Fast Relational Learning using Neural-Symbolic Systems (CILP++)

Artur S. d'Avila Garcez

City, University of London
a.garcez@city.ac.uk

ACAI, Ferrara
30 August 2018

Relational Learning

- Learning a *first-order logic* theory from examples (in the presence of uncertainty)
  - By searching for candidate hypotheses at 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.
- Hypothesis search (sound and sometimes complete at inducing theories) can be costly (but see streaming ILP (ILP'2013), online relational learning (ECML'12) and more recent developments (including differentiable (d)ILP, JAIR 2018).

Relational Learning in Neural-Symbolic Systems

- A number of attempts at first-order (and higher-order) logic learning in neural nets: semantic approach (Hitzler et al), fibring, topos (Osnabruck), association (SHRUTI), unification, etc.
- Propositionalization offers a trade-off between information loss and efficiency; it seems a natural choice for use with neural nets.
- 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)

Efficient Relational Learning using CILP++

- CILP++ 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 representations (ICCSW 2013, Dagstuhl OASIcs, 2013)

Summary of Results on Relational Learning

- CILP++ vs. ILP system Aleph
  - CILP++ achieved comparable accuracy while being consistently faster
  - Standard backprop vs. *early stopping*: trade-off between accuracy and 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 Relational Knowledge Extraction

- Initial evaluation of FOL rules learned by CILP++ (BCP-rules), extracted from the neural network trained using BCP propositionalization:
  - We compared BCP-rules with rules produced by Aleph (we used RIPPER (Cohen, 1995) as the rule learner, but other propositional rule learners 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 network, the lifting of FOL rules from trained neural nets can improve accuracy!
- Experiments on FOL knowledge extraction from neural networks have been limited and are ongoing. Next step: use macro operators (R. Mooney) with CILP++

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

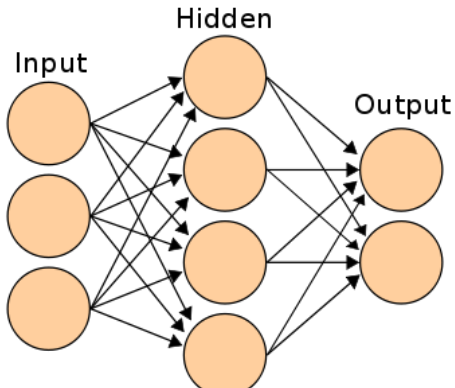
Positive Examples:
motherInLaw(mom1, husband1)

Negative Examples:
motherInLaw(daughter1, husband2)

- Using Progol's bottom clause algorithm (Muggleton, 1995):
  - $\perp_+ = [motherInLaw(A, B) \leftarrow mother(A, C), wife(C, B)]$
  - $\perp_- = [motherInLaw(A, B) \leftarrow wife(A, C)]$
- BCP features: mother(A, C), wife(C, B), wife(A, C)
- Hypothesis: Is the use of a **set** of bottom clauses useful for learning and generalization?

Neural networks and Backpropagation

- Popular connectionist models with application in many areas: pattern recognition, games, vision, speech, control
- CILP++ uses a multi-layer perceptron (MLP) with three layers of artificial neurons connected in a feed-forward manner; one could use a deep network instead, or even a recurrent network as used by CILP.
- 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 to use early stopping



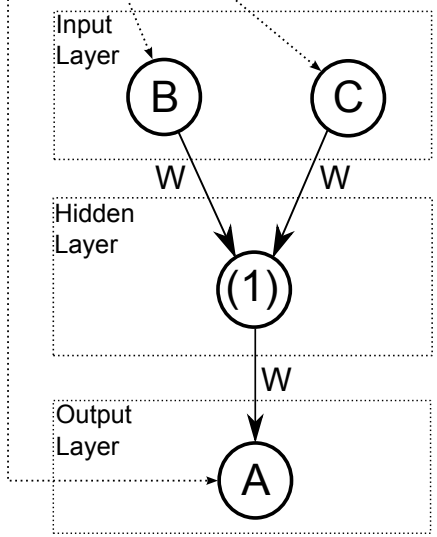
Early stopping

- Early stopping aims to avoid overfitting (Caruana et. al., 2000) by using a validation set as stopping criteria
- The main early stopping criteria 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 used by CILP++

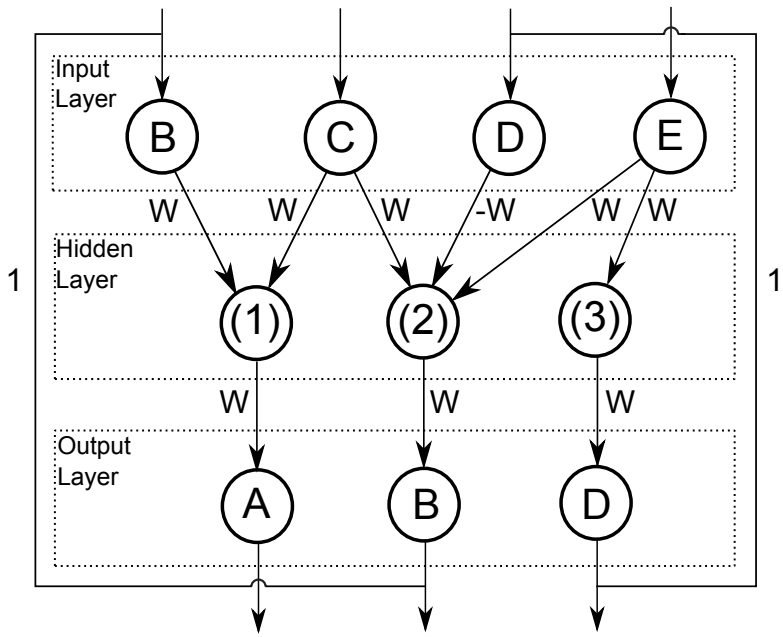
The CILP Translation Algorithm (revisited)

(1) (2) (3)

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



N:



Bottom Clause Propositionalization (BCP)

- BCP uses Progol's bottom clause generation to create features for each ILP example
- Each positive example is saturated normally and labeled as positive (1); negative examples are processed just like positive ones, but are labeled as negative (-1)
- 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 created
- Finally, a binary input vector v of size $|F|$ is created for each example (set of features) $(\forall i, v(i) \in \{0, 1\})$

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:
  $\perp_+ = [motherInLaw(A, B) \leftarrow mother(A, C), wife(C, B)]$
  $\perp_- = [motherInLaw(A, B) \leftarrow wife(A, C)]$
  - From $\perp_+$, $hash_+$:

Key	Value
mom1	A
husband1	B
daughter1	C

  - From $\perp_-$, $hash_-$:

Key	Value
daughter1	A
husband2	B
husband1	C

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

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 label and network output, respectively
4. Both standard stopping criteria and early stopping have been evaluated

Stopping criteria

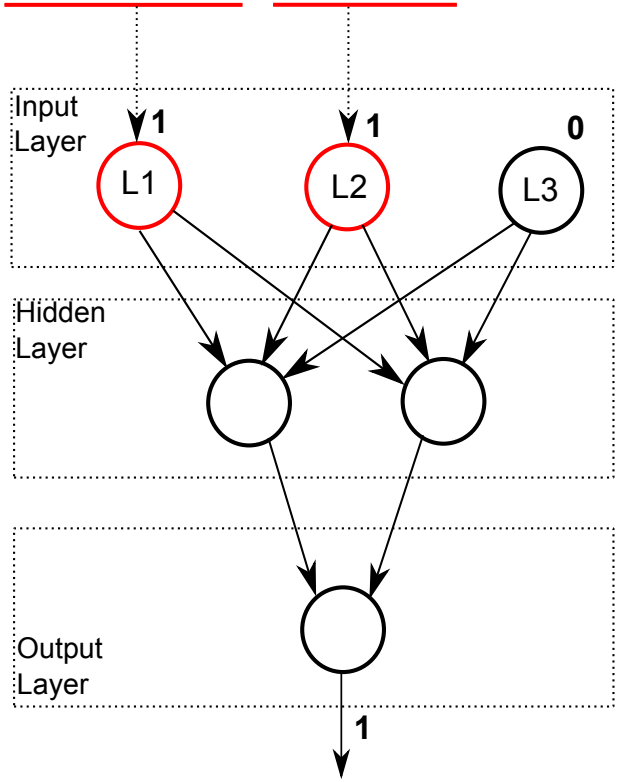
- Standard stopping criteria:
  - 300 training epochs have elapsed
  - 95% of the training data satisfies $E < 0.1$
  - 90% of the training data is correctly classified by the network and no improvements can be seen for 5 training epochs
- Early stopping:
  - A validation set with $\frac{1}{5}$ of the total set of examples 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

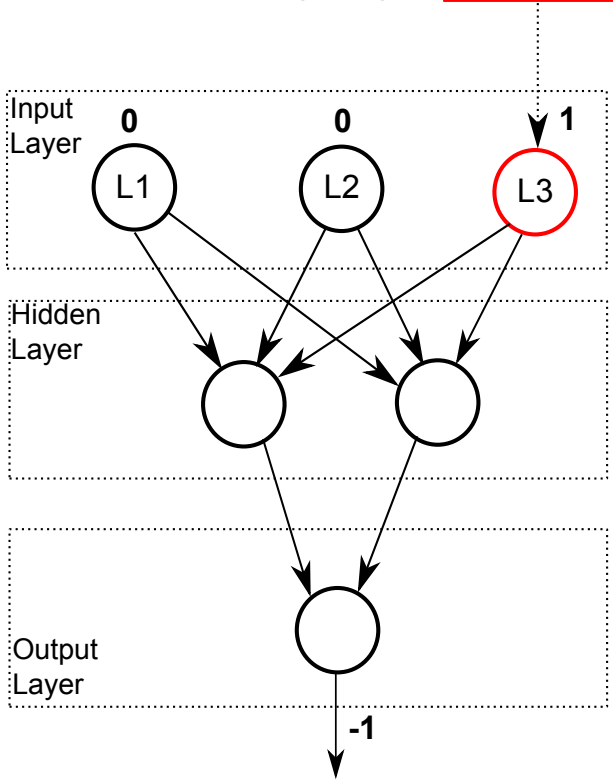
Building the network and training

1) motherInLaw(A,B) :- mother(A,C), wife(C,B)

L1: mother(A,C)
L2: wife(C,B)
L3: wife(A,C)



2) ~motherInLaw(A,B) :- wife(A,C)



Description of the Experiments

- 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 in the *2h* configurations?
  - It can help avoid overfitting in large networks
  - Accuracy doesn't seem to increase substantially with 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∗(±10.5) **0:08:15**	**91.70**(±5.84) 0:10:34	90.65(±8.97) 0:11:15	89.20(±8.92) 0:10:16
krk	**99.6**(±0.51) 0:11:03	98.31∗(±1.23) 0:04:38	98.32∗(±1.25) **0:04:34**	98.42(±1.26) 0:04:40
uw-cse	**84.91**(±7.32) 0:45:47	66.24∗(±7.01) **0:08:47**	66.08∗(±2.48) 0:10:19	70.01∗(±2.2) 0:08:54
alz-amine	78.71(±5.25) 1:31:05	**78.99**(±4.46) 1:23:42	76.02∗(±3.79) 2:07:04	77.08(±5.17) **1:14:21**
alz-acetyl	**69.46**(±3.6) 8:06:06	63.64∗(±4.01) 4:20:28	63.49∗(±4.16) 5:49:51	63.30∗(±5.09) **2:47:52**
alz-memory	**68.57**(±5.7) 3:47:55	60.44∗(±4.11) 1:41:36	59.19∗(±5.91) 2:12:14	59.82∗(±6.76) **1:19:27**
alz-toxic	80.5(±3.98) 6:02:05	79.92(±3.09) 3:04:53	80.49(±3.65) 3:33:17	**81.73**(±4.68) **2:12:17**

bold = best result; * indicates statistically significant difference

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

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

Comparison with another Propositionalization

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 BCP features (and neurons) are first-order literals, it should be possible to extend the CILP knowledge extraction algorithm directly to obtain first-order rules from trained CILP++ networks
- *...but such rules will not obey any ILP restrictions as imposed by a language bias*
- França et. al., ICCSW'13, AAAI Spring Symposium 2015, contain a first proposal towards an efficient relational knowledge extraction algorithm using the propositional rule learner RIPPER (other rule learner could be used)

From CILP++ networks to FOL rules

- Search for literals whose variables do not obey the ILP-style variable chaining
- Using BCP's hash tables, replace such variables by a disjunction of ground terms
- This transformation is provably correct: BCP feature equivalence theorem

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 is:

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

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 literals that are always true, and literals containing no variables
- If applied on $R_{\perp}^{C}$:

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

Experimental settings

- We have evaluated the 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(±5.25) 1:31:05, 36.1	69.46(±4.6) 8:06:06, 47.3	68.57(±5.7) 3:47:55, 45.7	80.5(±4.83) 6:02:05, 37.9
BCP+RIP$_{prop}$	73.35(±4.32) 0:19:49, 30	67.8(±3.77) 0:23:21, 20.1	65.27(±7.11) 0:25:11, 14.4	78.44(±5.44) 0:17:41, 35.2
BCP+RIP$_{FOL}$	77.73(±4.57) 0:21:59, 30.4	63.56(±5.06) 0:26:39, 18.7	57.64(±5.7) 0:28:45, 13.8	66.45(±6.93) 0:20:57, 18

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

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

In Summary

- CILP++: Competitive accuracy results can be obtained efficiently using a neural-symbolic approach; in principle, any ILP dataset can be used with CILP++
- BCP can be used with any propositional learner (although some don't seem to like the idea of bottom clauses very much)
- Relational Knowledge Extraction: capable of generating compact first-order rule sets with negation from trained neural nets
- Considerably faster than Aleph, but sometimes with (sometimes considerable) accuracy loss

Next Steps

- Evaluation of noise robustness and evaluation in theory revision tasks (with negation)
- Apply to first-order knowledge extraction from neural networks in general (compare with TREPAN and its variations)
- Compare accuracy results with other fast and hybrid ILP approaches, including differentiable ILP
- A lot of the magic seems to happen in the choice (and number of copies) of first-order literals to place in the input and output layers: use macro-operators!?