‘18]

- **Language frame** $(target, P_e, arity_e, C)$
  - *target*: the target predicate, the intensional predicate we are trying to learn
  - P_e: a set of extensional predicates
  - $arity_e$: a map $P_e \cup \{target\} \rightarrow \mathbb{N}$ specifying the arity of each predicate
  - C: a set of constants

Learning explanatory rules from noisy data [Evans and Grefenstette '18]

- **ILP problem** $(\mathcal{L}, \mathcal{B}, \mathcal{P}, \mathcal{N})$
  - L: a language frame
  - B: a set of background assumptions, ground atoms formed from the predicates in P_e and the constants in C
  - P: a set of positive examples, ground atoms formed from the target predicate and the constants in C
  - N: a set of negative examples, ground atoms formed from the target predicate and the constants in C

$$\begin{aligned} \mathcal{P} &= \{\text{even}(0), \text{even}(2), \text{even}(4), \text{even}(6), \ldots\} \\ \mathcal{N} &= \{\text{even}(1), \text{even}(3), \text{even}(5), \text{even}(7), \ldots\} \end{aligned}$$

Learning explanatory rules from noisy data [Evans and Grefenstette '18]

- Rule template $\tau = (v, int)$: a range of clauses that can be generated
  - $v \in N$: specifies the number of existentially quantified variables allowed in the clause
  - $int \in \{0, 1\}$: specifies whether the atoms in the body of the clause can use intensional predicates (int = 1) or only extensional predicates (int = 0)
- Program template Π = $(P_a, arity_a, rules, T)$
  - P_a: a set of auxiliary (intensional) predicates; these are the additional invented predicates used to help define the target predicate
  - $arity_a$: a map $P_a \to \mathbb{N}$ pecifying the arity of each auxiliary predicate
  - $rules$: a map from each intensional predicate p to a pair of rule templates (τ_p^1, τ_p^2)
  - $T \in N$: specifies the max number of steps of forward chaining inference

Learning explanatory rules from noisy data [Evans and Grefenstette ‘18]

– $rules$: a map from each intensional predicate p to a pair of rule templates (τ_p^1, τ_p^2)

– $rules$

- defines each intensional predicate by a pair of rule templates.
- In our system, we insist, without loss of generality that each predicate can be defined by exactly two clauses.

Learning explanatory rules from noisy data [Evans and Grefenstette '18]

- **Language**: $(P_e, P_i, arity, C) \quad P = P_e \cup P_i$

  $$P_i = P_a \cup \{target\} \quad arity = arity_e \cup arity_a$$

  – Combine the extensional predicates from the language-frame $\quad \Pi = (P_a, arity_a, rules, T)$

  – with the intensional predicates from the program templete $\quad \mathcal{L} = (target, P_e, arity_e, C)$

  – A language determines the set G of all ground atoms

    - E.g.) If we restrict ourselves to nullary, unary, and dyadic predicates

$$\begin{aligned} G = \{\gamma_i\}_{i=1}^{n} \quad = \quad & \{p() \mid p \in P,\ \text{arity}(p) = 0\} \cup \\ & \{p(k) \mid p \in P,\ \text{arity}(p) = 1,\ k \in C\} \cup \\ & \{p(k_1, k_2) \mid p \in P,\ \text{arity}(p) = 2,\ k_1, k_2 \in C\} \cup \\ & \{\bot\} \end{aligned}$$

Learning explanatory rules from noisy data [Evans and Grefenstette '18]

- **Generating Clauses**
  - For each rule template τ, we can generate a set $cl(\tau)$ of clauses that satisfy the template
  - Restrictions to keep $cl(\tau)$ manageable
    - 1) we only consider clauses composed of atoms involving free variables
      - We do not allow any constants in any of our clauses ➔
      - If we need a predicate whose meaning depends on particular constants, then we treat it as an extensional predicate, rather than an intensional predicate.

$$zero(0)$$

    - 2) we only allow predicates of arity 0, 1, or 2
      - We do not currently support ternary predicates or higher
    - 3) we insist that all clauses have exactly two atoms in the body

Learning explanatory rules from noisy data [Evans and Grefenstette '18]: Example

arity$(q) = 2$

– The language-frame: $\mathcal{L} = (target, P_e, arity_e, C)$

$$target = q/2 \quad P_e = \{p/2\} \quad C = \{a, b, c, d\}$$

– The ILP problem: $(\mathcal{L}, \mathcal{B}, \mathcal{P}, \mathcal{N})$

$$\mathcal{B} = \{p(a,b), p(b,c), p(c,d)\}$$

$$\mathcal{P} = \{q(a,b), q(a,c), q(a,d), q(b,c), q(b,d), q(c,d)\}$$

$$\mathcal{N} = \{q(X,Y) \mid (X,Y) \in \{a,b,c,d\}^2, q(X,Y) \notin \mathcal{P}\}$$

Learning explanatory rules from noisy data [Evans and Grefenstette ‘18]: Example

– Use the program template $\Pi = (P_a, arity_a, rules, T)$

$P_a = \{\}$ $arity_a = \{\}$ $rules = \{\tau_q^1, \tau_q^2\}$ $T = 3$

– Suppose template τ_q^1 for q is $(v = 0, int = 0)$

- Then, clauses generated after pruning are:

1. $q(X,Y) \leftarrow p(X,X), p(X,Y)$
2. $q(X,Y) \leftarrow p(X,X), p(Y,X)$
3. $q(X,Y) \leftarrow p(X,X), p(Y,Y)$
4. $q(X,Y) \leftarrow p(X,Y), p(X,Y)$
5. $q(X,Y) \leftarrow p(X,Y), p(Y,X)$
6. $q(X,Y) \leftarrow p(X,Y), p(Y,Y)$
7. $q(X,Y) \leftarrow p(Y,X), p(Y,X)$
8. $q(X,Y) \leftarrow p(Y,X), p(Y,Y)$

Learning explanatory rules from noisy data [Evans and Grefenstette '18]: Example

– Suppose template τ_q^2 for q is $(v = 1, int = 1)$
  - Then, there are 58 clauses generated after pruning, of which the first 16 are:

1. $q(X,Y) \leftarrow p(X,X), q(Y,X)$
2. $q(X,Y) \leftarrow p(X,X), q(Y,Y)$
3. $q(X,Y) \leftarrow p(X,X), q(Y,Z)$
4. $q(X,Y) \leftarrow p(X,X), q(Z,Y)$
5. $q(X,Y) \leftarrow p(X,Y), q(X,X)$
6. $q(X,Y) \leftarrow p(X,Y), q(X,Z)$
7. $q(X,Y) \leftarrow p(X,Y), q(Y,X)$
8. $q(X,Y) \leftarrow p(X,Y), q(Y,Y)$
9. $q(X,Y) \leftarrow p(X,Y), q(Y,Z)$
10. $q(X,Y) \leftarrow p(X,Y), q(Z,X)$
11. $q(X,Y) \leftarrow p(X,Y), q(Z,Y)$
12. $q(X,Y) \leftarrow p(X,Y), q(Z,Z)$
13. $q(X,Y) \leftarrow p(X,Z), q(Y,X)$
14. $q(X,Y) \leftarrow p(X,Z), q(Y,Y)$
15. $q(X,Y) \leftarrow p(X,Z), q(Y,Z)$
16. $q(X,Y) \leftarrow p(X,Z), q(Z,Y)$

Learning explanatory rules from noisy data [Evans and Grefenstette ‘18]: Differentiable ILP (∂ILP)

- **Valuations**
  - Given a set G of n ground atoms, a vector $[0,1]^n$ mapping each ground atom $\gamma_i \in G$ to the real unit interval
  - E.g.)

$$L = (P_e, P_i, \text{arity}, C)$$

$$P_e = \{r/2\} \qquad P_i = \{p/0, q/1\} \quad C = \{a, b\}$$

  - One possible valuation on the ground atoms G of L

$$\begin{array}{llll} \bot \mapsto 0.0 & p() \mapsto 0.0 & q(a) \mapsto 0.1 & q(b) \mapsto 0.3 \\ r(a,a) \mapsto 0.7 & r(a,b) \mapsto 0.1 & r(b,a) \mapsto 0.4 & r(b,b) \mapsto 0.2 \end{array}$$

Learning explanatory rules from noisy data [Evans and Grefenstette '18]: Differentiable ILP (∂ILP)

- **Induction by Gradient Descent**

$$\Lambda = \{(\gamma, 1) \mid \gamma \in \mathcal{P}\} \cup \{(\gamma, 0) \mid \gamma \in \mathcal{N}\}$$

  - given an ILP problem $(\mathcal{L}, \mathcal{B}, \mathcal{P}, \mathcal{N})$, a program template Π and a set of clause weights W, we construct a differentiable model that implements the conditional probability of λ for a ground atom α

$$\boxed{p(\lambda \mid \alpha, W, \Pi, \mathcal{L}, \mathcal{B})}$$

Clause weights Program template Language frame Background

  - Loss: The expected negative log likelihood

$$loss = - \mathop{\mathbb{E}}_{(\alpha,\lambda)\sim\Lambda} [\lambda \cdot \log p(\lambda \mid \alpha, W, \Pi, \mathcal{L}, \mathcal{B}) + (1 - \lambda) \cdot \log(1 - p(\lambda \mid \alpha, W, \Pi, \mathcal{L}, \mathcal{B}))]$$

Learning explanatory rules from noisy data [Evans and Grefenstette ‘18]: Differentiable ILP (∂ILP)

- **Induction by Gradient Descent**
  - To calculate the probability of the label λ given the atom α, we infer the consequences of applying the rules to the background facts (using T steps of forward chaining).
  - These consequences are called the Conclusion Valuation
  - Then, we extract λ as the probability of α in this valuation.

Learning explanatory rules from noisy data [Evans and Grefenstette '18]: Differentiable ILP (∂ILP)

$$p(\lambda \mid \alpha, W, \Pi, \mathcal{L}, \mathcal{B}) = f_{extract}(f_{infer}(f_{convert}(\mathcal{B}), f_{generate}(\Pi, \mathcal{L}), W, T), \alpha)$$

Differentiable

Non-differentiable

$$f_{extract} : [0,1]^n \times G \rightarrow [0,1]$$

Takes a valuation x and an atom γ and extracts the value for that atom

Valuation

Atom

$$f_{extract}(\mathbf{x}, \gamma) = \mathbf{x}[\text{index}(\gamma)]$$

$$\text{index} : G \rightarrow \mathbb{N}$$

a function that assigns each ground atom a unique integer index

$$f_{convert} : 2^G \rightarrow [0,1]^n$$

Takes a set of atoms and converts it into a valuation mapping the elements of B to 1 and all other elements of G to 0

$$f_{convert}(\mathcal{B}) = \mathbf{y} \quad \text{where} \quad \mathbf{y}[i] = \begin{cases} 1 \text{ if } \gamma_i \in \mathcal{B} \\ 0 \text{ otherwise} \end{cases}$$

the i'th ground atom in G for i = 1..n

Learning explanatory rules from noisy data [Evans and Grefenstette ‘18]: Differentiable ILP (∂ILP)

$$p(\lambda \mid \alpha, W, \Pi, \mathcal{L}, \mathcal{B}) = f_{extract}(f_{infer}(f_{convert}(\mathcal{B}), f_{generate}(\Pi, \mathcal{L}), W, T), \alpha)$$

Differentiable Non-differentiable

$f_{generate}$ produces a set of clauses from a program template Π and a language L

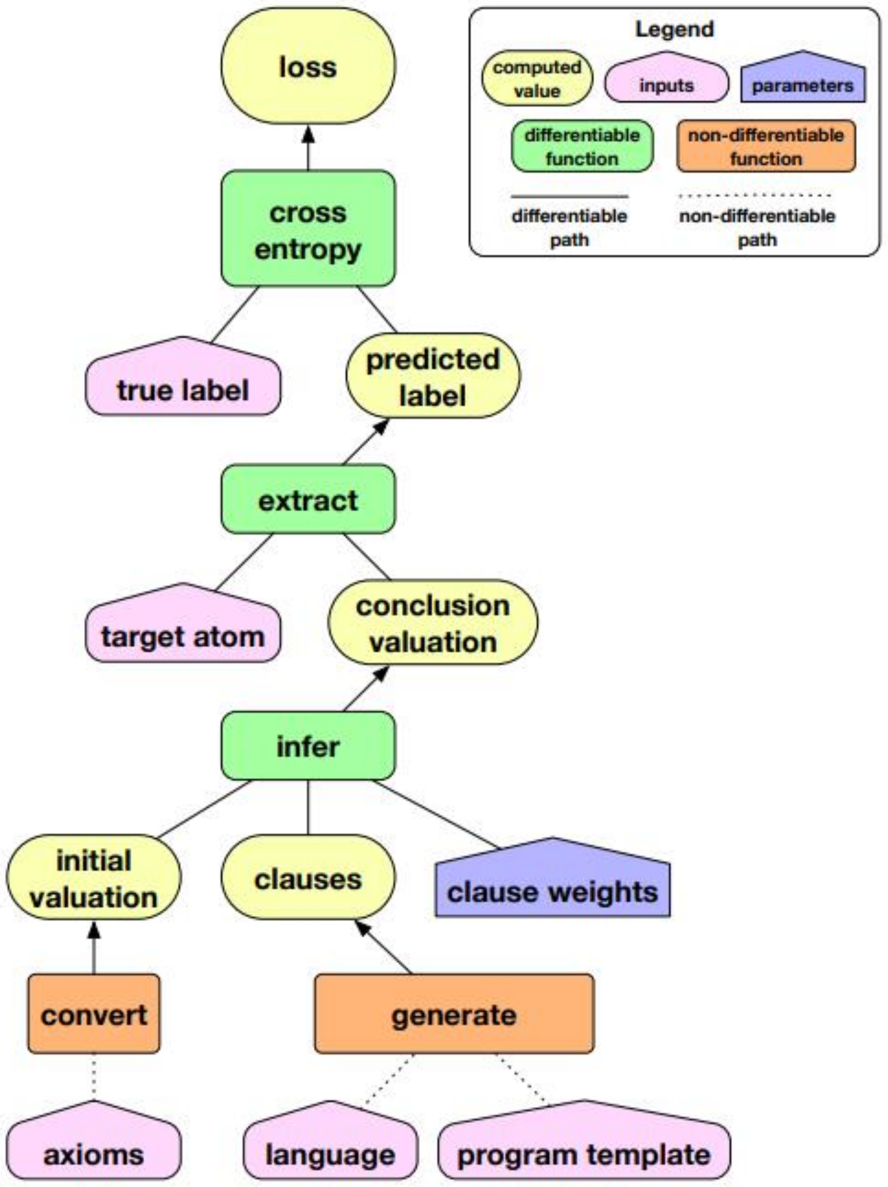
$$f_{generate}(\Pi, \mathcal{L}) = \{cl(\tau_p^i) \mid p \in P_i, i \in \{1,2\}\}$$

$$P_i = P_a \cup \{target\}$$

$$f_{infer} : [0,1]^n \times C \times W \times \mathbb{N} \to [0,1]^n$$

All the heavy-lifting takes place.
It performs T steps of forward-chaining inference using the generated clauses, amalgamating the various conclusions together using the clause weights W

Differentiable ILP (∂ILP) [Evans and Grefenstette ‘18]



Differentiable ILP (∂ILP) [Evans and Grefenstette '18]

- **Rule Weights**
  - Weights W: $\{\mathbf{W}_1, ..., \mathbf{W}_{|P_i|}\}$

$\mathbf{W}_p \in \mathbb{R}^{|cl(\tau_p^1)| \times |cl(\tau_p^2)|}$ one matrix for $p \in P_i$

the number of clauses generated by the first rule templates τ_p^1, τ_p^2

$\mathbf{W}_p[j,k]$ represents how strongly the system believes that the pair of clauses $(C_p^{1,j}, C_p^{2,k})$ is the right way to define the intensional predicate p

note that each predicate is defined by exactly two clauses

$$\mathbf{W}_p^*[j,k] = \frac{e^{\mathbf{W}_p[j,k]}}{\sum_{j',k'} e^{\mathbf{W}_p[j',k']}}$$

Differentiable ILP (∂ILP) [Evans and Grefenstette '18]

- **Inference**
  - The idea: each clause *c* induces a function $\mathcal{F}_c : [0,1]^n \rightarrow [0,1]^n$ on valuations
  - E.g.) $p(X) \leftarrow q(X)$

Applying $c = p(X) \leftarrow q(X)$ treated as a function

G	$\mathbf{a}_0$	$\mathcal{F}_c(\mathbf{a_0})$	$\mathbf{a}_1$	$\mathcal{F}_c(\mathbf{a}_1)$
$p(a)$	0.0	0.1	0.2	0.7
$p(b)$	0.0	0.3	0.9	0.4
$q(a)$	0.1	0.0	0.7	0.0
$q(b)$	0.3	0.0	0.4	0.0
$\perp$	0.0	0.0	0.0	0.0

Differentiable ILP (∂ILP) [Evans and Grefenstette '18]

- $\mathcal{F}_p^{i,j}$: the valuation function corresponding to the clause $C_p^{i,j}$

the j'th clause of the i'th rule template τ_p^i for intensional predicate p.

- $\mathcal{G}_p^{j,k}$:another indexed set of functions that combines the application of two functions $\mathcal{F}_p^{1,j}$ & $\mathcal{F}_p^{2,k}$

$$\mathcal{G}_p^{j,k}(\mathbf{a}) = \mathbf{x} \quad \text{where} \quad \mathbf{x}[i] = \max\left(\mathcal{F}_p^{1,j}(\mathbf{a})[i], \mathcal{F}_p^{2,k}(\mathbf{a})[i]\right)$$

- The initial value $\boldsymbol{a}_0$:

$$\mathbf{a}_0[x] = \begin{cases} 1 \text{ if } \gamma_x \in \mathcal{B} \\ 0 \text{ otherwise} \end{cases}$$

$$\mathbf{c}_t^{p,j,k} = G_p^{j,k}(\mathbf{a}_t)$$

Intuitively, $\boldsymbol{c}_t^{p,j,k}$ the result of applying one step of forward chaining inference to at using clauses $C_p^{1,j}$ & $C_p^{2,k}$

Differentiable ILP (∂ILP) [Evans and Grefenstette '18]

- The weighted average of the $\mathbf{c}_t^{p,j,k}$, using the softmax of the weights

$\boldsymbol{b}_t^p$ is also zero everywhere except for the p'th intensional predicate

$$\mathbf{b}_t^p = \sum_{j,k} \mathbf{c}_t^{p,j,k} \cdot \frac{e^{\mathbf{W}_p[j,k]}}{\sum_{j',k'} e^{\mathbf{W}_p[j',k']}}$$

$$\Downarrow$$

$\boldsymbol{b}_t^p$ are disjoint for different p, so we can simply sum these valuations

- The successor function:

$$\mathbf{a}_{t+1} = f_{amalgamate}(\mathbf{a}_t, \sum_{p \in P_i} \mathbf{b}_t^p)$$

$$f_{amalgamate}(\mathbf{x}, \mathbf{y}) = \max(\mathbf{x}, \mathbf{y})$$

$$f_{amalgamate}(\mathbf{x}, \mathbf{y}) = \mathbf{x} + \mathbf{y} - \mathbf{x} \cdot \mathbf{y}$$

Differentiable ILP (∂ILP) [Evans and Grefenstette '18]

- Computing the F_c Functions
  - Let $X_c = \{x_k\}_{k=1}^{n}$ be a set of sets of pairs of indices of ground atoms for clause c
  - Each x_k contains all the pairs of indices of atoms that justify atom γ_k according to the current clause c:

$$x_k = \{(a, b) \mid \text{satisfies}_c(\gamma_a, \gamma_b) \wedge \text{head}_c(\gamma_a, \gamma_b) = \gamma_k\}$$

$\text{satisfies}_c(\gamma_1, \gamma_2)$ if the pair of ground atoms (γ_1, γ_2) satisfies the body of clause c

true if given $c = \alpha \leftarrow \alpha_1, \alpha_2$

there is a substitution θ such that $\alpha_1[\theta] = \gamma_1$ & $\alpha_2[\theta] = \gamma_2$

$\text{head}_c(\gamma_1, \gamma_2)$: the head atom produced when applying clause c to the pair of atoms (γ_1, γ_2)

If $c = \alpha \leftarrow \alpha_1, \alpha_2$, $\alpha_1[\theta] = \gamma_1$ & $\alpha_2[\theta] = \gamma_2$

$$\text{head}_c(\gamma_1, \gamma_2) = \alpha[\theta]$$

Differentiable ILP (∂ILP) [Evans and Grefenstette '18]

- E.g.) Suppose $P = \{p, q, r\}$ & $C = \{a, b\}$
  - Then our ground atoms G are:

k	0	1	2	3	4	5	6	7	8
γ_k	$\perp$	$p(a,a)$	$p(a,b)$	$p(b,a)$	$p(b,b)$	$q(a,a)$	$q(a,b)$	$q(b,a)$	$q(b,b)$
k	9	10	11	12					
γ_k	$r(a,a)$	$r(a,b)$	$r(b,a)$	$r(b,b)$					

  - Suppose clause c is: $r(X,Y) \leftarrow p(X,Z), q(Z,Y)$
  - Then $X_c = \{x_k\}_{k=1}^{n}$ is:

k	γ_k	x_k
0	$\perp$	$\{\}$
1	$p(a,a)$	$\{\}$
2	$p(a,b)$	$\{\}$
3	$p(b,a)$	$\{\}$
4	$p(b,b)$	$\{\}$

k	γ_k	x_k
5	$q(a,a)$	$\{\}$
6	$q(a,b)$	$\{\}$
7	$q(b,a)$	$\{\}$
8	$q(b,b)$	$\{\}$

k	γ_k	x_k
9	$r(a,a)$	$\{(1,5), (2, 7)\}$
10	$r(a,b)$	$\{(1, 6), (2, 8)\}$
11	$r(b,a)$	$\{(3, 5), (4, 7)\}$
12	$r(b,b)$	$\{(3, 6), (4, 8)\}$

Differentiable ILP (∂ILP) [Evans and Grefenstette '18]

– Transform X_c into $\mathbf{X} \in \mathbb{N}^{n \times w \times 2}$

$$\mathbf{X}[k,m] = \begin{cases} x_k[m] \text{ if } m < |x_k| \\ (0,0) \text{ otherwise} \end{cases}$$

k	γ_k	$\mathbf{X}[k]$
0	$\bot$	$\begin{bmatrix}(0,0)\\(0,0)\end{bmatrix}$
1	$p(a,a)$	$\begin{bmatrix}(0,0)\\(0,0)\end{bmatrix}$
2	$p(a,b)$	$\begin{bmatrix}(0,0)\\(0,0)\end{bmatrix}$
3	$p(b,a)$	$\begin{bmatrix}(0,0)\\(0,0)\end{bmatrix}$
4	$p(b,b)$	$\begin{bmatrix}(0,0)\\(0,0)\end{bmatrix}$

k	γ_k	$\mathbf{X}[k]$
5	$q(a,a)$	$\begin{bmatrix}(0,0)\\(0,0)\end{bmatrix}$
6	$q(a,b)$	$\begin{bmatrix}(0,0)\\(0,0)\end{bmatrix}$
7	$q(b,a)$	$\begin{bmatrix}(0,0)\\(0,0)\end{bmatrix}$
8	$q(b,b)$	$\begin{bmatrix}(0,0)\\(0,0)\end{bmatrix}$

k	γ_k	$\mathbf{X}[k]$
9	$r(a,a)$	$\begin{bmatrix}(1,5)\\(2,7)\end{bmatrix}$
10	$r(a,b)$	$\begin{bmatrix}(1,6)\\(2,8)\end{bmatrix}$
11	$r(b,a)$	$\begin{bmatrix}(3,5)\\(4,7)\end{bmatrix}$
12	$r(b,b)$	$\begin{bmatrix}(3,6)\\(4,8)\end{bmatrix}$

Differentiable ILP (∂ILP) [Evans and Grefenstette ‘18]

– Slicing $\boldsymbol{X}$ into $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$

$$\mathbf{X}_1 = \mathbf{X}[:, :, 0] \qquad \mathbf{X}_2 = \mathbf{X}[:, :, 1]$$

$$\mathsf{gather}_2 : \mathbb{R}^a \times \mathbb{N}^{b\times c} \to \mathbb{R}^{b\times c}$$

$$\mathsf{gather}_2(x, y)[i, j] = x[y[i, j]]$$

After assembling the elements of a according to the matrix of indices in $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$, we obtain $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$:

$$\mathbf{Y}_1 = \mathsf{gather}_2(\mathbf{a}, \mathbf{X}_1) \qquad \mathbf{Y}_2 = \mathsf{gather}_2(\mathbf{a}, \mathbf{X}_2)$$

$$\mathbf{Z} = \mathbf{Y}_1 \odot \mathbf{Y}_2$$

$\mathbf{Z}[k, :]$: the vector of fuzzy conjunctions of all the pairs of atoms that contribute to the truth of γ_k, according to the current clause.

$$\boxed{F_c(\mathbf{a}) = \mathbf{a}' \quad \text{where} \quad \mathbf{a}'[k] = \max(\mathbf{Z}[k, :])}$$

Differentiable ILP (∂ILP) [Evans and Grefenstette ‘18]

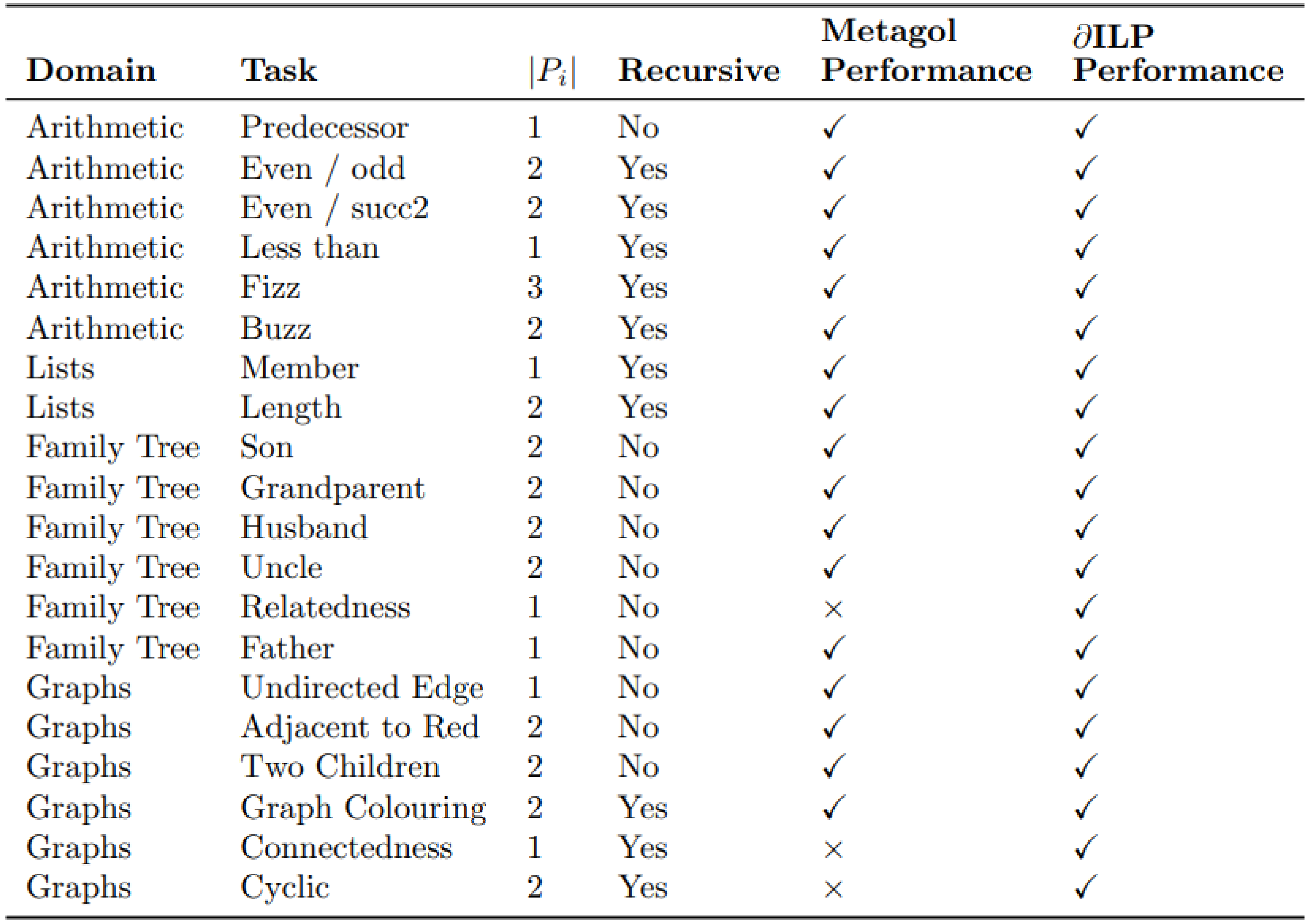
k	γ_k	$\mathbf{a}[k]$	$\mathbf{X}_1[k]$	$\mathbf{X}_2[k]$	$\mathbf{Y}_1[k]$	$\mathbf{Y}_2[k]$	$\mathbf{Z}[k]$	$F_c(\mathbf{a})[k]$
0	$\perp$	0.0	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	0.00
1	$p(a,a)$	1.0	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	0.00
2	$p(a,b)$	0.9	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	0.00
3	$p(b,a)$	0.0	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	0.00
4	$p(b,b)$	0.0	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	0.00
5	$q(a,a)$	0.1	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	0.00
6	$q(a,b)$	0.0	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	0.00
7	$q(b,a)$	0.2	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	0.00
8	$q(b,b)$	0.8	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	$[0\ \ 0]$	0.00
9	$r(a,a)$	0.0	$[1\ \ 2]$	$[5\ \ 7]$	$[1.0\ \ 0.9]$	$[0.1\ \ 0.2]$	$[0.1\ \ 0.18]$	0.18
10	$r(a,b)$	0.0	$[1\ \ 2]$	$[6\ \ 8]$	$[1.0\ \ 0.9]$	$[0\ \ 0.8]$	$[0\ \ 0.72]$	0.72
11	$r(b,a)$	0.0	$[3\ \ 4]$	$[5\ \ 7]$	$[0\ \ 0]$	$[0.1\ \ 0.2]$	$[0\ \ 0]$	0.00
12	$r(b,b)$	0.0	$[3\ \ 4]$	$[6\ \ 8]$	$[0\ \ 0]$	$[0\ \ 0.8]$	$[0\ \ 0]$	0.00

Differentiable ILP (∂ILP) [Evans and Grefenstette '18]

- **Defining fuzzy conjunction** $\mathbf{Z} = \mathbf{Y}_1 \odot \mathbf{Y}_2$
  - For other choices, need an operator $* : [0,1]^2 \rightarrow [0,1]$ satisfying the conditions on a t-norm [Esteva & Godo, 2001]
    - commutativity: x ∗ y = y ∗ x
    - associativity: (x ∗ y) ∗ z = x ∗ (y ∗ z)
    - monotonicity (i): x1 ≤ x2 implies x1 ∗ y ≤ x2 ∗ y
    - monotonicity (ii): y1 ≤ y2 implies x ∗ y1 ≤ x ∗ y
    - unit (i): x ∗ 1 = x
    - unit (ii): x ∗ 0 = 0
  - Operators satisfying these conditions include:
    - Godel t-norm: x ∗ y = min(x, y)
    - Lukasiewicz t-norm: x ∗ y = max(0, x + y − 1)
    - Product t-norm: x ∗ y = x · y

Differentiable ILP (∂ILP) [Evans and Grefenstette ‘18]

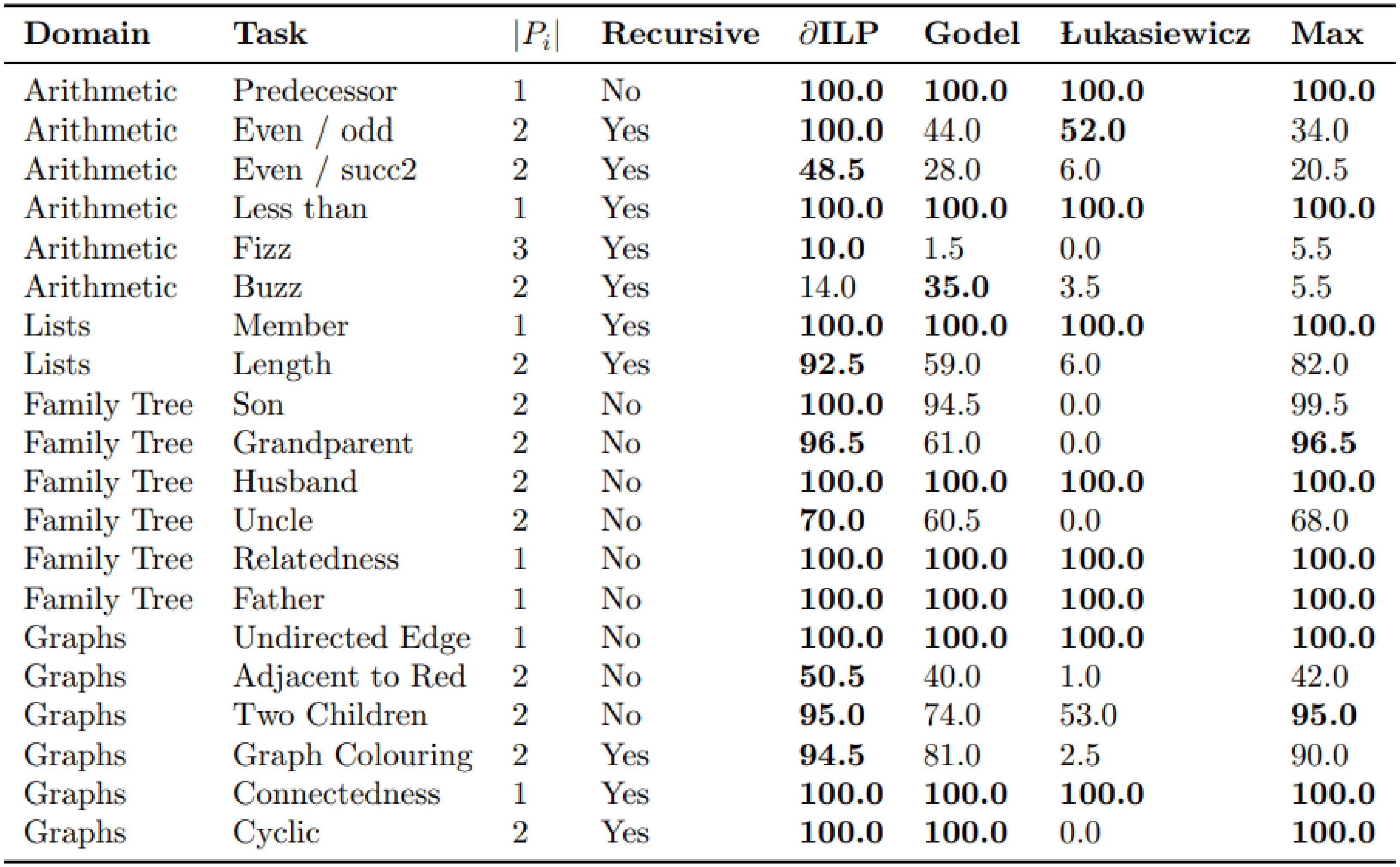
- Experiments



Domain	Task	$\lvert P_i \rvert$	Recursive	Metagol Performance	∂ILP Performance
Arithmetic	Predecessor	1	No	✓	✓
Arithmetic	Even / odd	2	Yes	✓	✓
Arithmetic	Even / succ2	2	Yes	✓	✓
Arithmetic	Less than	1	Yes	✓	✓
Arithmetic	Fizz	3	Yes	✓	✓
Arithmetic	Buzz	2	Yes	✓	✓
Lists	Member	1	Yes	✓	✓
Lists	Length	2	Yes	✓	✓
Family Tree	Son	2	No	✓	✓
Family Tree	Grandparent	2	No	✓	✓
Family Tree	Husband	2	No	✓	✓
Family Tree	Uncle	2	No	✓	✓
Family Tree	Relatedness	1	No	×	✓
Family Tree	Father	1	No	✓	✓
Graphs	Undirected Edge	1	No	✓	✓
Graphs	Adjacent to Red	2	No	✓	✓
Graphs	Two Children	2	No	✓	✓
Graphs	Graph Colouring	2	Yes	✓	✓
Graphs	Connectedness	1	Yes	×	✓
Graphs	Cyclic	2	Yes	×	✓

Differentiable ILP (∂ILP) [Evans and Grefenstette ‘18]

- Experiments



Domain	Task	$\lvert P_i \rvert$	Recursive	∂ILP	Godel	Łukasiewicz	Max
Arithmetic	Predecessor	1	No	**100.0**	**100.0**	**100.0**	**100.0**
Arithmetic	Even / odd	2	Yes	**100.0**	44.0	**52.0**	34.0
Arithmetic	Even / succ2	2	Yes	**48.5**	28.0	6.0	20.5
Arithmetic	Less than	1	Yes	**100.0**	**100.0**	**100.0**	**100.0**
Arithmetic	Fizz	3	Yes	**10.0**	1.5	0.0	5.5
Arithmetic	Buzz	2	Yes	14.0	**35.0**	3.5	5.5
Lists	Member	1	Yes	**100.0**	**100.0**	**100.0**	**100.0**
Lists	Length	2	Yes	**92.5**	59.0	6.0	82.0
Family Tree	Son	2	No	**100.0**	94.5	0.0	99.5
Family Tree	Grandparent	2	No	**96.5**	61.0	0.0	**96.5**
Family Tree	Husband	2	No	**100.0**	**100.0**	**100.0**	**100.0**
Family Tree	Uncle	2	No	**70.0**	60.5	0.0	68.0
Family Tree	Relatedness	1	No	**100.0**	**100.0**	**100.0**	**100.0**
Family Tree	Father	1	No	**100.0**	**100.0**	**100.0**	**100.0**
Graphs	Undirected Edge	1	No	**100.0**	**100.0**	**100.0**	**100.0**
Graphs	Adjacent to Red	2	No	**50.5**	40.0	1.0	42.0
Graphs	Two Children	2	No	**95.0**	74.0	53.0	**95.0**
Graphs	Graph Colouring	2	Yes	**94.5**	81.0	2.5	90.0
Graphs	Connectedness	1	Yes	**100.0**	**100.0**	**100.0**	**100.0**
Graphs	Cyclic	2	Yes	**100.0**	**100.0**	0.0	**100.0**

Neural Programmer-Interpreters [Reed & Freitas ‘16]

- Neural Programmer-Interpreter (NPI)
  - a recurrent and compositional neural network that learns to represent and execute programs
  - Three learnable components:
  - 1) a task-agnostic recurrent core
  - 2) a persistent key-value program memory
  - 3) domain-specific encoders that enable a single NPI to operate in multiple perceptually diverse environment
  - In experiment, a single NPI learns to execute three compositional programs (addition, sorting, and canonicalizing 3D models) and all 21 associated subprograms.

Neural Programmer-Interpreters [Reed & Freitas ‘16]

- Neural Programmer-Interpreter (NPI)
  - a compositional architecture that learns to represent and interpret programs.
  - The core module: an LSTM-based sequence model
    - Input: a learnable program embedding, program arguments passed on by the calling program, and a feature representation of the environment.
    - Output: a key indicating what program to call next, arguments for the following program and a flag indicating whether the program should terminate.
  - Includes a learnable key-value memory of program embeddings.
    - Essential for learning and re-using programs in a continual manner.

Neural Programmer-Interpreters [Reed & Freitas ‘16]

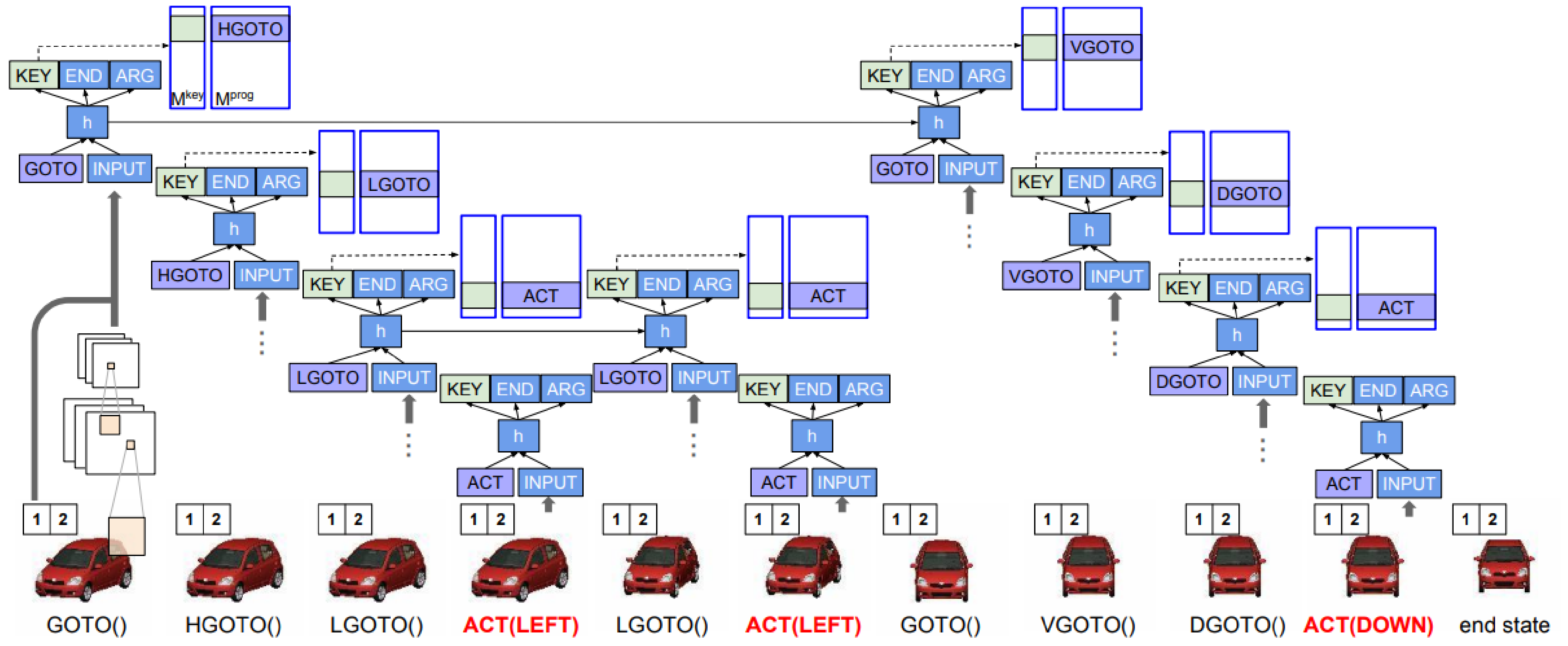
- Neural Programmer-Interpreter (NPI)
  - In experiments, can learn 21 programs, including addition, sorting, and trajectory planning from image pixels
  - Crucially, this can be achieved using a single core model with the same parameters shared across all tasks.
  - Different environments (for example images, text, and scratch-pads) may require specific perception modules or encoders to produce the features used by the shared core, as well as environment-specific actuators.
    - Both perception modules and actuators can be learned from data when training the NPI architecture
  - To train the NPI we use curriculum learning and supervision via example execution traces.

Neural Programmer-Interpreters [Reed & Freitas ‘16]

- Neural Programmer-Interpreter (NPI)
  - Exhibit strong generalization.
    - Specifically, when trained to sort sequences of up to twenty numbers in length, they can sort much longer sequences at test time.
    - In contrast, standard sequence to sequence LSTMs only exhibit weak generalization
  - Act both as an interpreter and as a programmer
    - A trained NPI with fixed parameters and a learned library of programs, can act both as an interpreter and as a programmer.
    - As an interpreter, it takes input in the form of a program embedding and input data and subsequently executes the program.
    - As a programmer, it uses samples drawn from a new task to generate a new program embedding that can be added to its library of programs.

Neural Programmer-Interpreters [Reed & Freitas '16]

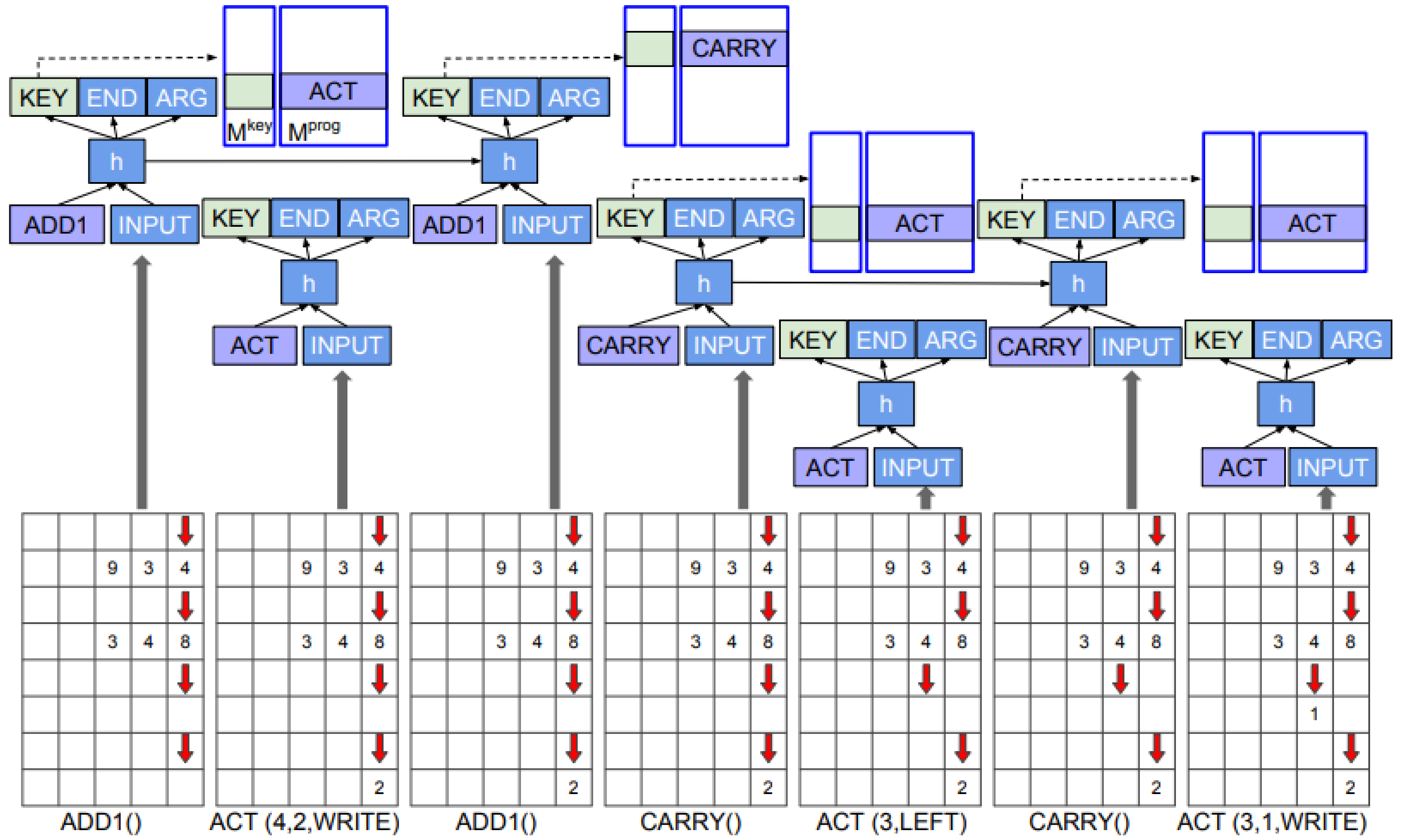
- Example execution of canonicalizing 3D car models



The task is to move the camera such that a target angle and elevation are reached. There is a read-only scratch pad containing the target (angle 1, elevation 2 here). The image encoder is a convnet trained from scratch on pixels

Neural Programmer-Interpreters [Reed & Freitas '16]

- Example execution trace of single-digit addition



The task is to perform a single-digit add on the numbers at pointer locations in the first two rows. At each time step, an observation of the environment (viewed from each pointer on a scratch pad) is encoded into a fixed-length vector.

Neural Programmer-Interpreters [Reed & Freitas ‘16]

- Inference
  - $e_t \in \mathcal{E}$: the environment observation at time t
  - $a_t \in \mathcal{A}$: the current program arguments
    - Here, consider only 3-tuple of integers $(a_t(1), a_t(2), a_t(3))$
  - $f_{enc} : \mathcal{E} \times \mathcal{A} \rightarrow \mathbb{R}^D$: domain-specific encoder
  - $p_t \in \mathbb{R}^P$: the current program embedding

  - key-value memory structures
    - $M^{\text{key}} \in \mathbb{R}^{N \times K}$: program keys
    - $M^{\text{prog}} \in \mathbb{R}^{N \times P}$: program embeddings

Neural Programmer-Interpreters [Reed & Freitas '16]

- s_t: state encoding

$$s_t = f_{enc}(e_t, a_t)$$
$$h_t = f_{lstm}(s_t, p_t, h_{t-1})$$
$$r_t = f_{end}(h_t),\; k_t = f_{prog}(h_t),\; a_{t+1} = f_{arg}(h_t)$$

- Given k_t, the program embedding is retrieved

$$i^* = \arg\max_{i=1..N}(M^{\text{key}}_{i,:})^T k_t \quad , \quad p_{t+1} = M^{\text{prog}}_{i^*,:}$$

- The next environmental state e_{t+1} will be determined by the dynamics of the env

$$e_{t+1} \sim f_{env}(e_t, p_t, a_t)$$

Neural Programmer-Interpreters [Reed & Freitas ‘16]

Algorithm 1 Neural programming inference

```
1: Inputs: Environment observation $e$, program id $i$, arguments $a$, stop threshold $\alpha$
2: function RUN($i, a$)
3:     $h \leftarrow \mathbf{0}, r \leftarrow 0, p \leftarrow M^{\texttt{prog}}_{i,:}$        ▷ Init LSTM and return probability.
4:     while $r < \alpha$ do
5:         $s \leftarrow f_{enc}(e, a), h \leftarrow f_{lstm}(s, p, h)$        ▷ Feed-forward NPI one step.
6:         $r \leftarrow f_{end}(h), k \leftarrow f_{prog}(h), a_2 \leftarrow f_{arg}(h)$
7:         $i_2 \leftarrow \arg\max_{j=1..N} (M^{\texttt{key}}_{j,:})^T k$        ▷ Decide the next program to run.
8:         if $i$ == ACT then $e \leftarrow f_{env}(e, p, a)$        ▷ Update the environment based on ACT.
9:         else RUN($i_2, a_2$)        ▷ Run subprogram $i_2$ with arguments $a_2$
```

Neural Programmer-Interpreters [Reed & Freitas ‘16]

- Training
  - Use execution traces $\xi_t^{inp} : \{e_t, i_t, a_t\}$ and $\xi_t^{out} : \{i_{t+1}, a_{t+1}, r_t\}$

Program IDs i_t and i_{t+1} are row-indices in M^{key} and M^{prog} of the programs to run at time t and t+1, respectively

$$\theta^* = \arg\max_{\theta} \sum_{(\xi^{inp},\xi^{out})} \log P(\xi^{out}|\xi^{inp};\theta)$$

$$\log P(\xi_{out}|\xi_{inp};\theta) = \sum_{t=1}^{T} \log P(\xi_t^{out}|\xi_1^{inp}, ..., \xi_t^{inp};\theta)$$

$$\log P(\xi_t^{out}|\xi_1^{inp}, ..., \xi_t^{inp}) = \log P(i_{t+1}|h_t) + \log P(a_{t+1}|h_t) + \log P(r_t|h_t$$

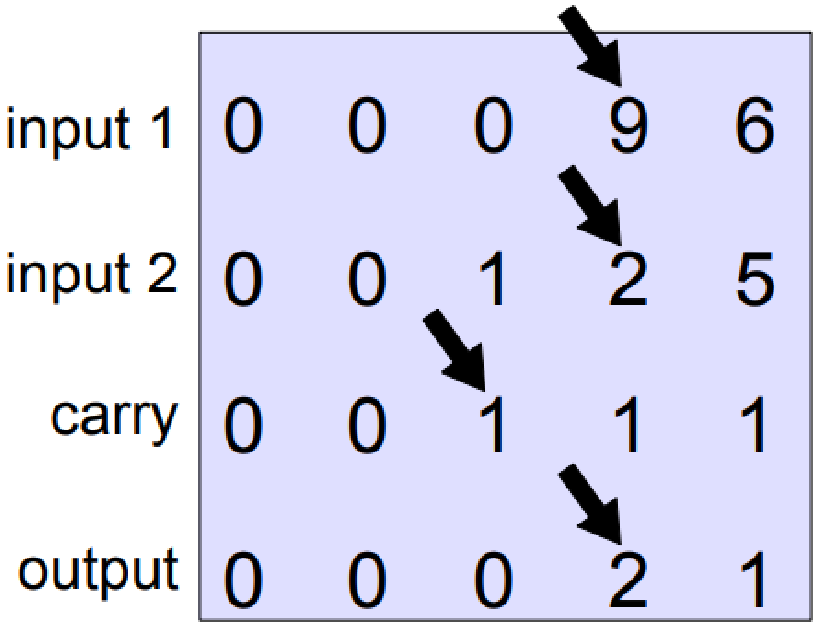
Neural Programmer-Interpreters [Reed & Freitas '16]

- Task: Addition

pointers, one per scratch pad row

$$f_{enc}(Q, i_1, i_2, i_3, i_4, a_t) = MLP([Q(1, i_1), Q(2, i_2), Q(3, i_3), Q(4, i_4), a_t(1), a_t(2), a_t(3)])$$

$$Q \in \mathbb{R}^{4 \times N \times K}$$



```
ADD ─────────────────────────────────────────────────────────
└→ADD1                      └→ADD1                       └→ADD1
   WRITE OUT 1                 WRITE OUT 2                  WRITE OUT 2
   CARRY                       CARRY                       LSHIFT
     PTR CARRY LEFT              PTR CARRY LEFT               PTR INP1 LEFT
     WRITE CARRY 1               WRITE CARRY 1                PTR INP2 LEFT
     PTR CARRY RIGHT             PTR CARRY RIGHT              PTR CARRY LEFT
  LSHIFT                      LSHIFT                          PTR OUT LEFT
   PTR INP1 LEFT               PTR INP1 LEFT
   PTR INP2 LEFT               PTR INP2 LEFT
   PTR CARRY LEFT              PTR CARRY LEFT
   PTR OUT LEFT                PTR OUT LEFT
```

Example scratch pad and pointers used for computing "96 + 125 = 221"

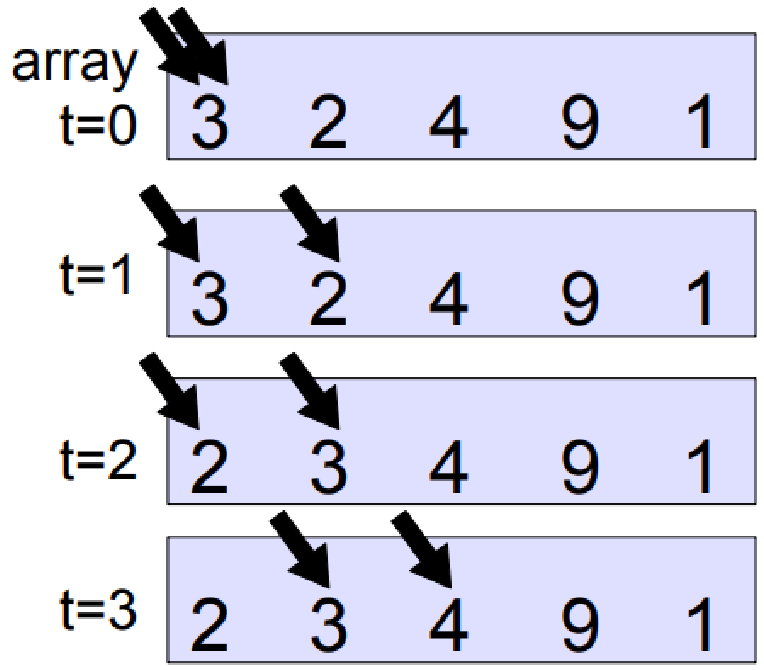
Actual trace of addition program generated by NPI

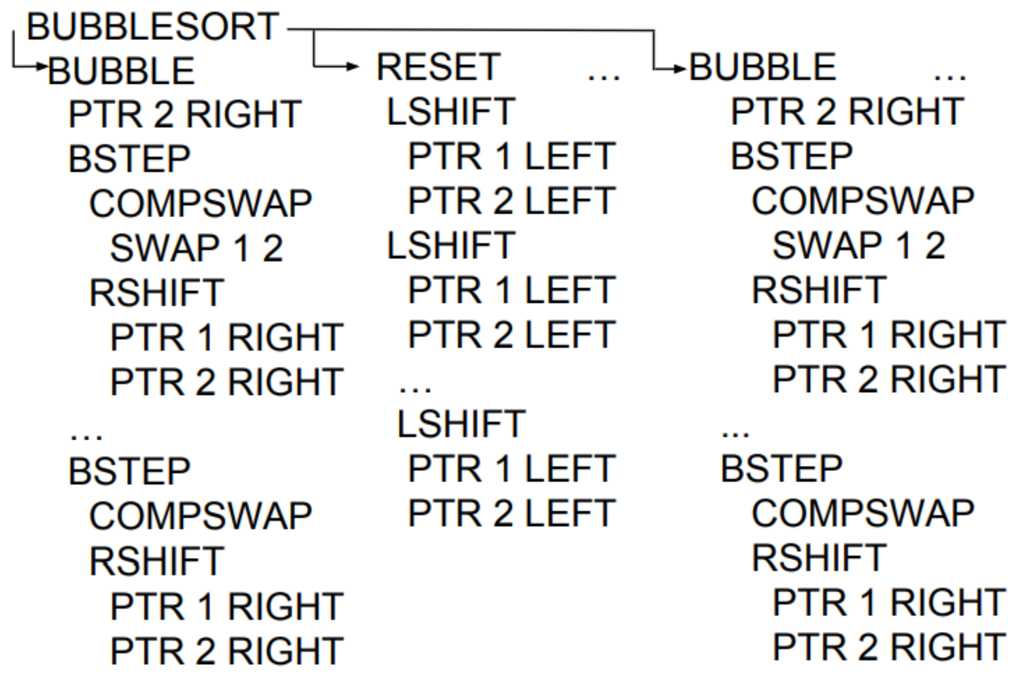
Neural Programmer-Interpreters [Reed & Freitas ‘16]

- Task: Sorting

$$Q \in \mathbb{R}^{1 \times N \times K}$$

$$f_{enc}(Q, i_1, i_2, a_t) = MLP([Q(1, i_1), Q(1, i_2), a_t(1), a_t(2), a_t(3)])$$





```
BUBBLESORT
└→BUBBLE              └→ RESET        ...  └→BUBBLE         ...
    PTR 2 RIGHT           LSHIFT                PTR 2 RIGHT
    BSTEP                   PTR 1 LEFT          BSTEP
      COMPSWAP              PTR 2 LEFT            COMPSWAP
        SWAP 1 2          LSHIFT                    SWAP 1 2
      RSHIFT                PTR 1 LEFT            RSHIFT
        PTR 1 RIGHT         PTR 2 LEFT              PTR 1 RIGHT
        PTR 2 RIGHT       ...                       PTR 2 RIGHT
    ...                   LSHIFT                ...
    BSTEP                   PTR 1 LEFT          BSTEP
      COMPSWAP              PTR 2 LEFT            COMPSWAP
      RSHIFT                                      RSHIFT
        PTR 1 RIGHT                                 PTR 1 RIGHT
        PTR 2 RIGHT                                 PTR 2 RIGHT
```

Excerpt from the trace of the learned bubblesort program

Neural Programmer-Interpreters [Reed & Freitas '16]

- Task: Canonicalizing 3D models

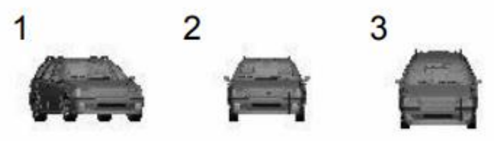
$Q \in \mathbb{R}^{2\times 1\times K}$ $K = 24$ corresponding to 15° pose increments

the pad containing canonical azimuth and elevation

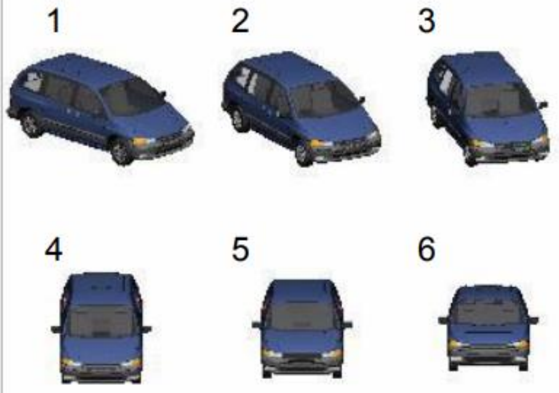
x: a car rendering at the current pose

$$f_{enc}(Q, x, i_1, i_2, a_t) = MLP([Q(1, i_1), Q(2, i_2), f_{CNN}(x), a_t(1), a_t(2), a_t(3)])$$

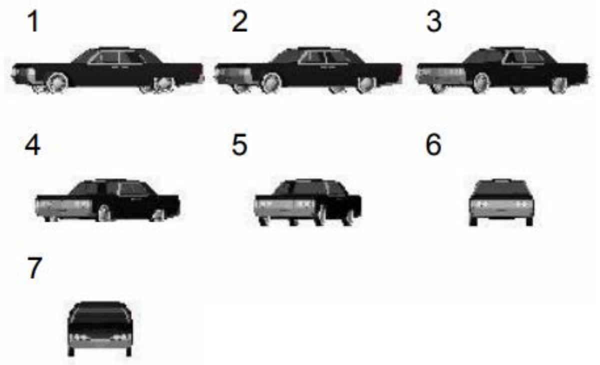
```
GOTO 1 2
  HGOTO
    RGOTO
      ACT(RIGHT)
  VGOTO
    UGOTO
      ACT(UP)
```



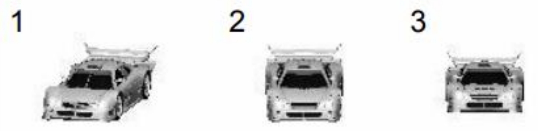
```
GOTO 1 2
  HGOTO
    RGOTO
      ACT(RIGHT)
      ACT(RIGHT)
      ACT(RIGHT)
  VGOTO
    DGOTO
      ACT(DOWN)
      ACT(DOWN)
```



```
GOTO 1 2
  HGOTO
    LGOTO
      ACT(LEFT)
      ACT(LEFT)
      ACT(LEFT)
      ACT(LEFT)
      ACT(LEFT)
  VGOTO
    UGOTO
      ACT(UP)
```



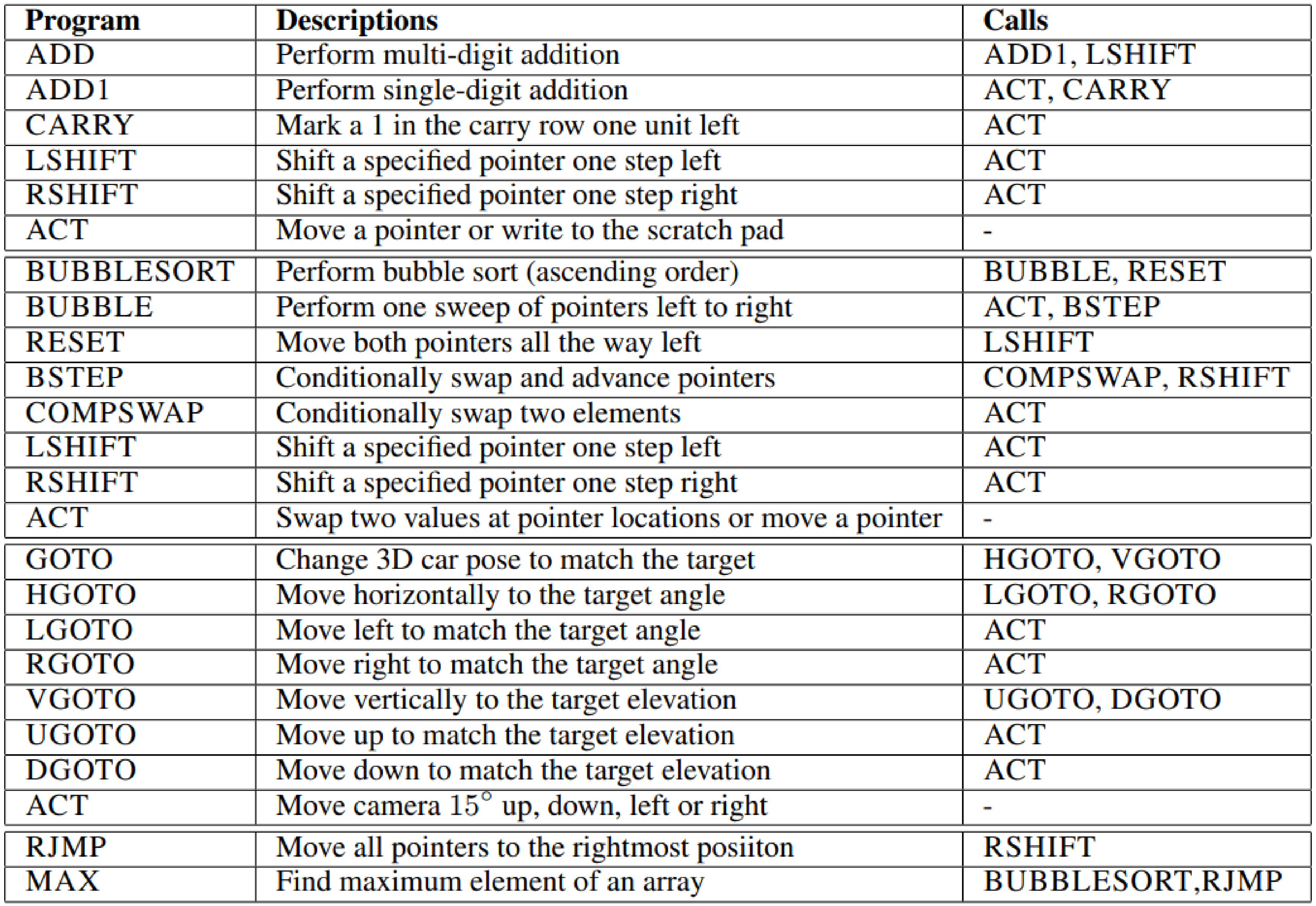
```
GOTO 1 2
  HGOTO
    LGOTO
      ACT(LEFT)
  VGOTO
    DGOTO
      ACT(DOWN)
```



Importantly, NPI can generalize to car appearances not encountered in the training set

Neural Programmer-Interpreters [Reed & Freitas '16]

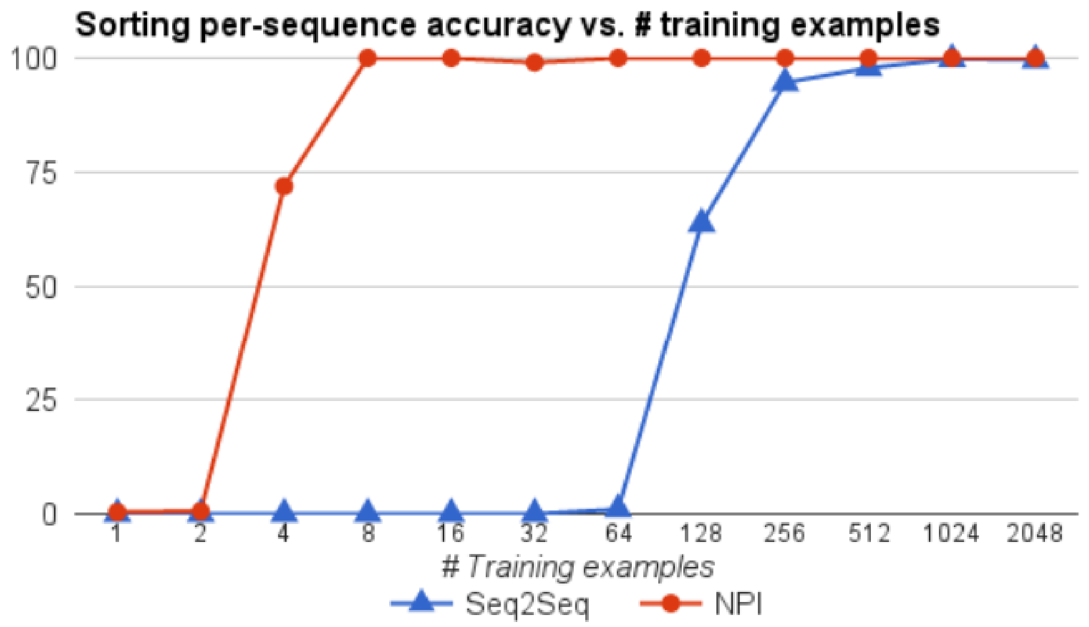
– Programs learned for addition, sorting and 3D car canonicalization.



Program	Descriptions	Calls
ADD	Perform multi-digit addition	ADD1, LSHIFT
ADD1	Perform single-digit addition	ACT, CARRY
CARRY	Mark a 1 in the carry row one unit left	ACT
LSHIFT	Shift a specified pointer one step left	ACT
RSHIFT	Shift a specified pointer one step right	ACT
ACT	Move a pointer or write to the scratch pad	-
BUBBLESORT	Perform bubble sort (ascending order)	BUBBLE, RESET
BUBBLE	Perform one sweep of pointers left to right	ACT, BSTEP
RESET	Move both pointers all the way left	LSHIFT
BSTEP	Conditionally swap and advance pointers	COMPSWAP, RSHIFT
COMPSWAP	Conditionally swap two elements	ACT
LSHIFT	Shift a specified pointer one step left	ACT
RSHIFT	Shift a specified pointer one step right	ACT
ACT	Swap two values at pointer locations or move a pointer	-
GOTO	Change 3D car pose to match the target	HGOTO, VGOTO
HGOTO	Move horizontally to the target angle	LGOTO, RGOTO
LGOTO	Move left to match the target angle	ACT
RGOTO	Move right to match the target angle	ACT
VGOTO	Move vertically to the target elevation	UGOTO, DGOTO
UGOTO	Move up to match the target elevation	ACT
DGOTO	Move down to match the target elevation	ACT
ACT	Move camera 15° up, down, left or right	-
RJMP	Move all pointers to the rightmost posiiton	RSHIFT
MAX	Find maximum element of an array	BUBBLESORT,RJMP

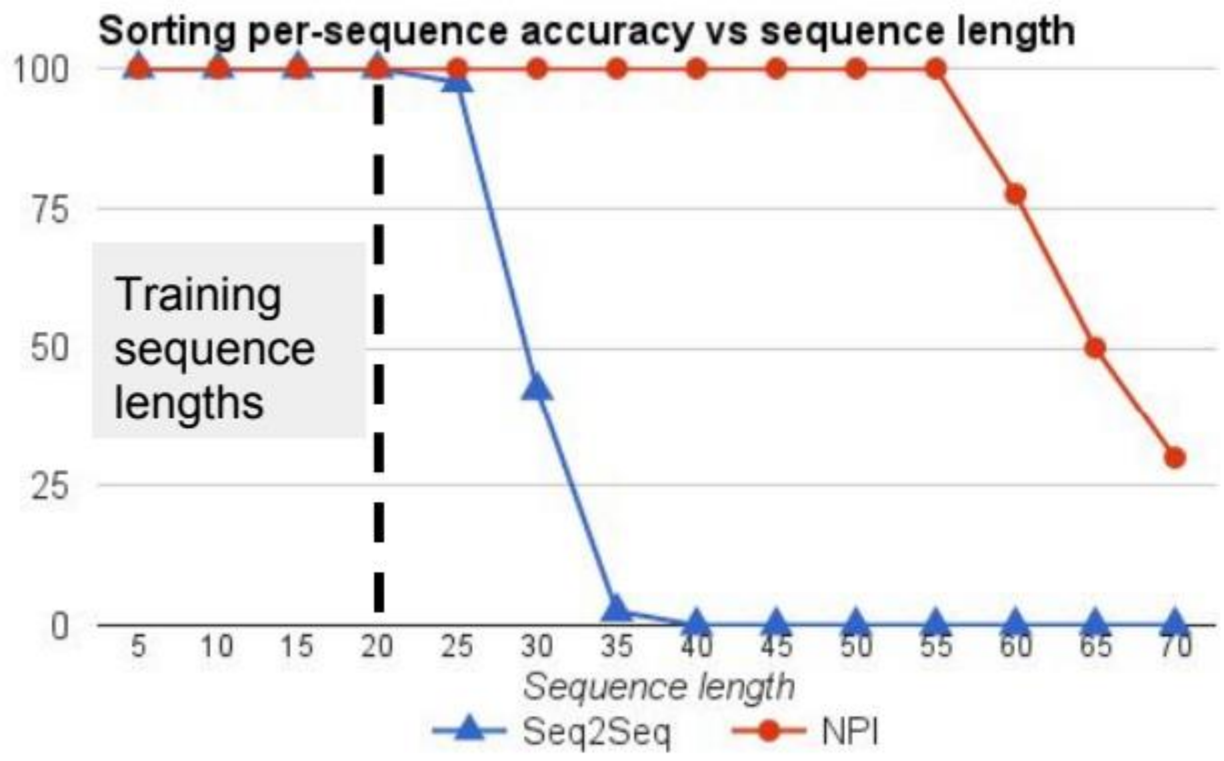
Neural Programmer-Interpreters [Reed & Freitas ‘16]

- Sample complexity
  - Sorting task



Neural Programmer-Interpreters [Reed & Freitas '16]

- Strong vs. weak generalization
  - Sorting task

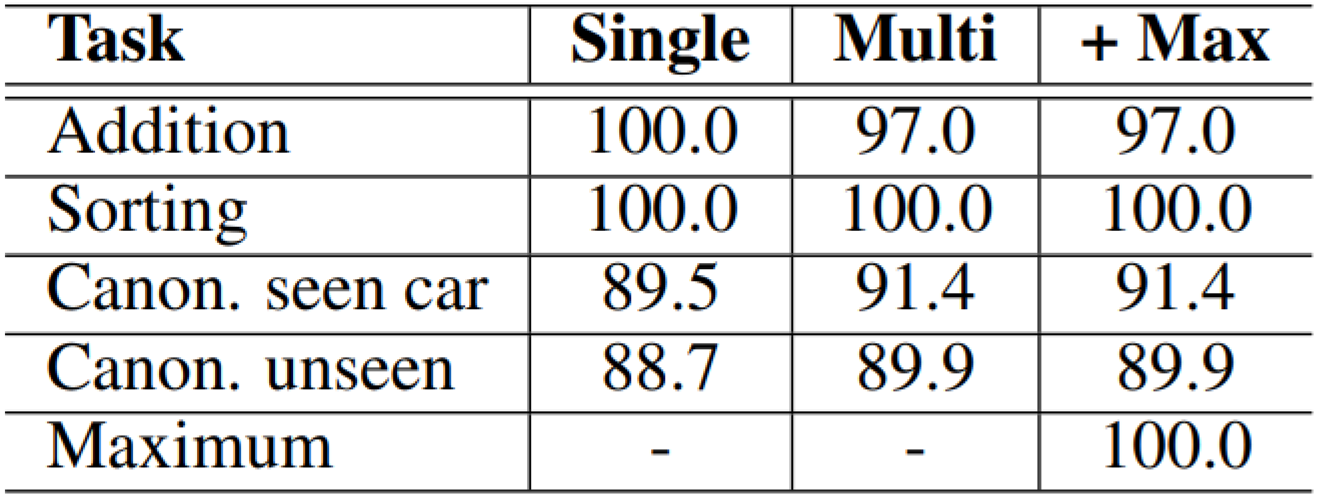


Neural Programmer-Interpreters [Reed & Freitas '16]

- Learning new programs with a fixed core
  - Adding a maximum-finding program MAX
    - MAX first calls BUBBLESORT and then a new program RJMP, which moves pointers to the right of the sorted array
  - To avoid catastrophic forgetting
    - Fix the weights of the core routing module, and only make sparse updates to the program memory
    - When adding a new program the core module's routing computation will be completely unaffected;
      - All the learning for a new task occurs in program embedding space.
  - An old program could mistakenly call a newly added program
    - The addition of new programs to the memory adds a new choice of program at each time step, and an old program could mistakenly call a newly added program
    - To overcome this, two methods are considered
      - 1) when learning a new set of program vectors with a fixed core, in practice we train not only on example traces of the new program, but also traces of existing programs
      - 2) Alternatively, a simpler approach is to prevent existing programs from calling subsequently added programs, allowing addition of new programs without ever looking back at training data for known programs.
    - In either case, note that **only the memory slots of the new programs** are updated, and all other weights, including other program embeddings, are fixed

Neural Programmer-Interpreters [Reed & Freitas ‘16]

- Solving multi-task with a single network
  - Perform a controlled experiment to compare the performance of a multi-task NPI with several single-task NPI models



Task	**Single**	**Multi**	**+ Max**
Addition	100.0	97.0	97.0
Sorting	100.0	100.0	100.0
Canon. seen car	89.5	91.4	91.4
Canon. unseen	88.7	89.9	89.9
Maximum	-	-	100.0

Towards Deep Symbolic Reinforcement Learning [Garnelo et al ‘16]

- Deep reinforcement learning (DRL)
  - Recently been shown to be effective in a number of domains, including Atari video games, robotics, and the game of Go
  - Can be thought of as a step towards instantiating the formal characterisation of universal artificial intelligence presented by Hutter, a theoretical framework for AGI founded on reinforcement learning
- Contemporary DRL systems: Shortcomings
  - 1) they inherit from deep learning the need for very large training sets, which entails that they learn very slowly
  - 2) they are brittle in the sense that a trained network that performs well on one task often performs very poorly on a new task, even if the new task is very similar to the one it was originally trained on.
  - 3) they are strictly reactive, meaning that they do not use high-level processes such as planning, causal reasoning, or analogical reasoning to fully exploit the statistical regularities present in the training data.
  - 4) they are opaque. It is typically difficult to extract a humanly-comprehensible chain of reasons for the action choice the system makes

Towards Deep Symbolic Reinforcement Learning [Garnelo et al ‘16]

- Propose a novel hybrid reinforcement learning architecture that combines neural network learning with aspects of classical symbolic AI
- Classical symbolic AI: Pros
  - The use of language-like propositional representations to encode knowledge
  - Thanks to their compositional structure, such representations are amenable to endless extension and recombination
    - This is an essential feature for the acquisition and deployment of high-level abstract concepts, which are key to general intelligence
  - Knowledge expressed in propositional form can be exploited by multiple high-level reasoning processes and has general-purpose application across multiple tasks and domains.
  - Features such as these, derived from the benefits of human language, motivated several decades of research in symbolic AI

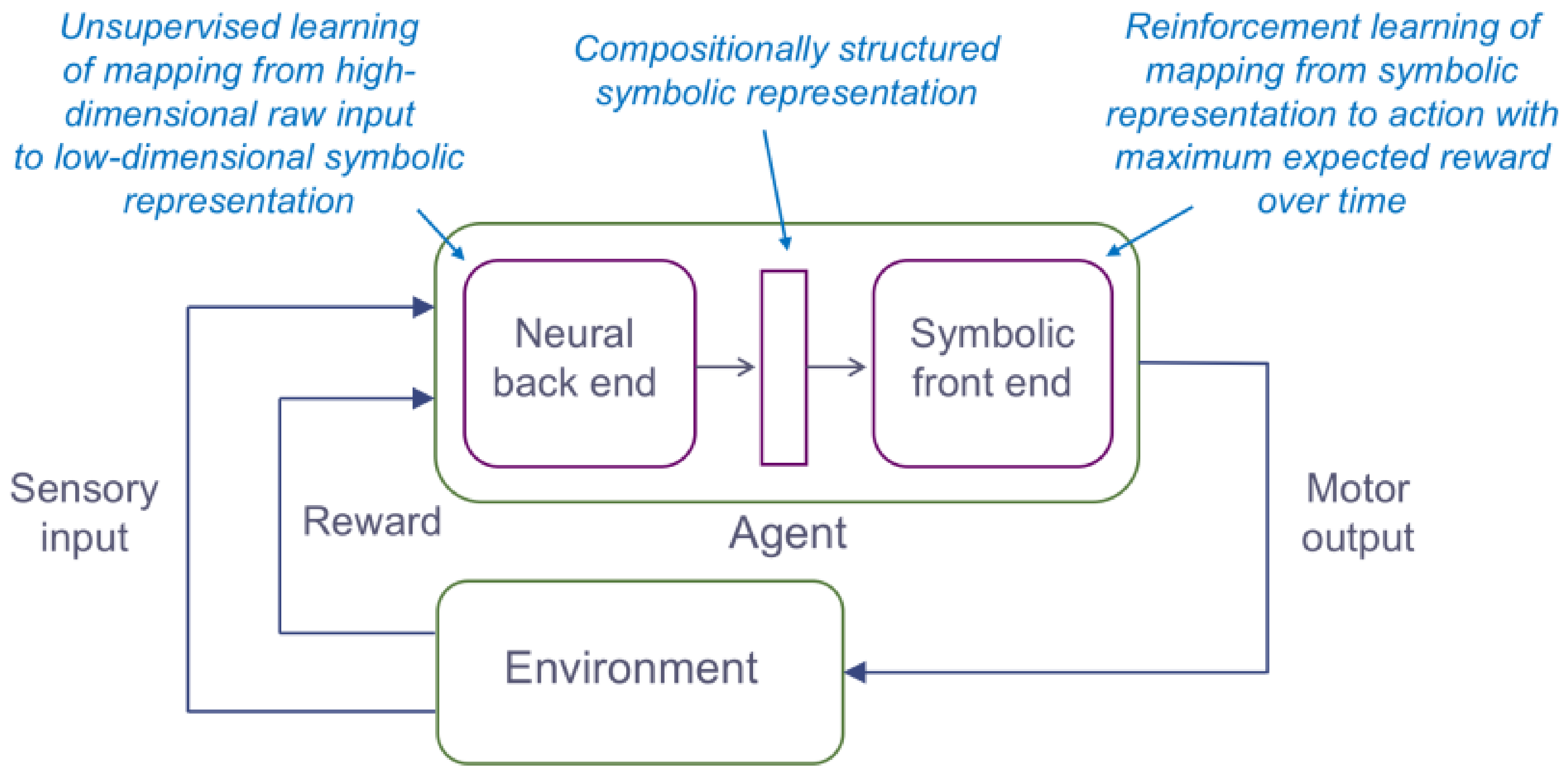
Towards Deep Symbolic Reinforcement Learning [Garnelo et al '16]

- Classical symbolic AI: Limits
  - The symbol grounding problem
    - The symbolic elements of a representation in classical AI – the constants, functions, and predicates – are typically hand-crafted, rather than grounded in data from the real world.
    - This means their semantics are parasitic on meanings in the heads of their designers rather than deriving from a direct connection with the world
    - Hand-crafted representations cannot capture the rich statistics of realworld perceptual data, cannot support ongoing adaptation to an unknown environment, and are an obvious barrier to full autonomy

Towards Deep Symbolic Reinforcement Learning [Garnelo et al ‘16]

- Deep learning
  - Have proven to be remarkably effective for supervised learning from large datasets using backpropagation.
  - Deep learning is therefore already a viable solution to the symbol grounding problem in the supervised case, and for the unsupervised case, which is essential for a full solution, rapid progress is being made
- The approach of this work for the hybrid approach
  - The hybrid neural-symbolic reinforcement learning relies on a deep learning solution to the symbol grounding problem.

Towards Deep Symbolic Reinforcement Learning [Garnelo et al '16]


Unsupervised learning of mapping from high-dimensional raw input to low-dimensional symbolic representation
Compositionally structured symbolic representation
Reinforcement learning of mapping from symbolic representation to action with maximum expected reward over time
Neural back end
Symbolic front end
Agent
Sensory input
Reward
Motor output
Environment

Towards Deep Symbolic Reinforcement Learning [Garnelo et al ‘16]

- Four fundamental principles of the architectural manifesto.
  - 1) Conceptual abstraction
    - Determining that a new situation is similar or analogous to one (or several) encountered previously is an operation fundamental to general intelligence, and to reinforcement learning in particular
    - The present architecture maps high-dimensional raw input into a lower-dimensional conceptual state space
      - It is possible to establish similarity between states using symbolic methods that operate at a higher level of abstraction.
      - Facilitates both data efficient learning and transfer learning as well as providing a foundation for other high-level cognitive processes such as planning, innovative problem solving, and communication with other agents (including humans).

Towards Deep Symbolic Reinforcement Learning [Garnelo et al '16]

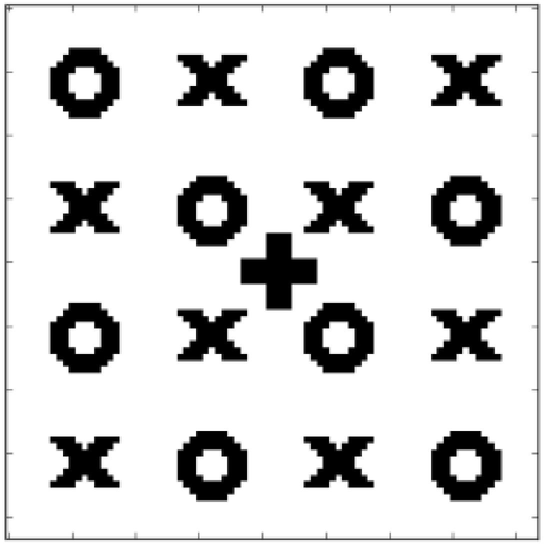
- 2) Compositional structure
  - A representational medium is required that has a compositional structure.
  - Thus use probabilistic first-order logic for the semantic underpinnings of the low-dimensional conceptual state space representation into which the neural front end must map the system's high-dimensional raw input
- 3) Common sense priors
  - The everyday physical world is structured according to many other common sense priors
    - Consisting mostly of empty space, it contains a variety of objects that tend to persist over time and have various attributes such as shape, colour, and texture.
    - Objects frequently move, typically in continuous trajectories. Objects participate in a number of stereotypical events, such as starting to move or coming to a halt, appearing or disappearing, and coming into contact with other objects.
  - Thus graft a suitable ontology onto the underlying representational language, greatly reducing the learning workload and facilitating various forms of common sense reasoning

Towards Deep Symbolic Reinforcement Learning [Garnelo et al ‘16]

– 4) Causal reasoning
  - The current generation of DRL architectures eschews model-based reinforcement learning, ensuring that the resulting systems are purely reactive
  - Instead, the current architecture attempts to discover the causal structure of the domain, and to encode this as a set of symbolic causal rules expressed in terms of the common sense ontology described above
  - These causal rules enable conceptual abstraction
  - The narrative structure of the ongoing situation needs to be mapped to the causal structure of a set of previously encountered situations
    – Carry out analogical inference at a more abstract level, and thereby facilitate the transfer of expertise from one domain to another

Towards Deep Symbolic Reinforcement Learning [Garnelo et al ‘16]

- Game environment
  - The agent (shaped as a ‘+’) has to learn either to avoid or to collect objects depending on their shape.
  - Once the agent reaches an object using one of four possible move actions (up, down, left, or right), this object disappears and the agent obtains either a positive or a negative reward.
  - Encountering a circle (‘o’) results in a negative reward while collecting a cross (‘x’) yields a positive reward

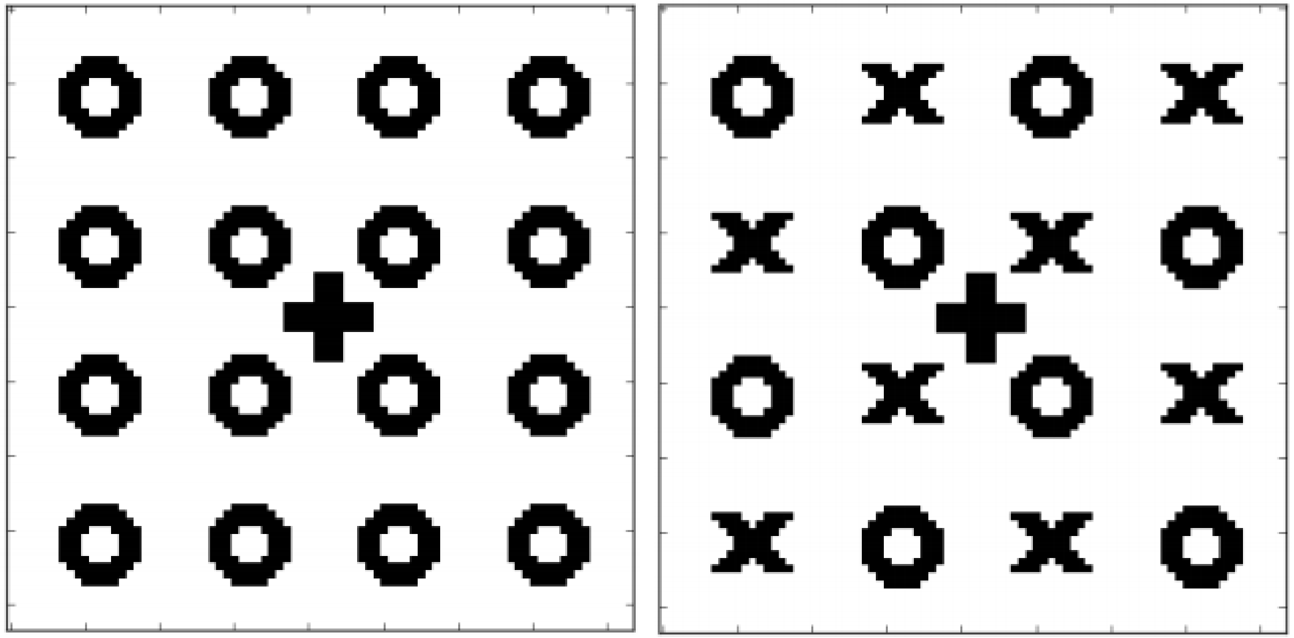


Towards Deep Symbolic Reinforcement Learning [Garnelo et al ‘16]

- The four different game environments
  - **Variant 1**
    - In this environment there are only objects that return negative rewards (‘o’) and they are positioned in a grid across the screen. This layout is the same for every new game. Encountering an object returns a score of -1 and at the beginning of the game the player is located in the middle of the board.
  - **Variant 2.**
    - The layout is the same as in version 1 but there are two types of objects. As before, circles give -1 points and we introduce crosses that return 1 points.
  - **Variant 3.**
    - As in version 1 this game only contains objects that return a negative reward. In order to increase the difficulty of the learning process however, the position of these objects is determined at random and changes at every new game.
  - **Variant 4.**
    - This version combines the randomness from environment 3 and the different object types from version 2.

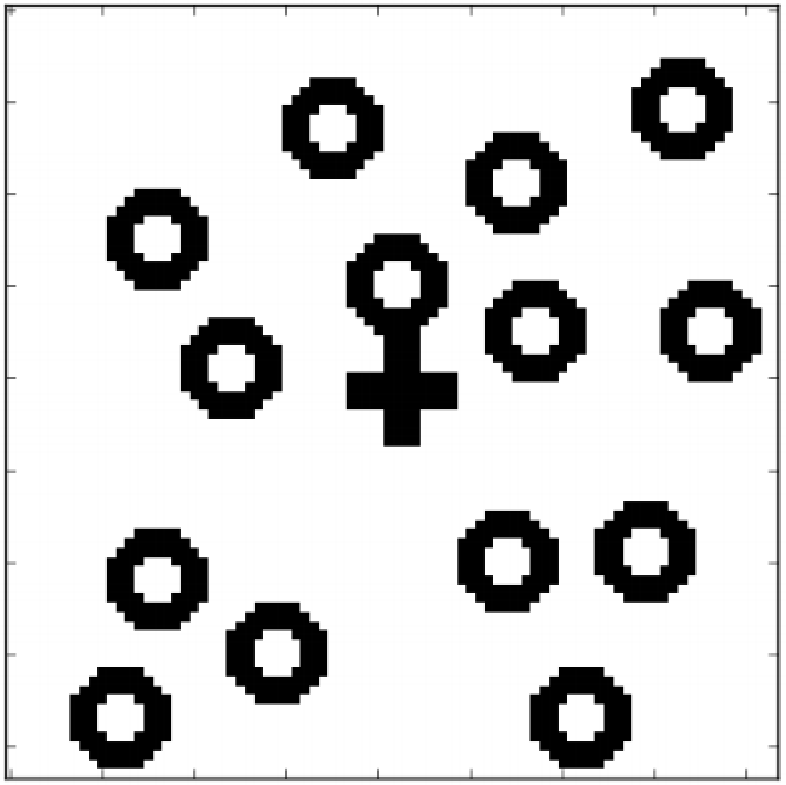
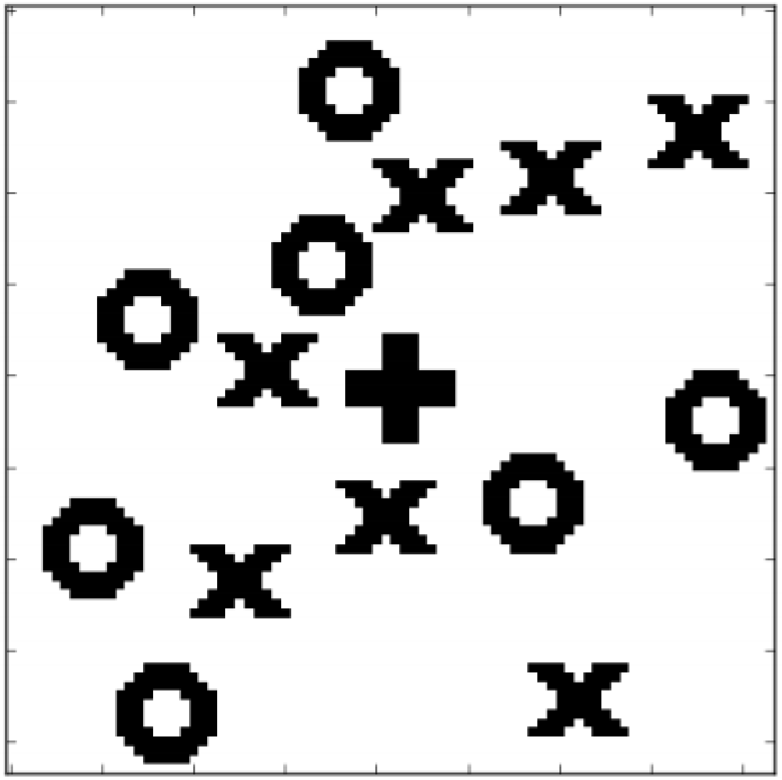
Towards Deep Symbolic Reinforcement Learning [Garnelo et al '16]

- The four different game environments
  - The agent is represented by the '+' symbol. The static objects return positive or negative reward depending on their shape ('x' and 'o' respectively).



Towards Deep Symbolic Reinforcement Learning [Garnelo et al ‘16]

- The four different game environments
  - The agent is represented by the ‘+’ symbol. The static objects return positive or negative reward depending on their shape (‘x’ and ‘o’ respectively).

Towards Deep Symbolic Reinforcement Learning [Garnelo et al ‘16]

- Low-level symbol generation
  - Generate, in an unsupervised manner, a set of symbols that can be used to represent the objects in a scene
  - Use a CNN, since such networks are well-suited to feature extraction, especially from images.
    - Train a convolutional autoencoder on 5000 randomly generated images of varying numbers of game objects sattered across the screen
  - The CNN consists of a 5x5 convolutional layer followed by a 2x2 pooling layer plus the corresponding decoding layers
  - Directly use the activations across features in the middle layer of the CNN for the detection of the objects in the scene.

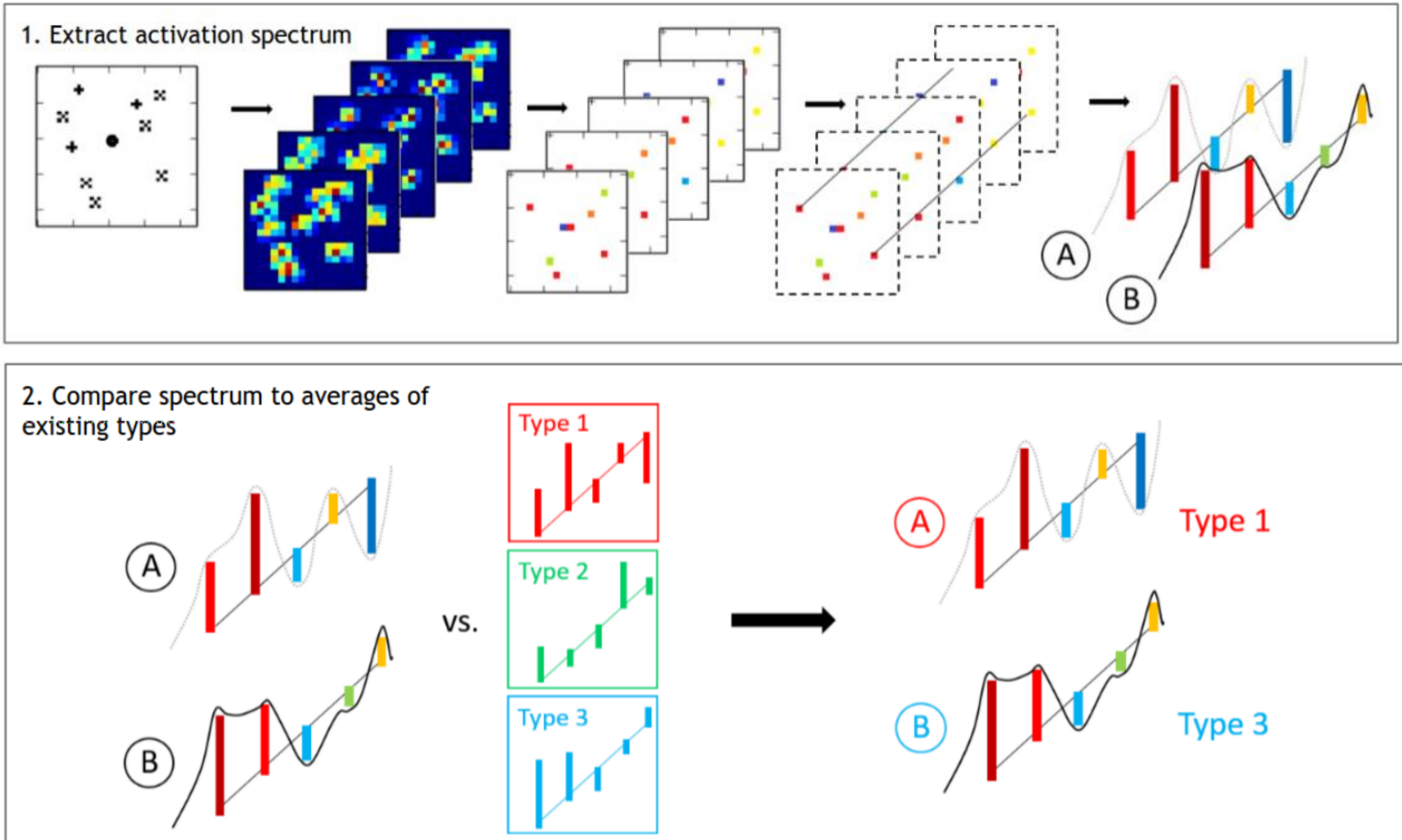
Towards Deep Symbolic Reinforcement Learning [Garnelo et al '16]

- Low-level symbol generation
  - Object detection and characterization
    - 1) first select, for each pixel, the feature with the highest activation
    - 2) Then threshold these activation values, forming a list of those that are sufficiently salient
      - Ideally, each member of this list is a representative pixel for a single object.
    - 3) The objects identified this way are then assigned a symbolic type according to the geometric properties computed by the autoencoder
      - Procedures for the object identification
      - Compare the activation spectra of the salient pixels across features.
        - » This comparison is carried out using the sum of the squared distances, which involves setting an ad hoc threshold for the maximal ditance between two objects of the same type.

➔ The information extracted at this stage consists of a symbolic representation of the positions of salient objects in the frame along with their types

Towards Deep Symbolic Reinforcement Learning [Garnelo et al '16]

Unsupervised extraction of low-level symbols from the information provided by the convolutional autoencoder


1. Extract activation spectrum
A
B
2. Compare spectrum to averages of existing types
A
B
vs.
Type 1
Type 2
Type 3
A
Type 1
B
Type 3

Towards Deep Symbolic Reinforcement Learning [Garnelo et al '16]

- Representation building
  - Track the low-level symbols across frames in order to observe and learn from their dynamics
  - Take account of the first common sense prior: object persistence across time
    - Based on three measures
  - 1) **Spatial proximity**
    - Build in the notion of continuity by defining the likelihood to be inversely proportional to the distance between two objects in consecutive frames

$$L_{dist} = \frac{1}{1+d}$$

the Euclidean distance between two objects i_1^t and i_2^t in consecutive frames t and $t+1$ respectively.

Towards Deep Symbolic Reinforcement Learning [Garnelo et al ‘16]

- Representation building
  - 2) **Type transitions**
    - Given the types of two objects $\tau(i_1^t) = \tau_{i1}$ and $\tau(i_2^t) = \tau_{i2}$ in consecutive frames, we can determine the probability that they are the same object that has changed from one type to the other
      - By learning a transition matrix T from previously observed frames

$$L_{trans} = T_{\tau_{i1},\tau_{i2}}$$

    - Introduce the object type 0: corresponds to ’non-existent’
      - To describe all transitions, including the ones that correspond to objects appearing and disappearing

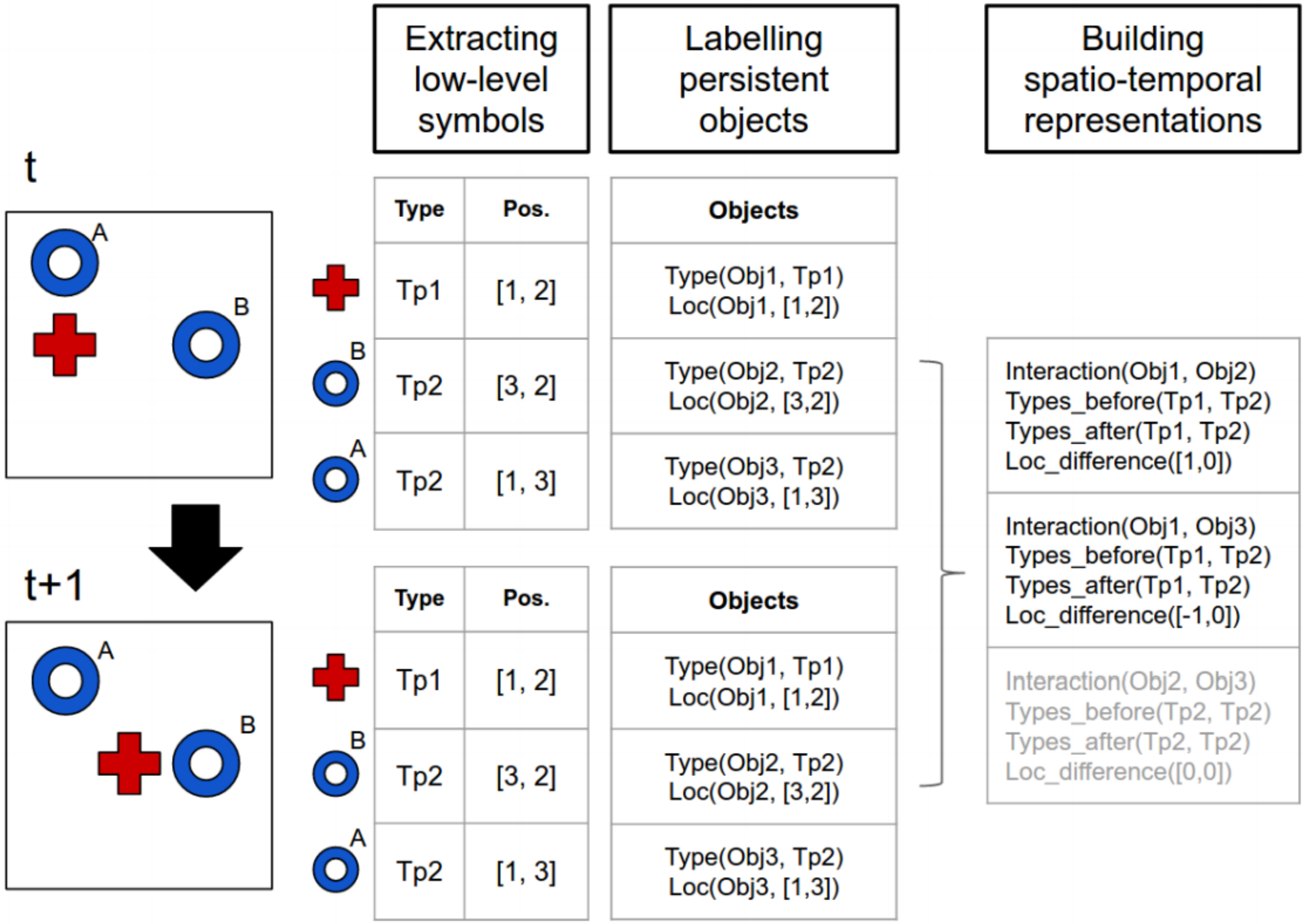
Towards Deep Symbolic Reinforcement Learning [Garnelo et al '16]

- Representation building
  - 3) **Neighbourhood**
    - The neighbourhood of an object will typically be similar from one frame to the next.
    - ΔN: The difference in the number of neighbours between two objects
      - Define a neighbour to be any object, i_n, within a distance $d_{\max}$ of another object i_1.

$$L_{neigh} = \frac{1}{1 + \Delta N}$$

$$L = w_1 L_{dist} + w_2 L_{trans} + w_3 L_{neigh}$$

Towards Deep Symbolic Reinforcement Learning [Garnelo et al '16]



Extracting low-level symbols

Labelling persistent objects

Building spatio-temporal representations

t

	Type	Pos.
✚	Tp1	[1, 2]
◯ B	Tp2	[3, 2]
◯ A	Tp2	[1, 3]

Objects
Type(Obj1, Tp1) Loc(Obj1, [1,2])
Type(Obj2, Tp2) Loc(Obj2, [3,2])
Type(Obj3, Tp2) Loc(Obj3, [1,3])

t+1

	Type	Pos.
✚	Tp1	[1, 2]
◯ B	Tp2	[3, 2]
◯ A	Tp2	[1, 3]

Objects
Type(Obj1, Tp1) Loc(Obj1, [1,2])
Type(Obj2, Tp2) Loc(Obj2, [3,2])
Type(Obj3, Tp2) Loc(Obj3, [1,3])

Interaction(Obj1, Obj2) Types_before(Tp1, Tp2) Types_after(Tp1, Tp2) Loc_difference([1,0])
Interaction(Obj1, Obj3) Types_before(Tp1, Tp2) Types_after(Tp1, Tp2) Loc_difference([-1,0])
Interaction(Obj2, Obj3) Types_before(Tp2, Tp2) Types_after(Tp2, Tp2) Loc_difference([0,0])

Towards Deep Symbolic Reinforcement Learning [Garnelo et al ‘16]

- **Symbolic interactions and dynamics**
  - The final, reinforcement learning stage of the algorithm will require information about the dynamics of objects and their spatial interactions
  - 1) consider the difference between frames rather than working with single frames, thus moving to a temporally extended representation
  - 2) Represent the positions of objects relative to other objects rather than using absolute coordinates.
    - only record relative positions of objects that lie within a certain maximum distance of each other.

Towards Deep Symbolic Reinforcement Learning [Garnelo et al ‘16]

- Approximate the optimal policy using tabular Q-learning with the update rule for the interaction between objects of types i and j

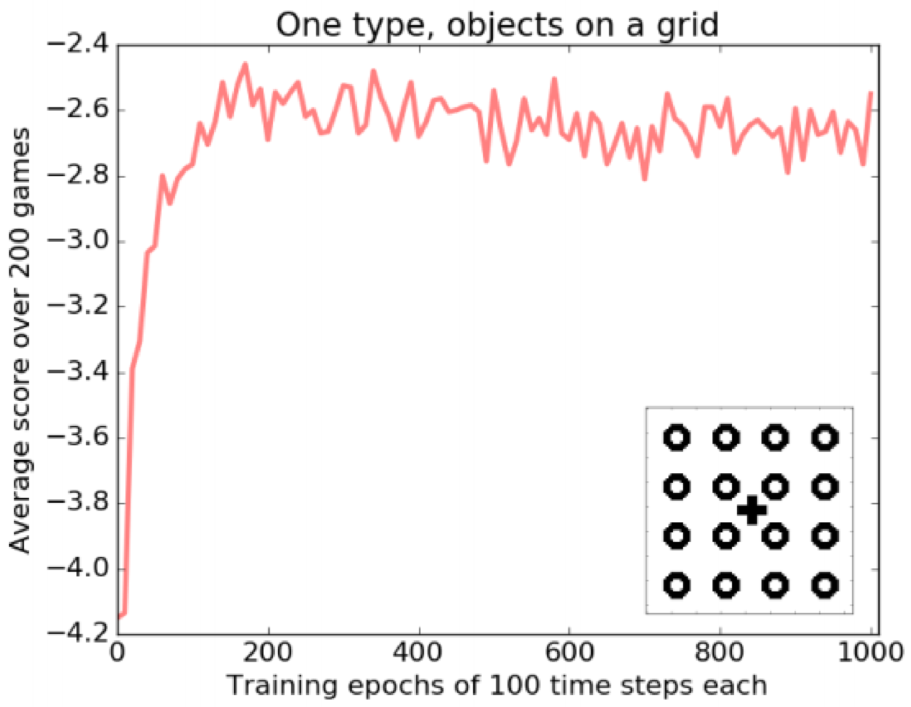
$$Q^{ij}(s_t^{ij}, a_t) \leftarrow Q^{ij}(s_t^{ij}, a_t) + \alpha \left(r_{t+1} + \gamma(\max_a Q^{ij}(s_{t+1}^{ij}, a) - Q^{ij}(s_t^{ij}, a_t)) \right)$$

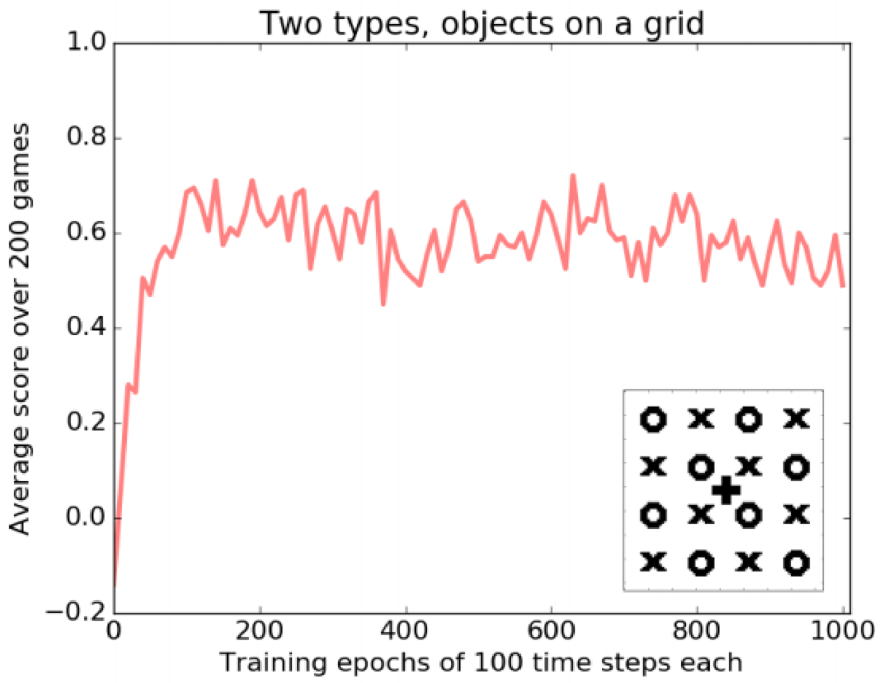
- choose the next action that will return the highest reward overall

$$a_{t+1} = \arg\max_a (\sum_Q (Q(s_{t+1}, a))$$

Towards Deep Symbolic Reinforcement Learning [Garnelo et al '16]

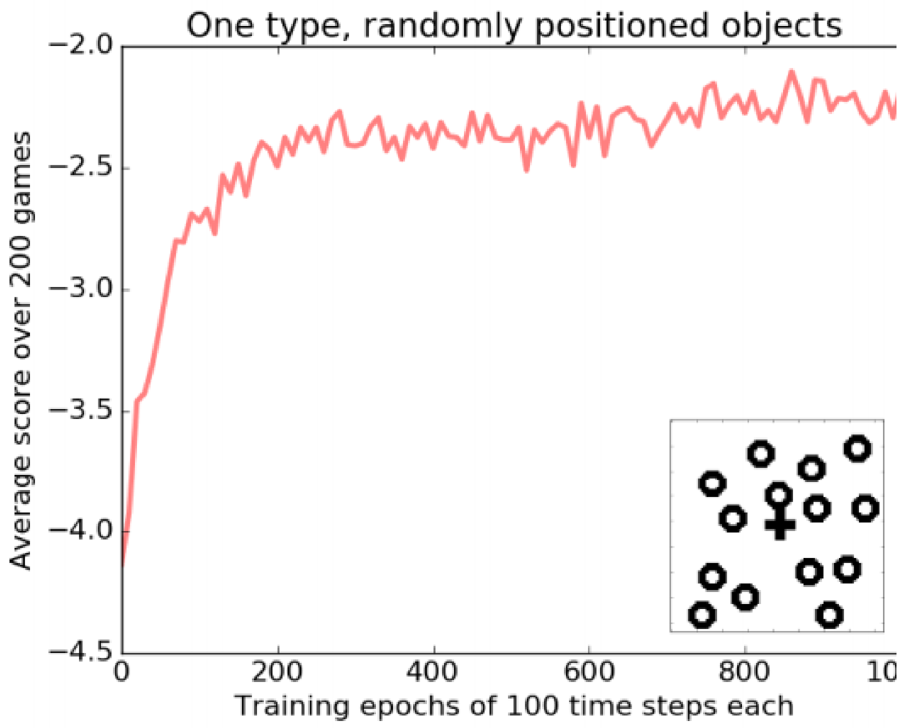
- Experiment results

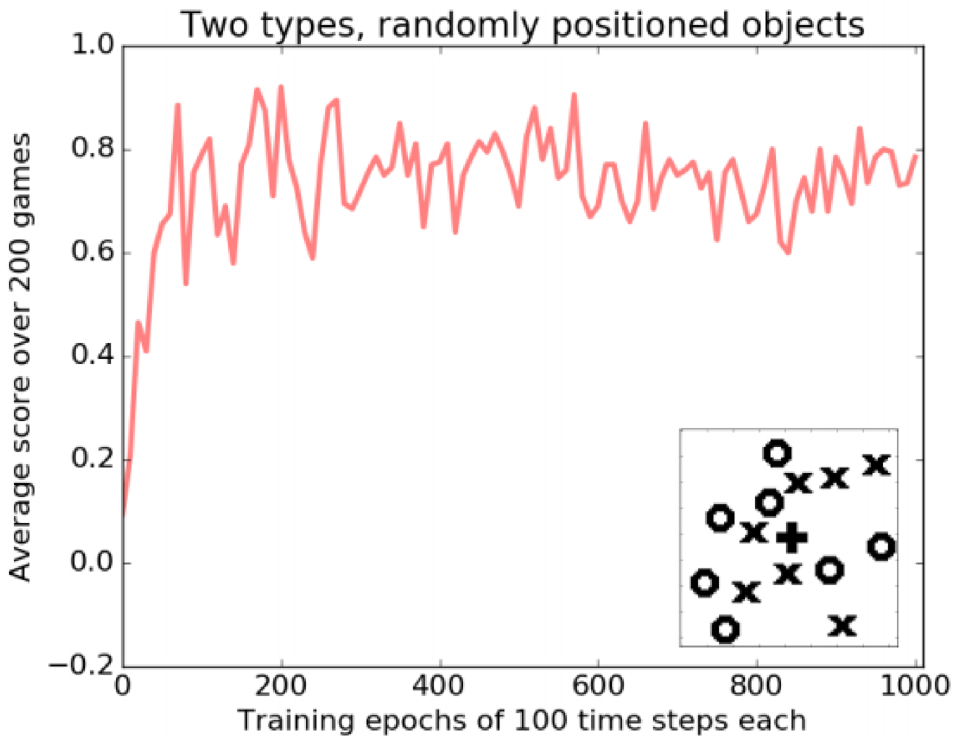




Towards Deep Symbolic Reinforcement Learning [Garnelo et al '16]

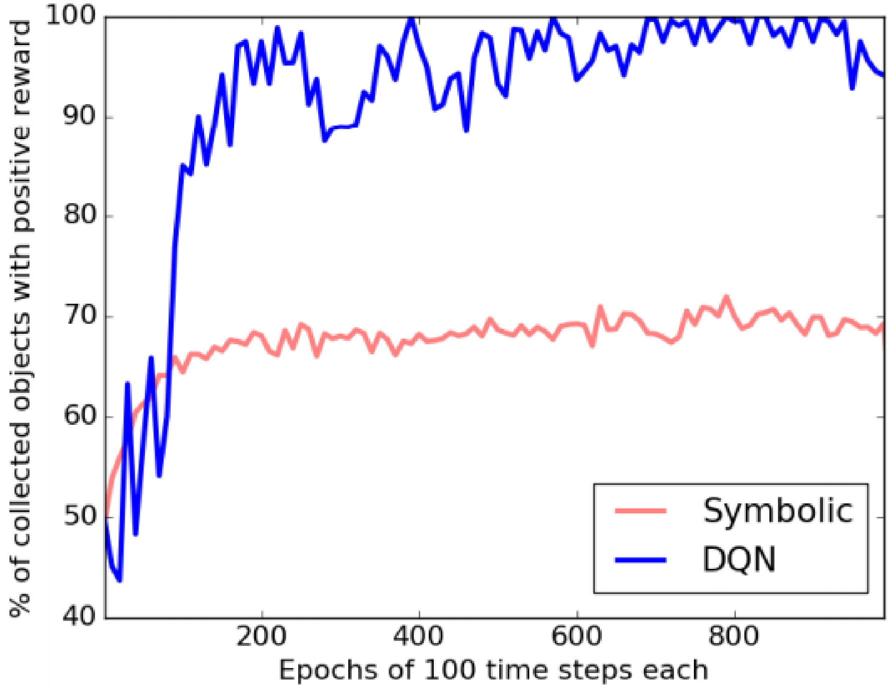
- Experiment results



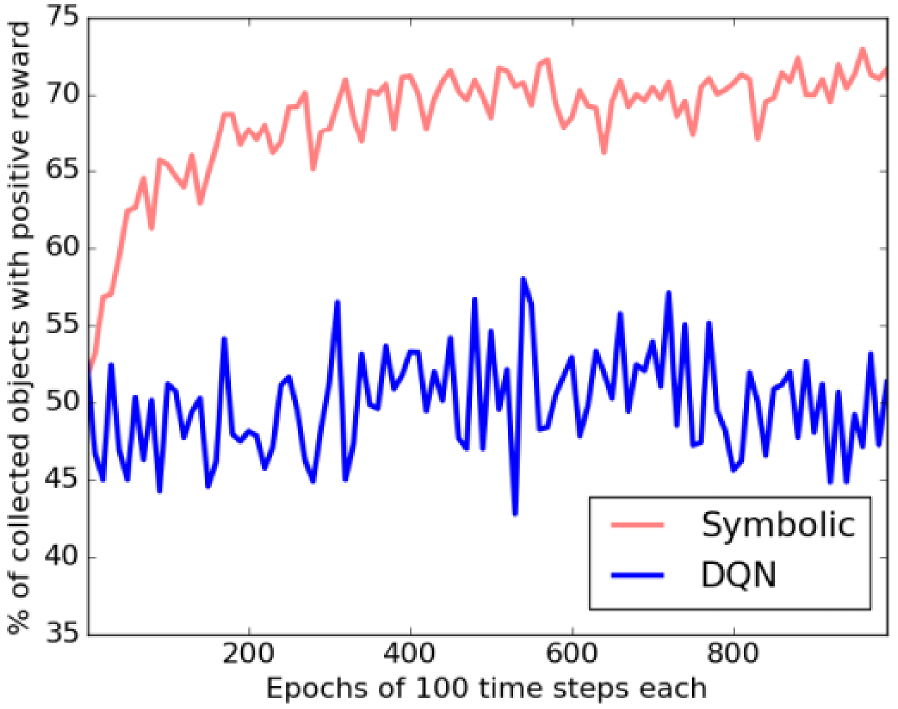


Towards Deep Symbolic Reinforcement Learning [Garnelo et al '16]

- Comparison between DQN and symbolic approach
  - Average percentage of objects collected over 200 games



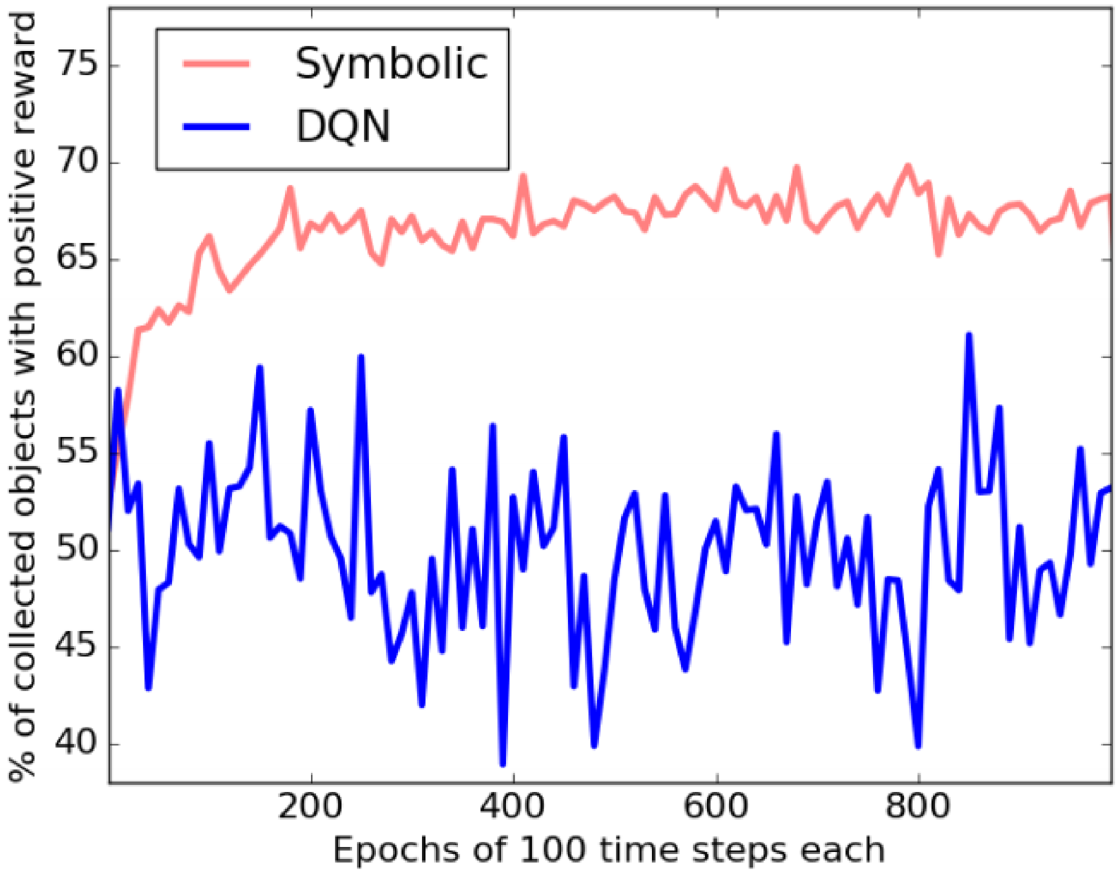
The grid environment



The random environment

Towards Deep Symbolic Reinforcement Learning [Garnelo et al '16]

- Average percentage of objects collected over 200 games that return positive reward by an agent that is trained on the grid environment and tested on random environments. (domain transfer)



TensorLog [Cohen '16]

- Goal: Integrate probabilistic logical reasoning with the powerful infrastructure of the deep learning
  - To enable deep learners to incorporate first-order probabilistic KBs
  - conversely, to enable probabilistic reasoning over the outputs of deep learners
- Deductive database (DDB): a database DB with a theory T defines a set of facts $f_1, \cdots, f_n$ which can be derived by accessing the database and reasoning using T
- Probabilistic deductive database (PrDDB)
  - A soft extension of a DDB, where
  - Derived facts have a numeric confidence, augmenting DB with a set of parameters Θ
  - Computation of confidences is computationally expensive, often not conductive to learning Θ

TensorLog [Cohen ‘16]

- TensorLog:
  - Probabilistic deductive database (PrDDB) in which reasoning uses a differentiable process
  - Each clause is converted into certain type of factor graph
    - Each logical variable is associated with a random variable in the factor graph
    - Each literal is associated with a factor
  - Inference is linear in database size and the num of massage-passing steps used in BP
  - Inference is also differentiable
  - Subsumes some prior probabilistic logic programming models, including several variants of stochastic logic programs (SLPs)

TensorLog [Cohen '16]

- A motivating example: A simple theory for QA against a KB
    - The goal of learning is to find appropriate weights for the soft predicate facts
    - TensorLog: can learn from 10,000 questions against a KB of 420,00 triples, in around 200 seconds per epoch in a single GPU

```
answer(Question,Answer) :-
    classification(Question,aboutActedIn),
    mentionsEntity(Question,Entity), actedIn(Answer,Entity).
answer(Question,Answer) :-
    classification(Question,aboutDirected),
    mentionsEntity(Question,Entity), directed(Answer,Entity).
answer(Question,Answer) :-
    classification(Question,aboutProduced),
    mentionsEntity(Question,Entity), produced(Answer,Entity).
...
mentionsEntity(Question,Entity) :-
    containsNGram(Question,NGram), matches(NGram,Name),
    possibleName(Entity,Name), popular(Entity).

classification(Question,Y) :-
    containsNGram(Question,NGram), indicatesLabel(NGram,Y).
matches(NGram,Name) :-
    containsWord(NGram,Word), containsWord(Name,Word), important(Word).
```

For NLP tasks, the KB stores word n-grams in the question, the strings that are possible names of an entity, and the words that are contained in these names and n-grams

Soft KB predicates

TensorLog [Cohen '16]: Motivation

- Inefficiency in integration of PrDDB and deep learning
  - Integration of probabilistic logics into deep learners is that most existing first-order probabilistic logics are not easily adapted to evaluation on a GPU
- The most common approach to first-order inference: the grounding
  - To ground a first-order logic by converting it to a zeroth-order format, such as a boolean formula or a probabilistic graphical model

$$p(X,Y) \leftarrow q(Y,Z), r(Z,Y)$$

$$\Big\Downarrow \text{ Grounding}$$

Even this small rule gives a grounding of size $o(|C|^3)$

$$\bigvee_{\exists x,y,z \in C} (p(x,y) \vee \neg q(y,z) \vee \neg r(z,y))$$

the set of objects in the KB

But, groundings can be very large: a grounding of size $o(|C|^n)$ is produced by a rule:

$$p(X_0, X_n) \leftarrow q_1(X_0, X_1), q_2(X_1, X_2), \ldots, q_n(X_{n-1}, X_n)$$

TensorLog [Cohen ’16]: Main Idea

- 1) Propose the use of a restricted family of probabilistic deductive databases (PrDDBs) called polytree-limited stochastic deductive knowledge graphs (ptree-SDKGs)
  - Ptree-SDKGs are tractable, but still reasonably expressive
  - Ptree-SDKGs are in some sense maximally expressive
    - we cannot drop the polytree restriction, or switch to a more conventional possible-worlds semantics, without making inference intractable
- 2) Present an inference algorithm for ptree-SDKGs
  - performs inference with a dynamic-programming method
    - Formalize as belief propagation on a certain factor graph

TensorLog [Cohen ’16]

- Deductive database (DDB)

```
1. uncle(X,Y):-child(X,W),brother(W,Y).
2. uncle(X,Y):-aunt(X,W),husband(W,Y).
3. status(X,tired):-child(W,X),infant(W).
```

child(liam,eve)	0.99
child(dave,eve)	0.99
child(liam,bob)	0.75
husband(eve,bob)	0.9
infant(liam)	0.7
infant(dave)	0.1
aunt(joe,eve)	0.9
brother(eve,chip)	0.9

TensorLog [Cohen ’16]

- Deductive database (DDB)
  - $\mathcal{DB}$: A database, which is a set $\{f_1, \ldots, f_N\}$ oг ground facts
  - $\mathcal{T}$: A theory, which is a set of function-free Horn clauses
  - A clause is written as: $A\text{:-}B_1, \ldots, B_k$
    - A: the head of the clause, $B_1, \ldots, B_k$: the body
    - A & B_i: literals
    - Clauses can be understood as logical implications
  - A literal has the form: $p(X_1, \ldots, X_k)$
    - p: a predicate symbol
    - X_i: either logical variables or database constants
    - *Arity*: the number of arguments k to a literal
  - $\mathcal{C}$: the set of all database constants
    - We assume that constants appear only in the database, not in the theory

TensorLog [Cohen ’16]

- Deductive database (DDB)
  - *Knowledge graph (KG)*
    - The database where all literals are binary or unary
    - *A deductive knowledge graph (DKG)*
      - The program for KG
  - σ : a substitution, a mapping from logical variables to constants in C
  - $\sigma(L)$: the result of replacing all logical variables X in the literal L with $\sigma(X)$
  - A set of tuples S is deductively closed with respect to the clause $A \leftarrow B_1, \ldots, B_k$ iff for all substitutions σ either $\sigma(A) \in S$ or $\exists B_i : \sigma(B_i) \notin S$

Deductively closed ➔ KG completeness

TensorLog [Cohen '16]

- Deductive database (DDB)

```
1. uncle(X,Y):-child(X,W),brother(W,Y).
2. uncle(X,Y):-aunt(X,W),husband(W,Y).
3. status(X,tired):-child(W,X),infant(W).
```

`child(liam,eve)`	0.99
`child(dave,eve)`	0.99
`child(liam,bob)`	0.75
`husband(eve,bob)`	0.9
`infant(liam)`	0.7
`infant(dave)`	0.1
`aunt(joe,eve)`	0.9
`brother(eve,chip)`	0.9

Not deductively closed with respect to the clause 1
unless it also contains `uncle(chip,liam)` and `uncle(chip,dave)`

TensorLog [Cohen '16]

- Deductive database (DDB)
    - E.g.) Deductively closed: If S contains the facts in our example, S is not deductively closed with respect to the clause 1 unless it also contains $uncle(liam, chip)$ and $uncle(dave, chip)$
  - $Model(\mathcal{DB}, \mathcal{T})$: the smallest superset of DB that is deductively closed with respect to every clause in T
    - This least model is unique, and in the usual DDB semantics
    - a ground fact f is considered "true" iff $f \in Model(\mathcal{DB}, T)$
  - Bottom-up inference
    - Explicitly generates the set $Model(DB, T)$ iteratively
      - Repeatedly extends a set of facts S, which initially contains just the database facts, by looking for rules which "fire" on S and using them derive new facts
    - This can be much larger than the original database

TensorLog [Cohen ’16]

- Deductive database (DDB)
  - Top-down inference
    - Does not compute a least model explicitly
    - instead, it takes as input a querv fact f and determines whether f is derivable, i.e., if $f \in Model(\mathcal{DB}, \mathcal{T})$
    - E.g.) find all values of Y such that $uncle(joe, Y)$ holds:
      - Formally, given $Q = \texttt{uncle(joe,Y)}$ find all $f \in Model(\mathcal{DB}, \mathcal{T})$ which are instances of Q, where an f is defined to be an instance of Q iff $\exists \sigma : f = \sigma(Q)$
    - a unit clause: a fact clause $p(a, b) \leftarrow$
    - $\mathcal{T}^{+\mathcal{DB}}$: denote the theory T augmented with unit clauses for each database fact

TensorLog [Cohen '16]

- Top-down theorem prover
  - Root vertex: a pair $(S, L) = (Q, [Q])$
  - For any vertex (S, L) where $L = [G_1, \cdots, G_n]$, there is a child vertex (S', L') for each rule $A \leftarrow B_1, \ldots, B_k \in \mathcal{T}^{+\mathcal{DB}}$ and σ for which $\sigma(G_i) = \sigma(A)$. Then, $S' = \sigma(S)$

    $$L' = [\sigma(G_1), \ldots, \sigma(G_{i-1}), \sigma(B_1), \ldots, \sigma(B_k), \sigma(G_{i+1}), \ldots, \sigma(G_n)]$$

    - L' is empty: a solution vertex
    - L' is smaller than L if the clause selected is a unit clause (i.e., a fact).
  - In any solution vertex (S, L), if S contains no variables, then S is an instance of Q and is in $Model(T, DB)$

TensorLog [Cohen '16]

- Top-down theorem prover
  - An example proof tree.

$$\begin{tikzcd}
 & (\text{S=uncle(liam,Y), L=[uncle(liam,Y)]}) \arrow[d] & \\
 & (\text{S=uncle(liam,Y), L=[child(liam,W),brother(W,Y)]}) \arrow[dl] \arrow[dr] & \\
(\text{S=uncle(liam,Y), L=[brother(bob,Y)]}) \arrow[d] & & (\text{S=uncle(liam,Y), L=[brother(eve,Y)]}) \arrow[d] \\
\textit{dead end} & & (\text{S=uncle(liam,chip), L=[]})
\end{tikzcd}$$

TensorLog [Cohen '16]

- Stochastic logic programs (SLPs) [Cussens '01]
  - Putting probabilistic reasoning in first-order logics
    - Theory T is extended by associating with each rule r a non-negative scalar weight θ_r
  - The weight of an edge: when a rule r is used to create an edge $(S, L) \rightarrow (S', L')'$, this edge is given weight θ_r
  - The weight of a path $v_1 \rightarrow \ldots \rightarrow v_n$:
    - The product of the weights of the edges in the path
  - The weight of a node v in the proof graph for Q
    - the sum of the weights of the paths from the root node $v_0 = (Q, [Q])$ to v

TensorLog [Cohen '16]

- If $r_{v,v'}$ is the rule used for the edge from v to v'
- The weight of $w_Q(v_n)$

$$w_Q(v_n) \equiv \sum_{v_0 \rightarrow \ldots \rightarrow v_n} \prod_{i=0}^{n-1} \theta_{r_{v_i,v_{i+1}}}$$

- The weight of an answer f to query Q

$$w_Q(f) \equiv \sum_{v:v=(f,[])} w_Q(v)$$

- The conditional probability distribution over answers f to the query Q:

$$\Pr(f|Q) \equiv \frac{1}{Z} w_Q(f)$$

TensorLog [Cohen ’16]

- Stochastic logic programs (SLPs)
  - Thought of as logic-program analogs to probabilistic programming languages like Church [Goodman et al ’12]
  - Normalized SLPs are also conceptually quite similar to stochastic grammars such as PCFG
- Stochastic deductive knowledge graph (SDKG)
  - Three restrictions on SLPs:
  - 1) restrict the program to be in DDB form
    - Consist of a theory T which contains function-free clauses, and a database DB (of unit clauses)
  - 2) restrict all predicates to be unary or binary
  - 3) restrict the clauses in the theory T to have weight 1, so that the only meaningful weights are associated with database facts.

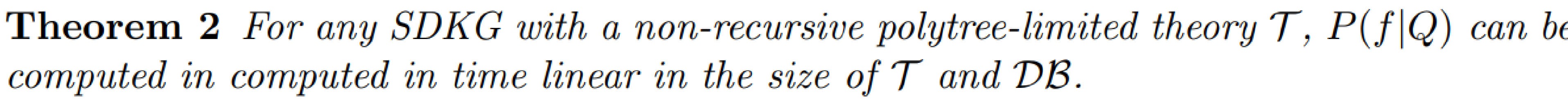
TensorLog [Cohen ’16]

- Complexity of reasoning with stochastic deductive KGs
  - The similarity between SLPs and probabilistic context-free grammars suggests that efficient schemes might exist, since there are efficient dynamic programming methods for probabilistic parsing. Unfortunately, this is not the case:
  - even for the restricted case of SDKGs, computing P(f|Q) is #P-hard

Theorem 1 *Computing $P(f|Q)$ (relative to a SDKG $\mathcal{T}, \mathcal{DB}$) for all possible answers f of the query Q is #P-hard, even if there are only two such answers, the theory contains only two non-recursive clauses, and the KG contains only 13 facts.*

TensorLog [Cohen '16]

- Fortunately, one further restriction makes SLP theorem-proving efficient.
  - For a theory clause $r = A \leftarrow B_1, \ldots, B_k$ define the literal influence graph for r to be a graph where each B_i is a vertex, and there is an edge from B_i to B_j iff they share a variable
  - A graph is a *polytree* iff there is at most one path between any pair of vertices: if each strongly connected component of the graph is a tree
- A theory is polytree-limited iff the influence graph for every clause is a polytree ➔ This additional restriction makes inference tractable


Theorem 2 *For any SDKG with a non-recursive polytree-limited theory $\mathcal{T}$, $P(f|Q)$ can be computed in computed in time linear in the size of $\mathcal{T}$ and $\mathcal{DB}$.*

TensorLog [Cohen ’16]: Differentiable inference for polytree-limited SDKGs

- Inference for polytree-limited SDKGs
  - Formalize this method as belief propagation on a certain factor graph
    - The random variables in the factor graph correspond to possible bindings to a logical variable in a proof
    - The factors correspond to database predicates
  - Though simple, a novel method for first-order probabilistic inference
    - While other common methods use Bernoulli random variables which correspond to potential ground database facts (i.e., elements of the Herbrand base of the program.)

TensorLog [Cohen '16]: Differentiable inference for polytree-limited SDKGs

- Numeric encoding of PrDDB's and queries
  - $\mathbf{u}_c$ for a constant c ∈ C,
    - a one hot row-vector representation for c
    - $\boldsymbol{u}[c] = 1$ & $\boldsymbol{u}[c'] = 0$ for $c' \neq c$
  - $\mathbf{M}_p$: a sparse matrix for a binary predicate p

    $$\mathbf{M}_p[a,b] = \theta_{p(a,b)} \text{ if } p(a,b) \in \mathcal{DB}$$

  - $\mathbf{v}_q$: a row vector for a unary predicate q
  - Collectively, $\underbrace{M_{p_1}, \ \ldots, \ M_{p_n}}_{\text{Parameters}}$ are viewed as a three-dimensional tensor

TensorLog [Cohen '16]: Differentiable inference for polytree-limited SDKGs

- An *argument-retrieval* query
  - query of the form $p(c,Y)$ or $p(Y,c)$
    - p(c,Y) has an input-output mode of in,out (io)
    - p(Y, c) has out,in (oi)
  - The response to a query p(c,Y)
    - a distribution over possible substitutions for Y
    - encoded as a vector $\mathbf{v}_Y$ such that for all constants $d \in C$

$$\mathbf{v}_Y[d] = \Pr(p(c,d) | Q = p(x,Y), \mathcal{T}, \mathcal{DB}, \Theta)$$

  - Notation $\mathbf{v}_{Y|c}$: formally if $U_{p(c,Y)}$ is the set of facts f that "match" (are instances of) $p(c,Y)$

$$\mathbf{v}_{Y|c}[d] = \Pr(f = p(c,d) | f \in U_{p(c,Y)}, \mathcal{T}, \mathcal{DB}, \Theta) \equiv \frac{1}{Z} w_{\overset{Q = p(c,Y)}{Q}}(f = p(c,d))$$

TensorLog [Cohen ’16]: Differentiable inference for polytree-limited SDKGs

- More complex queries can be answered by extending the theory
  - To find `{Y: uncle(joe,X),husband(X,Y)}`
  - We add the clause $q1(Y)$ to the theory

    ```
    q1(Y):-uncle(joe,X),husband(X,Y)
    ```

  - and find the answer to `q1(Y)`

TensorLog [Cohen '16]: Differentiable inference for polytree-limited SDKGs

- $f^p_{\mathtt{io}}$: for predicate p, a query response function for al queries with predicate p and mode $\mathtt{io}$

$$f^p_{\mathtt{io}}(\mathbf{u}_c) \equiv \mathbf{v}_{Y|c}$$

- $g^p_{\mathtt{io}}$ the unnormalized version of this function, i.e., the weight of *f* according to $w_Q(f)$

$$g^p_{\mathtt{io}}(\mathbf{u}_c) \equiv w_Q(f)$$

- $\mathtt{any}$: special DB predicate

  $\mathtt{any}(a, b)$ is conceptually true for any pair of a,b

  $\mathbf{M}_{\mathtt{any}}$ need not be explicitly stored.

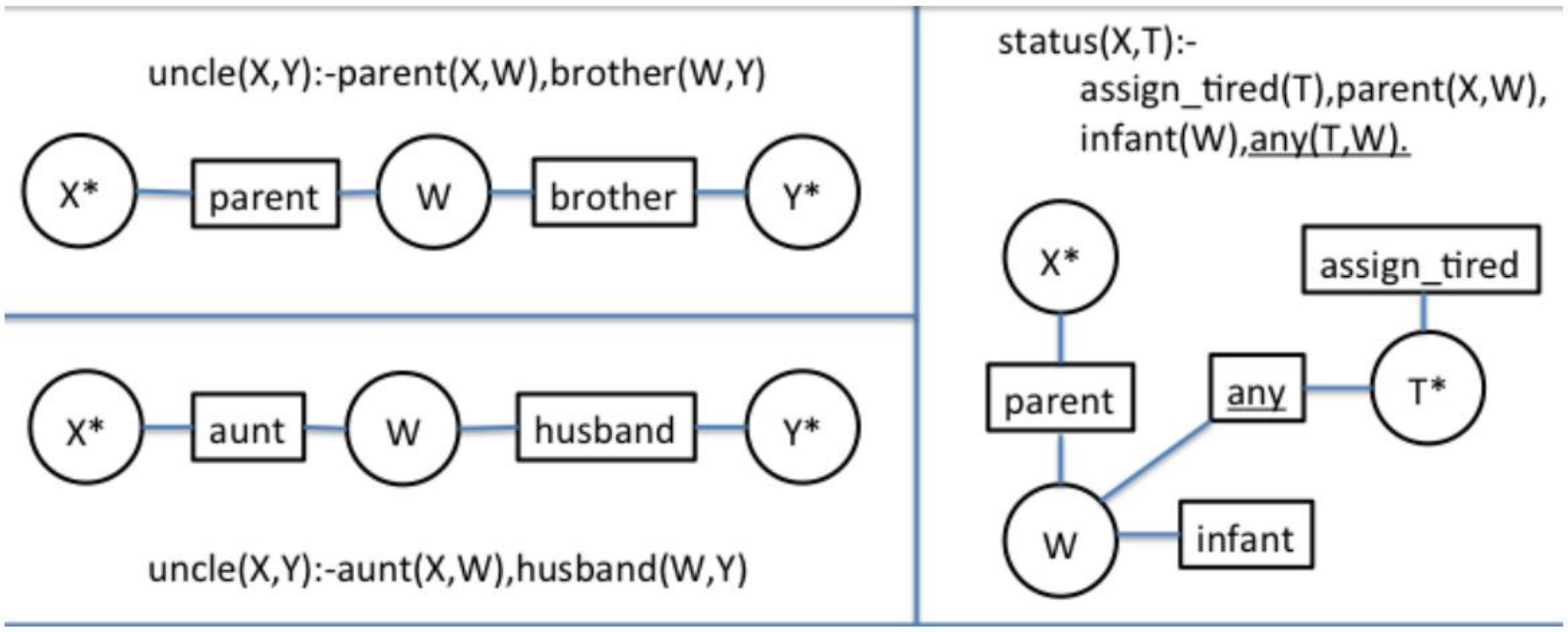
TensorLog [Cohen '16]: Differentiable inference for polytree-limited SDKGs

- **Efficient inference for one-clause theories**
  - Consider first programs containing only one non-recursive polytree-limited clause r
  - Build a factor graph G_r for r
    - for each logical variable W in the body ➔ there is a random variable W
    - for every literal $q(W_i, W_j)$ in the body of the clause, there is a factor with potentials M_q linking variables W_i and W_j
    - Finally, if the factor graph is disconnected, we add **any** factors between the components until it is connected

TensorLog [Cohen '16]: Differentiable inference for polytree-limited SDKGs

- Examples of factor graphs for the example theory

The variables appearing in the clause's head are starred


uncle(X,Y):-parent(X,W),brother(W,Y)
X*
parent
W
brother
Y*
X*
aunt
W
husband
Y*
uncle(X,Y):-aunt(X,W),husband(W,Y)
status(X,T):-
assign_tired(T),parent(X,W),
infant(W),any(T,W).
X*
assign_tired
parent
any
T*
W
infant

TensorLog [Cohen '16]: Differentiable inference for polytree-limited SDKGs

- **Efficient inference for one-clause theories**
  - BP on the factor graph G_r
    - Compute the conditional vectors $f^p_{\mathtt{io}}(\mathbf{u}_c)$ and $f^p_{\mathtt{oi}}(\mathbf{u}_c)$
    - E.g.) to compute $f^p_{io}(\boldsymbol{u}_c)$ for clause 1,
      - 1) set the message for the evidence variable X to $\boldsymbol{u}_c$,
      - 2) run BP
      - 3) read out as the value of f the marginal distribution for Y
    - The correctness of BP inference follow immediately from the convergence of belief propagation on factor graphs for polytrees [Kschischang et al '01]

TensorLog [Cohen '16]: Differentiable inference for polytree-limited SDKGs

- **Differentiable inference for one-clause theories**
  - "unroll" the message-passing steps into a series of operations

```
define compileMessage($L \rightarrow X$):
  assume wolg that $L = q(X)$ or $L = p(X_i, X_o)$
  generate a new variable name $\mathbf{v}_{L,X}$
  if $L = q(X)$ then
    emitOperation( $\mathbf{v}_{L,X} = \mathbf{v}_q$)
  else if $X$ is the output variable $X_o$ of $L$ then
    $\mathbf{v}_i$ = compileMessage($X_i \rightarrow L$)
    emitOperation( $\mathbf{v}_{L,X} = \mathbf{v}_i \cdot \mathbf{M}_p$ )
  else if $X$ is the input variable $X_i$ of $L$ then
    $\mathbf{v}_o$ = compileMessage($X_i \rightarrow L$)
    emitOperation( $\mathbf{v}_{L,X} = \mathbf{v}_o \cdot \mathbf{M}_p^T$ )
  return $\mathbf{v}_{L,X}$
```

If $L = p(X_o, X_i)$ then replace $\mathbf{M}_p$ with $\mathbf{M}_p^T$

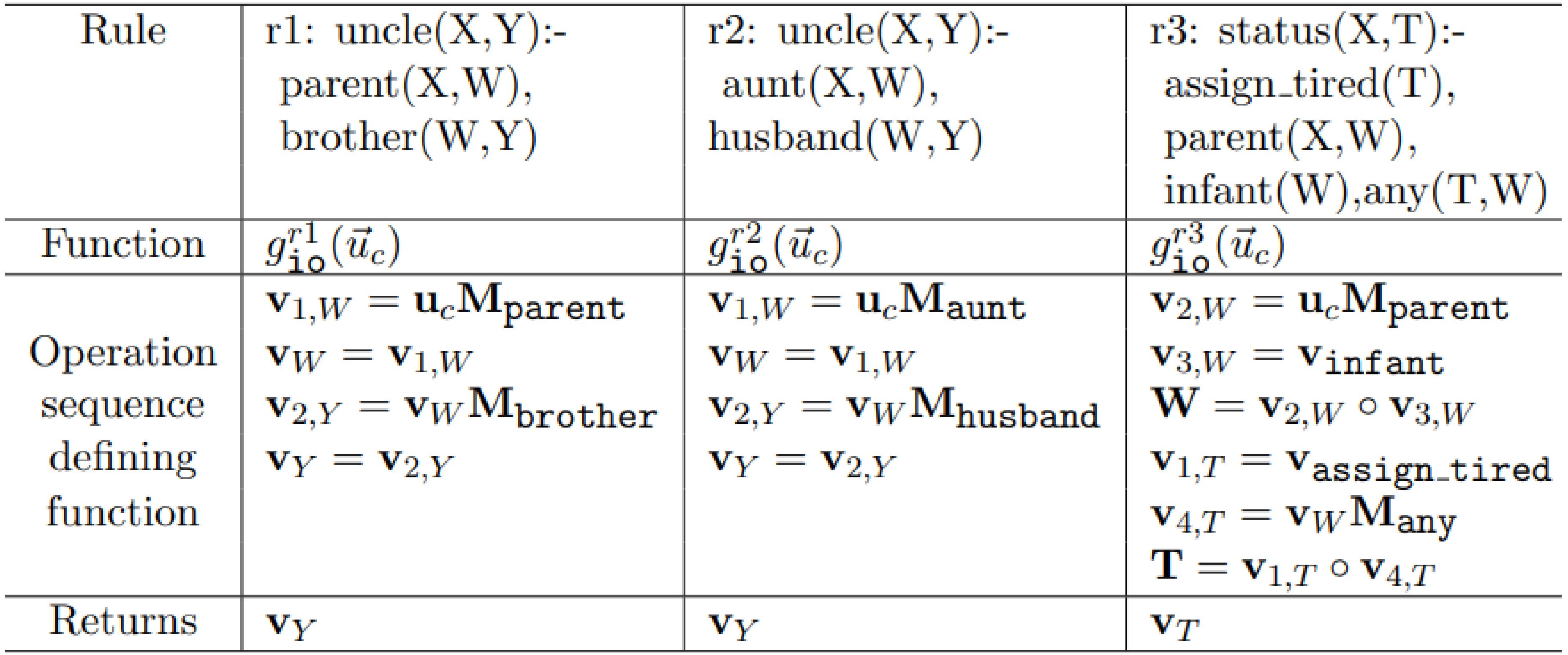
TensorLog [Cohen '16]: Differentiable inference for polytree-limited SDKGs

```
define compileMessage($X \to L$):
  if $X$ is the input variable $X$ then
    return $\mathbf{u}_c$, the input
  else
    generate a new variable name $\mathbf{v}_X$
    assume $L_1, L_2, \ldots, L_k$ are the
      neighbors of $X$ excluding $L$
    for $i = 1, \ldots, k$ do
      $\mathbf{v}_i$ = compileMessage($L_i \to X$)
    emitOperation($\mathbf{v}_X = \mathbf{v}_1 \circ \cdots \circ \mathbf{v}_k$)
    return $\mathbf{v}_X$
```

the Hadamard (componentwise) product, and if k = 0 an all-ones vector is returned

TensorLog [Cohen '16]: Differentiable inference for polytree-limited SDKGs

- **Chains of messages constructed for the three sample clauses**

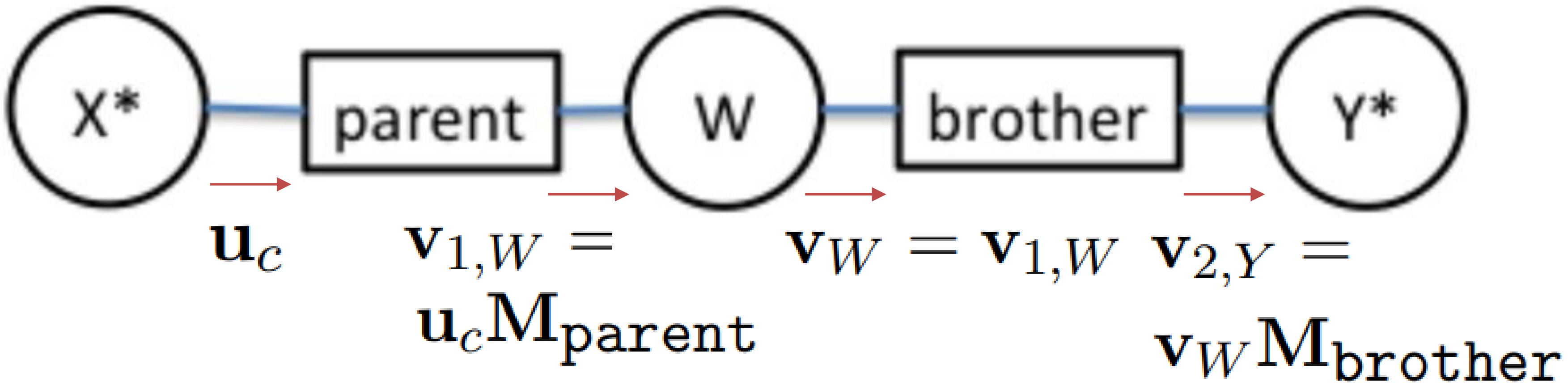


Rule	r1: uncle(X,Y):- parent(X,W), brother(W,Y)	r2: uncle(X,Y):- aunt(X,W), husband(W,Y)	r3: status(X,T):- assign_tired(T), parent(X,W), infant(W),any(T,W)
Function	$g^{r1}_{\mathtt{io}}(\vec{u}_c)$	$g^{r2}_{\mathtt{io}}(\vec{u}_c)$	$g^{r3}_{\mathtt{io}}(\vec{u}_c)$
Operation sequence defining function	$\mathbf{v}_{1,W} = \mathbf{u}_c\mathbf{M}_{\mathtt{parent}}$ $\mathbf{v}_W = \mathbf{v}_{1,W}$ $\mathbf{v}_{2,Y} = \mathbf{v}_W\mathbf{M}_{\mathtt{brother}}$ $\mathbf{v}_Y = \mathbf{v}_{2,Y}$	$\mathbf{v}_{1,W} = \mathbf{u}_c\mathbf{M}_{\mathtt{aunt}}$ $\mathbf{v}_W = \mathbf{v}_{1,W}$ $\mathbf{v}_{2,Y} = \mathbf{v}_W\mathbf{M}_{\mathtt{husband}}$ $\mathbf{v}_Y = \mathbf{v}_{2,Y}$	$\mathbf{v}_{2,W} = \mathbf{u}_c\mathbf{M}_{\mathtt{parent}}$ $\mathbf{v}_{3,W} = \mathbf{v}_{\mathtt{infant}}$ $\mathbf{W} = \mathbf{v}_{2,W} \circ \mathbf{v}_{3,W}$ $\mathbf{v}_{1,T} = \mathbf{v}_{\mathtt{assign\_tired}}$ $\mathbf{v}_{4,T} = \mathbf{v}_W\mathbf{M}_{\mathtt{any}}$ $\mathbf{T} = \mathbf{v}_{1,T} \circ \mathbf{v}_{4,T}$
Returns	$\mathbf{v}_Y$	$\mathbf{v}_Y$	$\mathbf{v}_T$

$$f^{p}_{\mathtt{io}}(\vec{u}_c) \equiv g^{r}_{\mathtt{io}}(\vec{u}_c)/\|g^{r}_{\mathtt{io}}(\vec{u}_c)\|_1$$

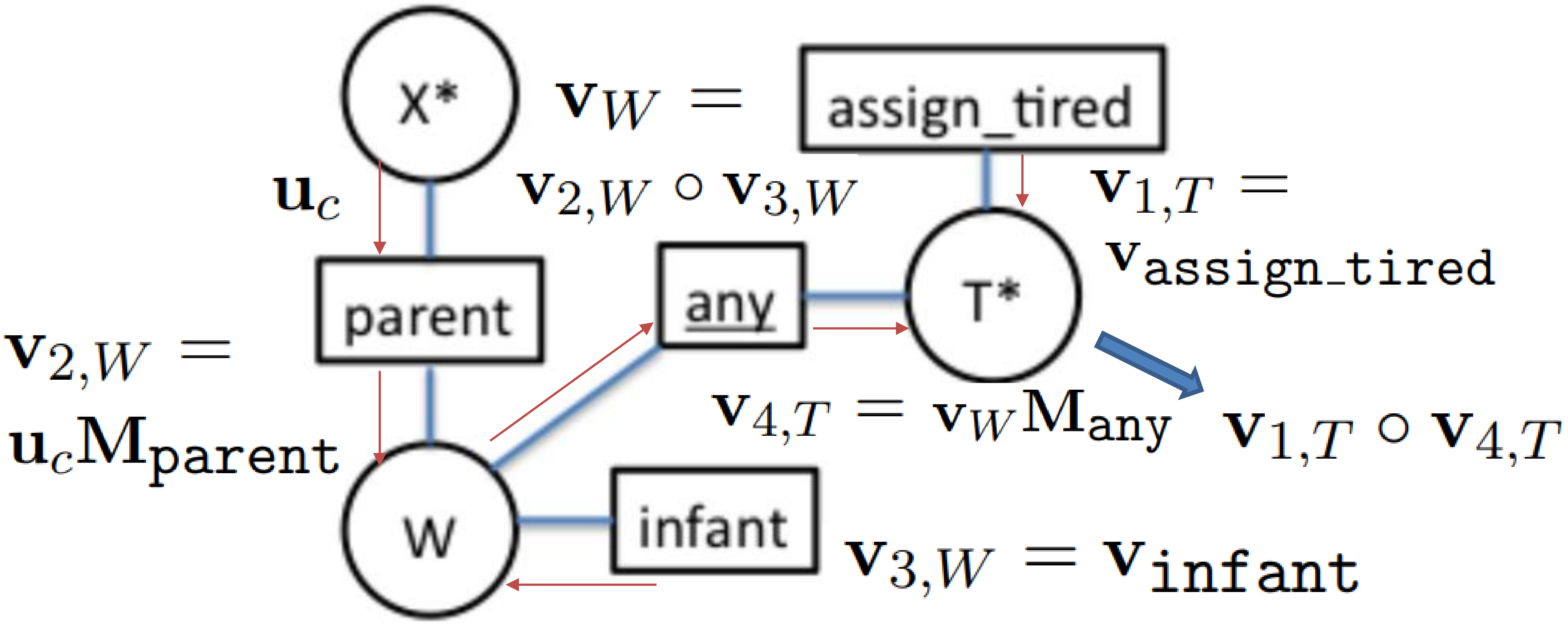
TensorLog [Cohen '16]: Differentiable inference for polytree-limited SDKGs

uncle(X,Y):-parent(X,W),brother(W,Y)


X*
parent
W
brother
Y*
$\mathbf{u}_c$
$\mathbf{v}_{1,W} = \mathbf{u}_c \mathrm{M}_{\texttt{parent}}$
$\mathbf{v}_W = \mathbf{v}_{1,W}$
$\mathbf{v}_{2,Y} = \mathbf{v}_W \mathrm{M}_{\texttt{brother}}$

TensorLog [Cohen '16]: Differentiable inference for polytree-limited SDKGs

```
status(X,T):-
    assign_tired(T),parent(X,W),
    infant(W),any(T,W).
```


X*
assign_tired
$\mathbf{v}_W =$
$\mathbf{v}_{2,W} \circ \mathbf{v}_{3,W}$
$\mathbf{u}_c$
parent
any
T*
$\mathbf{v}_{1,T} =$
$\mathbf{v}_{\mathtt{assign\_tired}}$
$\mathbf{v}_{2,W} =$
$\mathbf{u}_c \mathbf{M}_{\mathtt{parent}}$
$\mathbf{v}_{4,T} = \mathbf{v}_W \mathbf{M}_{\mathtt{any}}$
$\mathbf{v}_{1,T} \circ \mathbf{v}_{4,T}$
W
infant
$\mathbf{v}_{3,W} = \mathbf{v}_{\mathtt{infant}}$

TensorLog [Cohen '16]: Differentiable inference for polytree-limited SDKGs

- **Differentiable inference for one-clause theories**
  - Belief propagation to compute $f^p_{\mathtt{io}}(\mathbf{u}_c)$
    - Emit a series of operations, and return the name of a register that contains the unnormalized conditional probability vector for the output variable.
    - Use $g^r_{\mathtt{io}}(\vec{u}_c)$ for the unnormalized version of the query response function build from G_r

$$f^p_{\mathtt{io}}(\vec{u}_c) \equiv g^r_{\mathtt{io}}(\vec{u}_c)/\|g^r_{\mathtt{io}}(\vec{u}_c)\|_1$$

TensorLog [Cohen '16]: Differentiable inference for polytree-limited SDKGs

- **Extension to Multi-clause programs**
  - Extend to theories with many clauses
  - if there are several clauses with the same predicate symbol in the head, we simply sum the unnormalized query response functions
  - E.g) for the predicate **uncle**, defined by rules r_1 and r_2, we would define

$$g_{\texttt{io}}^{\texttt{uncle}} = g_{\texttt{io}}^{r1} + g_{\texttt{io}}^{r2}$$

  - This is equivalent to building a new factor graph G,
    - Approximately $\cup_i G_{ri}$, together global input and output variables,
    - plus a factor that constrains the input variables of the G_{ri}'s to be equal,
    - plus a factor that constrains the output variable of G to be the sum of the outputs of the G_{ri}'s.

TensorLog [Cohen '16]: Differentiable inference for polytree-limited SDKGs

- **Extension to Multi-clause programs**
  - A more complex situation is when the clauses for one predicate, p, use a second theory predicate q, in their body
  - E.g.) the case if aunt was also defined in the theory, rather than the database
    - Replace the message-passing operations $\mathbf{v}_Y = \mathbf{v}_X \mathbf{M}_q$ with $\mathbf{v}_Y = g^q_{\mathtt{io}}(\mathbf{v}_X)$, or $\mathbf{v}_Y = \mathbf{v}_X \mathbf{M}_q^T$ with $\mathbf{v}_Y = g^q_{\mathtt{oi}}(\mathbf{v}_X)$
    - This is equivalent to taking the factor graph for q and "splicing" it into the graph for p

TensorLog [Cohen '16]: Differentiable inference for polytree-limited SDKGs

- **Extension to Multi-clause programs**
  - Adding depth argument d
    - Allow function calls to recurse to a fixed maximum depth
    - Ensure function calls inside $g^p_{\mathtt{io},d}$ to q always call the next-deeper version of the function for q, i.e., $g^q_{\mathtt{io},d+1}$
  - Computationally, the algorithm we describe is quite efficient.
    - Assuming the matrices $\boldsymbol{M}_p$ exist, the additional memory needed for the factor-graph G_r is linear in the size of the clause r
      - The compilation to response functions is linear in the theory size and the number of steps of BP.
    - For ptree-SDKGs, G_r is a tree, the number of message-passing steps is also linear.
      - Message size is (by design) limited to |C|, and is often smaller in practice, due to sparsity or type restrictions

TensorLog [Cohen ’16]

```
tlog = tensorlog.simple.Compiler(db=”data.db”, prog=”rules.tlog”)
train_data = tlog.load_dataset(”train.exam”)
test_data = tlog.load_dataset(”test.exam”)
# data is stored dictionary mapping a function specification, like $p_{io}$,
# to a pair X, Y. The rows of X are possible inputs $f^p_{io}$, and the rows of
# Y are desired outputs.
function_spec = train_data.keys()[0]
# assume only one function spec
X,Y = train_data[function_spec]

# construct a tensorflow version of the loss function, and function used for inference
unregularized_loss = tlog.loss(function_spec)
f = tlog.inference(function_spec)

             ...
# run the optimizer for 10 epochs
session = tf.Session()
session.run(tf.global_variables_initializer())
for i in range(10):
    session.run(train_step, feed_dict=train_step_input)

# now run the learned function on some new data
result = session.run(f, feed_dict={tlog.input_placeholder_name(function_spec): X2})
```

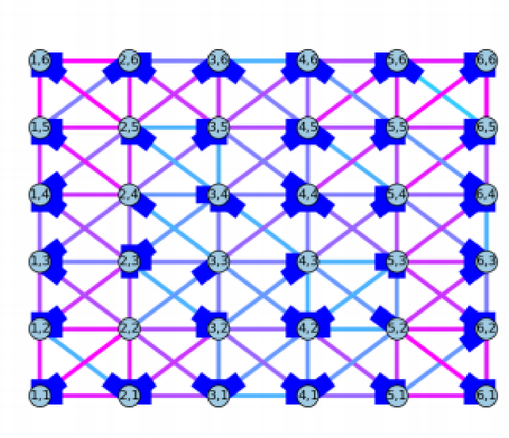
TensorLog [Cohen ’16]

- Experiments
  - Inference task
    - “Friends and smokers” inference task.

	Social Influence Task	
ProbLog2	20 nodes	40-50 sec
TensorLog	3327 nodes	**9.2 msec**

    - Path-finding task in a grid

	Path-finding		
	Size	Time	Acc
ProbLog2	16x16 grid, $d = 10$	100-120 sec	
TensorLog	16x16 grid, $d = 10$	**2.1 msec**	
	64x64 grid, $d = 99$	2.2 msec	
trained	16x16 grid, $d = 10$	6.2 msec	99.89%



TensorLog [Cohen '16]

- Experiments
  - Learning task
    - Relational learning tasks

	ProPPR	TensorLog
CORA (13k facts,10 rules)	AUC 83.2	AUC **97.6**
UMLS (5k facts, 226 rules)	acc 49.8	acc **52.5**
Wordnet (276k facts)		
Hypernym (46 rules)	acc **93.4**	acc 93.3
Hyponym (46 rules)	acc 92.1	acc **92.8**

  - Path finding after learning

Grid Size	Max Depth	# Graph Nodes		Acc		Time (30 epochs)	
		Local	TF	Local	TF	Local	TF
16	10	68	2696	**99.9**	97.2	37.6 sec	**1.1 sec**
18	12	80	3164	93.9	**96.9**	126.1 sec	**1.8 sec**
20	14	92	3632	25.2	**99.1**	144.9 sec	**2.8 sec**
22	16	104	4100	8.6	**98.4**	83.8 sec	**4.2 sec**
24	18	116	4568	**2.4**	0.0	611.7 sec	**6.3 sec**

TensorLog [Cohen '16]

- Experiments
  - KBQA on WikiMovies
    - The KB consists of over 420k tuples containing information about 10 relations and 16k movies

- Question: Who acted in the movie Wise Guys?
  Answers: "Harvey Keitel", "Danny DeVito", "Joe Piscopo", . . .

- Question: what is a film written by Luke Ricci?
  Answer: "How to be a Serial Killer"

    - Encode the questions into the KB by extending it with two additional relations

`mentionsEntity(Q,E)` true if question `Q` mentions entity `E`

`hasFeature(Q,W)` true if question `Q` contains feature `W`

TensorLog [Cohen '16]

- Experiments
  - KBQA on WikiMovies
    - The entities mentioned in a question were extracted by looking for every longest match to a name in the KB
    - The features of a question are simply the words in the question (minus a short stoplist)
    - The simple longest-exact-match heuristic described above identifies entities accurately for this dataset, ➔ take `mentionsEntity` as a hard KB predicate.
    - The final theory contains two rules and two "soft" unary relations $\texttt{QuestionType}_{R,1}$, $\texttt{indicatesQuestionType}_{R,2}$ for each relation R in the original movie KB

TensorLog [Cohen '16]

- Experiments
  - KBQA on WikiMovies
    - The final theory contains two rules and two "soft" unary relations $\texttt{QuestionType}_{R,1}$, $\texttt{indicatesQuestionType}_{R,2}$ for each relation R in the original movie KB
    - for the relation `directedBy` the theory has the two rules

```
answer(Question,Movie) :-
    mentionsEntity(Question,Entity), directedBy(Movie,Entity),
    hasFeature(Question,Word), indicatesQuestionType_{directedBy,1}(Word)
answer(Question,Entity) :-
    mentionsEntity(Question,Movie), directedBy(Movie,Entity),
    hasFeature(Question,Word), indicatesQuestionType_{directedBy,2}(Word)
```

acts as a linear classifier for that rule.

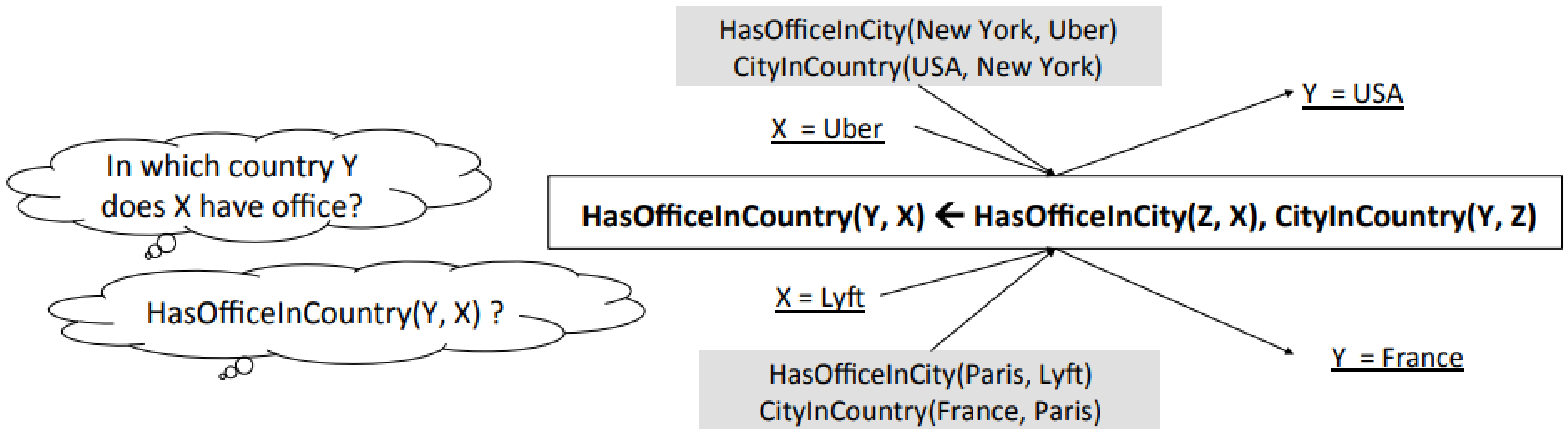
TensorLog [Cohen ’16]

- Experiments
  - KBQA on WikiMovies

Original KB		Extended KB		Num Examples		
Num Tuples	Num Relations	Num Tuples	Num Relations	Train	Devel	Test
421,243	10	1,362,670	12	96,182	20,000	10,000

Method	Accuracy	Time per epoch
Subgraph/question embedding	93.5%	
Key-value memory network	93.9%	
TensorLog (1,000 training examples)	89.4%	6.1 sec
TensorLog (10,000 training examples)	94.8%	1.7 min
TensorLog (96,182 training examples)	**95.0%**	49.5 min

Neural Logic Programming [Yang et al '17]


HasOfficeInCity(New York, Uber)
CityInCountry(USA, New York)
Y = USA
X = Uber
In which country Y does X have office?
HasOfficeInCountry(Y, X) ← HasOfficeInCity(Z, X), CityInCountry(Y, Z)
HasOfficeInCountry(Y, X) ?
X = Lyft
Y = France
HasOfficeInCity(Paris, Lyft)
CityInCountry(France, Paris)

- TensorLog
  - Limited as a learning system because it only learns parameters, not rules
- Neural Logic Programming
  - learn parameters and structure simultaneously in a differentiable framework
  - Based on a neural controller system with an attention mechanism and memory
    - Sequentially compose the primitive differentiable operations used by TensorLog
  - At each stage of the computation, the controller uses attention to "softly" choose a subset of TensorLog's operations, and then performs the operations with contents selected from the memory

Neural Logic Programming [Yang et al '17]

- Knowledge base
  - Collections of relational data of the format `Relation(head,tail)`
    - `head` and `tail`: entities
    - `Relation`: a binary relation between entities
    - E.g.) `HasOfficeInCity(New York,Uber) and CityInCountry(USA,New York).`
- Knowledge base reasoning task
  - Consists of a `query` , an entity `tail` that the `query` is about, and an entity `head` that is the answer to the query
  - Goal: to retrieve a ranked list of entities based on the query such that the desired answer (i.e. head) is ranked as high as possible.

Neural Logic Programming [Yang et al ‘17]

- Knowledge base reasoning task
  - for each query, we are interested in learning weighted chain-like logical rules of the following form:

$$\alpha \quad \texttt{query}(Y, X) \leftarrow R_n(Y, Z_n) \wedge \cdots \wedge R_1(Z_1, X)$$

confidence relations in the knowledge base.

  - During inference, given an entity x, the score of each y:
    - The sum of the confidence of rules that imply query(y,x)
  - Then return a ranked list of entities
    - Where higher the score implies higher the ranking.

Neural Logic Programming [Yang et al '17]

- TensorLog for KB reasoning
  - $\mathbf{E}$: the set of all entities,
    - each entity i is associated with a one-hot vector $\mathbf{v_i} \in \{0,1\}^{|\mathbf{E}|}$
  - $\mathbf{R}$: the set of all binary relations
  - $\mathbf{M_R}$: a matrix in $\{0,1\}^{|\mathbf{E}|\times|\mathbf{E}|}$ for each relation R
    - 1 if $\mathtt{R(i,j)}$ in the KB
  - For each query, we want to learn $\boxed{\sum_l \alpha_l \Pi_{\mathrm{k}\in\beta_l} \mathbf{M}_{\mathrm{R_k}}}$

indexes over all possible rules

an ordered list of all relations in this particular rule

Neural Logic Programming [Yang et al '17]

- TensorLog for KB reasoning

$$\mathtt{R(Y,X)} \leftarrow \mathtt{P(Y,Z)} \wedge \mathtt{Q(Z,X)}$$

  - For any entity $\mathtt{X} = \mathtt{x}$, $\mathbf{M}_{\mathtt{P}} \cdot \mathbf{M}_{\mathtt{Q}} \cdot \mathbf{v}_{\mathtt{x}} \doteq \mathbf{s}$
  - During inference, given an entity v_x, the score of each retrieved entity is then equivalent to the entries in the vector ***s***

$$\mathbf{s} = \sum_{l} \left(\alpha_l \left(\Pi_{\mathrm{k}\in\beta_l} \mathbf{M}_{\mathrm{R_k}} \mathbf{v}_{\mathrm{x}}\right)\right), \ \mathrm{score}(\mathrm{y} \mid \mathrm{x}) = \mathbf{v}_{\mathrm{y}}^{T} \mathbf{s}$$

  - Then, learning problem for each query is:

$$\max_{\{\alpha_l,\beta_l\}} \sum_{\{\mathrm{x},\mathrm{y}\}} \mathrm{score}(\mathrm{y} \mid \mathrm{x}) = \max_{\{\alpha_l,\beta_l\}} \sum_{\{\mathrm{x},\mathrm{y}\}} \mathbf{v}_{\mathrm{y}}^{T} \left(\sum_{l} \left(\alpha_l \left(\Pi_{\mathrm{k}\in\beta_l} \mathbf{M}_{\mathrm{R_k}} \mathbf{v}_{\mathrm{x}}\right)\right)\right)$$

To be learned

Neural Logic Programming [Yang et al '17]

- Learning the logical rules
  - Difficult to formulate a differentiable process to directly learn the parameters and the structure $\{\alpha_l, \beta_l\}$
    - because each parameter is associated with a particular rule, and enumerating rules is an inherently discrete task.
  - Alternatively, rewrite the original equation with a different parametrization

the number of relations in the KB

$$\prod_{t=1}^{T} \sum_{k}^{|\mathbf{R}|} a_t^k \mathbf{M}_{\mathbf{R}_k}$$

The max length of rules

- The key difference is that here we associate each relation in the rule with a weight
- But, this parameterization is not sufficiently expressive, as it assumes that all rules are of the same length
- Thus, we introduce a recurrent formulation

Neural Logic Programming [Yang et al ‘17]

- The recurrent formulation
  - Use auxiliary memory vectors $\boldsymbol{u}_t$
  - Initially the memory vector is set to the given entity $\boldsymbol{v}_x$.
  - The model first computes a weighted average of previous memory vectors using the memory attention vector $\boldsymbol{b}_t$
  - Then the model “softly” applies the TensorLog operators using the operator attention vector $\boldsymbol{a}_t$
    - This formulation allows the model to apply the TensorLog operators on all previous partial inference results, instead of just the last step’s.
  - Finally, the model computes a weighted average of all memory vectors, thus using attention to select the proper rule length
  - Given the recurrent formulation, the learnable parameters for each query are:

$$\{\mathbf{a_t} \mid 1 \leq t \leq T\} \text{ and } \{\mathbf{b_t} \mid 1 \leq t \leq T+1\}$$

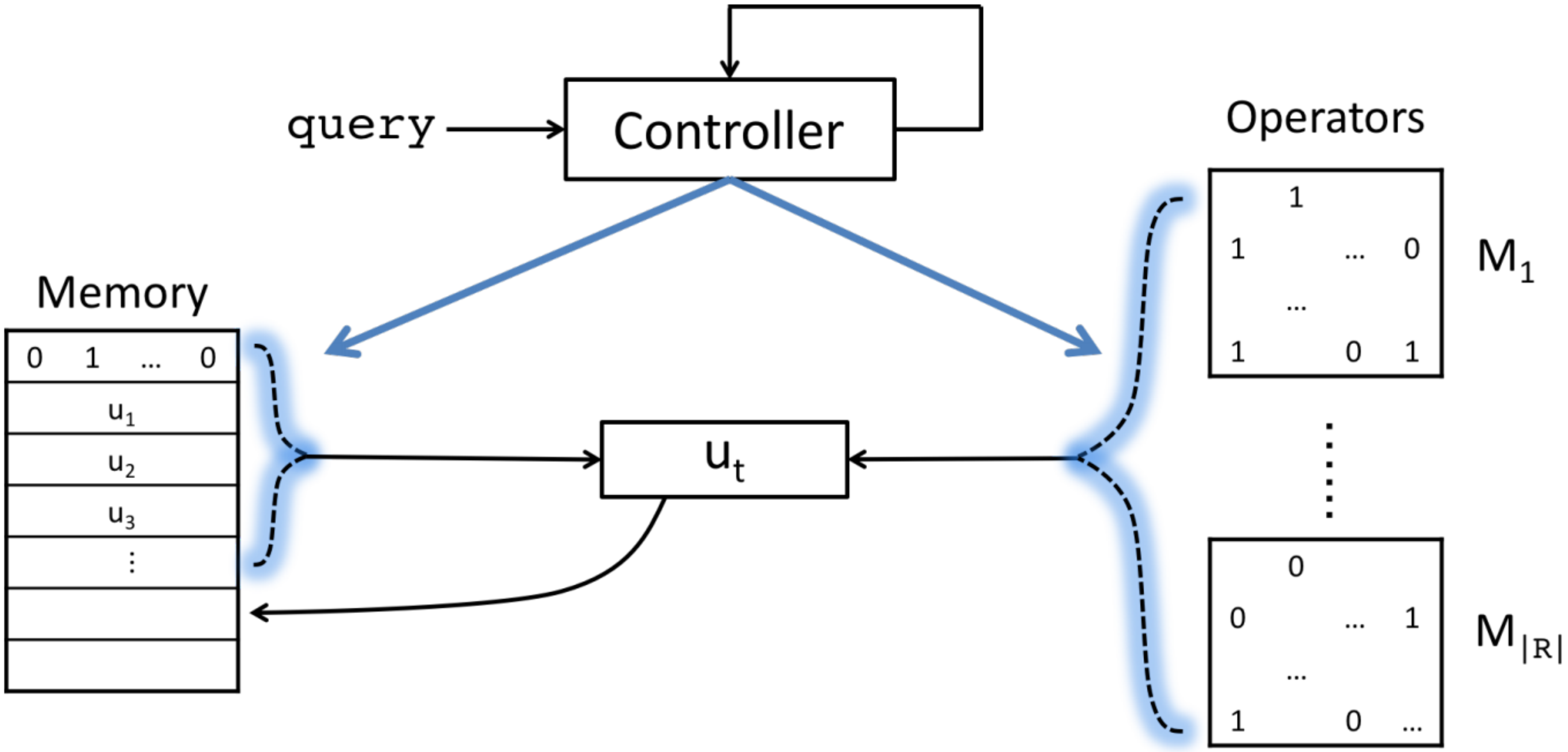
Neural Logic Programming [Yang et al ‘17]

- A neural controller system
  - Learn the operator and memory attention vectors.
    - Use recurrent neural networks not only because they fit with the recurrent formulation, but also because it is likely that current step’s attentions are dependent on previous steps’.
  - The network predicts operator and memory attention vectors:

$$\mathbf{h_t} = \text{update}\left(\mathbf{h_{t-1}}, \text{input}\right)$$
$$\mathbf{a_t} = \text{softmax}\left(W\mathbf{h_t} + b\right)$$
$$\mathbf{b_t} = \text{softmax}\left([\mathbf{h_0}, \ldots, \mathbf{h_{t-1}}]^T \mathbf{h_t}\right)$$

Neural Logic Programming [Yang et al ‘17]

- The neural controller system



Neural Logic Programming [Yang et al '17]

- The recurrent formulation

$$\mathbf{u_0} = \mathbf{v_x}$$

$$\mathbf{u_t} = \sum_{\mathrm{k}}^{|\mathbf{R}|} a_t^{\mathrm{k}} \mathbf{M}_{\mathrm{R_k}} \left(\sum_{\tau=0}^{t-1} b_t^{\tau} \mathbf{u}_{\tau} \right) \quad \text{for } 1 \leq t \leq T$$

$$\mathbf{u_{T+1}} = \sum_{\tau=0}^{T} b_{T+1}^{\tau} \mathbf{u}_{\tau}$$

$$\mathbf{h_t} = \text{update}\left(\mathbf{h_{t-1}}, \text{input}\right)$$

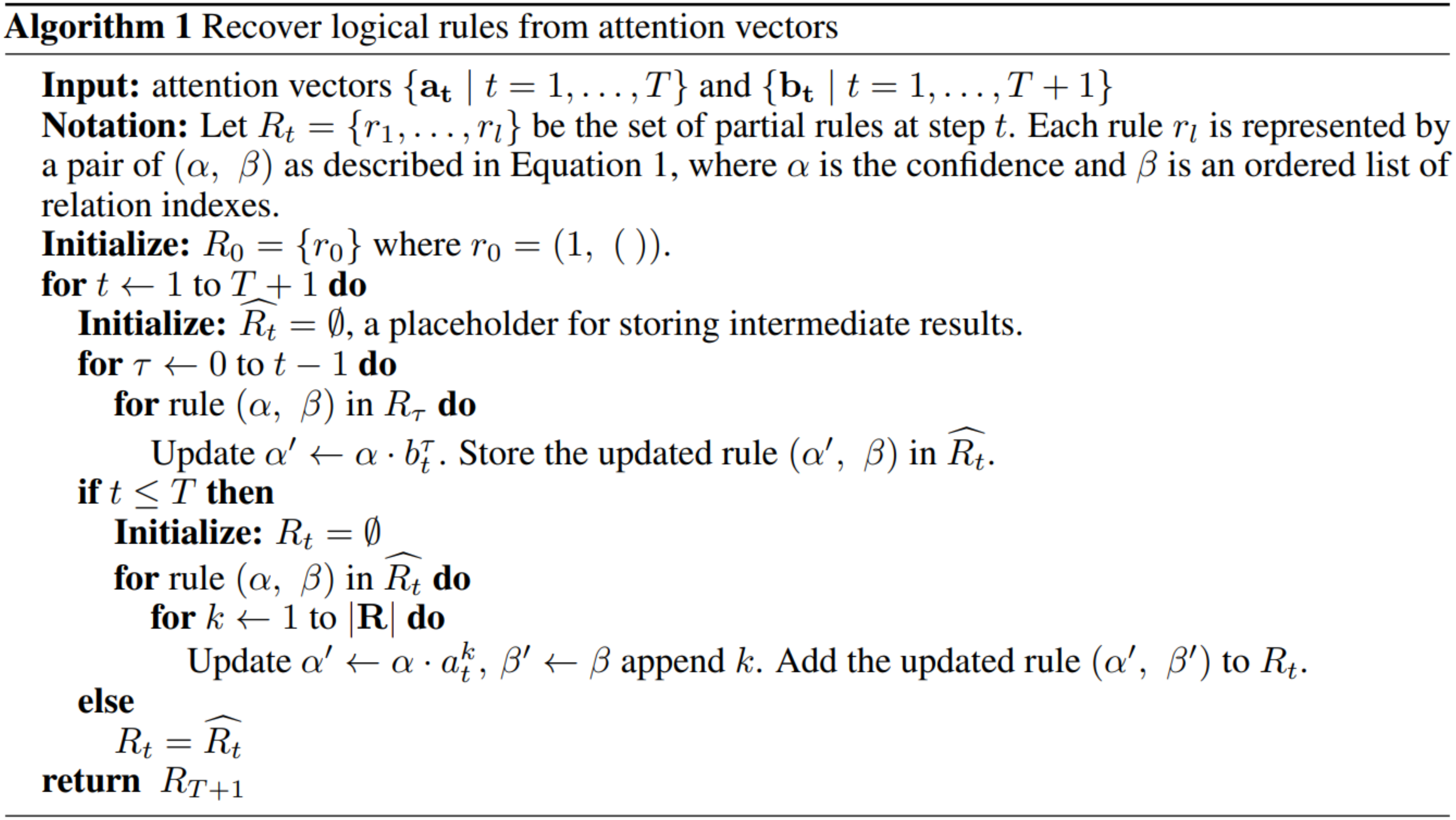
$$\mathbf{a_t} = \text{softmax}\left(W\mathbf{h_t} + b\right)$$

$$\mathbf{b_t} = \text{softmax}\left(\left[\mathbf{h_0}, \ldots, \mathbf{h_{t-1}}\right]^T \mathbf{h_t}\right)$$

  - The input is the query for 1 ≤ t ≤ T and a special END token when t = T + 1
  - The memory holds each step's partial inference results $\{\mathbf{u_0}, \ldots, \mathbf{u_t}, \ldots, \mathbf{u_{T+1}}\}$
  - The final inference result $\boldsymbol{u}$ is just the last vector in memory $\mathbf{u_{T+1}}$
  - The objective is to maximize $\log \mathbf{v_y}^T \mathbf{u}$

Neural Logic Programming [Yang et al '17]

- Recovering logical rules from the neural controller system
  - write rules and their confidences $\{\alpha_l, \beta_l\}$ in terms of the attention vectors $\{\mathbf{a_t}, \mathbf{b_t}\}$



Algorithm 1 Recover logical rules from attention vectors

```
Input: attention vectors $\{\mathbf{a_t} \mid t = 1, \ldots, T\}$ and $\{\mathbf{b_t} \mid t = 1, \ldots, T+1\}$
Notation: Let $R_t = \{r_1, \ldots, r_l\}$ be the set of partial rules at step $t$. Each rule $r_l$ is represented by
a pair of $(\alpha, \ \beta)$ as described in Equation 1, where $\alpha$ is the confidence and $\beta$ is an ordered list of
relation indexes.
Initialize: $R_0 = \{r_0\}$ where $r_0 = (1, \ ())$.
for $t \leftarrow 1$ to $T+1$ do
    Initialize: $\widehat{R_t} = \emptyset$, a placeholder for storing intermediate results.
    for $\tau \leftarrow 0$ to $t-1$ do
        for rule $(\alpha, \ \beta)$ in $R_\tau$ do
            Update $\alpha' \leftarrow \alpha \cdot b_t^\tau$. Store the updated rule $(\alpha', \ \beta)$ in $\widehat{R_t}$.
    if $t \leq T$ then
        Initialize: $R_t = \emptyset$
        for rule $(\alpha, \ \beta)$ in $\widehat{R_t}$ do
            for $k \leftarrow 1$ to $|\mathbf{R}|$ do
                Update $\alpha' \leftarrow \alpha \cdot a_t^k$, $\beta' \leftarrow \beta$ append $k$. Add the updated rule $(\alpha', \ \beta')$ to $R_t$.
    else
        $R_t = \widehat{R_t}$
return $R_{T+1}$
```

Neural Logic Programming [Yang et al ‘17]

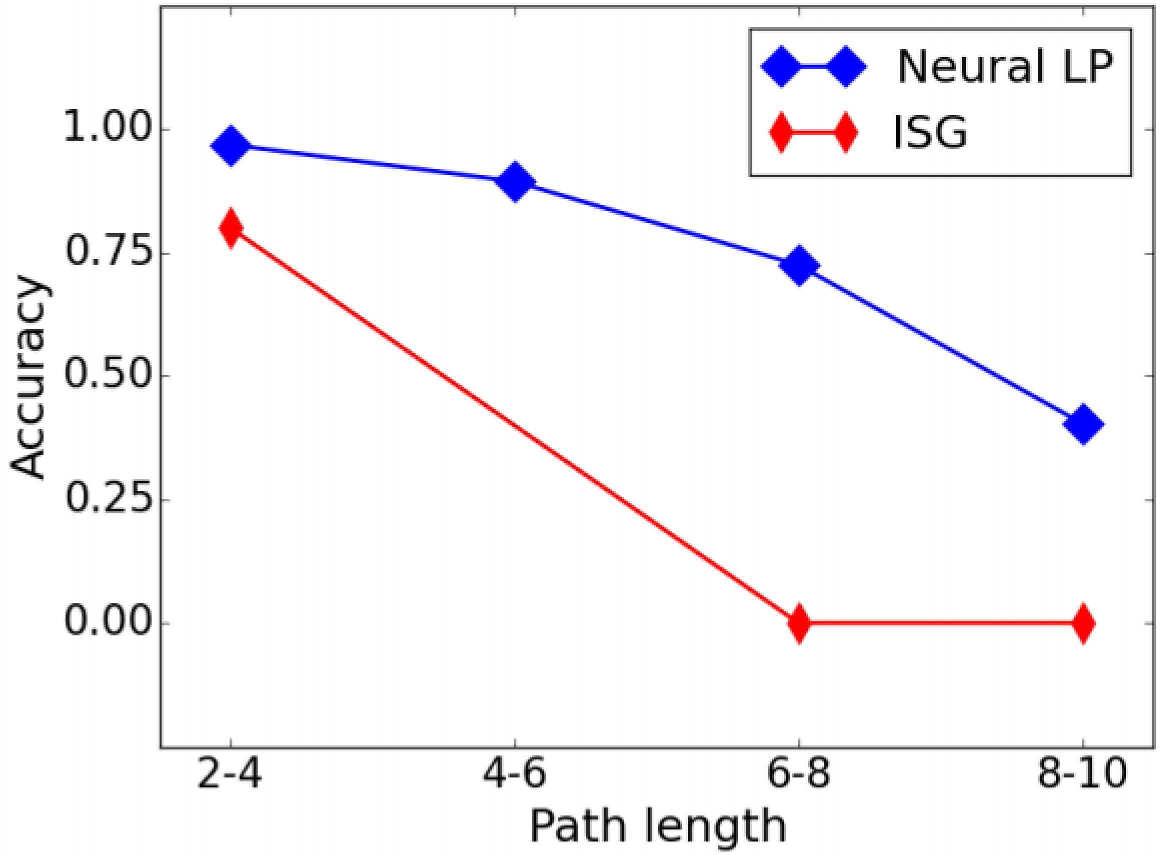
- Experiments: Statistical relation learning

	# Data	# Relation	# Entity
UMLS	5960	46	135
Kinship	9587	25	104

	ISG		Neural LP	
	$T = 2$	$T = 3$	$T = 2$	$T = 3$
UMLS	43.5	43.3	92.0	**93.2**
Kinship	59.2	59.0	**90.2**	90.1

Neural Logic Programming [Yang et al ‘17]

- Grid path finding
  - query: randomly generated by combining a series of directions, such as North_SouthWest



Neural Logic Programming [Yang et al '17]

- Knowledge base completion
  - query: E.g.) HasOfficeInCountry and Uber
    - Use an embedding lookup table for each query

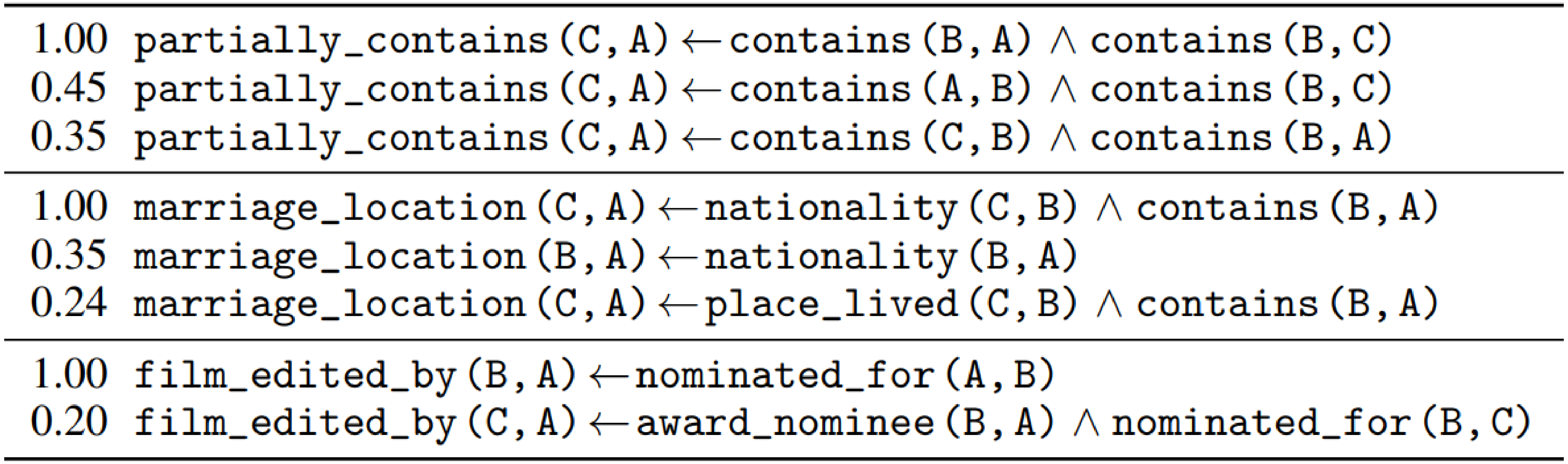
`HasOfficeInCountry(USA,Uber)`

Dataset	# Facts	# Train	# Test	# Relation	# Entity
WN18	106,088	35,354	5,000	18	40,943
FB15K	362,538	120,604	59,071	1,345	14,951
FB15KSelected	204,087	68,028	20,466	237	14,541

	WN18		FB15K		FB15KSelected	
	MRR	Hits@10	MRR	Hits@10	MRR	Hits@10
Neural Tensor Network	0.53	66.1	0.25	41.4	-	-
TransE	0.38	90.9	0.32	53.9	-	-
DISTMULT [29]	0.83	94.2	0.35	57.7	**0.25**	**40.8**
Node+LinkFeat [25]	0.94	94.3	**0.82**	87.0	0.23	34.7
Implicit ReasoNets [23]	-	**95.3**	-	**92.7**	-	-
Neural LP	**0.94**	94.5	0.76	83.7	0.24	36.2

Neural Logic Programming [Yang et al '17]

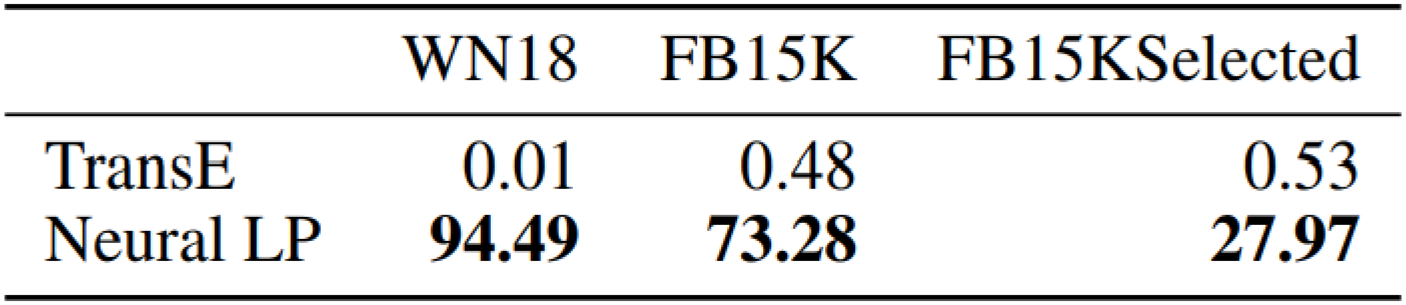
- Examples of logical rules learned by Neural LP on FB15KSelected. The letters A,B,C are ungrounded logic variables.



1.00	partially_contains(C,A) $\leftarrow$ contains(B,A) $\wedge$ contains(B,C)
0.45	partially_contains(C,A) $\leftarrow$ contains(A,B) $\wedge$ contains(B,C)
0.35	partially_contains(C,A) $\leftarrow$ contains(C,B) $\wedge$ contains(B,A)
1.00	marriage_location(C,A) $\leftarrow$ nationality(C,B) $\wedge$ contains(B,A)
0.35	marriage_location(B,A) $\leftarrow$ nationality(B,A)
0.24	marriage_location(C,A) $\leftarrow$ place_lived(C,B) $\wedge$ contains(B,A)
1.00	film_edited_by(B,A) $\leftarrow$ nominated_for(A,B)
0.20	film_edited_by(C,A) $\leftarrow$ award_nominee(B,A) $\wedge$ nominated_for(B,C)

Neural Logic Programming [Yang et al ‘17]

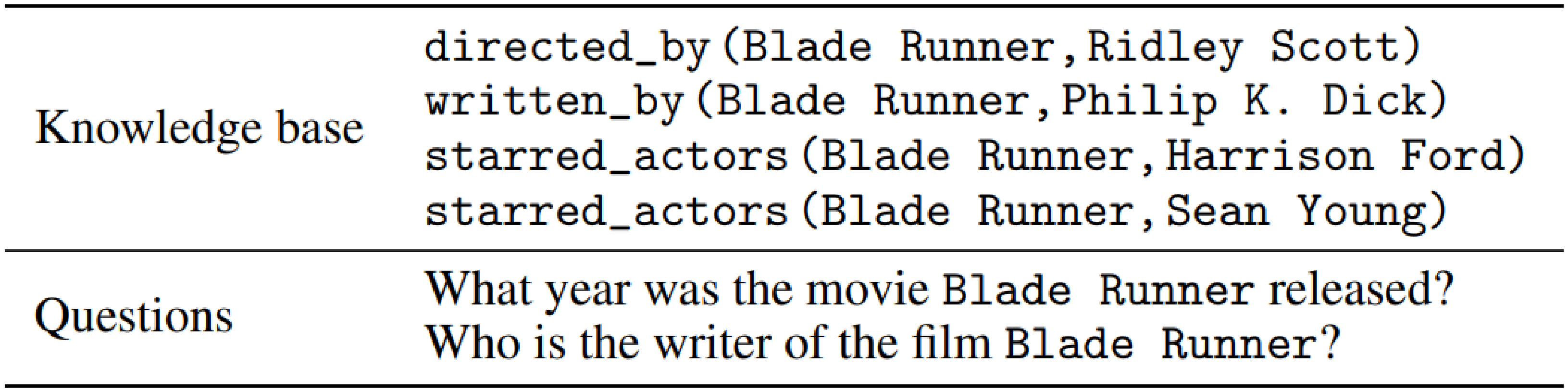
- Inductive knowledge base completion
  - conduct experiments where training and testing use disjoint sets of entities



	WN18	FB15K	FB15KSelected
TransE	0.01	0.48	0.53
Neural LP	**94.49**	**73.28**	**27.97**

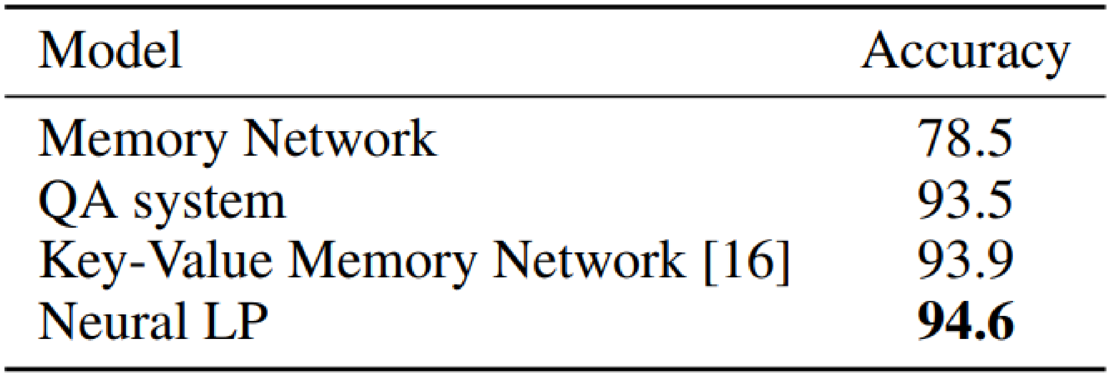
Neural Logic Programming [Yang et al '17]

- Question answering against knowledge base
  - WikiMovies



Knowledge base	directed_by (Blade Runner,Ridley Scott)
	written_by (Blade Runner,Philip K. Dick)
	starred_actors (Blade Runner,Harrison Ford)
	starred_actors (Blade Runner,Sean Young)
Questions	What year was the movie Blade Runner released?
	Who is the writer of the film Blade Runner?

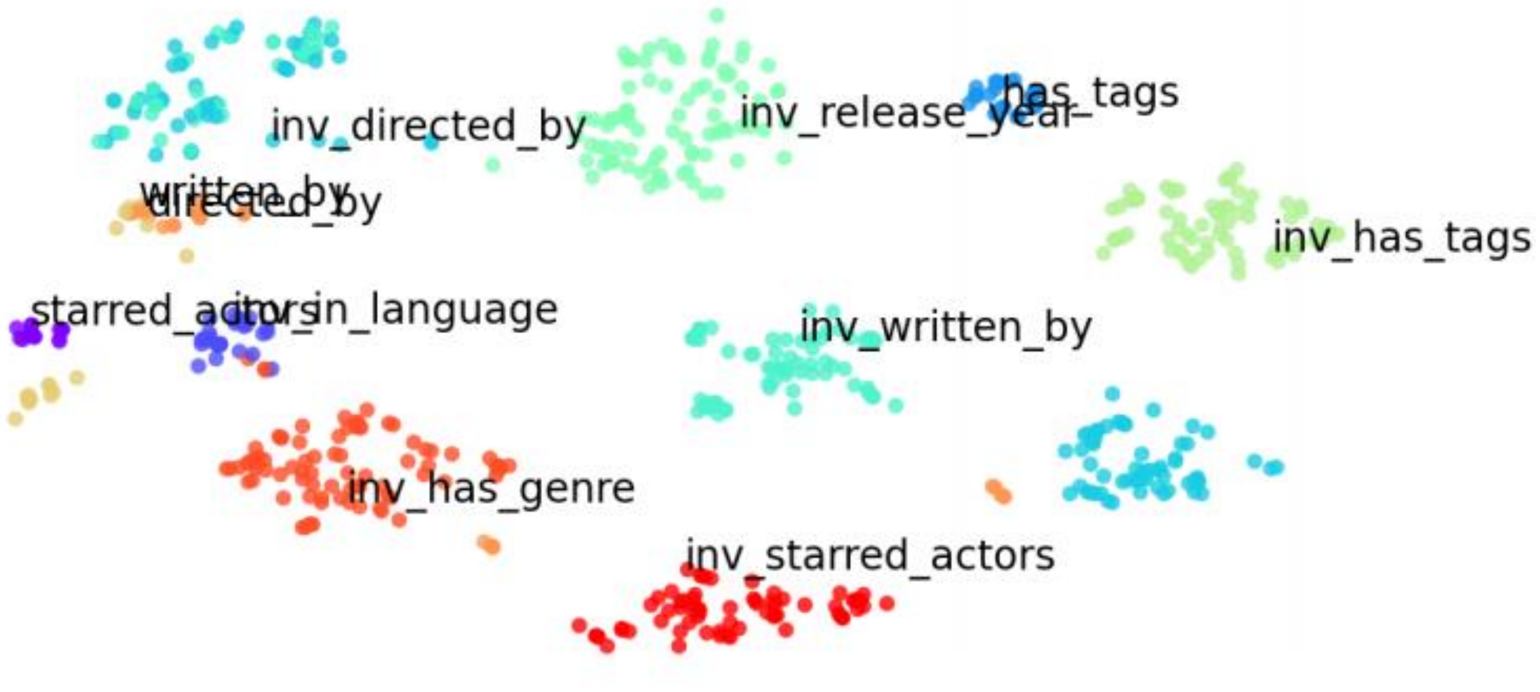
  - query: the average of the embeddings of the words



Model	Accuracy
Memory Network	78.5
QA system	93.5
Key-Value Memory Network [16]	93.9
Neural LP	**94.6**

Neural Logic Programming [Yang et al ‘17]

- Question answering against knowledge base
  - Visualization of learned logical rules

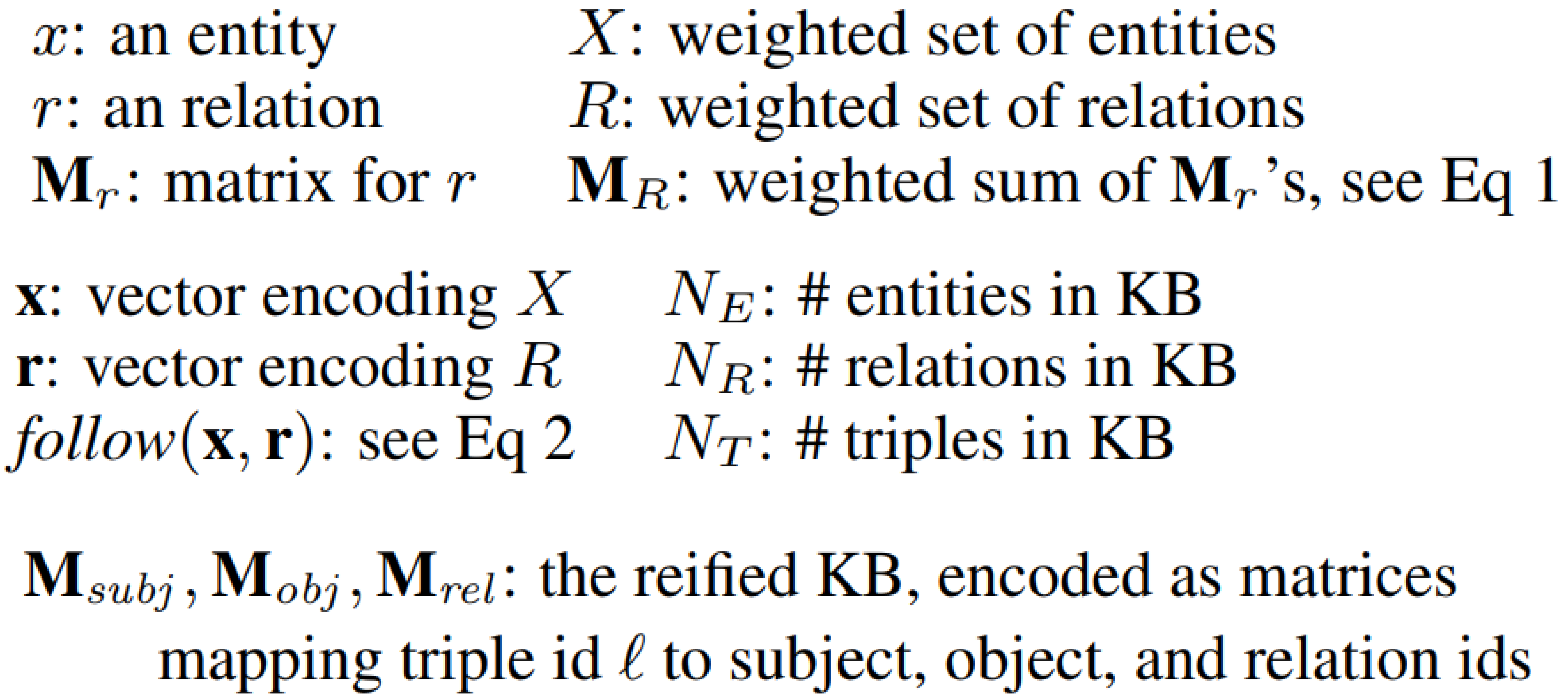


Scalable Neural Methods for Reasoning With a Symbolic Knowledge Base [Cohen et al ’20]

- Propose a **sparse-matrix reified KB**
  - representing a symbolic KB
  - Enables neural modules that are
    - 1) fully differentiable,
    - 2) faithful to the original semantics of the KB,
    - 3) expressive enough to model multi-hop inferences, and
    - 4) scalable enough to use with realistically large KBs

Scalable Neural Methods for Reasoning With a Symbolic Knowledge Base [Cohen et al ’20]

- Summary of notation



x: an entity	X: weighted set of entities
r: an relation	R: weighted set of relations
$\mathbf{M}_r$: matrix for r	$\mathbf{M}_R$: weighted sum of $\mathbf{M}_r$’s, see Eq 1

$\mathbf{x}$: vector encoding X	N_E: # entities in KB
$\mathbf{r}$: vector encoding R	N_R: # relations in KB
$\mathit{follow}(\mathbf{x}, \mathbf{r})$: see Eq 2	N_T: # triples in KB

$\mathbf{M}_{subj}, \mathbf{M}_{obj}, \mathbf{M}_{rel}$: the reified KB, encoded as matrices mapping triple id ℓ to subject, object, and relation ids

Scalable Neural Methods for Reasoning With a Symbolic Knowledge Base [Cohen et al ’20]

- **Weighted sets** as “k-hot” vectors
  - Each element x of weighted set X is associated with a non-negative real number
  - a weight less than 1: a confidence that the set contains x
  - weights more than 1: make X a multiset
  - X is a **hard set** if all elements of X have weight 1
  - If X is a hard entity set, then this will be a “k-hot” vector, for k = |X|
  - The **support** of x: the set of indices of $\boldsymbol{x}$ with non-zero values

Scalable Neural Methods for Reasoning With a Symbolic Knowledge Base [Cohen et al ’20]

- Sparse vs. dense matrices for relations
  - for all but the smallest KBs, a relation matrix must be implemented using a sparse matrix data structure
    - a sparse coordinate pair (COO) encoding: with a COO encoding, each KB fact requires storing only two integers and one float

Scalable Neural Methods for Reasoning With a Symbolic Knowledge Base [Cohen et al '20]

- The relation-set following operation
  - r-neighbors of an entity x_i: the set of entities x_j that are connected to x_i by an edge labeled r

$$r\text{-}neighbors(x) \equiv \{x_j : (x_i, x_j) \in r\}$$

  - R-neighbors: Extension to relation sets

$$R\text{-}neighbors(X) \equiv \{x_j : \exists r \in R, x_i \in X \text{ so that } (x_i, x_j) \in r\}$$

  - E.g)

$$q = \text{“}\textit{what movies were produced or directed by Quentin Tarantino}\text{”}$$

  - The answer to q is the set R-neighbors(X) with

$$R = \{producer\_of, writer\_of\} \quad X = \{Quentin\_Tarantino\}$$

Scalable Neural Methods for Reasoning With a Symbolic Knowledge Base [Cohen et al ’20]

- The relation-set following operation
  - Approximate the R-neighbors computation with differentiable operations
    - Can be performed on the vectors encoding the sets X and R

$$\mathbf{M}_R \equiv (\sum_{k=1}^{N_R} \mathbf{r}[k] \cdot \mathbf{M}_{r_k})$$

  - The relation-set following operation for x and r

$$follow(\mathbf{x}, \mathbf{r}) \equiv \mathbf{x}\mathbf{M}_R = \mathbf{x}(\sum_{k=1}^{N_R} \mathbf{r}[k] \cdot \mathbf{M}_{r_k})$$

Scalable Neural Methods for Reasoning With a Symbolic Knowledge Base [Cohen et al ’20]

- The relation-set following operation
  - Baseline implementations: Not efficient
    - The naive mixing
      - But, tensorflow does not support general sparse tensor contractions, it is not always possible to extend sparse-matrix computations to minibatches
    - The late mixing: mixes the output of many single-relation following steps, rather than mixing the KB itself

$$\mathit{follow}(\mathbf{x}, \mathbf{r}) = \sum_{k=1}^{N_R} (\mathbf{r}[k] \cdot \mathbf{x}\mathbf{M}_{r_k})$$

      - can be extended easily to a minibatches

Scalable Neural Methods for Reasoning With a Symbolic Knowledge Base [Cohen et al ’20]

- The relation-set following operation
  - A reified knowledge base
    - Represent each KB assertion $r_k(x_i, x_j)$ as a tuple (i, j, k)
    - (i_ℓ, j_ℓ, k_ℓ) : l-th triple

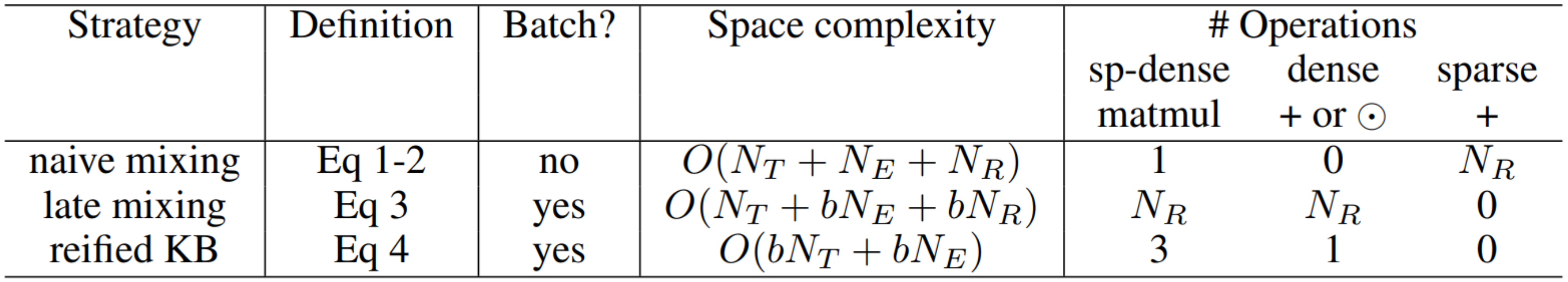
$$\mathbf{M}_{subj}[\ell, m] \equiv \begin{cases} 1 & \text{if } m = i_\ell \\ 0 & \text{else} \end{cases} \qquad \mathbf{M}_{obj}[\ell, m] \equiv \begin{cases} 1 & \text{if } m = j_\ell \\ 0 & \text{else} \end{cases}$$

$$\mathbf{M}_{rel}[\ell, m] \equiv \begin{cases} 1 & \text{if } m = k_\ell \\ 0 & \text{else} \end{cases}$$

$$follow(\mathbf{x}, \mathbf{r}) = (\mathbf{x}\mathbf{M}_{subj}^T \odot \mathbf{r}\mathbf{M}_{rel}^T)\mathbf{M}_{obj}$$

Scalable Neural Methods for Reasoning With a Symbolic Knowledge Base [Cohen et al ’20]

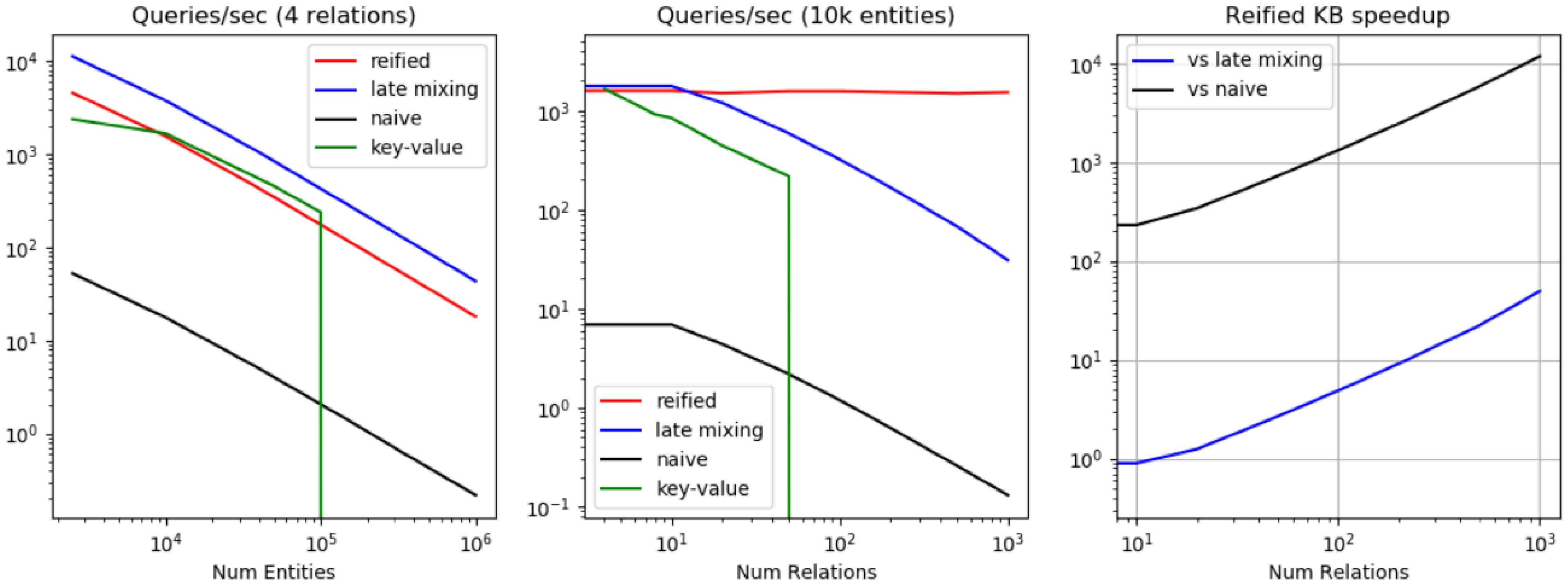
- The relation-set following operation



Strategy	Definition	Batch?	Space complexity	# Operations		
				sp-dense matmul	dense + or $\odot$	sparse +
naive mixing	Eq 1-2	no	$O(N_T + N_E + N_R)$	1	0	N_R
late mixing	Eq 3	yes	$O(N_T + bN_E + bN_R)$	N_R	N_R	0
reified KB	Eq 4	yes	$O(bN_T + bN_E)$	3	1	0

Scalable Neural Methods for Reasoning With a Symbolic Knowledge Base [Cohen et al '20]

- Experiments: Scalability
  - the speed of the key-value network is similar to the reified KB for only four relations, however it is about 7x slower for 50 relations and 10k entities
  - Comparing to the key-value network, the reified KB scales much better, and can handle 10x as many entities and 20x as many relations



Scalable Neural Methods for Reasoning With a Symbolic Knowledge Base [Cohen et al ’20]

Entity linking

- KBQA for multi-hop questions
  - $\boldsymbol{x}_0$: the set of entities associated with q

$$\text{for } t = 1, 2, 3: \quad \mathbf{r}^t = f^t(q); \quad \mathbf{x}^t = \mathit{follow}(\mathbf{x}^{t-1}, \mathbf{r}^t)$$

- KBQA on FreeBase
  - FreeBase contains two kinds of nodes: real-world entities, and compound value types (CVTs)
    - CVT: non-binary relationships or events

$$\mathbf{r}_{\mathrm{E}\to\mathrm{E}} = f_{\mathrm{E}\to\mathrm{E}}(q); \quad \mathbf{r}_{\mathrm{E}\to\mathrm{CVT}} = f_{\mathrm{E}\to\mathrm{CVT}}(q); \quad \mathbf{r}_{\mathrm{CVT}\to\mathrm{E}} = f_{\mathrm{CVT}\to\mathrm{E}}(q$$

$$\hat{\mathbf{a}} = \mathit{follow}(\mathit{follow}(\mathbf{x}, \mathbf{r}_{\mathrm{E}\to\mathrm{CVT}}), \mathbf{r}_{\mathrm{CVT}\to\mathrm{E}}) + \mathit{follow}(\mathbf{x}, \mathbf{r}_{\mathrm{E}\to\mathrm{E}})$$

Scalable Neural Methods for Reasoning With a Symbolic Knowledge Base [Cohen et al ’20]

- Knowledge base completion $\mathbf{x}_i^0 = \mathbf{x}$ for every i

  for $i = 1, \ldots, N$ and $t = 1, \ldots, T$: $\mathbf{r}_i^t = f_i^t(q)$; $\mathbf{x}_i^t = \mathit{follow}(\mathbf{x}_i^{t-1}, \mathbf{r}_i^t) + \mathbf{x}_i^{t-1}$

  – The final output: $\hat{\mathbf{a}} = \mathit{softmax}(\sum_{i\in\{1\ldots N\}} \mathbf{x}_i^T)$

- An encoder-decoder architecture for varying inferential structures
  - generated simple artificial natural-language sentences describing longer chains of relationships on a 10-by-10 grid.

    $\boldsymbol{h}_0$: The encoded values of the question with the final hidden state of an LSTM, written here

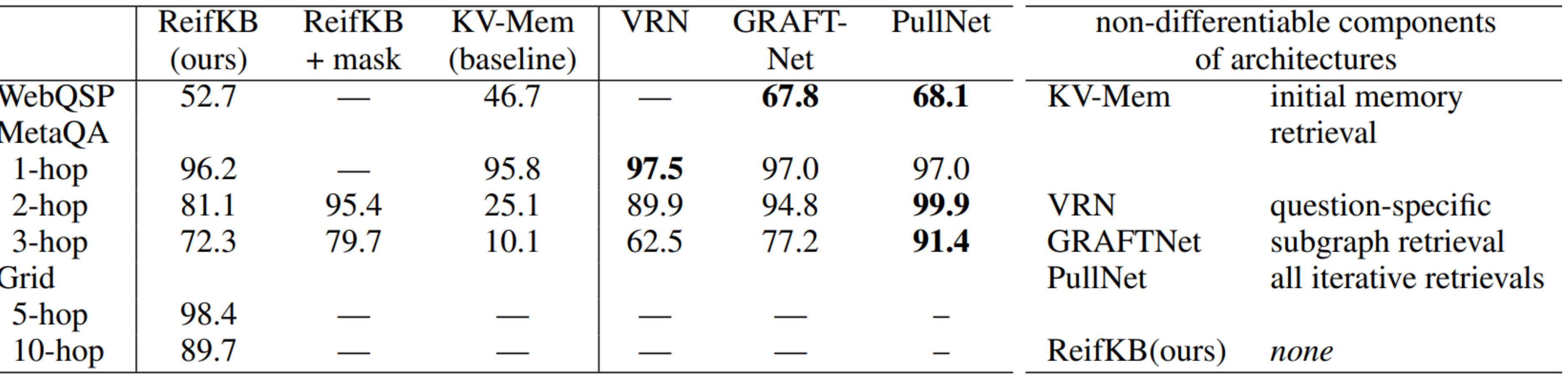
    $$p^t = f_p(\mathbf{h}^{t-1}); \quad \mathbf{r}^t = f_r(\mathbf{h}^{t-1});$$

    $$\mathbf{x}^t = \mathit{follow}(\mathbf{x}^{t-1}, \mathbf{r}^t); \quad \mathbf{h}^t = \mathrm{LSTM}(\mathbf{h}^{t-1}, \mathbf{r}^{t-1})$$

  – Final output: $\hat{\mathbf{a}} = \mathit{softmax}(\sum_{t=1}^{T} \mathbf{x}^t \cdot p^t \prod_{t'<t}(1 - p^{t'}))$

Scalable Neural Methods for Reasoning With a Symbolic Knowledge Base [Cohen et al '20]

- Experimental results

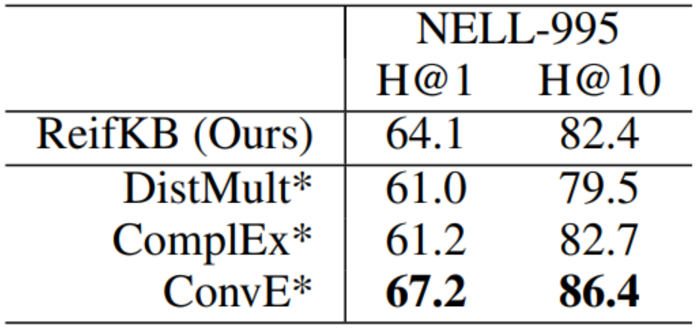


	ReifKB (ours)	ReifKB + mask	KV-Mem (baseline)	VRN	GRAFT-Net	PullNet
WebQSP	52.7	—	46.7	—	**67.8**	**68.1**
MetaQA						
1-hop	96.2	—	95.8	**97.5**	97.0	97.0
2-hop	81.1	95.4	25.1	89.9	94.8	**99.9**
3-hop	72.3	79.7	10.1	62.5	77.2	**91.4**
Grid						
5-hop	98.4	—	—	—	—	–
10-hop	89.7	—	—	—	—	–

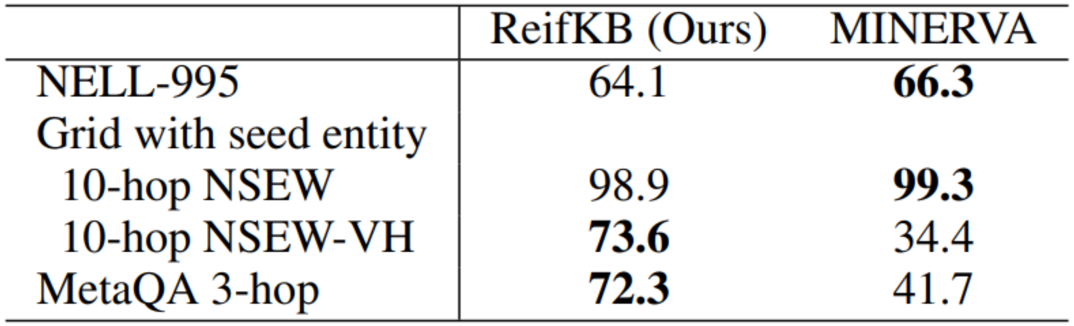
non-differentiable components of architectures	
KV-Mem	initial memory retrieval
VRN	question-specific
GRAFTNet	subgraph retrieval
PullNet	all iterative retrievals
ReifKB(ours)	*none*

Scalable Neural Methods for Reasoning With a Symbolic Knowledge Base [Cohen et al '20]

- Experimental results



	NELL-995	
	H@1	H@10
ReifKB (Ours)	64.1	82.4
DistMult*	61.0	79.5
ComplEx*	61.2	82.7
ConvE*	**67.2**	**86.4**



	ReifKB (Ours)	MINERVA
NELL-995	64.1	**66.3**
Grid with seed entity		
10-hop NSEW	98.9	**99.3**
10-hop NSEW-VH	**73.6**	34.4
MetaQA 3-hop	**72.3**	41.7

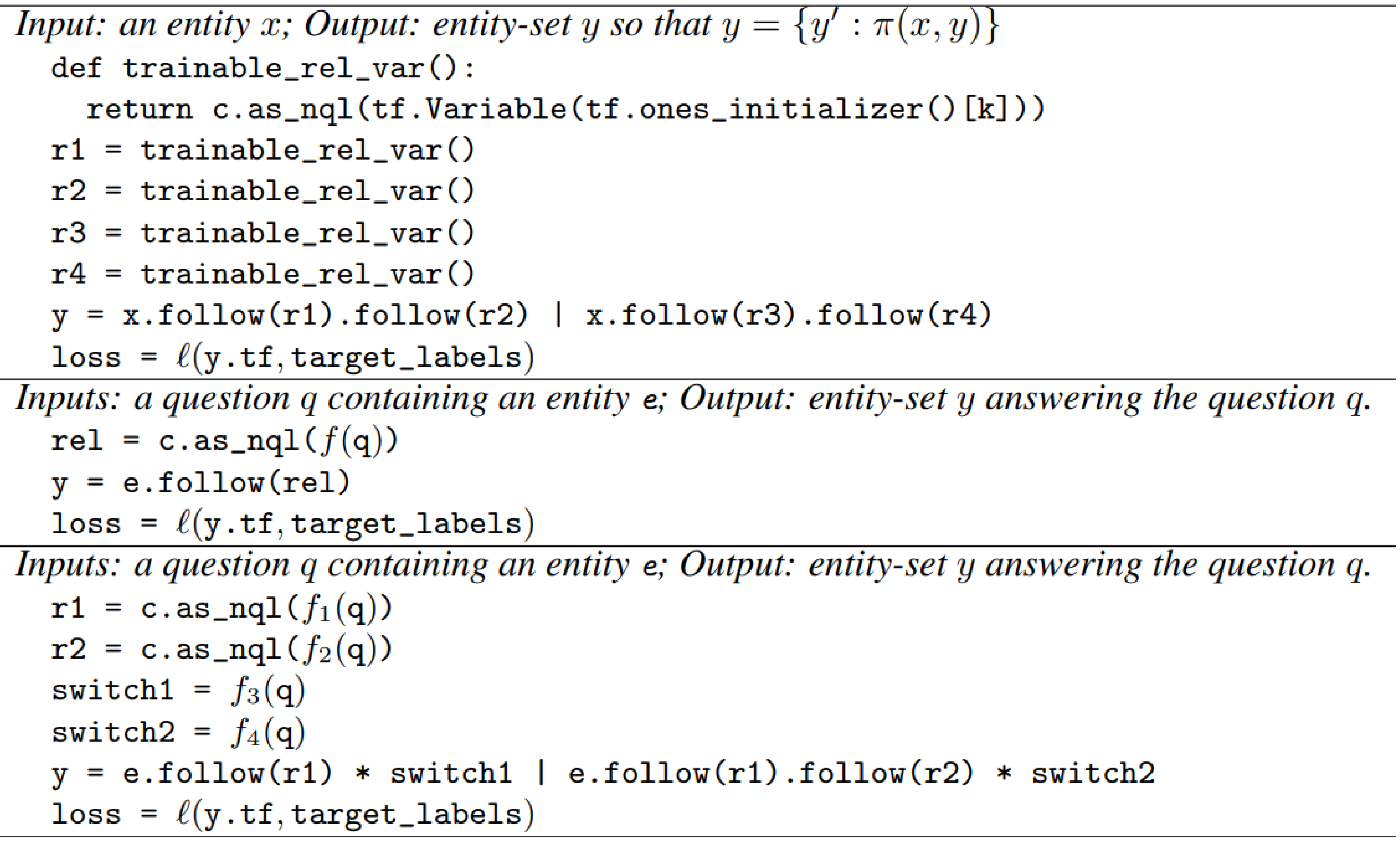
Neural Query Language: A Knowledge Base Query Language for Tensorflow [Cohen et al ‘19]

- Neural Query Language
  - A framework for accessing soft symbolic database using only differentiable operators
  - Simple NQL expressions
    - create singleton, unit-weighted sets:

```
henry8 = c.one(‘Henry_VIII of house of Tudor’, ’person_t’)
```

NQL expression		Vector-matrix specification	Comments
`s.rel()`	$\equiv$	$\mathbf{s}\mathbf{M}_\pi$	
`s.rel(-1)`	$\equiv$	$\mathbf{s}\mathbf{M}_\pi^T$	
`s | t`	$\equiv$	$\mathbf{s}+\mathbf{t}$	
`s & t`	$\equiv$	$\mathbf{s}\odot\mathbf{t}$	$\odot$ *is Hadamard product*
`s.follow(r)`	$\equiv$	$\mathbf{s}\left(\sum_{i=1}^{k}\mathbf{r}[i]M_{\pi_i}\right)$	
`s.if_any(t)`	$\equiv$	$\mathbf{s}\Vert\mathbf{t}\Vert_1$	
`s * a`	$\equiv$	$\mathbf{s}a$	a is a Tensorflow scalar

Neural Query Language: A Knowledge Base Query Language for Tensorflow [Cohen et al ‘19]



Input: an entity x; Output: entity-set y so that $y = \{y' : \pi(x, y)\}$

```
def trainable_rel_var():
  return c.as_nql(tf.Variable(tf.ones_initializer()[k]))
r1 = trainable_rel_var()
r2 = trainable_rel_var()
r3 = trainable_rel_var()
r4 = trainable_rel_var()
y = x.follow(r1).follow(r2) | x.follow(r3).follow(r4)
loss = ℓ(y.tf,target_labels)
```

Inputs: a question q containing an entity e; Output: entity-set y answering the question q.

```
rel = c.as_nql(f(q))
y = e.follow(rel)
loss = ℓ(y.tf,target_labels)
```

Inputs: a question q containing an entity e; Output: entity-set y answering the question q.

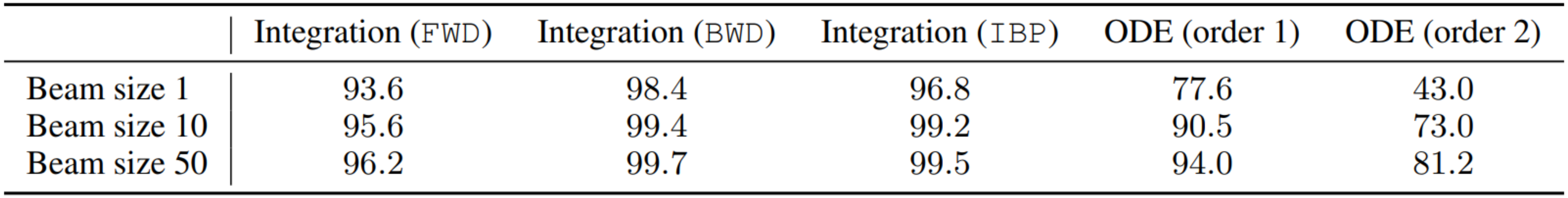
```
r1 = c.as_nql(f1(q))
r2 = c.as_nql(f2(q))
switch1 = f3(q)
switch2 = f4(q)
y = e.follow(r1) * switch1 | e.follow(r1).follow(r2) * switch2
loss = ℓ(y.tf,target_labels)
```

Neural Query Language: A Knowledge Base Query Language for Tensorflow [Cohen et al ‘19]

```
Input: an initial entity $e$ and encoded state $s$; Output: entity-set $y$
  p = 1; y = tf.zeros(k)
  for i in range(MAX_HOPS):
    s, r, p_stop = $f$(s)
    e = e.follow(r)
    y += p * p_stop * e.as_tf()
    p = p * (1 - p_stop)
```

Deep Learning for Symbolic Mathematics [Lample and Charton '19]

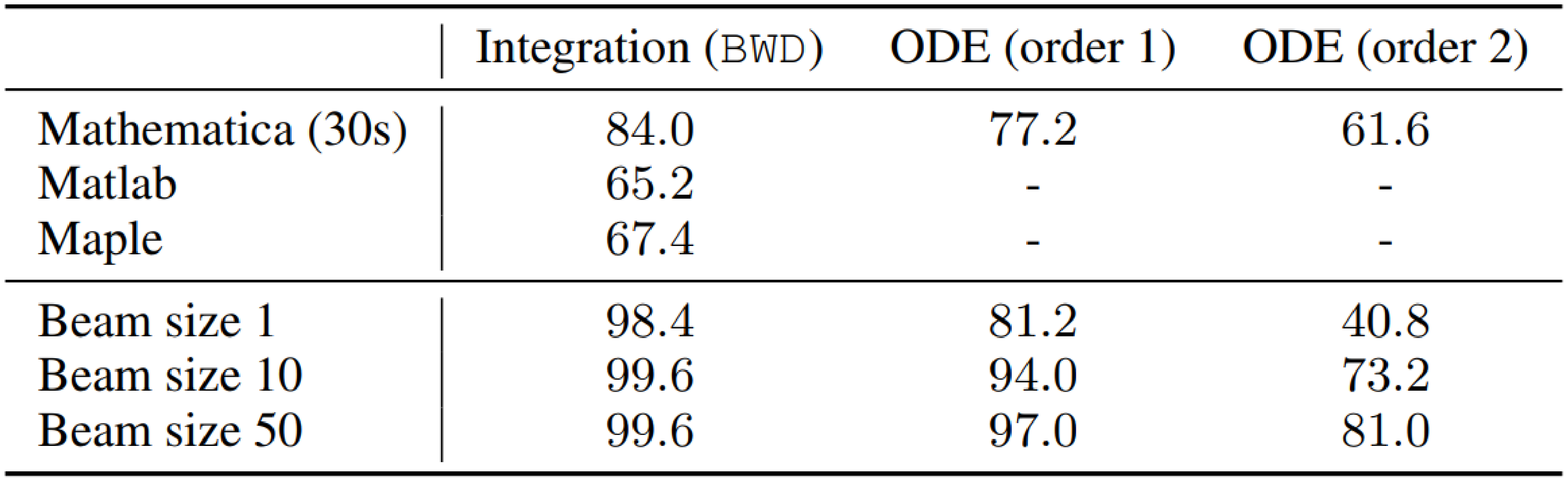
- Seq2seq model
  - Converting tree (math expression) to sequence
  - Use automatically generated training sets
- Accuracy of our models on integration and differential equation solving



	Integration (FWD)	Integration (BWD)	Integration (IBP)	ODE (order 1)	ODE (order 2)
Beam size 1	93.6	98.4	96.8	77.6	43.0
Beam size 10	95.6	99.4	99.2	90.5	73.0
Beam size 50	96.2	99.7	99.5	94.0	81.2

Deep Learning for Symbolic Mathematics [Lample and Charton ‘19]

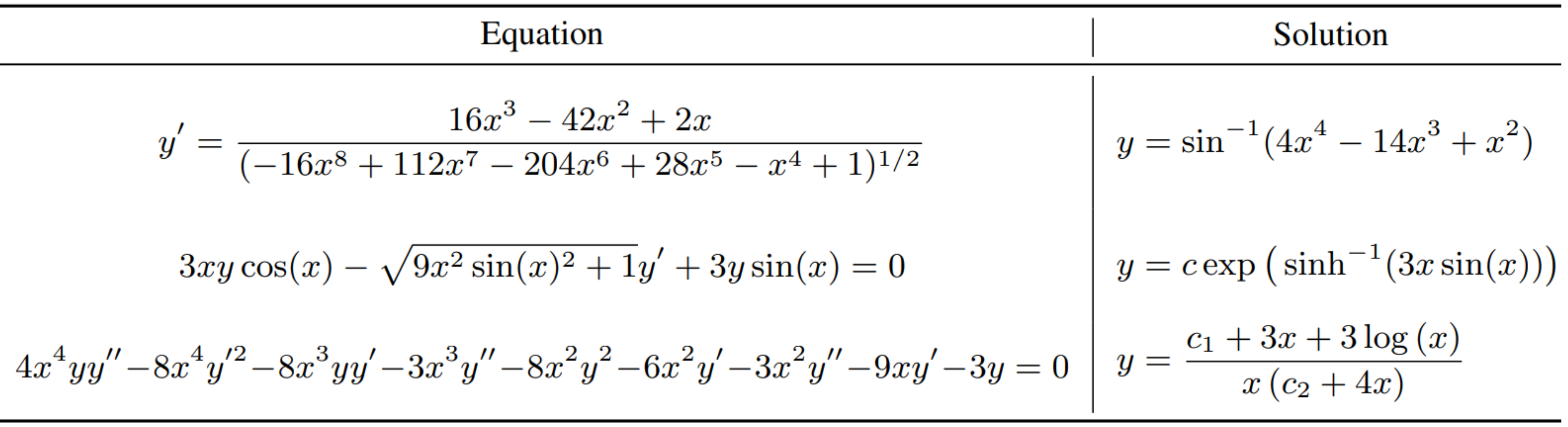
- Comparison of our model with Mathematica, Maple and Matlab on a test set of 500 equations



	Integration (BWD)	ODE (order 1)	ODE (order 2)
Mathematica (30s)	84.0	77.2	61.6
Matlab	65.2	-	-
Maple	67.4	-	-
Beam size 1	98.4	81.2	40.8
Beam size 10	99.6	94.0	73.2
Beam size 50	99.6	97.0	81.0

Deep Learning for Symbolic Mathematics [Lample and Charton '19]

- Examples of problems that our model is able to solve, on which Mathematica and Matlab were not able to find a soluti



Equation	Solution
$y' = \frac{16x^3 - 42x^2 + 2x}{(-16x^8 + 112x^7 - 204x^6 + 28x^5 - x^4 + 1)^{1/2}}$	$y = \sin^{-1}(4x^4 - 14x^3 + x^2)$
$3xy\cos(x) - \sqrt{9x^2\sin(x)^2 + 1}y' + 3y\sin(x) = 0$	$y = c\exp\left(\sinh^{-1}(3x\sin(x))\right)$
$4x^4yy'' - 8x^4y'^2 - 8x^3yy' - 3x^3y'' - 8x^2y^2 - 6x^2y' - 3x^2y'' - 9xy' - 3y = 0$	$y = \frac{c_1 + 3x + 3\log(x)}{x(c_2 + 4x)}$

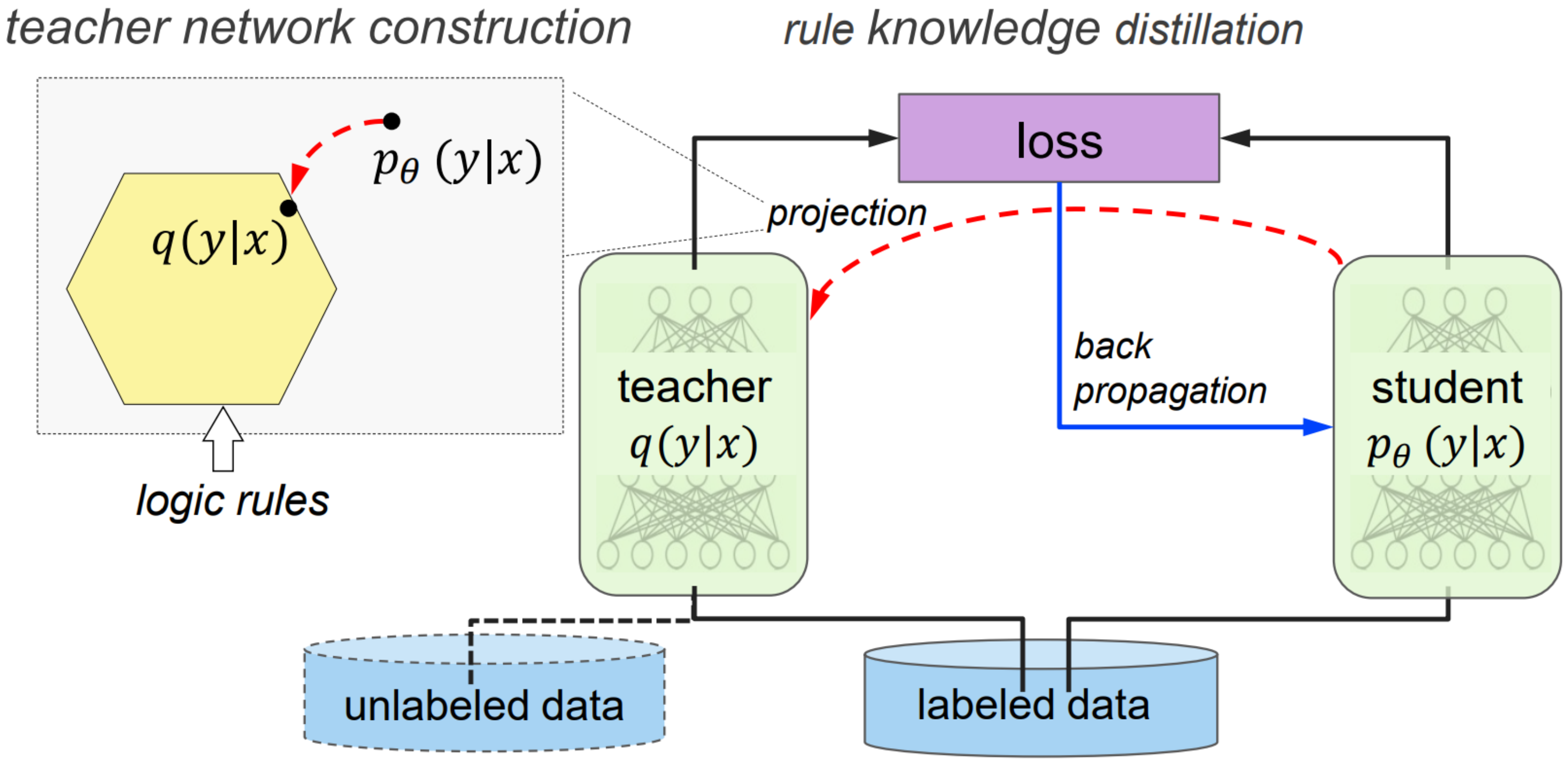
Harnessing deep neural networks with logic rules [Hu et al ‘16]

- Logic rules: provide a flexible declarative language for communicating high-level cognition and expressing structured knowledge
  - Desirable to integrate logic rules into DNNs, to transfer human intention and domain knowledge to neural models, and regulate the learning process
- Present a framework capable of enhancing general types of neural networks with logic rule knowledge.
- Previous works in exploiting a priori knowledge in general neural architectures
  - Augments each raw data instance with useful features while network training (Collobert et al., 2011)
    - However, is still limited to instance-label supervision and suffers from the same issues mentioned above.
    - Besides, a large variety of structural knowledge cannot be naturally encoded in the feature label form.

Harnessing deep neural networks with logic rules [Hu et al ‘16]

- Iterative rule knowledge distillation procedure
  - Learn simultaneously from labeled instances as well as logic rules
  - Transfers the structured information encoded in the logic rules into the network parameters
  - Combination of the knowledge distillation [Hinton et al., 2015; Bucilu et al., 2006] and the posterior regularization (PR) method [Ganchev et al., 2010]

Harnessing deep neural networks with logic rules [Hu et al '16]



At each iteration,

- **Teacher network construction (projection)**: the teacher network is obtained by projecting the student network to a rule-regularized subspace (red dashed arrow)
- **Back propagation**: the student network is updated to balance between emulating the teacher's output and predicting the true labels (black/blue solid arrows).

Harnessing deep neural networks with logic rules [Hu et al '16]

- Learning Resources: Instances and Rules
  - Assume we have input variable $\mathbf{x} \in \mathcal{X}$ and target variable $\mathbf{y} \in \mathcal{Y}$.
  - K-way classification setting:
    - $\mathcal{Y} = \Delta^K$: the K-dimensional probability simplex
    - $\mathbf{y} \in \{0,1\}^K \subset \mathcal{Y}$: a one-hot encoding of the class label
    - $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$: the training data

    the lth rule over the input-target space (X, Y)
  - $\mathcal{R} = \{(R_l, \lambda_l)\}_{l=1}^L$: a set of first-order logic (FOL) rules with confidences
  - $\{r_{lg}(\mathbf{X}, \mathbf{Y})\}_{g=1}^{G_l}$: the set of groundings of R_l

Harnessing deep neural networks with logic rules [Hu et al ‘16]

- Soft logic for the FOL rules
  - Allows continuous truth values from the interval [0, 1]
  - Reformulate Boolean logic operators:

$$A \& B = \max\{A + B - 1, 0\}$$

$$A \vee B = \min\{A + B, 1\}$$

$$A_1 \wedge \cdots \wedge A_N = \sum_i A_i / N$$

$$\neg A = 1 - A$$

Harnessing deep neural networks with logic rules [Hu et al ‘16]

- Rule Knowledge Distillation
  - $p_\theta(\boldsymbol{y}|\boldsymbol{x})$: a conditional probability using a softmax output layer that produces a K-dimensional soft prediction vector
  - $\boldsymbol{\sigma}_\theta(\mathbf{x})$: a K-dimensional soft prediction vector
  - $q(\boldsymbol{y}|\boldsymbol{x})$: a rule-regularized projection of $p_\theta(\boldsymbol{y}|\boldsymbol{x})$
  - In each iteration $\boldsymbol{q}$ is constructed by projecting p_θ into a subspace constrained by the rules, and thus has desirable properties
  - The prediction behavior of q reveals the information of the regularized subspace and structured rules

Harnessing deep neural networks with logic rules [Hu et al '16]

- Rule Knowledge Distillation
  - Emulating the q outputs serves to transfer this knowledge into p_θ.
  - The objective function

$$\boldsymbol{\theta}^{(t+1)} = \arg\min_{\theta\in\Theta} \frac{1}{N}\sum_{n=1}^{N}(1-\pi)\ell(\mathbf{y}_n, \boldsymbol{\sigma}_\theta(\mathbf{x}_n)) + \pi\ell(\mathbf{s}_n^{(t)}, \boldsymbol{\sigma}_\theta(\mathbf{x}_n)),$$

the imitation parameter calibrating the relative importance of the two objectives

the soft prediction vector of q on x_n at iteration t

  - $p_\theta(\boldsymbol{y}|\boldsymbol{x})$: the student
  - $q(\boldsymbol{y}|\boldsymbol{x})$: teacher

Harnessing deep neural networks with logic rules [Hu et al ‘16]

- Teacher Network Construction
  - Find the optimal q that fits the rules while at the same time staying close to p_θ
    - 1) $\mathbb{E}_{q(\mathbf{Y}|\mathbf{X})}[r_{lg}(\mathbf{X},\mathbf{Y})] = 1$, with confidence λ_l
    - 2) measure the closeness between q and p_θ with KL-divergence
  - The optimization problem:

$$\begin{aligned} \min_{q,\boldsymbol{\xi}\geq 0} \quad & \mathrm{KL}(q(\mathbf{Y}|\mathbf{X})\|p_\theta(\mathbf{Y}|\mathbf{X})) + C\sum\nolimits_{l,g_l}\xi_{l,g_l} \\ \text{s.t.} \quad & \lambda_l(1-\mathbb{E}_q[r_{l,g_l}(\mathbf{X},\mathbf{Y})]) \leq \xi_{l,g_l} \\ & g_l = 1,\ldots,G_l, \quad l=1,\ldots,L, \end{aligned}$$

    - Can be seen as projecting p_θ into the constrained subspace

$$\Rightarrow \quad q^*(\mathbf{Y}|\mathbf{X}) \propto p_\theta(\mathbf{Y}|\mathbf{X}) \exp\left\{-\sum_{l,g_l} C\lambda_l(1-r_{l,g_l}(\mathbf{X},\mathbf{Y}))\right\}$$

Harnessing deep neural networks with logic rules [Hu et al ‘16]

Algorithm 1 Harnessing NN with Rules

```
Input: The training data $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$,
       The rule set $\mathcal{R} = \{(R_l, \lambda_l)\}_{l=1}^L$,
       Parameters: $\pi$ – imitation parameter
                   $C$ – regularization strength
1: Initialize neural network parameter $\boldsymbol{\theta}$
2: repeat
3:     Sample a minibatch $(\mathbf{X}, \mathbf{Y}) \subset \mathcal{D}$
4:     Construct teacher network $q$ with Eq.(4)
5:     Transfer knowledge into $p_\theta$ by updating $\boldsymbol{\theta}$ with Eq.(2)
6: until convergence
Output: Distill student network $p_\theta$ and teacher network $q$
```

Harnessing deep neural networks with logic rules [Hu et al '16]

- Experiments: Sentiment classification
  - Logic rules
    - consider sentences S with an "A-but-B" structure
    - expect the sentiment of the whole sentence to be consistent with the sentiment of clause B

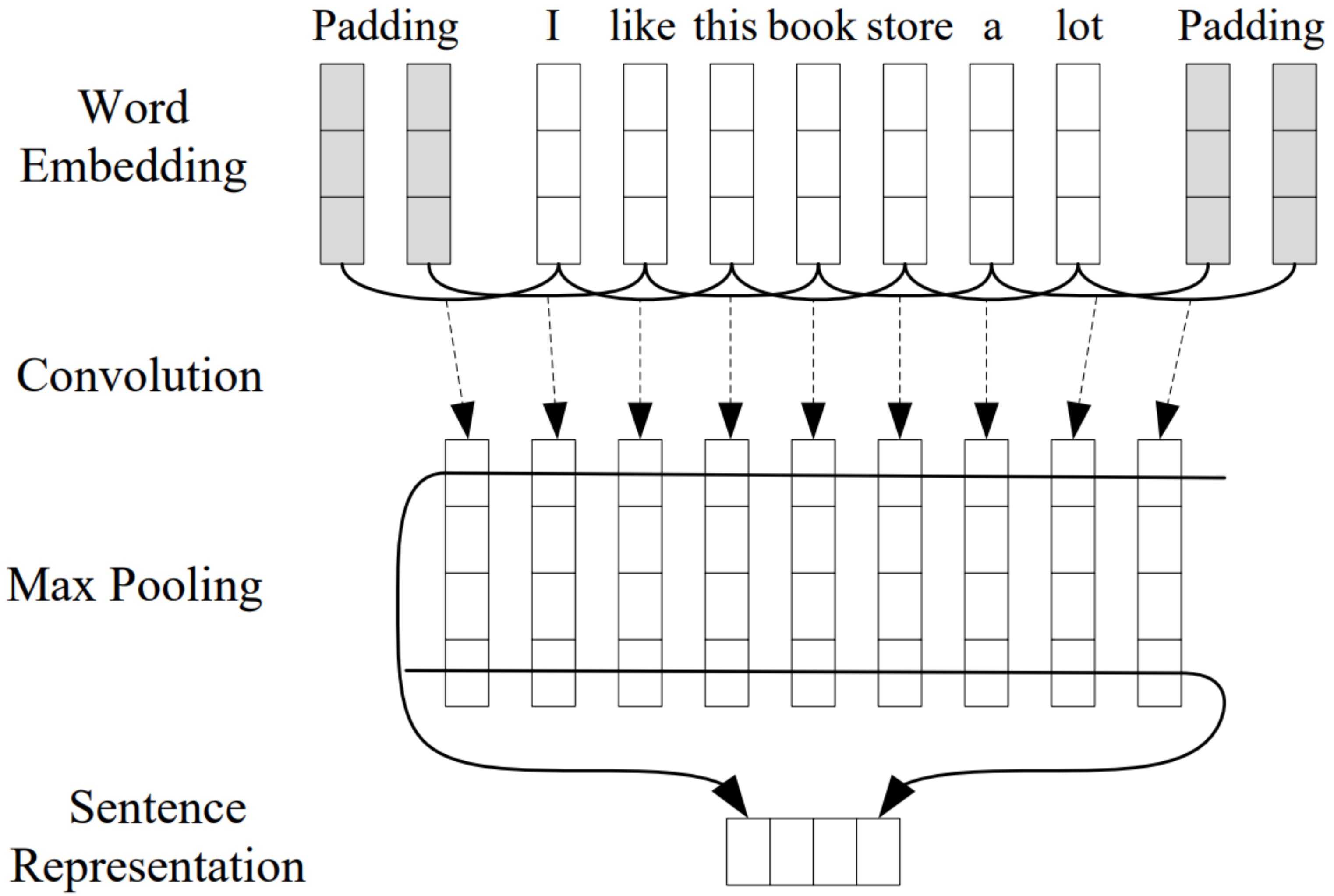
the element of $\sigma_\theta(B)$ for class '+'

$$\text{has-`A-but-B'-structure}(S) \Rightarrow$$
$$(\mathbf{1}(y=+) \Rightarrow \boldsymbol{\sigma}_\theta(B)_+ \;\wedge\; \boldsymbol{\sigma}_\theta(B)_+ \Rightarrow \mathbf{1}(y=+))$$

- According to the soft logic, the truth value of the logic rule when S has the 'A-but-B' structure:
- $(1+\boldsymbol{\sigma}_\theta(B)_+)/2$ for $y = +$
- $(2-\boldsymbol{\sigma}_\theta(B)_+)/2$ for $y = -$

Harnessing deep neural networks with logic rules [Hu et al '16]

- Sentiment classification



Harnessing deep neural networks with logic rules [Hu et al ‘16]

- Named Entity Recognition
  - Logic Rules

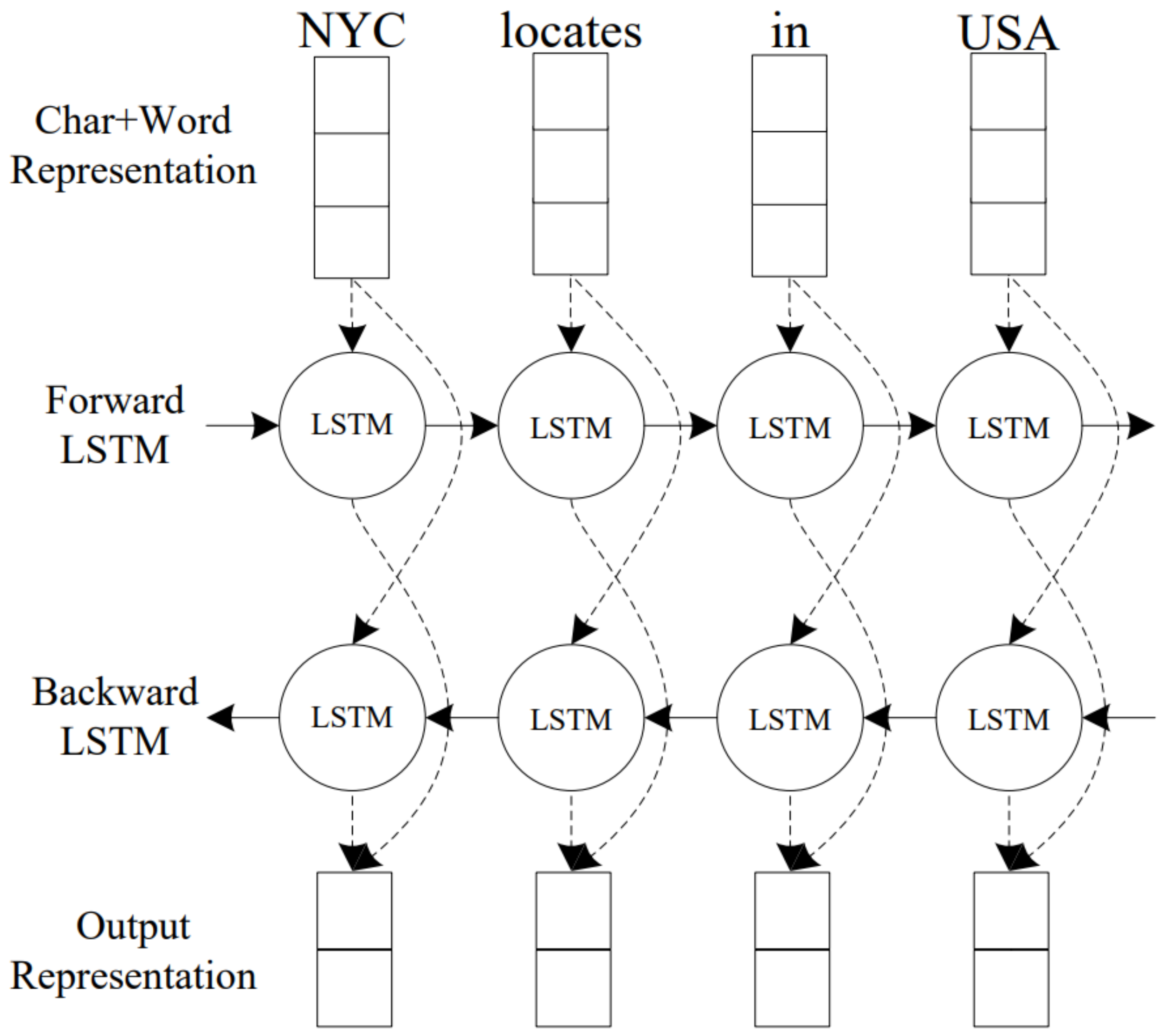
$$\text{equal}(y_{i-1}, \text{I-ORG}) \Rightarrow \neg\ \text{equal}(y_i, \text{B-PER})$$

  - Leverage the list structures within and across sentences of the same documents.
    - E.g.) “1. Juventus, 2. Barcelona, 3. ...”

$$\text{is-counterpart}(X, A) \Rightarrow 1 - \|c(\mathbf{e}_y) - c(\boldsymbol{\sigma}_\theta(A))\|_2$$

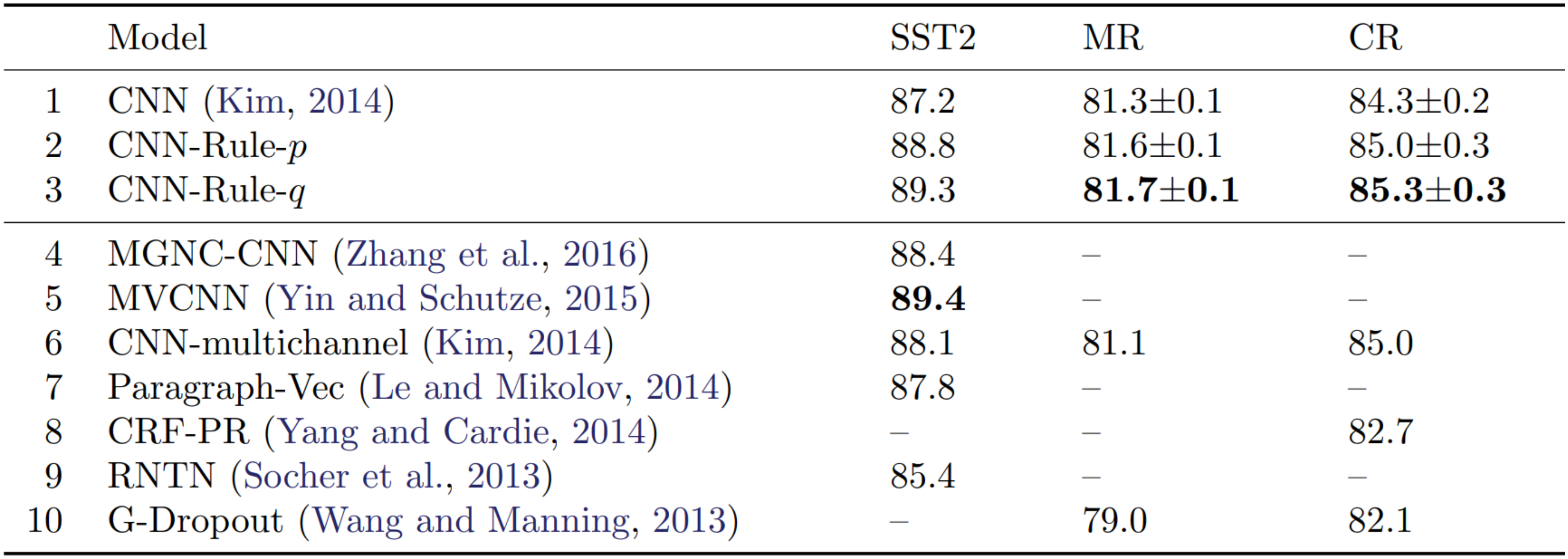
  - $c(\cdot)$ collapses the probability mass on the labels with the same categories into a single probability
    - yielding a vector with length equaling to the number of categories

Harnessing deep neural networks with logic rules [Hu et al '16]



Harnessing deep neural networks with logic rules [Hu et al '16]

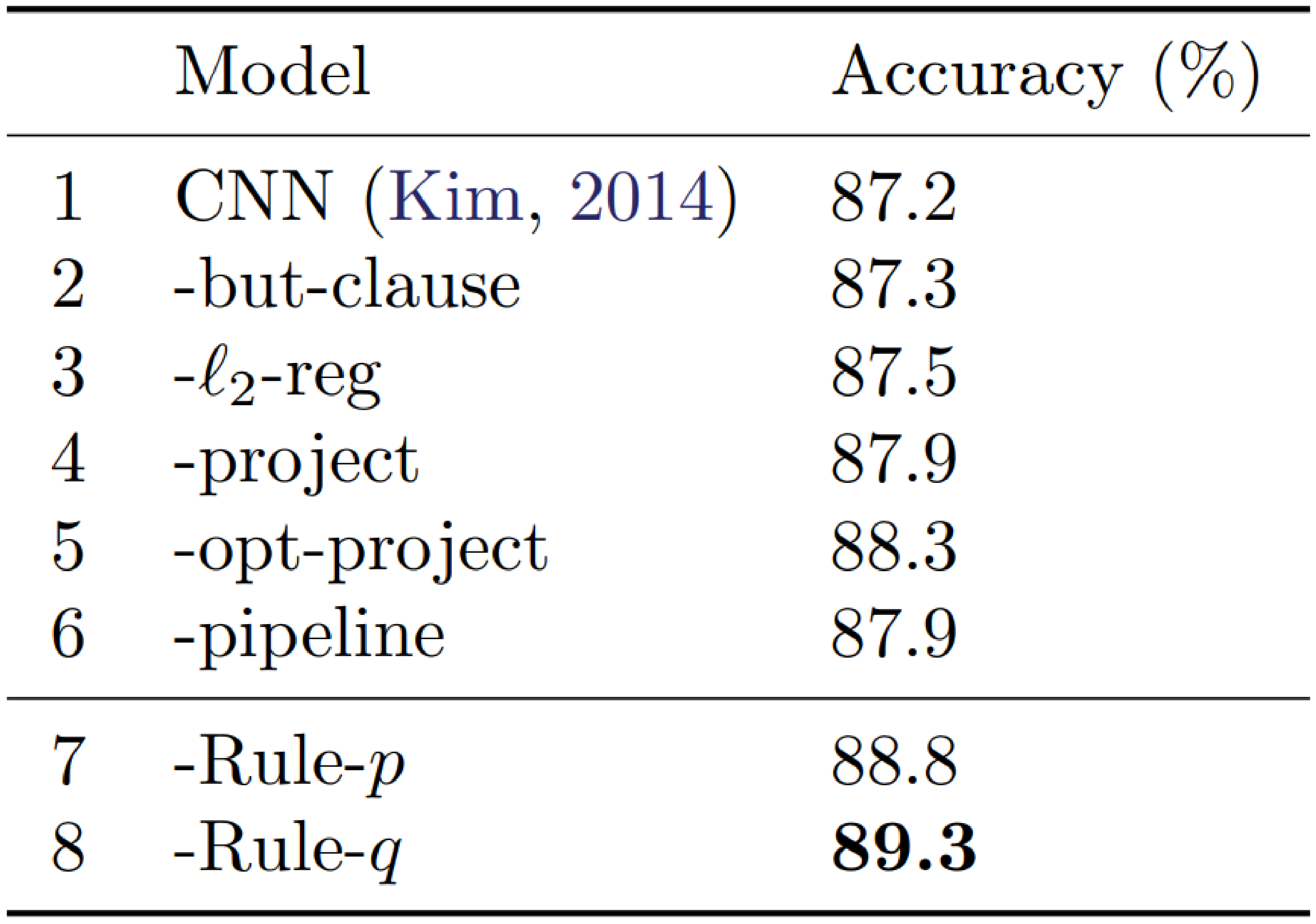
- Experiment results
  - Sentiment classification



	Model	SST2	MR	CR
1	CNN (Kim, 2014)	87.2	81.3±0.1	84.3±0.2
2	CNN-Rule-*p*	88.8	81.6±0.1	85.0±0.3
3	CNN-Rule-*q*	89.3	**81.7±0.1**	**85.3±0.3**
4	MGNC-CNN (Zhang et al., 2016)	88.4	–	–
5	MVCNN (Yin and Schutze, 2015)	**89.4**	–	–
6	CNN-multichannel (Kim, 2014)	88.1	81.1	85.0
7	Paragraph-Vec (Le and Mikolov, 2014)	87.8	–	–
8	CRF-PR (Yang and Cardie, 2014)	–	–	82.7
9	RNTN (Socher et al., 2013)	85.4	–	–
10	G-Dropout (Wang and Manning, 2013)	–	79.0	82.1

Harnessing deep neural networks with logic rules [Hu et al ‘16]

- Experiment results
  - Sentiment classification



	Model	Accuracy (%)
1	CNN (Kim, 2014)	87.2
2	-but-clause	87.3
3	-ℓ_2-reg	87.5
4	-project	87.9
5	-opt-project	88.3
6	-pipeline	87.9
7	-Rule-p	88.8
8	-Rule-q	**89.3**

Harnessing deep neural networks with logic rules [Hu et al '16]

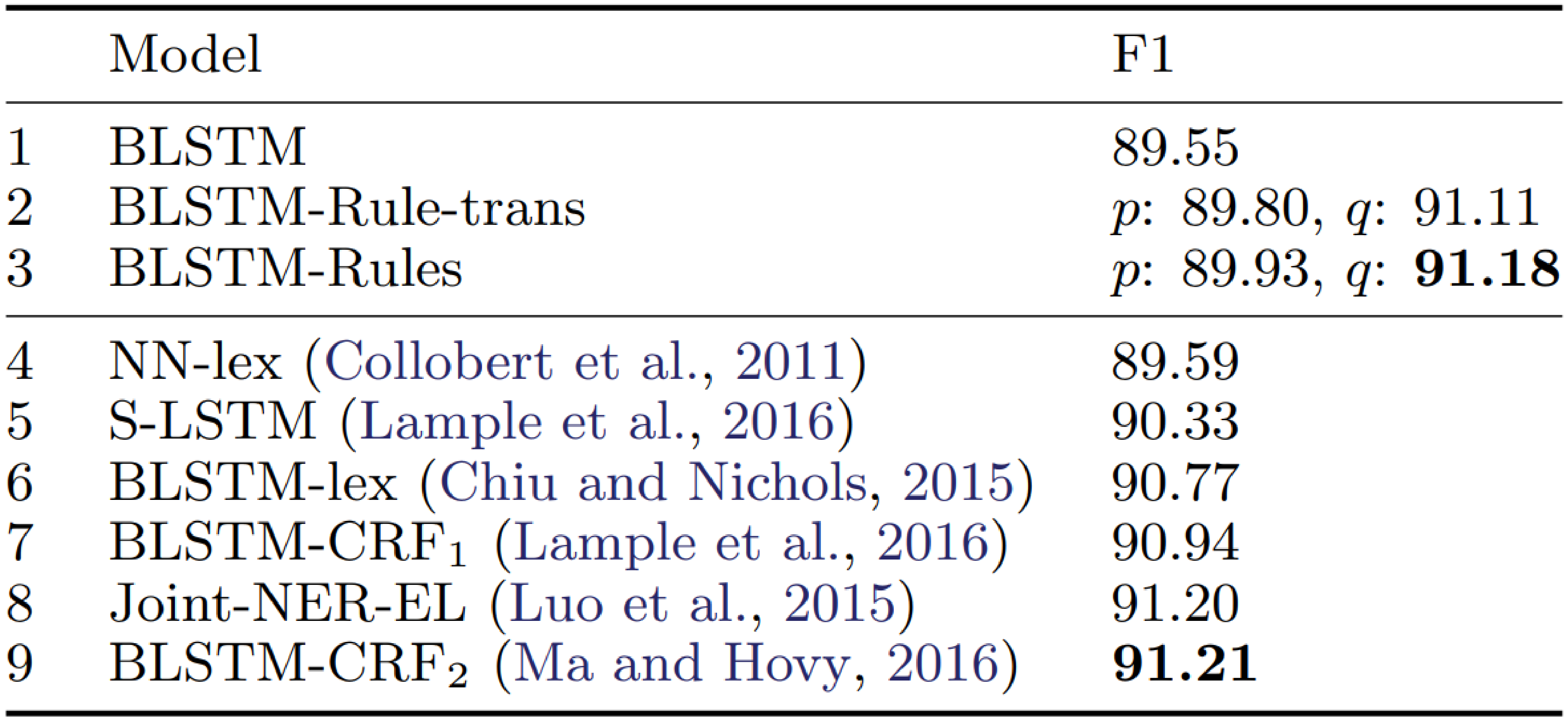
- Experiment results
  - Sentiment classification

	Data size	5%	10%	30%	100%
1	CNN	79.9	81.6	83.6	87.2
2	-Rule-p	81.5	83.2	84.5	88.8
3	-Rule-q	82.5	83.9	85.6	**89.3**
4	-semi-PR	81.5	83.1	84.6	–
5	-semi-Rule-p	81.7	83.3	84.7	–
6	-semi-Rule-q	**82.7**	**84.2**	**85.7**	–

semi-supervised learning where the remaining training data are used as unlabeled examples

Harnessing deep neural networks with logic rules [Hu et al ‘16]

- Experiment results
  - NER



	Model	F1
1	BLSTM	89.55
2	BLSTM-Rule-trans	p: 89.80, q: 91.11
3	BLSTM-Rules	p: 89.93, q: **91.18**
4	NN-lex (Collobert et al., 2011)	89.59
5	S-LSTM (Lample et al., 2016)	90.33
6	BLSTM-lex (Chiu and Nichols, 2015)	90.77
7	BLSTM-CRF$_1$ (Lample et al., 2016)	90.94
8	Joint-NER-EL (Luo et al., 2015)	91.20
9	BLSTM-CRF$_2$ (Ma and Hovy, 2016)	**91.21**

The Consciousness Prior [Bengio '17]

- Kahneman's system 2 cognitive abilities [Kahneman, 2011]
- System 1 tasks
  - Align well with the current successful applications of deep learning
    - E.g.)
      - low-level perception (and to a lesser extent low-level action)
      - intuitive knowledge (e.g. knowing that a particular Go move is good or that a given picture contains the image of a dog), i.e., knowledge which is difficult to verbalize, and which can typically be applied very quickly (in less than a second).

The Consciousness Prior [Bengio ’17]

- Kahneman’s system 2 cognitive abilities [Kahneman, 2011]
- System 2 cognitive abilities
  - can be described verbally, and thus includes the part of our cognitive abilities which we can communicate explicitly to a computer (typically as a sequence of computational steps)
  - Include things like reasoning, planning and imagination.
  - Typical system 2 tasks require a sequence of conscious steps, also means that they tend to take more time than system 1 tasks.
  - Closely related to consciousness

The Consciousness Prior [Bengio ’17]

- Global Workspace Theory [Baars, 1988, 1997, 2002, Dehaene and Naccache, 2001, Dehaene et al., 2017]
  - Posits that we become aware of specific pieces of information which will momentarily form the content of working memory.
  - A conscious thought is thus a set of these elements of which we have become aware, joined together and made globally available to other computational processes taking place in the brain at an unconscious level.
- Consciousness thus provides a form of bottleneck for information which has a strong influence on
  - Decision-making (voluntary action),
  - Memory (we tend to very quickly forget what we have not been consciously aware of), and
  - Perception (we may be blind to elements of our sensory input which may distract us from the current focus of conscious attention)

The Consciousness Prior [Bengio '17]

- System 2 Processing
  - Global Workspace Theory of Consciousness
- Consciousness Prior Theory
  - Extracting a Conscious State

$$h_t = F(x_t, h_{t-1})$$

    - x_t: the observation at time t for a learning agent
    - h_t: the **unconscious representation state.**
    - F: **Representation RNN**
  - Define the conscious state c_t
    - as a very low-dimensional set which is derived from h_t by a form of attention mechanism applied on h_t

$$c_t = C(h_t, c_{t-1}, m_{t-1}, z_t)$$

$$m_t = M(m_{t-1}, c_t).$$

z_t: a random noise source

m_t: the content of memory at time t.

    - The function C: the **consciousness process**

The Consciousness Prior [Bengio '17]

- Factor Graph $S = \{V_1, \ldots V_n\}$

$$P(S) = \frac{\prod_j f_j(S_j)}{Z}$$

- **Sparse Factor Graph**
  - the consciousness prior amounts to the assumption of Sparse Factor Graph:
    - the factor graph for the joint distribution between the elements in the set h_t (or more generally for the set containing all of the elements in mt and all those one could think of in the future) is sparse
    - The motivation comes from observing
      - The structure of natural language (broken down into phrases, statements or sentences, each of which involves very few words)
      - The structure of formal knowledge representations such as the sets of facts and rules studied in classical symbolic / logic AI or in ontologies and knowledge graphs

The Consciousness Prior [Bengio ’17]

- **Verifier network**
  - To capture the assumptiontion a conscious thought can encapsulate a statement about the future
  - match a current representation state h_t with a past conscious state c_{t-k} stored in memory m_{t-1}

$$V(h_t, c_{t-k}) \in \mathbb{R}$$

    - indicates the consistency of c_{t-k} with h_t, e.g., estimating the probability of the correspo nding statement being true, given ht