Fast Relational Learning using Neural-Symbolic Systems

Artur S. d'Avila Garcez

Department of Computer Science
City University London, EC1V 0HB London, United Kingdom
a.garcez@city.ac.uk

Imperial College London
29 November 2013

Neural-Symbolic Integration

- How can Computer Science reconcile the logical nature of reasoning with the statistical nature of learning?
- Neural-symbolic systems seek to offer high-level symbolic representations to efficient connectionist networks through *translation algorithms* and *extraction algorithms*

- Paul Smolensky, On the proper treatment of connectionism, BBS, 11(1), 1988
- John McCarthy, Epistemological challenges for connectionism, BBS 11(1) (commentary), 1988
- Andy Clark, Whatever next? Predictive brains, situated agents, and the future of cognitive science, BBS 36(13), 2013

Relational Learning

- Learning a *first-order logic* theory from examples (in the presence of uncertainty)
  - By searching for candidate hypotheses in first-order level or through *propositionalization*
  - Relevant for the analysis of complex networks: drug design in bioinformatics, link analysis in social networks, etc.
  - Related work: (probabilistic) ILP, deep networks, MLNs, BLOG, StarAI (lifted inference), etc, etc.
- Hypothesis search can be costly (but see streaming ILP (ILP'2013), online relational learning (ECML'12)), but sound and sometimes complete at inducing theories
- Propositionalization offers a trade-off between information loss and efficiency

Relational Learning in Neural-Symbolic Systems

- A number of attempts at first-order (and higher-order) logic learning in neural networks: semantic approach (Hitzler et al), fibring, topos (Osnabruck), association (SHRUTI), unification, etc.
- CILP++ is a neural-symbolic system that can solve ILP problems efficiently (through propositionalization) using a neural network trained with backpropagation (França, Zaverucha and d'Avila Garcez, Mach. Learn., July 2013)
- Neural-symbolic cognitive agent: situation -> pattern matching (efficient, handle uncertainty), learning (adaptation from data and prior knowledge, also uncertain), description (communication, transfer), considerate reasoning (change), explanation (synthesis) -> action

Efficient Relational Learning using CILP++

- CILP++ (available to download from sourceforge) is composed of:
  - Bottom Clause Propositionalization (BCP): creates a bottom clause for each positive and negative ILP example
  - Artificial Neural Networks (ANNs) trained with backpropagation: generalising from a set of bottom clauses given as binary vectors (c.f. Muggleton and Tamaddoni-Nezhad, QG-GA, Mach. Learn. 2008)
  - Presenting the trained knowledge in relational form: mapping trained features into first-oder logic representation (ICCSW 2013, Dagstuhl OASIcs, September 2013)

Summary of Results on Relational Learning

- CILP++ vs. state-of-the-art ILP system Aleph
  - CILP++ achieved comparable accuracy while being consistently faster
  - Standard backprop vs. *early stopping* produces a trade-off of accuracy vs. efficiency
- BCP vs. the propositionalization component of RSD (Železný and Lavrač, 2006)
  - BCP achieved overall better accuracy and was faster
  - BCP with neural nets is much better than RSD with neural nets; BCP with C4.5 is comparable to RSD with C4.5

Summary of Results on Knowledge Extraction

- Initial evaluation of how rules learned from data propositionalized with BCP (BCP-rules) performed when mapped into first-order rules:
    - We compared such first-order rules with rules produced by Aleph and with the original BCP-rules (we used RIPPER (Cohen, 1995) as the rule learner, but any other propositional rule learner can be used)
    - As expected, there is information loss in comparison with Aleph, but in exchange for considerable speed-ups and smaller (more compact) rule sets
    - Although our extraction algorithm shows good fidelity to the BCP-rules, the "lifting" can improve on the accuracy of BCP-rules
- Experiments using knowledge extraction from neural networks are ongoing

A simple example: Family Relationship

- BCP: generates a bottom clause for each (positive or negative) example; converts each bottom clause into a binary vector (similarly to QG-GA)

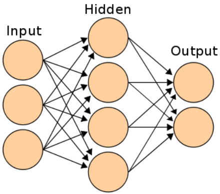
Background Knowledge:	**Positive Examples:**
mother(mom1, daughter1)	*motherInLaw(mom1, husband1)*
wife(daughter1, husband1)	**Negative Examples:**
wife(daughter2, husband2)	*motherInLaw(daughter1, husband2)*

- Using Progol's bottom clause algorithm (Muggleton, 1995):
  - $\bot_{+} = [motherInLaw(A, B) \leftarrow mother(A, C), wife(C, B)]$
  - $\bot_{-} = [motherInLaw(A, B) \leftarrow wife(A, C)]$
- BCP features: mother(A, C), wife(C, B), wife(A, C)
- "*mom1* is a mother-in-law because she has a married daughter", from $\bot_{+}$
- Is the set of bottom clauses useful for learning and generalization?

CITY UNIVERSITY LONDON

Neural networks and Backpropagation

- Artificial Neural Networks (ANNs) are popular connectionist models with applications in many areas, e.g. pattern recognition, control engineering.
- We use a multi-layer perceptron (MLP) with three layers of artificial neurons connected in a feed-forward fashion
- Error back-propagation is a widely used training algorithm for MLPs; it seeks to minimize an error function through gradient descent
- When using error back-propagation, a common way of dealing with overfitting is by using early stopping



Early stopping

- Early stopping aims to avoid overfitting (Caruana et. al., 2000) by using a validation set as stopping criteria
- The main early stopping criteria used are:
  - Stop when the ratio between the validation error and the current training epoch error gets higher than a specified α
  - Stop when the sum of the current epoch's training and validation errors becomes smaller than a specified β for m consecutive epochs
  - Stop when the validation error keeps increasing for n consecutive epochs
- There is empirical evidence that the first criterion leads to fair accuracy and faster convergence (Prechet (1997)), thus it is the one used by CILP++

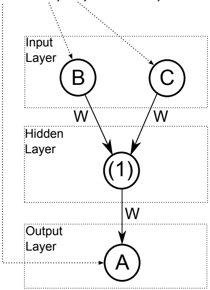
Connectionist Inductive Learning and Logic Programming (CILP)

- We are integrating (symbolic) logic and (numerical) ANN's
- CILP (Garcez and Zaverucha, 1999, Garcez, Broda and Gabbay, 2002) is a neural-symbolic system created by combining Logic Programming and ANN's, inspired by KBANN (Towell and Shavlik, 1994)
- A recursive ANN is built from a grounded logic program and it is then used for training, allowing learning from background knowledge and examples, reasoning in parallel, and knowledge extraction
- CILP was extended and applied to non-classical reasoning, argumentation, cognitive agents, bioinformatics, fault diagnosis, etc (Garcez, Lamb and Gabbay, 2009)

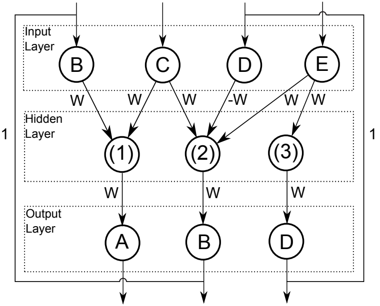
The CILP Translation Algorithm (example)

(1) (2) (3)

BK = {A ← B, C; B ← C, not D, E; D ← E}



N:



CITY UNIVERSITY LONDON

Bottom Clause Propositionalization (BCP)

- BCP uses Progol's bottom clause generation to create features for each first-order example
- Each positive example is saturated normally
- Negative examples are processed just like positive ones, but are labeled as negative
- All unifications made during bottom clause generation are stored into *hash* sets
- A feature table F is created, consisting of all distinct body literals from all the bottom clauses generated
- Then, a binary input vector v of size $|F|$ is created for each example $(\forall i, v(i) \in \{0, 1\})$

BCP (cont.)

Background Knowledge:
mother(mom1, daughter1)
wife(daughter1, husband1)
wife(daughter2, husband2)

Positive Examples
motherInLaw(mom1, husband1)

Negative Examples
motherInLaw(daughter1, husband2)

- Continuing the family relationship example:
  $\bot_+ = [motherInLaw(A, B) \leftarrow mother(A, C), wife(C, B)]$
  $\bot_- = [motherInLaw(A, B) \leftarrow wife(A, C)]$

  - From $\bot_+$, $hash_+$:

Key	Value
mom1	A
husband1	B
daughter1	C

  - From $\bot_-$, $hash_-$:

Key	Value
daughter1	A
husband2	B
husband1	C

  - $F = \{mother(A, C), wife(C, B), wife(A, C)\}$
  - $v_+ = (1, 1, 0)$
  - $v_- = (0, 0, 1)$

CITY UNIVERSITY LONDON

CILP++

- CILP++ is an extension of CILP to allow it to work with relational data
- ILP examples are propositionalized with BCP and used as learning patterns by the neural network
- CILP's translation algorithm is used to build an initial network using some of the examples, prior to training (optional)
- The network is trained with backpropagation and early stopping
- Both CILP and CILP++ are open-source systems, available at https://sourceforge.net/projects/cil2p/ and https://sourceforge.net/projects/cilppp/, respectively

Network training

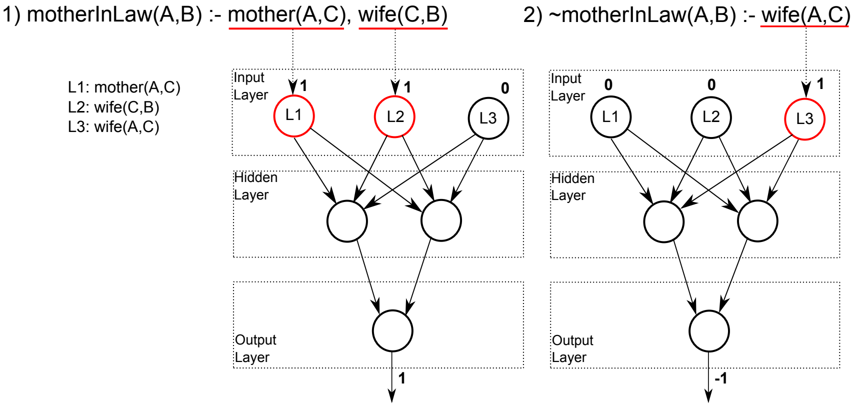
1. The network is fully-connected with near-zero weight connections
2. Positive examples are associate with a label $y = 1$ and negative examples with $y = -1$
3. The heuristic chosen to calculate the training error E is standard mean-squared error:

   $E = \sum_{i=1}^{n} \frac{(y_i - \bar{y}_i)^2}{2}$, where y_i and $\bar{y}_i$ are the example label and CILP++'s output, respectively
4. Both standard stopping criteria and early stopping have been evaluated

Stopping criteria

- Standard stopping criteria:
  - 300 training epochs have elapsed
  - 95% of the training data satisfies $E < 0.1$
  - 90% of the training data is correctly classified by the network and no improvements can be seen for 5 epochs
- Early stopping:
  - A validation set with $\frac{1}{5}$ of the total training set was created
  - At time t, the best model from epochs 1 to t is stored
  - If after epoch e, the condition (first criterion of Prechet, 1999)
    $GL(t) > \alpha,\ GL(t) = 0.1 * \left(\frac{E_{va}(e)}{E_{opt}(t)} - 1\right)$ is satisfied, training stops

Building the network and training (example)



CILP++ Evaluation: Datasets Used

Dataset statistics

Dataset	**# Positive Examples**	**# Negative Examples**	**# Predicates**	**# BCP Features**
Mutagenesis	125	63	34	1115
KRK	341	655	9	60
UW-CSE	113	226	37	430
Alz-amine	343	343	30	1090
Alz-acetyl	618	618	30	1363
Alz-memory	321	321	30	1052
Alz-toxic	443	443	30	1319

Experiments Description

- Four CILP++ configurations have been tested:
  - *st*: uses standard backpropagation stopping criteria
  - *es*: uses early stopping
  - *n%bk*: the network is created using *n%* of the examples in ($E_{\perp}^{train}$) as BK[1]
  - $2h$: when $n = 0$, the network uses 2 hidden neurons only!
- Why two hidden neurons on *2h* configurations?
  - It can help avoid overfitting in large ANN's
  - Accuracy doesn't seem to increase substantially for more than 2 hidden neurons when the input is a binary vector (Haykin, 2009)

[1] $n = 0$, 2.5 and 5 were used.

Experimental Results – Accuracy vs. Runtime

Dataset	***Aleph***	***CILP++***$_{st,2.5\%bk}$	***CILP++***$_{st,5\%bk}$	***CILP++***$_{st,2h}$
mutagenesis	80.85 $*$ ($\pm$10.5) **0:08:15**	**91.70**($\pm$5.84) 0:10:34	90.65($\pm$8.97) 0:11:15	89.20($\pm$8.92) 0:10:16
krk	**99.6**($\pm$0.51) 0:11:03	98.31 $*$ ($\pm$1.23) 0:04:38	98.32 $*$ ($\pm$1.25) **0:04:34**	98.42($\pm$1.26) 0:04:40
uw-cse	**84.91**($\pm$7.32) 0:45:47	66.24 $*$ ($\pm$7.01) **0:08:47**	66.08 $*$ ($\pm$2.48) 0:10:19	70.01 $*$ ($\pm$2.2) 0:08:54
alz-amine	78.71($\pm$5.25) 1:31:05	**78.99**($\pm$4.46) 1:23:42	76.02 $*$ ($\pm$3.79) 2:07:04	77.08($\pm$5.17) **1:14:21**
alz-acetyl	**69.46**($\pm$3.6) 8:06:06	63.64 $*$ ($\pm$4.01) 4:20:28	63.49 $*$ ($\pm$4.16) 5:49:51	63.30 $*$ ($\pm$5.09) **2:47:52**
alz-memory	**68.57**($\pm$5.7) 3:47:55	60.44 $*$ ($\pm$4.11) 1:41:36	59.19 $*$ ($\pm$5.91) 2:12:14	59.82 $*$ ($\pm$6.76) **1:19:27**
alz-toxic	80.5($\pm$3.98) 6:02:05	79.92($\pm$3.09) 3:04:53	80.49($\pm$3.65) 3:33:17	**81.73**($\pm$4.68) **2:12:17**

bold = best result; * indicates statistically significant difference from the best

- CILP++ achieves better accuracy AND better runtime in one *st* configuration!
- CILP++{st,2h} is generally faster than Aleph

CITY UNIVERSITY LONDON

Experimental Results with early stopping

Dataset	*Aleph*	*CILP++*$_{es,2.5\%bk}$	*CILP++*$_{es,5\%bk}$	*CILP++*$_{es,2h}$
mutagenesis	80.85(±10.51) 0:08:15	83.48(±7.68) **0:01:25**	83.01(±10.71) 0:01:43	**84.76**(±8.34) 0:01:50
krk	**99.6**(±0.51) 0:11:03	98.16∗(±0.83) **0:04:08**	96.33∗(±4.95) 0:04:28	98.31(±1.23) 0:04:18
uw-cse	**84.91**(±7.32) 0:45:47	68.16∗(±4.77) **0:04:08**	65.69∗(±1.81) 0:04:16	67.86∗(±1.79) **0:04:08**
alz-amine	**78.71**(±3.51) 1:31:05	65.33∗(±9.32) 0:35:27	65.44∗(±5.58) **0:08:30**	70.26∗(±7.1) 0:10:14
alz-acetyl	**69.46**(±3.6) 8:06:06	64.97∗(±5.81) 3:04:47	64.88∗(±4.64) 2:42:31	65.47∗(±2.43) **0:25:43**
alz-memory	**68.57**(±5.7) 3:47:55	53.43∗(±5.64) 1:40:51	54.84∗(±6.01) 3:57:39	51.57∗(±5.36) **1:33:35**
alz-toxic	**80.5**(±4.83) 6:02:05	67.55∗(±6.36) **0:12:33**	67.26∗(±7.5) 0:14:04	74.48∗(±5.62) 0:28:39

- Considerable speed-ups are obtained, in exchange for accuracy at times

Propositionalization Experimental Results: BCP vs. RSD

Dataset	***Aleph***	***BCP+ANN***	***RSD+ANN***	***BCP+C4.5***	***RSD+C4.5***
muta	80.85 ∗ (±10.51) 0:08:15	**89.20**(±8.92) 0:10:16	67.63 ∗ (±16.44) 0:11:11	85.43 ∗ (±11.85) **0:02:01**	87.77(±1.02) 0:02:29
krk	**99.6**(±0.51) 0:11:03	98.42 ∗ (±1.26) 0:04:40	72.38 ∗ (±12.94) 0:06:21	98.84 ∗ (±0.77) **0:01:59**	96.1 ∗ (±0.11) 0:05:54

- BCP achieves better accuracy than RSD overall, while being slightly faster
- Accuracy of BCP with ANNs is much higher than RSD with ANNs

Relational Knowledge Extraction

- Since the BCP features are first-order literals, the first step towards knowledge extraction is to evaluate the first-order meaning of those rules
- *...but the rules generated do not obey any ILP restrictions as imposed by a language bias*
- We introduce an efficient relational knowledge extraction algorithm from BCP-rules, in order to allow first-order inference (França et. al., ICCSW'13)
- We used the propositional rule learner RIPPER (although any rule learner could have been used)

From BCP-rules to first-order rules

- In order to illustrate our approach, consider the following BCP data (set of bottom clauses):

$$\begin{aligned} S_\perp = \{motherInLaw(A,B) &\ :- \ mother(C,B), wife(C,D);\\ motherInLaw(A,B) &\ :- \ mother(A,C), wife(C,B);\\ \sim motherInLaw(A,B) &\ :- \ wife(C,B), parents(C,B,D), dad(E,F)\}. \end{aligned}$$

- A possible BCP-rule generated by RIPPER is:

$$R_\perp = \{motherInLaw(A,B) :- mother(A,C), wife(C,D), not(dad(E,F))\}.$$

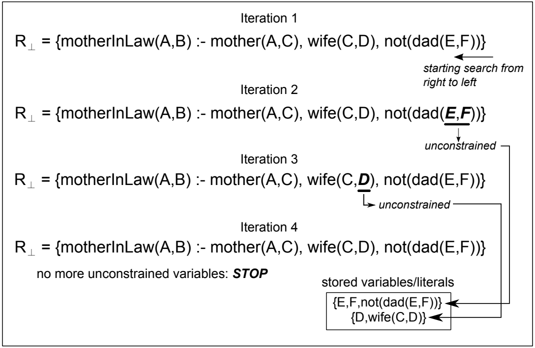
- Since BCP-rules are created by a propositional learner, they do not comply with variable chaining and/or language bias restrictions

From BCP to FOL (cont.)

- One needs a way of extracting first-order rules first in order to perform relational inference
- Our extraction algorithm can be divided into 3 steps:
  - *Unconstrained variable search*
  - *Unconstrained variable grounding*
  - *First-order filtering*

Unconstrained variable search

- Search for all the literals whose variables do not obey the ILP variable chaining



Unconstrained variable grounding

- All stored variable/literal pairs are replaced by an equivalent disjunction of ground terms
- BCP's hash tables will be used to obtain ground terms which are equivalent to the ones containing unconstrained variables
- This equivalence is proved by the BCP feature equivalence theorem (next)

Grounding unifier

- Let $e \in E$ be an example of an dataset E, post-BCP
- Let f be a BCP feature
- Let U be the set of all unconstrained variables which can be found inside f, having size k
- Let $hash_e$ be the unification hash generated for example e

Definition

The *grounding unifier* θ_e^f for a feature f w.r.t. e is defined as

$$\theta_e^f = \{v^1/c^1, v^2/c^2, \cdots, v^k/c^k\},$$

- where:
  - $v_i \in U, 1 \leq i \leq k$ is an unconstrained variable
  - c_i is the constant which is mapped to v_i, according to $hash_e$

BCP feature equivalence theorem

- Let E be an ILP example set, post-BCP and having size n
- Let f be one of the generated features with BCP when applied to E
- Let $v(f)$ be a valuation function, which associates a boolean truth-value to a feature f
- Let $hash_e$ be the unification hash generated for example e

Theorem

Then,

$$v(f) \equiv v(f\theta^f_{e_1}) \vee v(f\theta^f_{e_2}) \vee \cdots \vee v(f\theta^f_{e_n}),$$

where:

- $e_1, e_2 \cdots e_n \in E$ are the examples
- $f\theta^f_{e_i}$ is the unification of f with a grounding unifier $\theta^f_{e_i}$, $1 \leq i \leq n$

Example: Family relationship revisited

$$
\begin{aligned}
S_\perp = \{motherInLaw(A,B) &\; :- \; mother(C,B), wife(C,D);\\
motherInLaw(A,B) &\; :- \; mother(A,C), wife(C,B);\\
\sim motherInLaw(A,B) &\; :- \; wife(C,B), parents(C,B,D), dad(E,F)\}.
\end{aligned}
$$

$$R_\perp = \{motherInLaw(A,B) :- mother(A,C), wife(C,D), not(dad(E,F))\}.$$

$hash = \{D/husband2, D/daughter1, E/husband1, F/daughter1\}$

- From the BCP feature equivalence:
  $not(dad(E,F)) \mapsto not(dad(husband1, daughter1))$
  $wife(C,D) \mapsto wife(C, daughter1) \vee wife(C, husband2)$
- The resulting first-order rule set is:

$$R_\perp^C = \{motherInLaw(A,B) :- mother(A,C), (wife(C, daughter1); wife(C, husband2)), not(dad(husband1, daughter1))\}.$$

First-order filtering

- We can improve results further by making the theory more compact
- A modified version of the theory filtering algorithm *T-reduce* (A. Srinivasan, The Aleph System, version 5) is applied
  - Original T-reduce: removes sets of rules without positive coverage or that contribute negatively to training set accuracy
  - Modified T-reduce: also removes redundant literals, e.g. literals that are always true, and literals containing no variables
- If applied on $R_\perp^C$:

$$R_\perp^{FOL} = \{motherInLaw(A,B) \text{ :- } mother(A,C), (wife(C, daughter1); wife(C, husband2))\}$$

- With this, we can do consistent first-order inference from BCP-data

CITY UNIVERSITY LONDON

Experimental settings

- We have evaluated our approach (named BCP+RIP$_{FOL}$) in comparison with:
  - Aleph (as a baseline)
  - BCP + RIPPER (named BCP+RIP$_{prop}$)
- We use the Alzheimers benchmark for comparison
- The used metrics for evaluation are:
  - Classification accuracy
  - Runtime
  - Theory size (i.e. total number of literals)

Initial Results

	Alz-ami	*Alz-ace*	*Alz-mem*	*Alz-tox*
Aleph (baseline)	78.71($\pm$5.25) 1:31:05, 36.1	69.46($\pm$4.6) 8:06:06, 47.3	68.57($\pm$5.7) 3:47:55, 45.7	80.5($\pm$4.83) 6:02:05, 37.9
$BCP{+}RIP_{prop}$	73.35($\pm$4.32) 0:19:49, 30	67.8($\pm$3.77) 0:23:21, 20.1	65.27($\pm$7.11) 0:25:11, 14.4	78.44($\pm$5.44) 0:17:41, 35.2
$BCP{+}RIP_{FOL}$	77.73($\pm$4.57) 0:21:59, 30.4	63.56($\pm$5.06) 0:26:39, 18.7	57.64($\pm$5.7) 0:28:45, 13.8	66.45($\pm$6.93) 0:20:57, 18

Accuracy and theory size averaged over 10-fold cross-validation

- As expected, Aleph's accuracy is superior than *translation and extraction*
- But our approach seems to produce first-order theories that are more compact than Aleph
- And the accuracy of BCP+RIP$_{FOL}$ is even higher than BCP+RIP in one case

CITY UNIVERSITY LONDON

Conclusion and Future Work

- CILP++: Fair accuracy can be obtained efficiently by using a neural-symbolic approach
- In principle, any ILP dataset can be used with CILP++
- BCP can be used with any propositional learner capable of processing patterns with high feature dimensionality
- Next steps:
  - Evaluation of noise robustness and theory revision
  - Online learning, as an alternative to streaming ILP (Srinivasan and Bain, ILP'13)
- Relational Extraction: capable of generating compact first-order rule sets with negation
- Considerably faster than Aleph, but with some (sometimes considerable) accuracy losses
- Ongoing:
  - First-order knowledge extraction from neural networks
  - Comparing accuracy results with other fast ILP approaches

CITY UNIVERSITY LONDON

Thank You!

Questions?

Artur Garcez: a.garcez@city.ac.uk

CITY UNIVERSITY LONDON