Machine Learning Coalgebraic Proofs

Ekaterina Komendantskaya

School of Computing, University of Dundee

ILP'11,
1 August 2011

General motivation

We know that...

General motivation

We know that...

- Formal [manual or computerised] proofs follow the logic rules [rules of inference]. They have precise, rather than statistical, nature.

General motivation

We know that...

- Formal [manual or computerised] proofs follow the logic rules [rules of inference]. They have precise, rather than statistical, nature.
- ... but that concerns proofs as final products.

General motivation

We know that...

- Formal [manual or computerised] proofs follow the logic rules [rules of inference]. They have precise, rather than statistical, nature.
- ... but that concerns proofs as final products.

However, Process of proving [Proof search]

- ...does involve statistical learning
- ... which needs to be studied.

Technical Motivation

- ITPs are proof assistants based on higher-order logics/type theory.
- They are extensively used for purposes of verification in CS and mathematics.
- Not all proofs can be done automatically.
- In bigger industrial proofs there might be thousands of lemmas and theorems needing proofs, with about 5-20% needing programmer's intervention.
- Can statistical machine-learning methods help us to analyse why these fail (The models of "Why", Cliff Jones)?
- My experience with Coq was that the experts often justify the chosen combinations of tactics statistically rather than conceptually.

Overall feeling

- These are hard questions to solve.
- If solved, may change the way we look at proofs (automated and manual)...
- May enrich the methods of machine learning and proof theory.

Lessons learnt:

I have tried to implement several logic algorithms in Neural nets

- Semantic operators for first-order logic programs and many-valued logic programs;
- First-order Unification algorithm;
- First-order term-rewriting;
- Inductive definitions akin inductive dependent types.

Overall conclusion

It is inefficient to apply statistical methods to logic algorithms "as they are" — because statistical methods cannot compete with logical methods on the same grounds. Where can they compete?

Why is machine learning **UN**-suitable for Formal methods:

- Many logic algorithms have a precise, rather than statistical nature.

Example

Two formulae `list(x)` and `list(nil)` are unifiable: `x/nil`. We mean exactly this, and do not want it to be substituted by some approximate such as `nol`. (Although humans would tolerate this mis-spelling had it appeared in a written text...)

Why is machine learning **UN**-suitable for Formal methods:

- Many logic algorithms have a precise, rather than statistical nature.

Example

Two formulae `list(x)` and `list(nil)` are unifiable: `x/nil`. We mean exactly this, and do not want it to be substituted by some approximate such as `nol`. (Although humans would tolerate this mis-spelling had it appeared in a written text...)

- Many important logic algorithms are sequential, e.g. unification.

Example

If I have a goal: `list(cons(x,y))` $\wedge$ `list(x)`, my proof will never succeed — `x` will get substituted by some `nat` term, e.g. `0` or `S(0)`, which will make the second formula invalid. Note that the proof would have succeed had it been concurrent.

Machine-learn patterns of sequential proofs

Example

1. $\mathtt{nat(0)} \leftarrow$
2. $\mathtt{nat(s(x))} \leftarrow \mathtt{nat(x)}$
3. $\mathtt{list(nil)} \leftarrow$
4. $\mathtt{list(cons\ x\ y)} \leftarrow \mathtt{nat(x),\ list(y)}$

Given correct and incorrect sequences of the numbers 1, 2, 3, 4, 5, 6 how likely is it that we can train a neural network to recognise correct and incorrect proofs?

Example

Sequential algorithms of Unification and SLD-resolution drive the derivations:

Well-formed

4, 1, 4, 1, 5.

$$\begin{array}{c}
\mathtt{list}(\mathtt{cons}(\mathrm{x}, \mathtt{cons}(\mathrm{y}, \mathrm{x}))) \\
| \\
\mathtt{nat}(\mathrm{x}), \mathtt{list}(\mathtt{cons}(\mathrm{y}, \mathrm{x})) \\
| \\
\mathtt{nat}(\mathrm{y}), \mathtt{list}(0) \\
| \\
\mathtt{list}(0)
\end{array}$$

Example

Sequential algorithms of Unification and SLD-resolution drive the derivations:

Well-formed

4, 1, 4, 1, 5.

Ill-formed

4, 1, 4, 1, 6.

$$\begin{array}{c}
\mathtt{list}(\mathtt{cons}(\mathtt{x}, \mathtt{cons}(\mathtt{y}, \mathtt{x}))) \\
| \\
\mathtt{nat}(\mathtt{x}), \mathtt{list}(\mathtt{cons}(\mathtt{y}, \mathtt{x})) \\
| \\
\mathtt{nat}(\mathtt{y}), \mathtt{list}(0) \\
| \\
\mathtt{list}(0)
\end{array}$$

$$\begin{array}{c}
\mathtt{list}(\mathtt{cons}(\mathtt{x}, \mathtt{cons}(\mathtt{y}, \mathtt{x}))) \\
| \\
\mathtt{nat}(\mathtt{x}), \mathtt{list}(\mathtt{cons}(\mathtt{y}, \mathtt{x})) \\
| \\
\mathtt{nat}(\mathtt{y}), \mathtt{list}(0) \\
| \\
\mathtt{list}(0) \\
| \\
\square
\end{array}$$

Example 2

Well-formed

4, 1, 4, 1, 5.

$$\mathtt{list}(\mathtt{cons}(\mathtt{x}, \mathtt{cons}(\mathtt{y}, \mathtt{x})))$$
$$|$$
$$\mathtt{nat}(\mathtt{x}), \mathtt{list}(\mathtt{cons}(\mathtt{y}, \mathtt{x}))$$
$$|$$
$$\mathtt{nat}(\mathtt{y}), \mathtt{list}(\mathtt{0})$$
$$|$$
$$\mathtt{list}(\mathtt{0})$$

Example 2

Well-formed

4, 1, 4, 1, 5.

Ill-formed

4, 1, 4, 1, 5.

$$\begin{array}{c} \mathtt{list}(\mathtt{cons}(x, \mathtt{cons}(y, x))) \\ | \\ \mathtt{nat}(x), \mathtt{list}(\mathtt{cons}(y, x)) \\ | \\ \mathtt{nat}(y), \mathtt{list}(0) \\ | \\ \mathtt{list}(0) \end{array}$$

$$\begin{array}{c} \mathtt{list}(\mathtt{cons}(x, \mathtt{cons}(y, z))) \\ | \\ \mathtt{nat}(x), \mathtt{list}(\mathtt{cons}(y, x)) \\ | \\ \mathtt{nat}(y), \mathtt{list}(z) \\ | \\ \mathtt{list}(z) \end{array}$$

Conclusion

If the size of the training data set (= set of the examples used for training) is big and representative, derivations with "tactics" 4, 1, 4, 1, 5 are equally likely to be correct and incorrect.

What are the coinductive trees?

Lesson learnt

- The sequence of deductive rules alone does not help; and actually hides the structure of the proof.
- We need more “structural” representation of proofs.

Coinductive trees:

- They arose from coalgebraic semantics for derivations in logic programs, [Komendantskaya,Power CALCO'11, CSL'11].
- They also allow for concurrency.
- They offer very structured approach to automated proofs.

Example

For examples and explanations, please come along to the poster session.

Example: Sequential derivation versus coinductive:

Sequential SLD-tree

$$\mathtt{list}(\mathtt{cons}(x, \mathtt{cons}(y, z)))$$

$$\mathtt{nat}(x), \mathtt{list}(\mathtt{cons}(y, z))$$

$$\mathtt{nat}(y), \mathtt{list}(z)$$

$$\mathtt{list}(z)$$

$$\square$$

Coinductive tree

$$\mathtt{list}(\mathtt{cons}(x, \mathtt{cons}(y, z)))$$

$$\mathtt{nat}(x) \qquad \mathtt{list}(\mathtt{cons}(y, z))$$

$$\mathtt{nat}(y) \qquad \mathtt{list}(z)$$

Example: Sequential derivation versus coinductive:

Sequential SLD-tree

$\mathtt{list}(\mathtt{cons}(x, \mathtt{cons}(y, z)))$

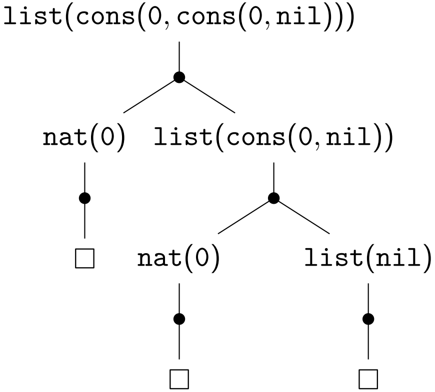
$\mathtt{nat}(x), \mathtt{list}(\mathtt{cons}(y, z))$

$\mathtt{nat}(y), \mathtt{list}(z)$

$\mathtt{list}(z)$

$\square$

Coinductive tree



More generally, Coalgebraic methods in proofs:

- - work with concurrent models of computations; e.g. Milner's CCS and π-calculus;
- - work with possibly infinite processes/proofs/objects/... — and therefore based on the idea of repeating patterns (rather than final answers or terminating computations). E.g., the notion of productiveness.
- These have good chances of yielding statistical analysis.

Problem definition and Results:

We use statistical pattern-recognition methods, e.g. Neural nets with backpropagation learning (gradient descent) to learn properties of proofs given by coinductive trees.

Based on problems arising in ITPs, there are three classes of problems we wish to machine-learn:

- Is a proof well-formed?
- Does a well-formed proof belong to a given family of proofs?
- Does a well-formed proof belong to the success family of proofs?

Problem definition and Results:

We use statistical pattern-recognition methods, e.g. Neural nets with backpropagation learning (gradient descent) to learn properties of proofs given by coinductive trees.

Based on problems arising in ITPs, there are three classes of problems we wish to machine-learn:

- Is a proof well-formed? Accuracy up to 73%
- Does a well-formed proof belong to a given family of proofs?
- Does a well-formed proof belong to the success family of proofs?

Problem definition and Results:

We use statistical pattern-recognition methods, e.g. Neural nets with backpropagation learning (gradient descent) to learn properties of proofs given by coinductive trees.

Based on problems arising in ITPs, there are three classes of problems we wish to machine-learn:

- Is a proof well-formed? Accuracy up to 73%
- Does a well-formed proof belong to a given family of proofs? Accuracy up to 98-100%
- Does a well-formed proof belong to the success family of proofs?

Problem definition and Results:

We use statistical pattern-recognition methods, e.g. Neural nets with backpropagation learning (gradient descent) to learn properties of proofs given by coinductive trees.

Based on problems arising in ITPs, there are three classes of problems we wish to machine-learn:

- Is a proof well-formed? Accuracy up to 73%
- Does a well-formed proof belong to a given family of proofs? Accuracy up to 98-100%
- Does a well-formed proof belong to the success family of proofs? Accuracy up to 95%

Problem definition and Results:

We use statistical pattern-recognition methods, e.g. Neural nets with backpropagation learning (gradient descent) to learn properties of proofs given by coinductive trees.

Based on problems arising in ITPs, there are three classes of problems we wish to machine-learn:

- Is a proof well-formed? Accuracy up to 73%
- Does a well-formed proof belong to a given family of proofs? Accuracy up to 98-100%
- Does a well-formed proof belong to the success family of proofs? Accuracy up to 95%

Surprisingly successful.

Conclusions:

- We have tried to machine learn concurrent (coalgebraic) algorithms — various properties of coinductive proof trees.
- Coalgebraic derivations look more promising than traditional sequential derivations from the point of view of statistical ML (offer both concurrency and structural approach).
- We have learned *both* from positive and negative examples. (models of "Why?")
- Big ambition: move on to tactics in ITPs.

Questions?

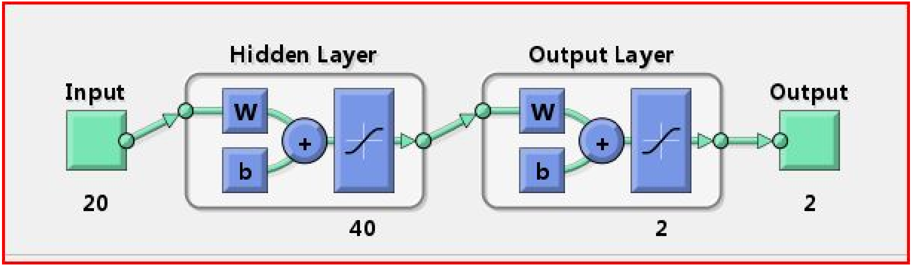
(Please join me for poster session
or contact me katya@computing.dundee.ac.uk if they arise later!)

Representation in vectors

	list	nat	$\bullet$	$\square$
$\mathtt{cons}(\mathrm{x}, \mathtt{cons}(\mathrm{y}, \mathrm{x}))$	- $\lvert\mathtt{cons}(\mathrm{x}, \mathtt{cons}(\mathrm{y}, \mathrm{x}))\rvert$	0	2	0
$\mathtt{cons}(\mathrm{y}, \mathrm{x}))$	- $\lvert\mathtt{cons}(\mathrm{y}, \mathrm{x}))\rvert$	0	2	0
x	-1	-1	1	0
y	-1	0	1	0
z	0	0	0	0

And then it is further flattened into a vector.

Features of Coinductive trees represented as vectors form an input to the neural network:



It's a two-layer feed-forward network, with sigmoid hidden and output neurons, that can classify vectors arbitrarily well, given enough neurons in its hidden layer.

The network was trained with scaled conjugate gradient back-propagation.