universität innsbruck

Computational Logic

Artificial Intelligence in Theorem Proving

Cezary Kaliszyk

VTSA 2019

Computer Theorem Proving

- Computer used to automate reasoning in a logic
- Traditionally part of artificial intelligence
  - (not machine learning)
- Field of research since the fifties
- Applications: program verification, mathematical deduction, ...
- Theorem proving logics, precision, automation, ... very varied.

Computer Theorem Proving: Historical Context

- 1940s: Algorithmic proof search (λ-calculus)
- 1960s: de Bruijn's Automath
- 1970s: Small Certifiers (LCF)
- 1990s: Resolution (Superposition)
- 2000s: Large proofs and theories
- 2010s: Machine Learning for Reasoning?

Covered Topics

Part I

Theorem proving systems

Machine learning problems

Lemma relevance

Deep learning for theorem proving

Part II

Guided Automated Reasoning

Lemma mining

Unsupervised methods

Longer proofs

What is a Proof Assistant? (1/2)

A Proof Assistant is a

- a computer program
- to assist a mathematician
- in the production of a proof
- that is mechanically checked

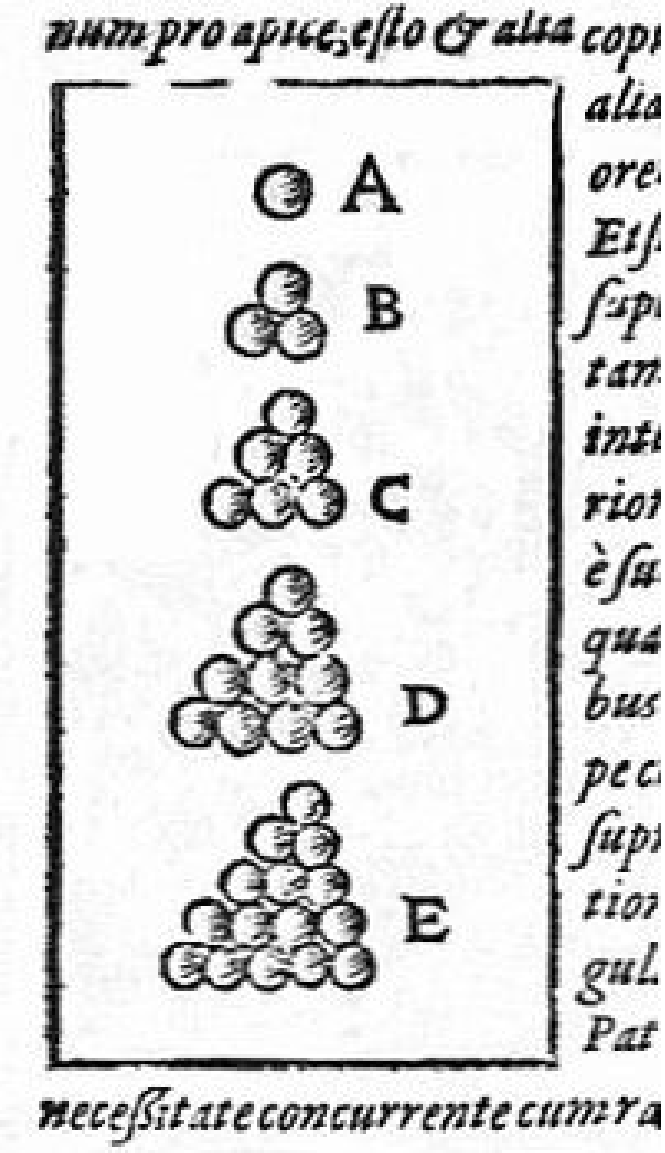
What does a Proof Assistant do?

- Keep track of theories, definitions, assumptions
- Interaction - proof editing
- Proof checking
- Automation - proof search

What does it implement? (And how?)

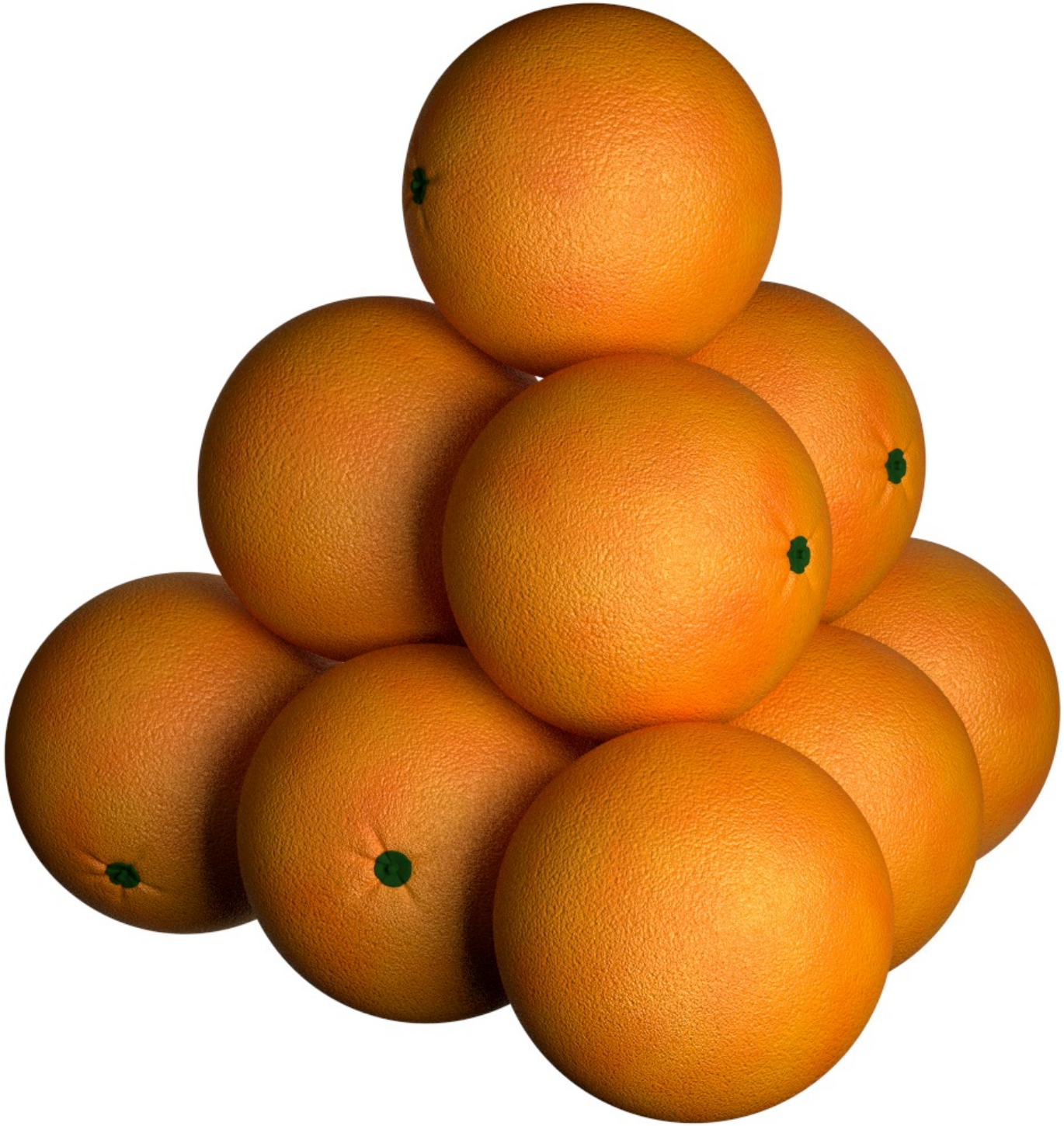
- a formal logical system intended as foundation for mathematics
- decision procedures

The Kepler Conjecture (year 1611)



The most compact way of stacking balls of the same size in space is a pyramid.

$$V = \frac{\pi}{\sqrt{18}} \approx 74\%$$



The Kepler Conjecture (year 1611)

Proved in 1998

- Tom Hales, 300 page proof using computer programs
- Submitted to the Annals of Mathematics

The Kepler Conjecture (year 1611)

Proved in 1998

- Tom Hales, 300 page proof using computer programs
- Submitted to the Annals of Mathematics
- 99% correct. . . but we cannot verify the programs

1039 equalities and inequalities

For example:

$$\frac{\begin{array}{c}-x_1x_3-x_2x_4+x_1x_5+x_3x_6-x_5x_6+\\+x_2(-x_2+x_1+x_3-x_4+x_5+x_6)\end{array}}{\sqrt{4x_2\begin{pmatrix}x_2x_4(-x_2+x_1+x_3-x_4+x_5+x_6)+\\+x_1x_5(x_2-x_1+x_3+x_4-x_5+x_6)+\\+x_3x_6(x_2+x_1-x_3+x_4+x_5-x_6)-\\-x_1x_3x_4-x_2x_3x_5-x_2x_1x_6-x_4x_5x_6\end{pmatrix}}} < \tan(\frac{\pi}{2}-0.74)$$

The Kepler Conjecture (year 1611)

Solution? Formalized Proof!

- Formalize the proof using Proof Assistants
- Implement the computer code in the system
- Prove the code correct
- Run the programs inside the Proof Assistant

Flyspeck Project

- Project results published 2017
- Many Proof Assistants and contributors

Intel Pentium® P5 (1994)

Superscalar; Dual integer pipeline; Faster floating-point, ...

Intel Pentium® P5 (1994)

Superscalar; Dual integer pipeline; Faster floating-point, ...

$$\frac{4159835}{3145727} = 1.333820...$$

$$\frac{4159835}{3145727} \overset{\text{P5}}{=} 1.333739...$$

Intel Pentium® P5 (1994)

Superscalar; Dual integer pipeline; Faster floating-point, ...

$$\frac{4159835}{3145727} = 1.333820... \qquad \frac{4159835}{3145727} \overset{\text{P5}}{=} 1.333739...$$

FPU division lookup table: for certain inputs division result off

Replacement

- Few customers cared, still cost of $475 million
- Testing and model checking insufficient:
  - Since then Intel and AMD processors formally verified (*)
  - HOL Light and ACL2 (along other techniques)

```
theorem sqrt2_not_rational:
  "sqrt (real 2) ∉ ℚ"
proof




qed
```

```
theorem sqrt2_not_rational:
  "sqrt (real 2) ∉ ℚ"
proof
  assume "sqrt (real 2) ∈ ℚ"
```

```
  thus False
qed
```

```
theorem sqrt2_not_rational:
  "sqrt (real 2) ∉ ℚ"
proof
  assume "sqrt (real 2) ∈ ℚ"
  then obtain m n :: nat where
    n_nonzero: "n ≠ 0" and sqrt_rat: "¦sqrt (real 2)¦ = real m / real
    and lowest terms: "gcd m n = 1" ..

  thus False
qed
```

```
theorem sqrt2_not_rational:
  "sqrt (real 2) ∉ ℚ"
proof
  assume "sqrt (real 2) ∈ ℚ"
  then obtain m n :: nat where
    n_nonzero: "n ≠ 0" and sqrt_rat: "¦sqrt (real 2)¦ = real m / real
    and lowest terms: "gcd m n = 1" ..

          have eq: "m² = 2 * n²"
  hence "2 dvd m²" ..
                  have dvd m: "2 dvd m"

  hence "2 dvd n²" ..
                  have "2 dvd n"
            have "2 dvd gcd m n"

  thus False
qed
```

```
theorem sqrt2_not_rational:
  "sqrt (real 2) ∉ ℚ"
proof
  assume "sqrt (real 2) ∈ ℚ"
  then obtain m n :: nat where
    n_nonzero: "n ≠ 0" and sqrt_rat: "¦sqrt (real 2)¦ = real m / real
    and lowest_terms: "gcd m n = 1" ..
  from n_nonzero and sqrt_rat have "real m = ¦sqrt (real 2)¦ * real n
  then have "real (m²) = (sqrt (real 2))² * real (n²)"
    by (auto simp add: power2_eq_square)
  also have "(sqrt (real 2))² = real 2" by simp
  also have "... * real (m²) = real (2 * n²)" by simp
  finally have eq: "m² = 2 * n²" ..
  hence "2 dvd m²" ..
  with two_is_prime have dvd_m: "2 dvd m" by (rule prime_dvd_power_tw
  then obtain k where "m = 2 * k" ..
  with eq have "2 * n² = 2² * k²" by (auto simp add: power2_eq_square
  hence "n² = 2 * k²" by simp
  hence "2 dvd n²" ..
  with two_is_prime have "2 dvd n" by (rule prime_dvd_power_two)
  with dvd_m have "2 dvd gcd m n" by (rule gcd_greatest)
  with lowest_terms have "2 dvd 1" by simp
  thus False by arith
qed
```

Proof Assistant (2/2)

- Keep track of theories, definitions, assumptions
  - set up a theory that describes mathematical concepts (or models a computer system)
  - express logical properties of the objects
- Interaction - proof editing
  - typically interactive
  - specified theory and proofs can be edited
  - provides information about required proof obligations
  - allows further refinement of the proof
  - often manually providing a direction in which to proceed.
- Automation - proof search
  - various strategies
  - decision procedures
- Proof checking
  - checking of complete proofs
  - sometimes providing certificates of correctness
- Why should we trust it?
  - small core

Can a Proof Assistant do all proofs?

Decidability!

- Validity of formulas is undecidable
- (for non-trivial logical systems)

Automated Theorem Provers

- Specific domains
- Adjust your problem
- Answers: Valid (Theorem with proof)
- Or: Countersatisfiable (Possibly with counter-model)

Proof Assistants

- Generally applicable
- Direct modelling of problems
- Interactive

What are the other classes of tools?

(Many already covered in the courses in past few days)

ATPs (tomorrow)

- Built in automation (model elimination, resolution)
- Vampire, Eprover, SPASS, ...
- Applications: Robbin's conjecture, Programs, and AIM

Users of Proof Assistants

Computer Science

- Modelling and specifying systems
- Proving properties of systems
- Proving software correct

Mathematics

- Defining concepts and theories
- Proving (mostly verifying) proofs
- (currently less common)

Theorems and programs that use ITP

Theorems

- Kepler Conjecture
- 4 color theorem
- Feit-Thomson theorem (2012)

Software

- Processors and Chips
- Security Protocols
- Project Cristal (Comp-Cert)
- L4-Verified
- Java Bytecode

Coverage of Basic Mathematics

Freek Wiedijk's list of 100 theorems

HOL Light	86
Isabelle	81
MetaMath	71
Coq	69
Mizar	69
any	94

Coverage by other tools

- much less as single steps
- (actually hard to compare)

[*Wiedijk'15*]

Proof Assistant Summary

Complicated Proofs (Math, Computer Science)

Human proofs

Proof Assistant Summary

Complicated Proofs (Math, Computer Science)

Proof Assistant

- a computer program to assist a mathematician
- in the production of a proof
- that is mechanically checked

Human proofs

Proof Assistant Summary

Complicated Proofs (Math, Computer Science)

Proof Assistant

- a computer program to assist a mathematician
  - keep track of theories, definitions, assumptions, check individual steps, provide decision procedures
- in the production of a proof
- that is mechanically checked

Human proofs

Proof Assistant Summary

Complicated Proofs (Math, Computer Science)

Proof Assistant

- a computer program to assist a mathematician
  - keep track of theories, definitions, assumptions, check individual steps, provide decision procedures
- in the production of a proof
- that is mechanically checked
  - formal logical system

Human proofs

Proof Assistant Summary

Complicated Proofs (Math, Computer Science)

Proof Assistant

- a computer program to assist a mathematician
  - keep track of theories, definitions, assumptions, check individual steps, provide decision procedures
- in the production of a proof
- that is mechanically checked
  - formal logical system

Human proofs

- Proof skeletons

Filling in the gaps: most of the work

- Small intermediate steps
- Sometimes also hard ones

Proof Assistant Summary

Complicated Proofs (Math, Computer Science)

Proof Assistant

- a computer program to assist a mathematician
  - keep track of theories, definitions, assumptions, check individual steps, provide decision procedures
- in the production of a proof
- that is mechanically checked
  - formal logical system

Human proofs

- Proof skeletons

Filling in the gaps: most of the work

- Small intermediate steps
  - General Purpose Automation!
- Sometimes also hard ones
  - Selected domains

Fast progress in machine learning

What is Machine Learning?

- Tuning a big number of parameters

Algorithms that improve their performance based on data

- Face detection
- Recommender systems
- Speech recognition
- Stock prediction
- Spam detection
- Molecule modeling
- Automated translation
- ...

Tasks related to proofs and reasoning

Tasks involving logical inference

- Natural language question answering [*Sukhbaatar+2015*]
- Knowledge base completion [*Socher+2013*]
- Automated translation [*Wu+2016*]

Games

AlphaGo (Zero) problems similar to proving [*Silver+2016*]

- Node evaluation
- Policy decisions

AI theorem proving techniques

High-level AI guidance

- premise selection: select the right lemmas to prove a new fact
- based on suitable features (characterizations) of the formulas
- and on learning lemma-relevance from many related proofs
- tactic selection

Mid-level AI guidance

- learn good ATP strategies/tactics/heuristics for classes of problems
- learning lemma and concept re-use
- learn conjecturing

Low-level AI guidance

- guide (almost) every inference step by previous knowledge
- good proof-state characterization and fast relevance

Problems for Machine Learning

- Is my conjecture true?

Problems for Machine Learning

- Is my conjecture true?
- Is a statement is useful?

$$a^n + b^n = c^n$$

Problems for Machine Learning

- Is my conjecture true?
- Is a statement is useful?
  - For a conjecture

$$a^n + b^n = c^n$$

Problems for Machine Learning

- Is my conjecture true? $a^n + b^n = c^n$
- Is a statement is useful?
  - For a conjecture
- What are the dependencies of statement? (premise selection)

Problems for Machine Learning

- Is my conjecture true? $a^n + b^n = c^n$
- Is a statement is useful?
  - For a conjecture
- What are the dependencies of statement? (premise selection)
- Should a theorem be named? How?

Problems for Machine Learning

- Is my conjecture true? $a^n + b^n = c^n$
- Is a statement is useful?
  - For a conjecture
- What are the dependencies of statement? (premise selection)
- Should a theorem be named? How?
- What should the next proof step be?
  - Tactic? Instantiation?

Problems for Machine Learning

- Is my conjecture true? $a^n + b^n = c^n$
- Is a statement is useful?
  - For a conjecture
- What are the dependencies of statement? (premise selection)
- Should a theorem be named? How?
- What should the next proof step be?
  - Tactic? Instantiation?
- What new problem is likely to be true?
  - Intermediate statement for a conjecture

Premise selection

Intuition

Given:

- set of theorems T (together with proofs)
- conjecture c

Find: minimal subset of T that can be used to prove c

More formally

$$\arg\min_{t \subseteq T} \{|t| \mid t \vdash c\}$$

(or $\emptyset$ if not provable)

Note: implicit assumption on a proving system. ATP in practice.

In machine learning terminology

Multi-label classification

Input: set of samples $\mathbb{S}$, where samples are triples $s, F(s), L(s)$

- s is the sample ID
- $F(s)$ is the set of features of s
- $L(s)$ is the set of labels of s

Output: function f : features $\rightarrow$ labels

Predicts n labels (sorted by relevance) for set of features

Sample features

Sample add_comm ($a + b = b + a$) characterized by:

- F(add_comm) = {"+", "=", "num"}
- L(add_comm) = {num_induct, add_0, add_suc, add_def}

Not exactly the usual machine learning problem

Labels correspond to premises and samples to theorems

- Very often same

Not exactly the usual machine learning problem

Labels correspond to premises and samples to theorems

- Very often same

Similar theorems are likely to be useful in the proof

- Also likely to have similar premises

Not exactly the usual machine learning problem

Labels correspond to premises and samples to theorems

- Very often same

Similar theorems are likely to be useful in the proof

- Also likely to have similar premises

Theorems sharing logical features are similar

- Theorems sharing rare features are very similar

Not exactly the usual machine learning problem

Labels correspond to premises and samples to theorems

- Very often same

Similar theorems are likely to be useful in the proof

- Also likely to have similar premises

Theorems sharing logical features are similar

- Theorems sharing rare features are very similar

Temporal order

- Recently considered theorems and premises are important
- Also in evaluation

Not exactly for the usual machine learning tools

Needs efficient learning and prediction

- Frequent major data updates
- Automation cannot wait more than 10 seconds, often less

Multi-label classifier output

- Often asked for 1000 or more most relevant lemmas

Easy to get many interesting features

- Complicated feature relations
- PCA / LSA / ...?

Premise Selection

- Syntactic methods
  - Neighbours using various metrics
  - Recursive — SInE, MePo
- Naive Bayes, k-Nearest Neighbours
- Linear / Logistic Regression
  - Needs feature and theorem space reduction
  - Kernel-based multi-output ranking
- Decision Trees (Random Forests)
- Neural Networks
  - Winnow, Perceptron — SNoW, MaLARea
  - DeepMath

Machine Learning Algorithms

- **k-Nearest Neighbours:**
  - finds a fixed number (k) of proved facts nearest to the conjecture c
  - weight the dependencies each such fact f by the distance between f and c
  - relevance is the sum of weights across the k nearest neighbors
- **Naive Bayes:**
  - probability of f being needed to prove c
  - based on the previous use of f in proving conjectures similar to c
  - assumes independence of features to use the Bayes theorem
- **MePo:** (*Me*ng–*P*aulso*n*)
  - score of a fact is $r/(r+i)$, where r is the number of relevant features and i the number of irrelevant features
  - iteratively select all top-scoring facts and add their features to the set of relevant features.
- **Combination**

k-NN (1/2)

Definition: Distance of two facts (**similarity**)

$$s(a,b) = \sum_{f \in F(a) \cap F(b)} 1$$

k-NN (1/2)

Definition: Distance of two facts (**similarity**)

$$s(a,b) = \sum_{f \in F(a) \cap F(b)} w(f)$$

k-NN (1/2)

Definition: Distance of two facts (**similarity**)

$$s(a,b) = \sum_{f \in F(a) \cap F(b)} w(f)^{\tau_1}$$

k-NN (1/2)

Definition: Distance of two facts (**similarity**)

$$s(a,b) = \sum_{f \in F(a) \cap F(b)} w(f)^{\tau_1}$$

Relevance of fact a for goal g

$$\left(\sum_{b \in N | a \in D(b)} \frac{s(b,g)}{|D(b)|} \right)$$

k-NN (1/2)

Definition: Distance of two facts (**similarity**)

$$s(a,b) = \sum_{f \in F(a) \cap F(b)} w(f)^{\tau_1}$$

Relevance of fact a for goal g

$$\left(\sum_{b \in N | a \in D(b)} \frac{s(b,g)}{|D(b)|} \right) + \begin{cases} s(a,g) & \text{if } a \in N \\ 0 & \text{otherwise} \end{cases}$$

k-NN (1/2)

Definition: Distance of two facts (**similarity**)

$$s(a,b) = \sum_{f \in F(a) \cap F(b)} w(f)^{\tau_1}$$

Relevance of fact a for goal g

$$\left(\tau_2 \sum_{b \in N | a \in D(b)} \frac{s(b,g)}{|D(b)|}\right) + \begin{cases} s(a,g) & \text{if } a \in N \\ 0 & \text{otherwise} \end{cases}$$

k-NN (2/2)

```
let knn_eval csyms (sym_ths, sym_wght) deps maxth no_adv =
  let neighbours = Array.init maxth (fun j -> (j, 0.)) in
  let ans = Array.copy neighbours in

  (* for each symbol, increase the importance of the theorems
     which contain the symbol by a given symbol weight *)
  List.iter (fun sym ->
      let ths = sym_ths sym and weight = sym_wght sym in
      List.iter (fun th ->
          if th < maxth then map_snd neighbours th ((+.) (weight ** 6.0))) ths) csyms;

  Array.fast_sort sortfun neighbours;

  let no_recommends = ref 0 in
  let add_ans k i o =
    if snd (ans.(i)) <= 0. then begin
      incr no_recommends;
      map_snd ans i (fun _ -> float_of_int (age k) +. o))
    end else map_snd ans i ((+.) o) in

  (* Additionally stop when given no_recommends reached *)
  Array.iteri (fun k (nn, o) ->
    add_ans k nn o;
    let ds = deps nn in
    let ol = 2.7 *. o /. (float_of_int (List.length ds)) in
    List.iter (fun d -> if d < maxth then add_ans k d ol) ds;
  ) neighbours;

  Array.fast_sort sortfun ans;
```

Naive Bayes

$$
\begin{aligned}
& P(f \text{ is relevant for proving } g) \\
= \quad & P(f \text{ is relevant} \mid g\text{'s features}) \\
= \quad & P(f \text{ is relevant} \mid f_1, \ldots, f_n) \\
\propto \quad & P(f \text{ is relevant}) \Pi_{i=1}^{n} P(f_i \mid f \text{ is relevant}) \\
\propto \quad & \#f \text{ is a proof dependency} \cdot \Pi_{i=1}^{n} \frac{\#f_i \text{ appears when } f \text{ is a proof dependency}}{\#f \text{ is a proof dependency}}
\end{aligned}
$$

Naive Bayes: adaptation to premise selection

extended features $\overline{F}(a)$ of a fact a

features of a and of the facts that were proved using a

More precise estimation of the relevance of ϕ to prove γ:

$$P(a \text{ is used in } \psi\text{'s proof})$$
$$\cdot \prod_{f \in F(\gamma) \cap \overline{F}(a)} P\big(\psi \text{ has feature } f \mid a \text{ is used in } \psi\text{'s proof}\big)$$
$$\cdot \prod_{f \in F(\gamma) - \overline{F}(a)} P\big(\psi \text{ has feature } f \mid a \text{ is not used in } \psi\text{'s proof}\big)$$
$$\cdot \prod_{f \in \overline{F}(a) - F(\gamma)} P\big(\psi \text{ does not have feature } f \mid a \text{ is used in } \psi\text{'s proof}\big)$$

Naive Bayes: adaptation to premise selection

extended features $\overline{F}(a)$ of a fact a

features of a and of the facts that were proved using a (only one iteration)

More precise estimation of the relevance of ϕ to prove γ:

$$P(a \text{ is used in } \psi\text{'s proof})$$
$$\cdot \prod_{f \in F(\gamma) \cap \overline{F}(a)} P\big(\psi \text{ has feature } f \mid a \text{ is used in } \psi\text{'s proof}\big)$$
$$\cdot \prod_{f \in F(\gamma) - \overline{F}(a)} P\big(\psi \text{ has feature } f \mid a \text{ is not used in } \psi\text{'s proof}\big)$$
$$\cdot \prod_{f \in \overline{F}(a) - F(\gamma)} P\big(\psi \text{ does not have feature } f \mid a \text{ is used in } \psi\text{'s proof}\big)$$

All these probabilities can be computed efficiently

Update two functions (tables):

- $t(a)$: number of times a fact a was dependency
- $s(a, f)$:
  number of times a fact a was dependency of a fact described by feature f

Then:

$$P(a \text{ is used in a proof of (any) } \psi) = \frac{t(a)}{K}$$

$$P\big(\psi \text{ has feature } f \mid a \text{ is used in } \psi\text{'s proof}\big) = \frac{s(a,f)}{t(a)}$$

$$P\big(\psi \text{ does not have feature } f \mid a \text{ is used in } \psi\text{'s proof}\big) = 1 - \frac{s(a,f)}{t(a)}$$

$$\approx 1 - \frac{s(a,f) - 1}{t(a)}$$

Naive Bayes “in practice”

```
double NaiveBayes::score(sample_t i, set<feature_t> symh) const {
  // number of times current theorem was used as dependency
  const long n      = tfreq[i];
  const auto sfreqh = sfreq[i];

  double s = 30 * log(n);

  for (const auto sv : sfreqh) {
    // sv.first ranges over all features of theorems depending on i
    // sv.second is the number of times sv.first appears among theorems
    // depending on i
    double sfreqv = sv.second;

    // if sv.first exists in query features
    if (symh.erase(sv.first) == 1)
      s += tfidf.get(sv.first) * log (5 * sfreqv / n);
    else
      s += tfidf.get(sv.first) * 0.2 * log (1 + (1 - sfreqv) / n);
  }

  // for all query features that did not appear in features of dependencies
  // of current theorem
  for (const auto f : symh) s -= tfidf.get(f) * 18;

  return s;
```

SInE

[*Hoder'09*]

Basic algorithm

If symbol s is d-relevant and appears in axiom a,
then a and all symbols in a become $d + 1$-relevant.

Problem: Common Symbols

- Simple *relevance* usually selects all axioms
- Because of common symbols, such as `subclass` or `subsumes`

```
subclass (beverage, liquid).
subclass (chair, furniture).
```

Solution: Trigger based selection

"appears" is changed to "triggers"

But how to know if s is common?

Approximate by number of occurrences in the current problem

SInE: Tolerance

- Only symbols with t-times more occurrences than the least common symbol trigger an axiom
- For $t = \infty$ this is the same as relevance

1: subclass(X,Y) ∧ subclass(Y,Z) → subclass(X,Z)
subclass(petrol,liquid)
¬subclass(stone,liquid)
2: subclass(beverage,liquid)
1: subclass(beer,beverage)
subclass(guinness,beer)
subclass(pilsner,beer)

? subclass(beer,liquid)

Occ.	Symbols
7	subclass
3	liquid, beer
2	beverage
1	petrol, stone, guinness, pilsner

[*Hoder*]

SInE in E

Implementation: GSInE in `e_axfilter`

- Parameterizable filters
  - Different generality measures (frequency count, generosity, benevolence)
  - Different limits (absolute/relative size, # of iterations)
  - Different seeds (conjecture/hypotheses)
- Efficient implementation
  - E data types and libraries
  - Indexing (symbol $\rightarrow$ formula, formula $\rightarrow$ symbol)
- Multi-filter support
  - Parse & index once (amortize costs)
  - Apply different independent filters

Primary use

- Initial over-approximation
  (efficiently reduce HUGE input files to manageable size)
- Secondary use: Filtering for individual E strategies

Regression in Theorem Proving

Premises: Classification

- Dimensions in the input
- Matrix QR decomposition

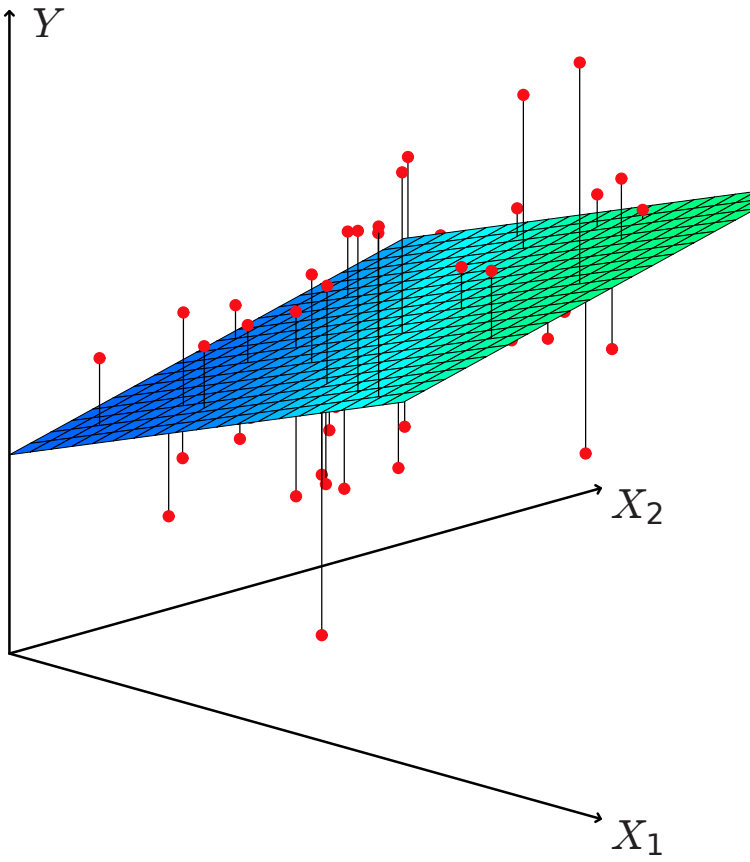
Probabilities: Logistic

Non-linearity

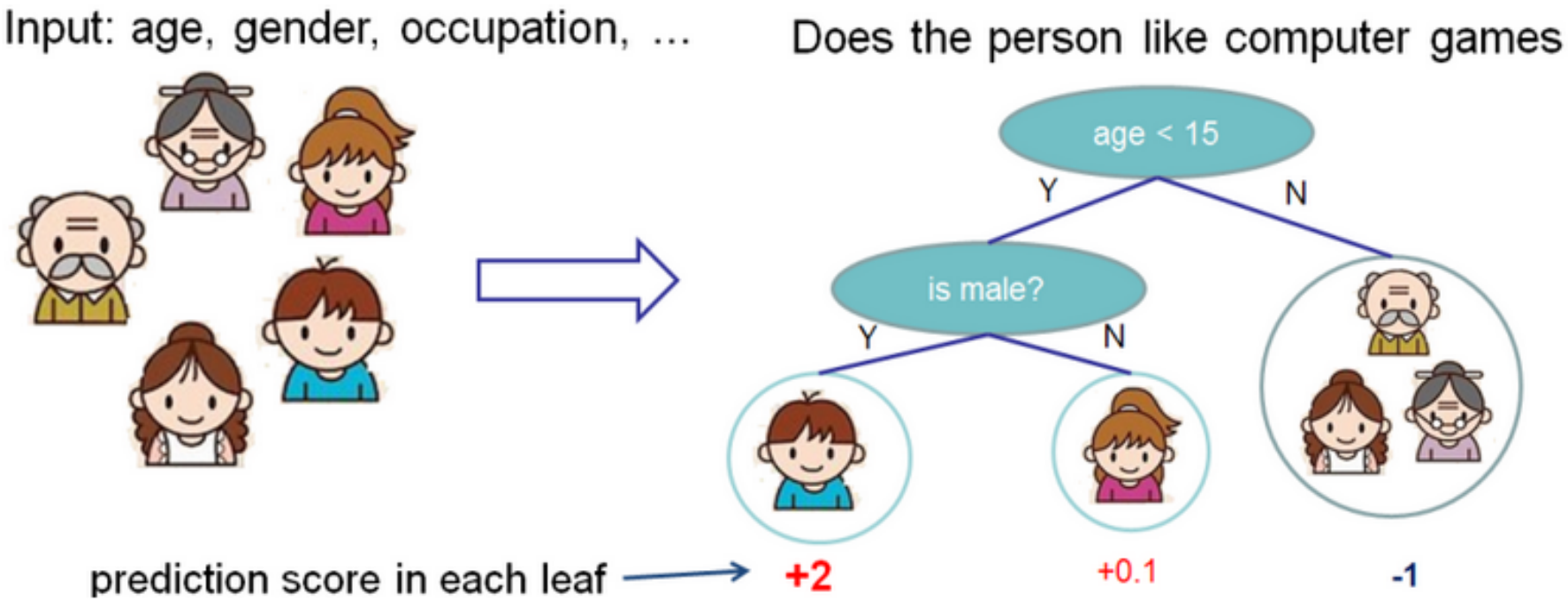
- Kernels [*Enigma*]
- Multi-output Ranking [*Kühlwein'14, ...*]

State space reduction

- Random projections [*VowpalWabbit*]
- Decomposition

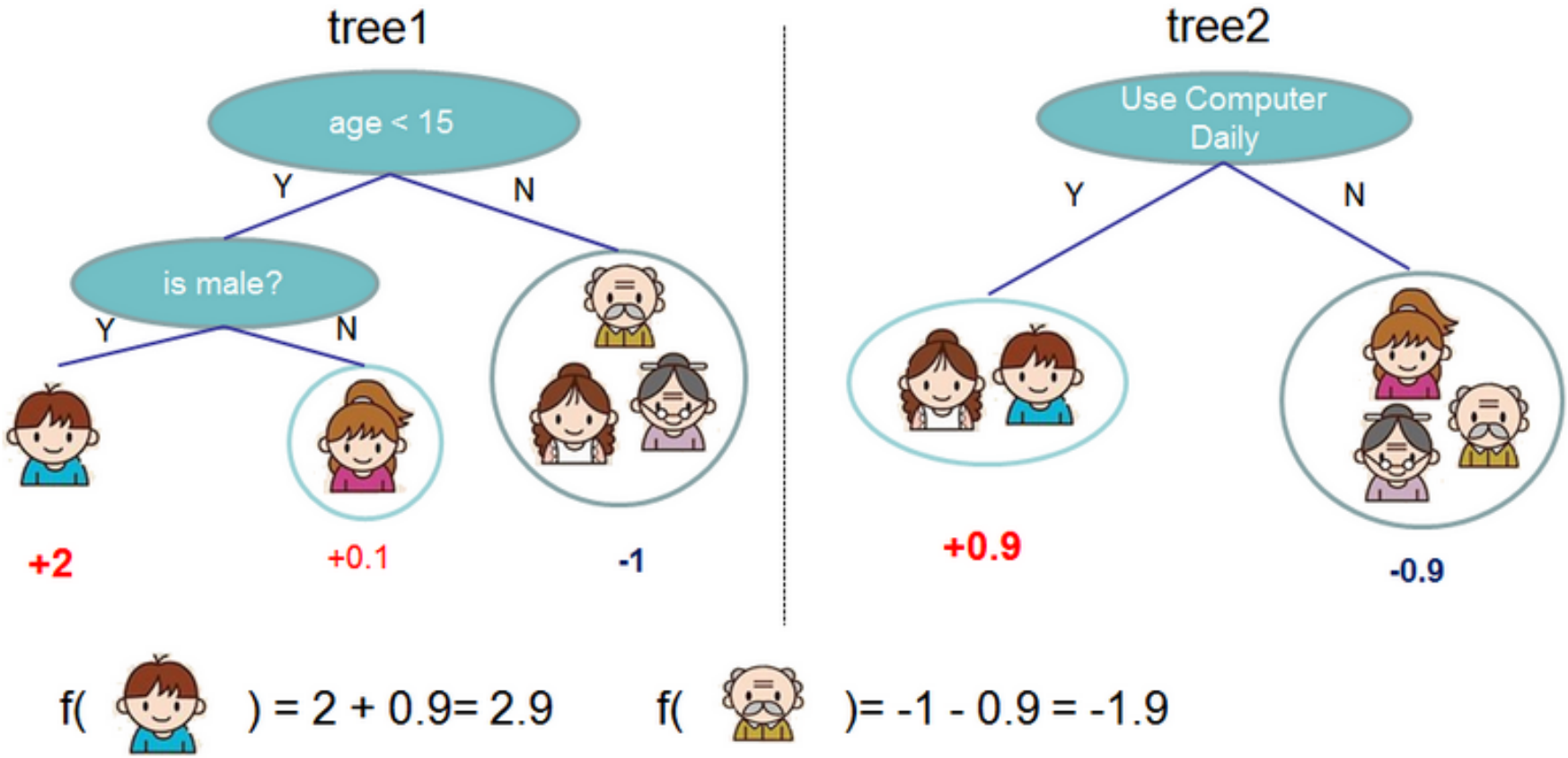


Decision Trees (1/2)



[*Chen, Guestrin*]

Decision Trees (2/2)



.

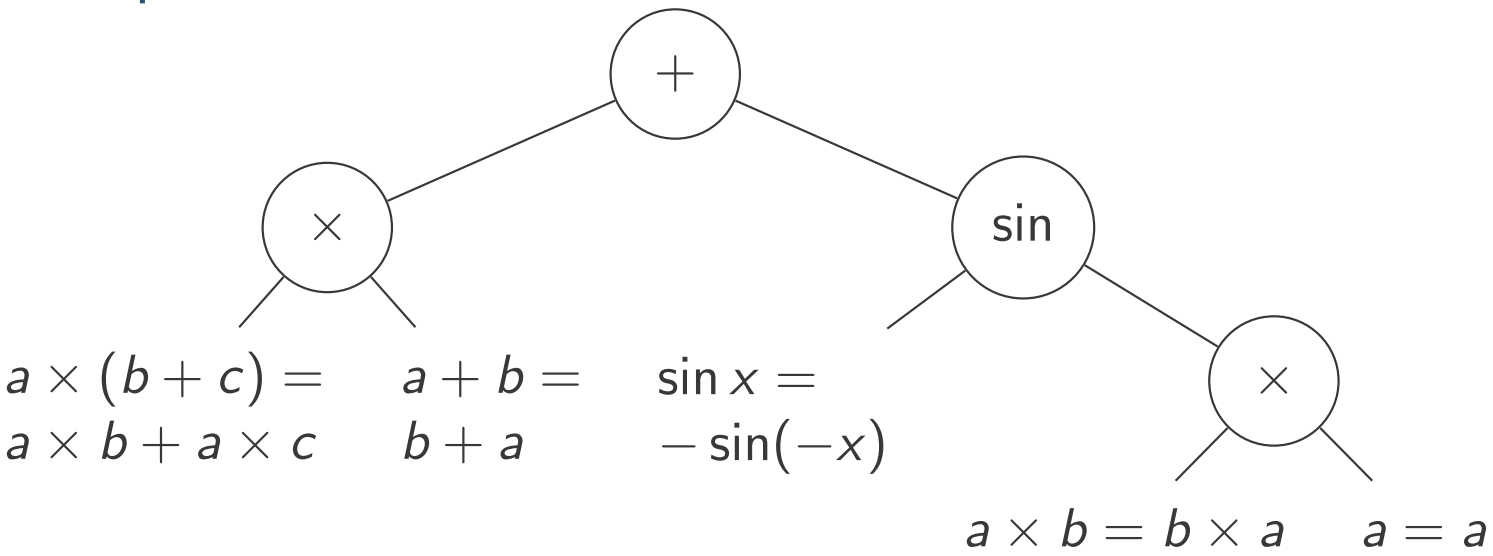
[*Chen, Guestrin*]

Decision Trees

Definition

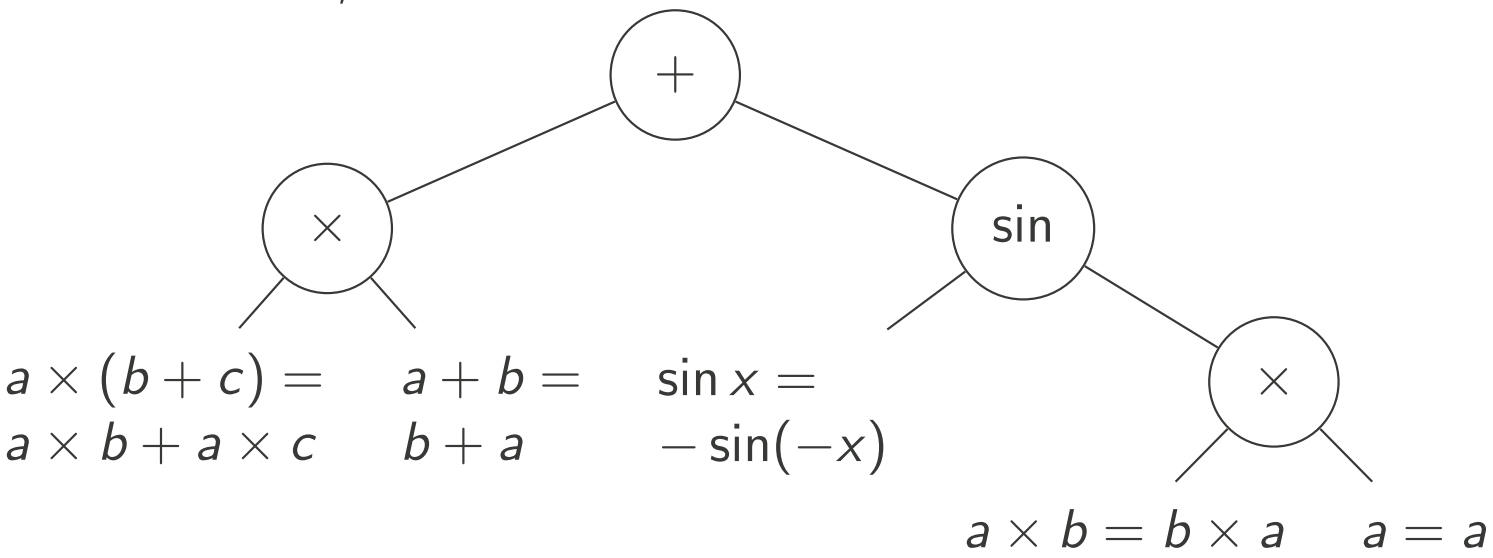
- each leaf stores a set of samples
- each branch stores a feature f and two subtrees, where:
  - the left subtree contains only samples having f
  - the right subtree contains only samples not having f

Example



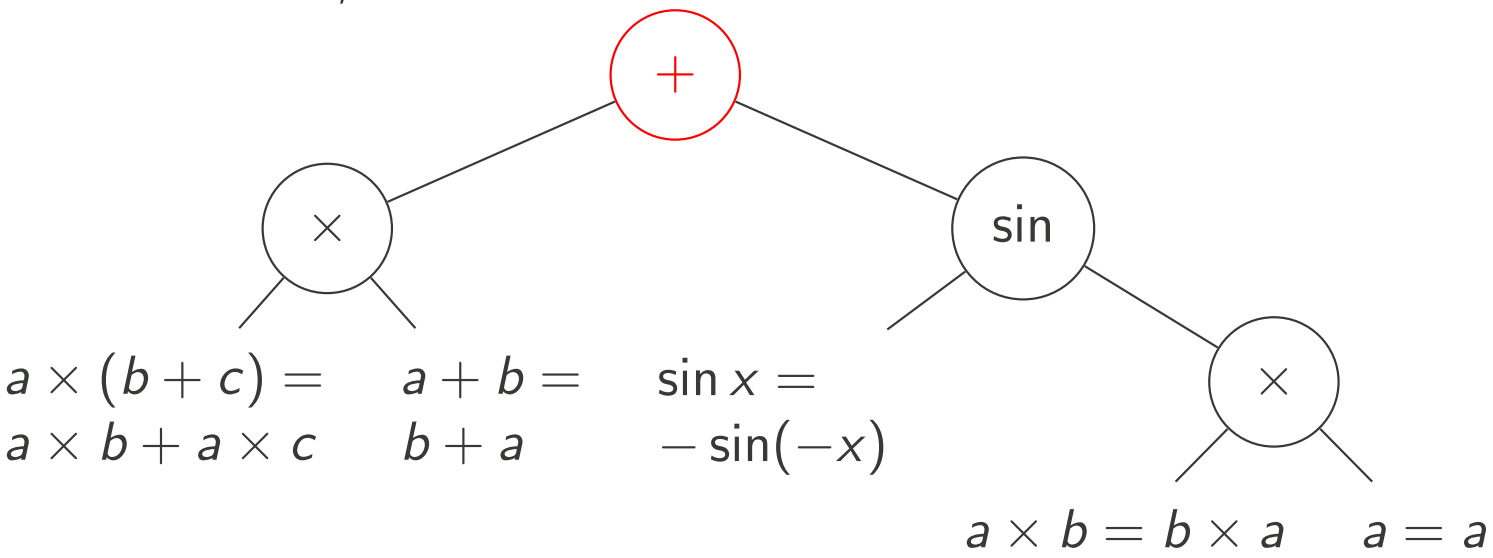
Single-path query

Query tree for conjecture "$\sin(0) = 0$".
Features: "sin", "0".



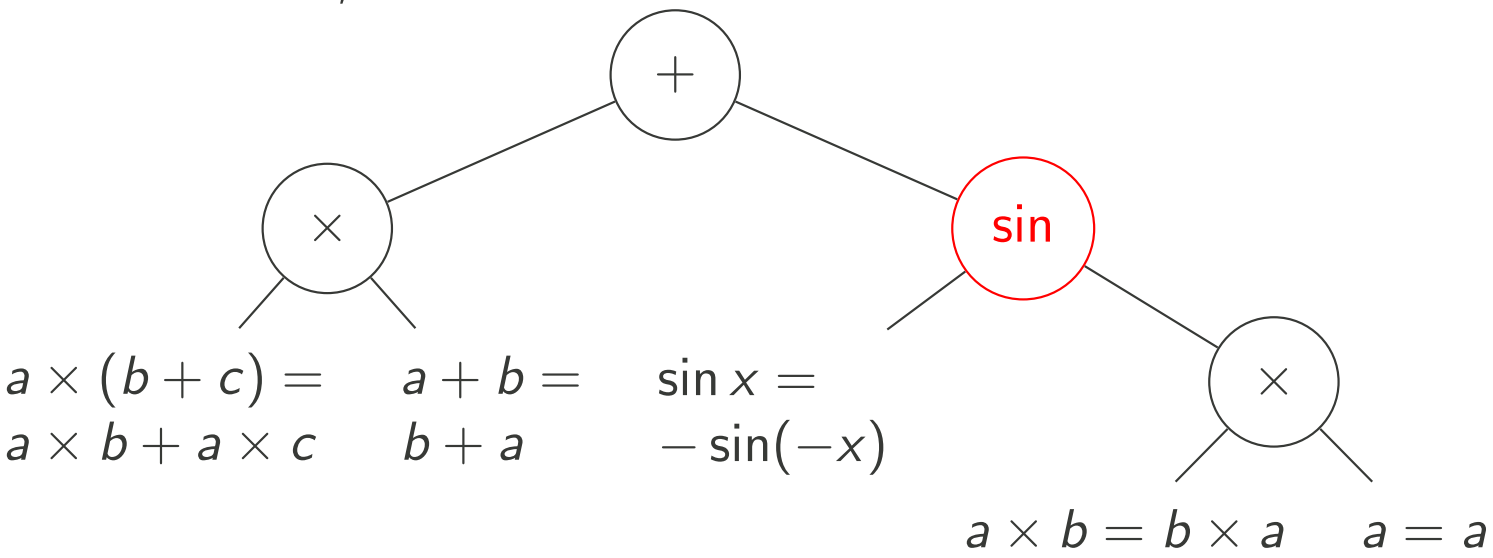
Single-path query

Query tree for conjecture "$\sin(0) = 0$".
Features: "sin", "0".



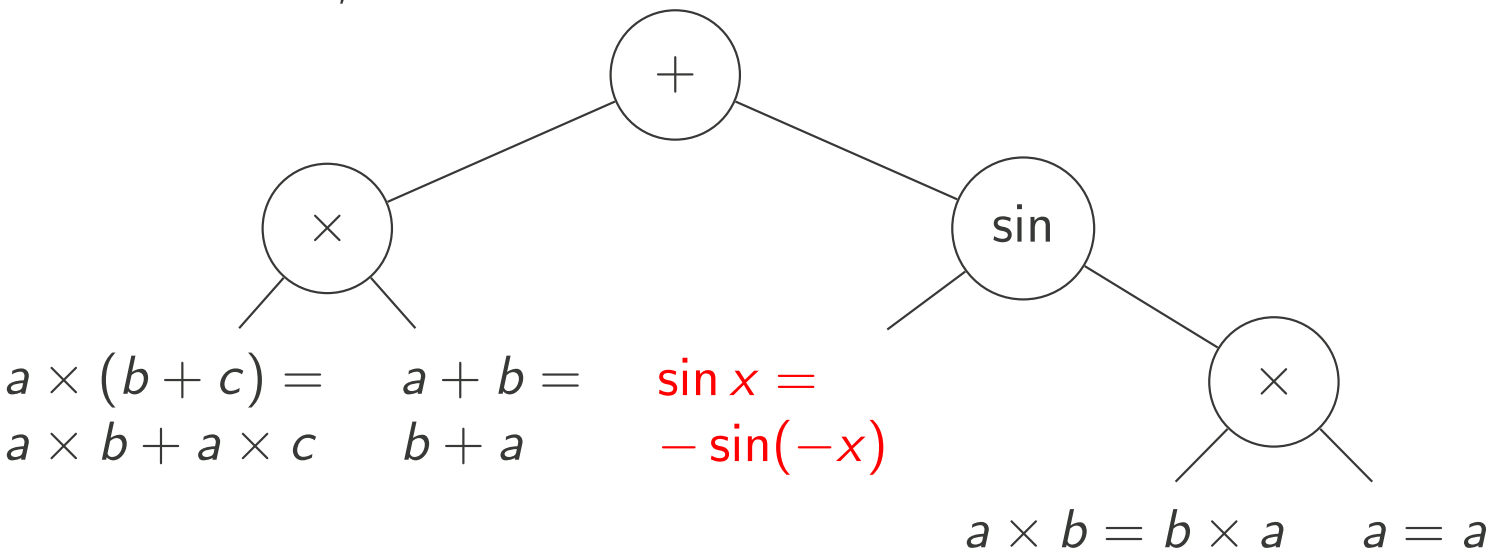
Single-path query

Query tree for conjecture "$\sin(0) = 0$".
Features: "sin", "0".



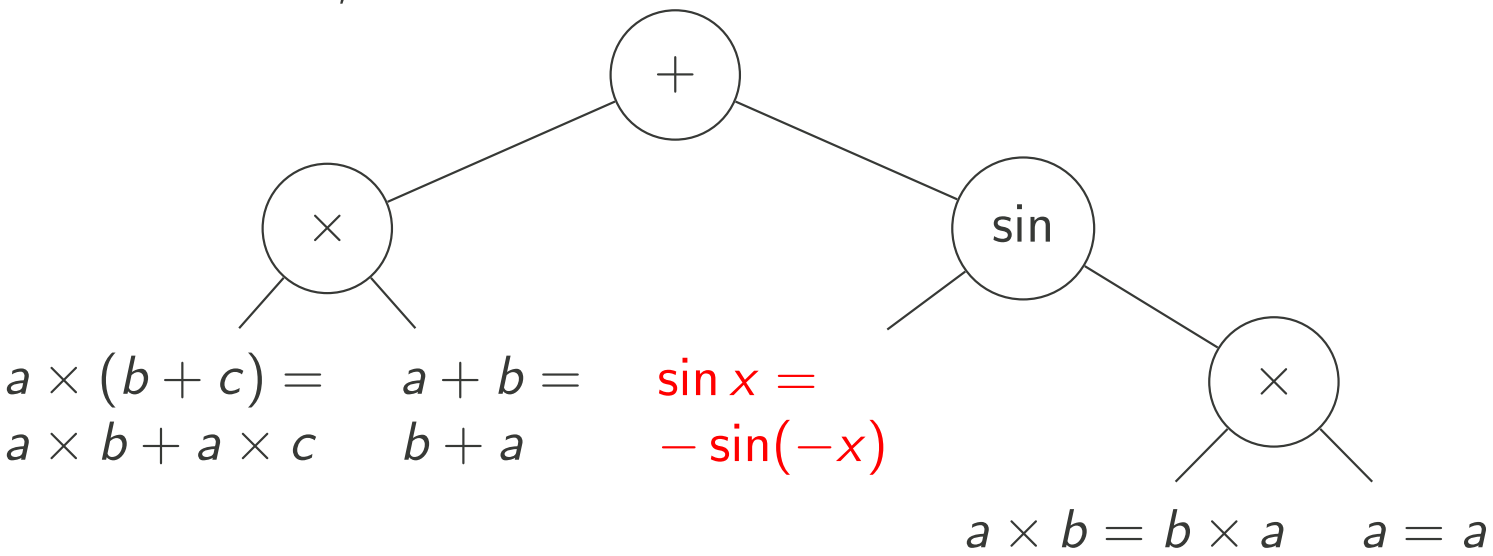
Single-path query

Query tree for conjecture "$\sin(0) = 0$".
Features: "sin", "0".



Single-path query

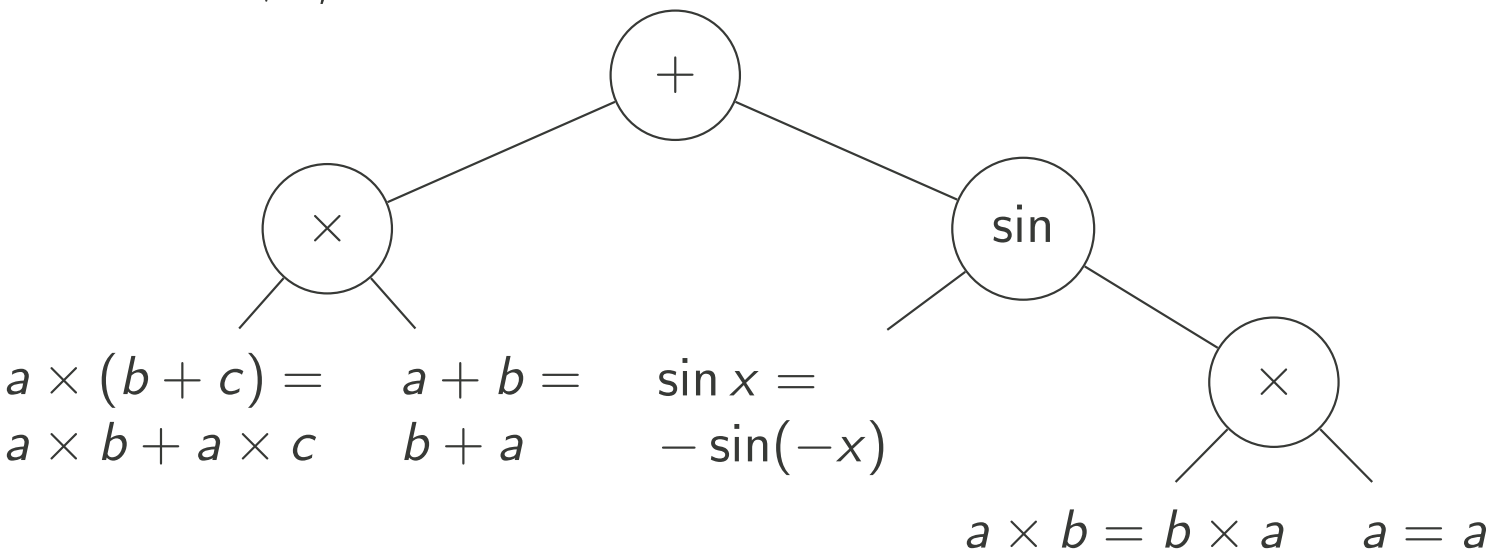
Query tree for conjecture "$\sin(0) = 0$".
Features: "sin", "0".



The overall result will be the premises of $\sin x = -\sin(-x)$.

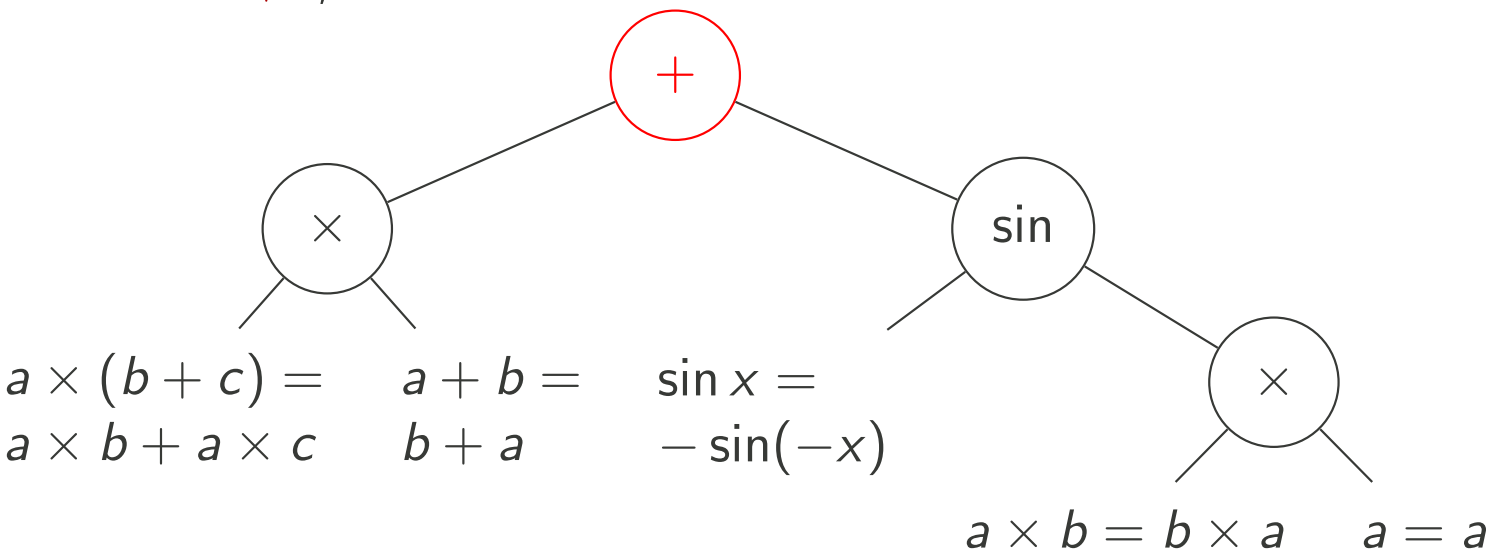
Single-path query (2)

Query tree for conjecture "$(a + b) \times c = a \times c + b \times c$".
Features: "$+$", "$\times$".



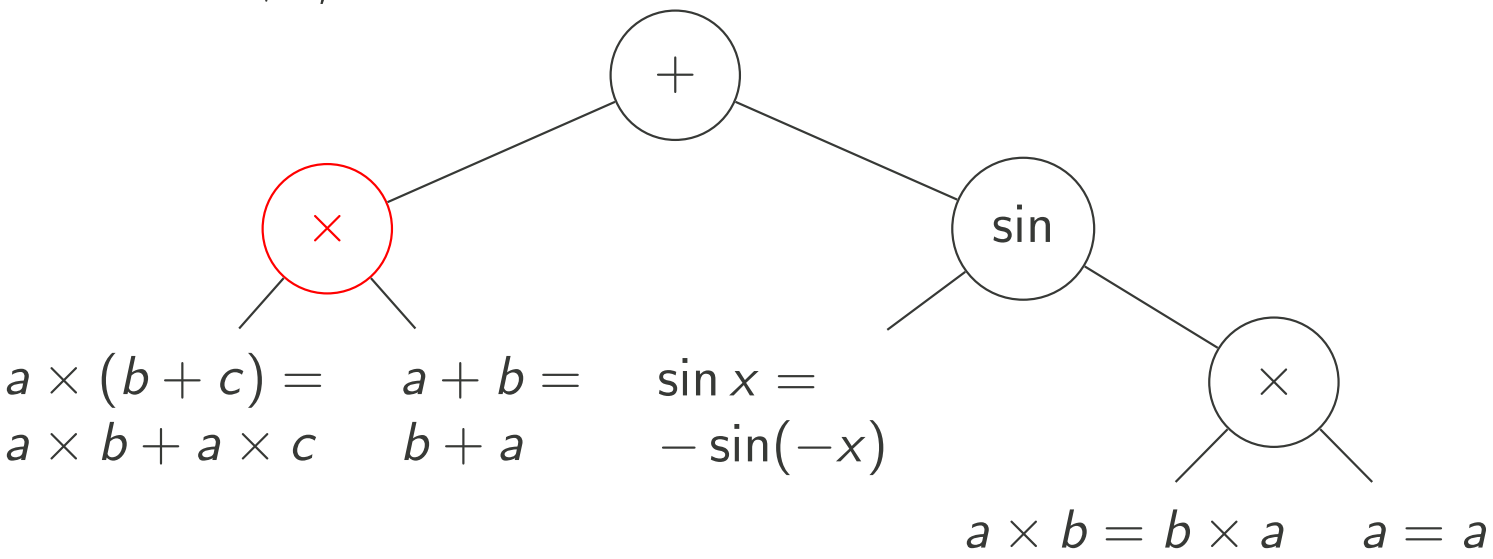
Single-path query (2)

Query tree for conjecture "$(a + b) \times c = a \times c + b \times c$".
Features: "$+$", "$\times$".



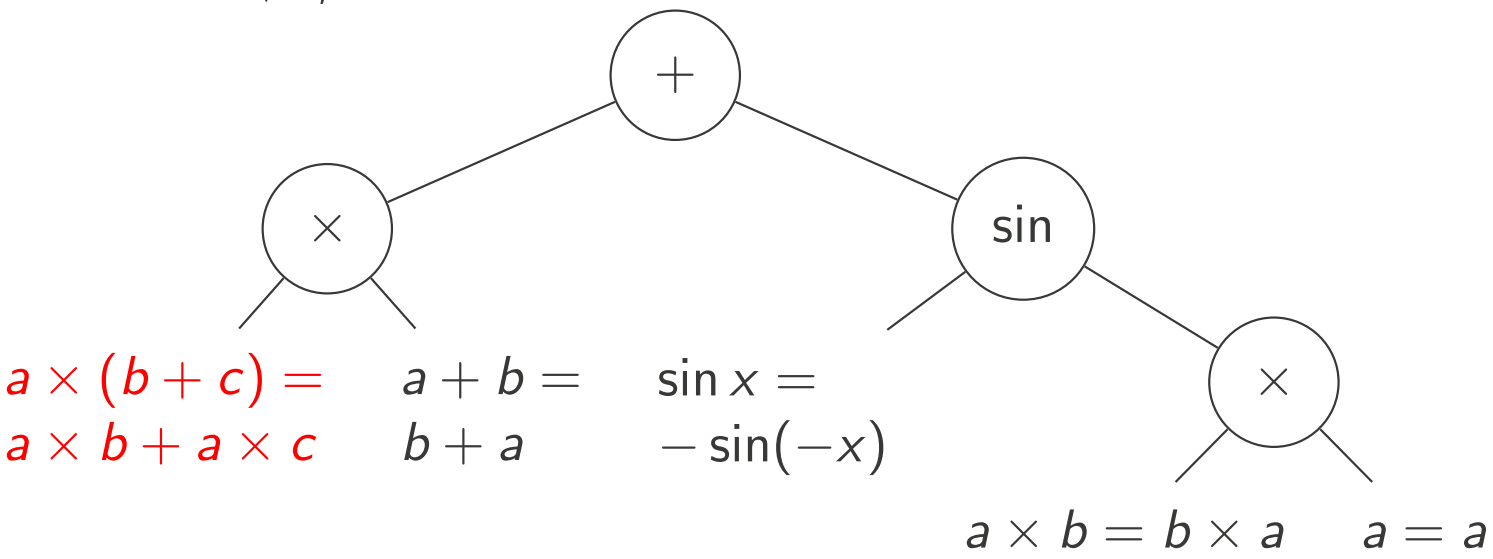
Single-path query (2)

Query tree for conjecture "$(a + b) \times c = a \times c + b \times c$".
Features: "$+$", "$\times$".



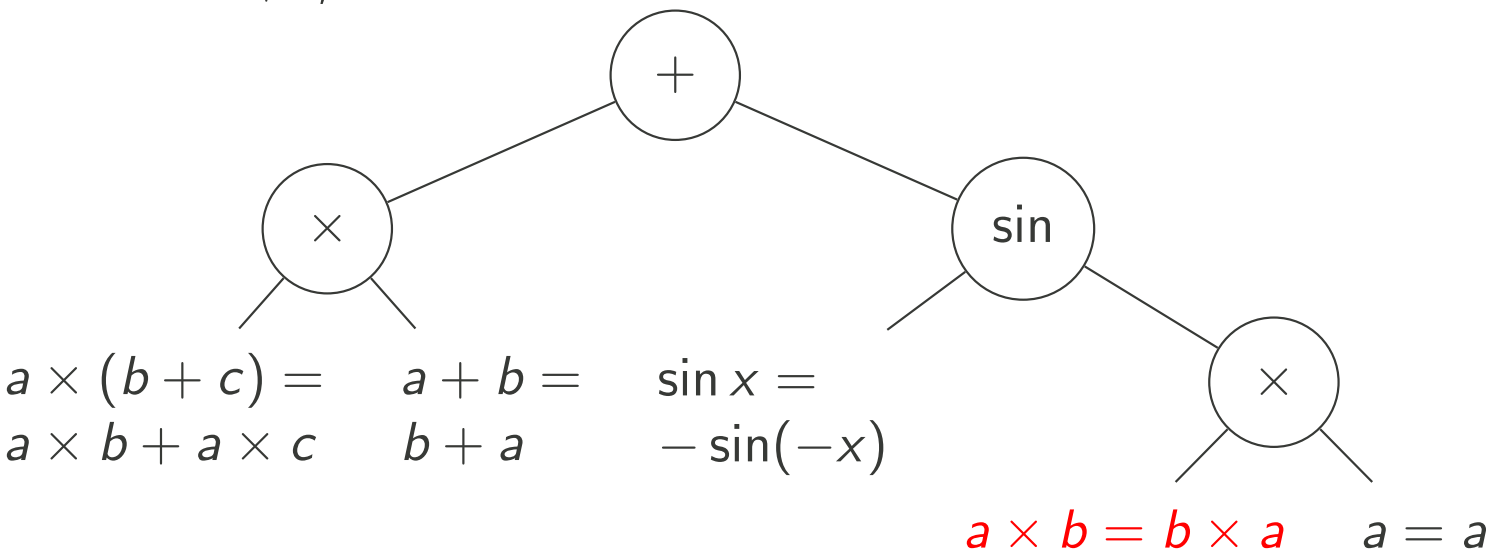
Single-path query (2)

Query tree for conjecture "$(a + b) \times c = a \times c + b \times c$".
Features: "$+$", "$\times$".



Single-path query (2)

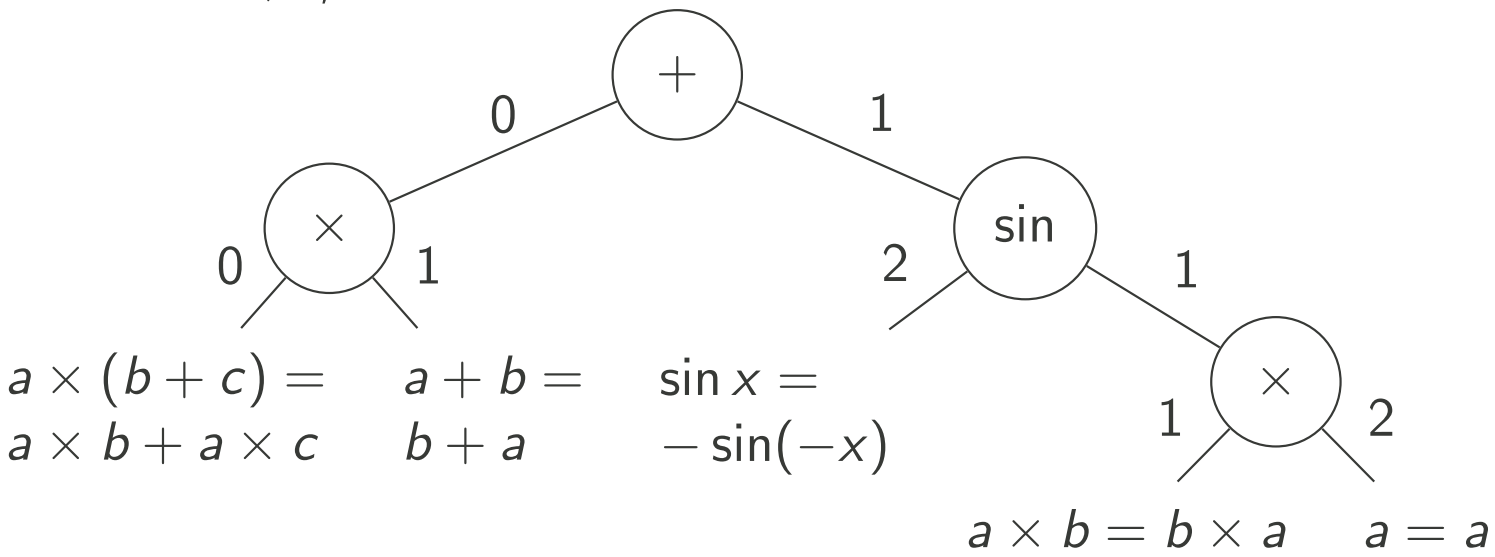
Query tree for conjecture "$(a + b) \times c = a \times c + b \times c$".
Features: "$+$", "$\times$".



$a \times b = b \times a$ is not considered!

Multi-path query

Weight samples by the number of errors on each path.
Features: “+”, “×”.



Splitting feature

Agrawal et al.

- Take n random features from samples and choose feature with lowest Gini impurity (probability of mis-labeling)
- Problem: Gini impurity calculation slow
- Choose feature that divides samples most evenly ($|S_f| \approx |S_{\neg f}|$)

Online / Offline forests

tree is updated or completely rebuilt [*Agraval, Saffari*]

Approach for premise selection

- when a branch learns new samples, check whether the branch feature is still an optimal splitting feature wrt. the new data
- if yes, update subtrees with new data
- if no, rebuild tree

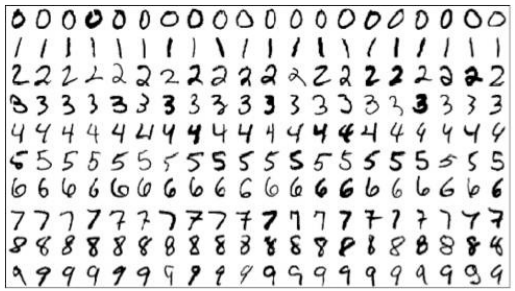
learning takes 21 min for the Mizar dataset...

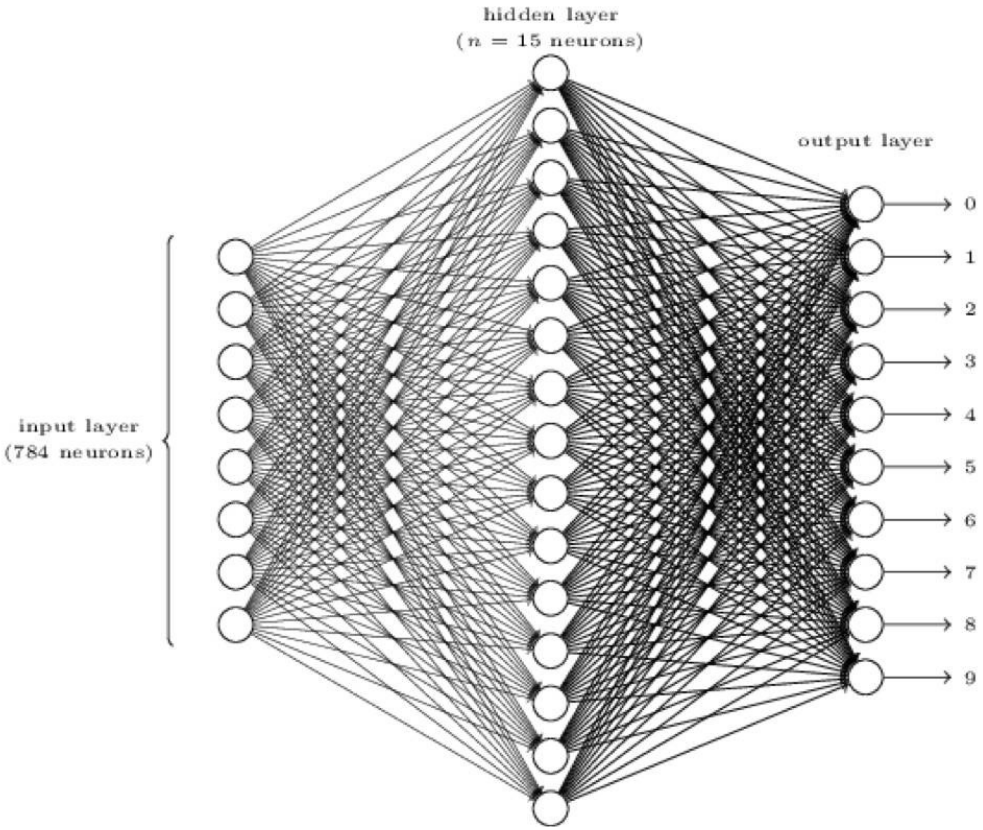
Neural Networks (Introduction in 2 slides)

Neural Networks (Introduction in 2 slides)

Recognize a handwritten character

Measure: recognition rate





Works ok on MNIST

Neural Networks: Third edition

Modelling of Neurophysiological Networks (1950s – 1960s)

- Simple networks of individual perceptrons, with basic learning
- Severe limitations [*Minsky,Papert*]

Paralled Distributed Processing (1990s)

- rejuvenated interest [*Rumelhart,MacClelland*]
- But statistical algorithms were comparably powerful (SVM)

Deep Learning (2010s)

- Data-oriented algorithms
- Data and processing were a limitation before

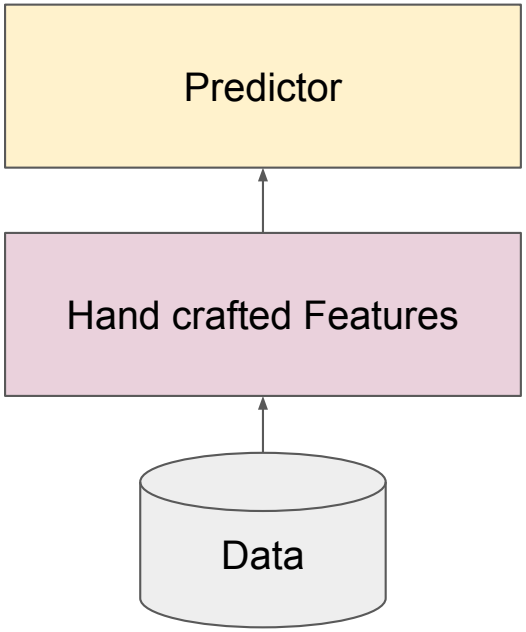
Expressiveness of multilayer perceptron networks

Perceptrons implement linear separators, but:

- Every continuous function modeled with three layers (= 1 hidden)
- Every function can be modeled with four layers
- But the layers are assumed to be arbitrarily large!

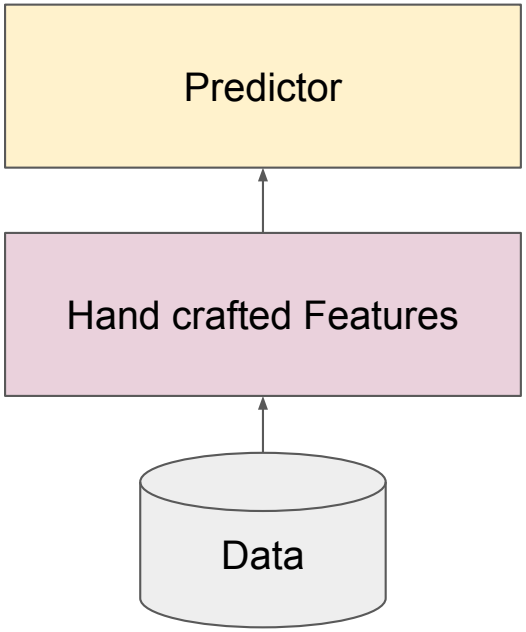
(Results recently formalized)

Deep Learning vs Shallow Learning

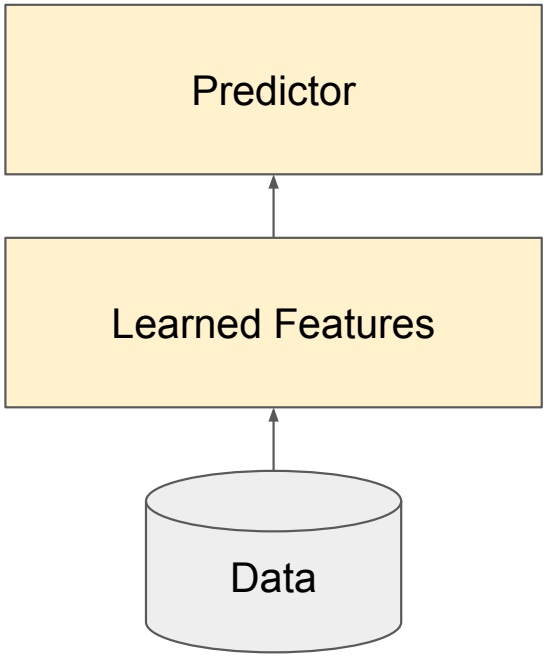


Traditional machine learning

Deep Learning vs Shallow Learning

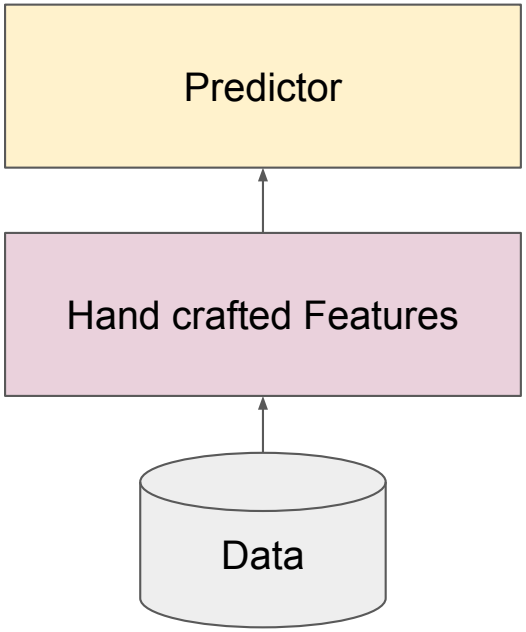


Traditional machine learning

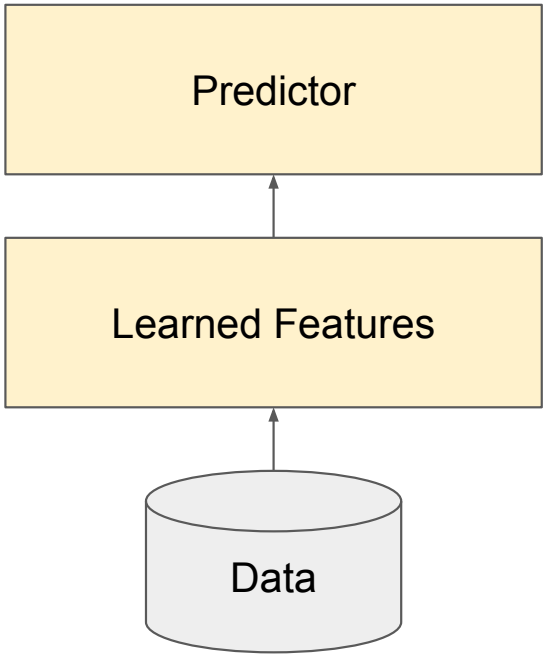


Deep Learning

Deep Learning vs Shallow Learning



Traditional machine learning



Deep Learning

- Mostly convex, provably tractable
- Special purpose solvers
- Non-layered architectures

- Mostly NP-Hard
- General purpose solvers
- Hierarchical models

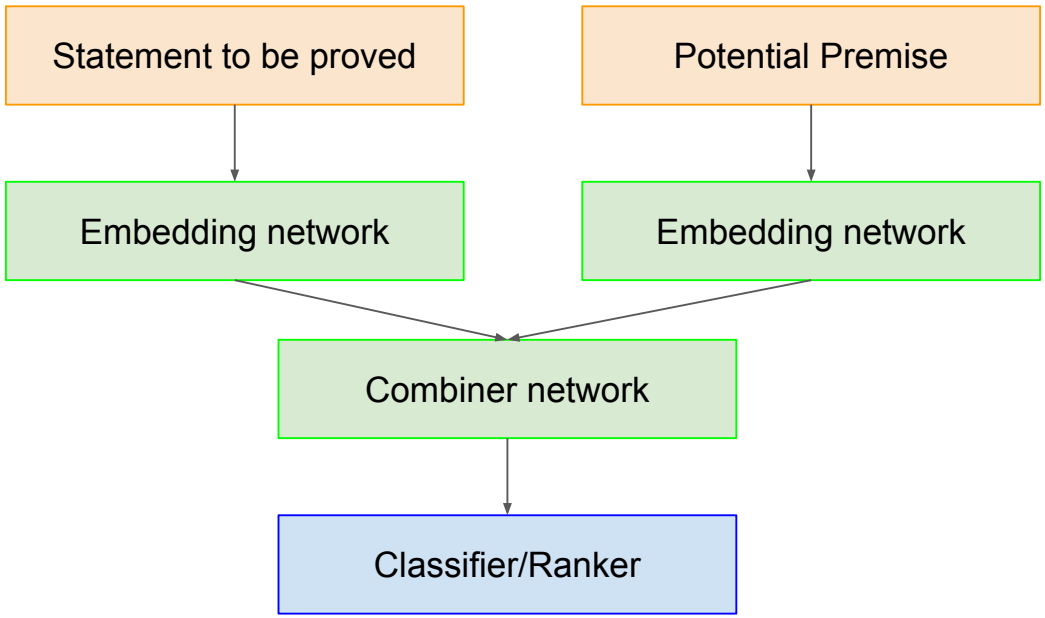
DeepMath intuition

[*Alemi'16*]

Simple classifier on top of concatenated embeddings

- different model of premise selection
- trained to estimate usefulness
- positive and negative examples

Architecture



Deep Learning for Mizar Lemma Selection

- No hand-engineered features
- Comparison of various neural architectures
- Semantic-aware definition embeddings
- Complementary to previous approaches
- Can be ensembled

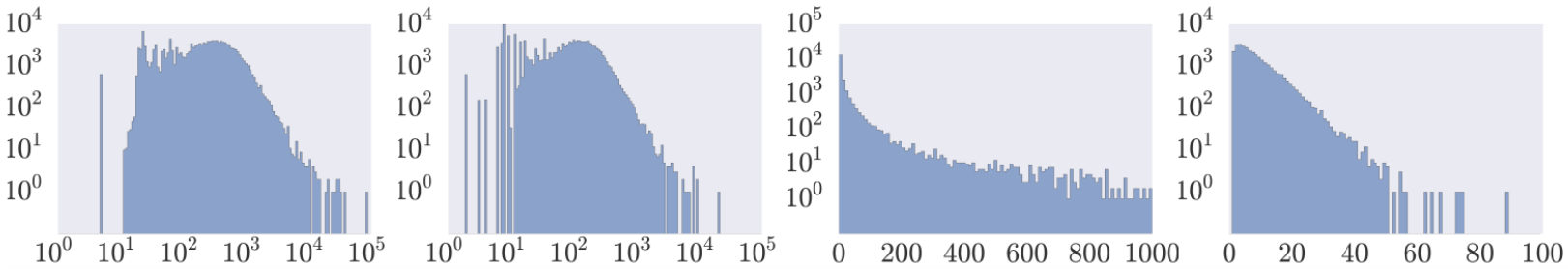
DeepMath: Dataset

The Mizar Mathematical Library (MML) is a corpus of formal mathematical proofs, containing 57,917 theorems from a wide varity of mathematical subjects. We worked with a version converted to first order logic in the TPTP format.

```
:: t99_jordan: Jordan curve theorem in Mizar
for C being Simple_closed_curve holds C is Jordan;

:: Translation to first order logic
fof(t99_jordan, axiom,  (! [A] :  ( (v1_topreal2(A) & m1_subset_1(A,
k1_zfmisc_1(u1_struct_0(k15_euclid(2))))  => v1_jordan1(A)) ) ).
```

Figure 1: (top) The final statement of the Mizar formalization of the Jordan curve theorem. (bottom) The translation to first-order logic, using name mangling to ensure uniqueness across the entire corpus.



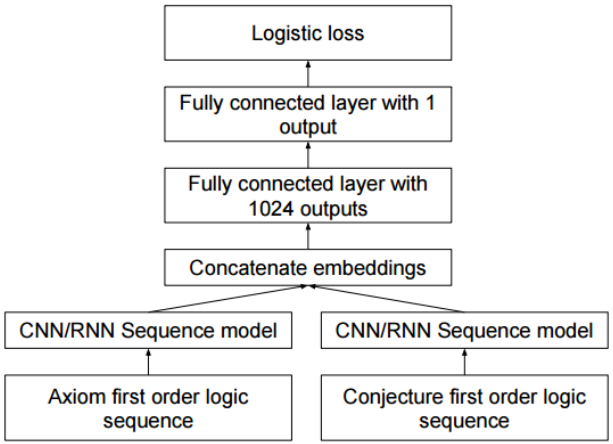
(a) Length in chars. (b) Length in words. (c) Word occurrences. (d) Dependencies.

Figure 2: Histograms of statement lengths, occurrences of each word, and statement dependencies in the Mizar corpus translated to first order logic. The wide length distribution poses difficulties for RNN models and batching, and many rarely occurring words make it important to take definitions of words into account.

DeepMath: Problem, Metric, Model

Definition (Premise selection problem)**.** *Given a large set of premises $\mathcal{P}$, an ATP system A with given resource limits, and a new conjecture C, predict those premises from $\mathcal{P}$ that will most likely lead to an automatically constructed proof of C by A.*

$$\text{aMRR} = \underset{C}{\text{mean}} \max_{P \in \mathcal{P}_{\text{test}}(C)} \frac{\text{rank}(P, \mathcal{P}_{\text{avail}}(C))}{|\mathcal{P}_{\text{avail}}(C)|}$$



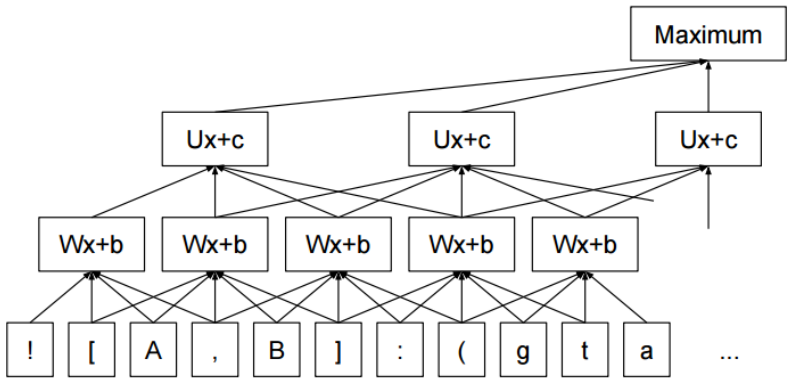


Figure 3: (left) Our network structure. The input sequences are either character-level (section 5.1) or word-level (section 5.2). We use separate models to embed conjecture and axiom, and a logistic layer to predict whether the axiom is useful for proving the conjecture. (right) A convolutional model.

Recurrent Neural Networks

Recurrent Neural Networks (RNN)

process sequences by feeding back the output into the next input

Long-Short Term Memory (LSTM)

add forgetting to RNNs

DeepMath: Architectures

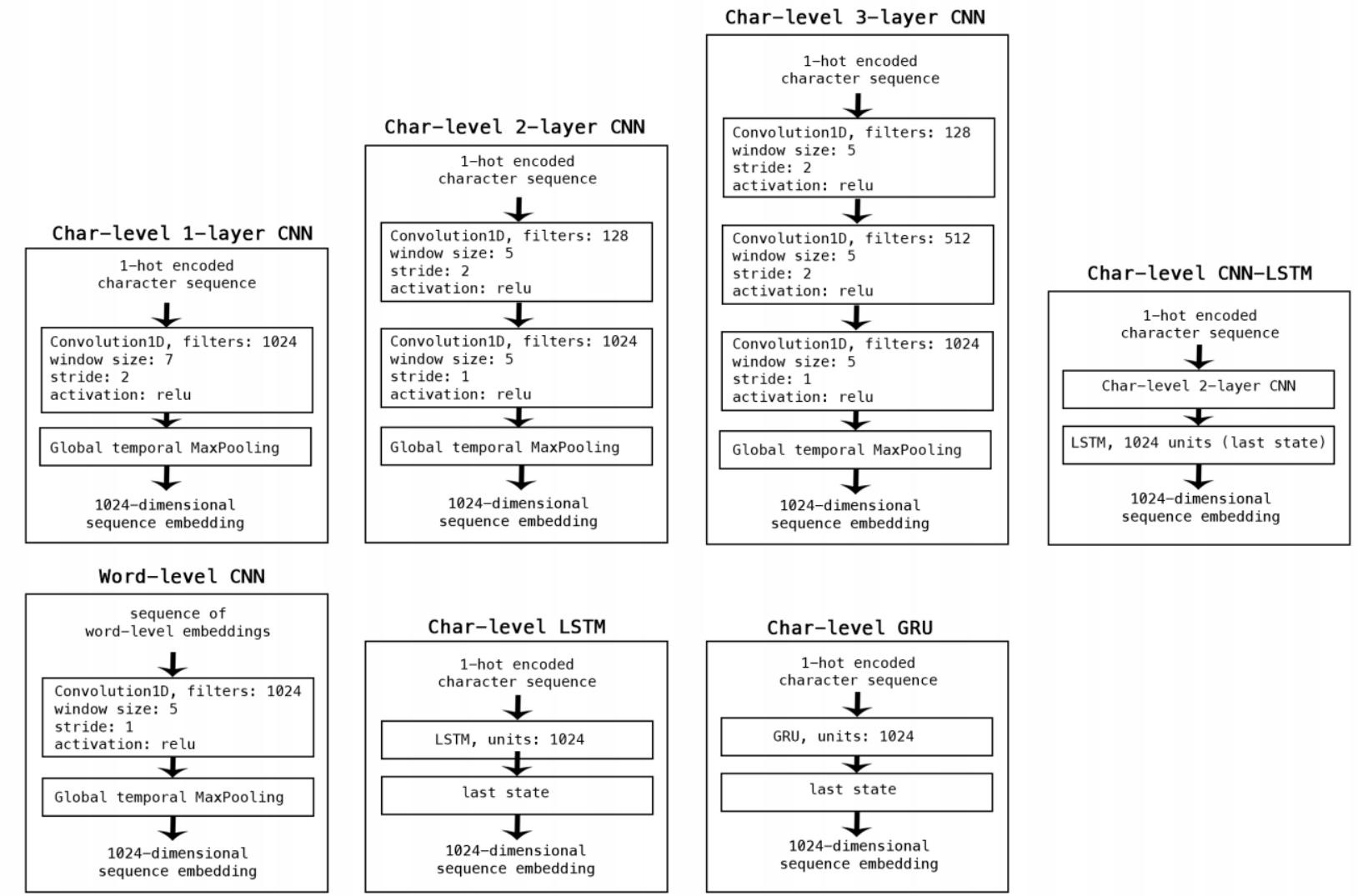


Figure 4: Specification of the different embedder networks.

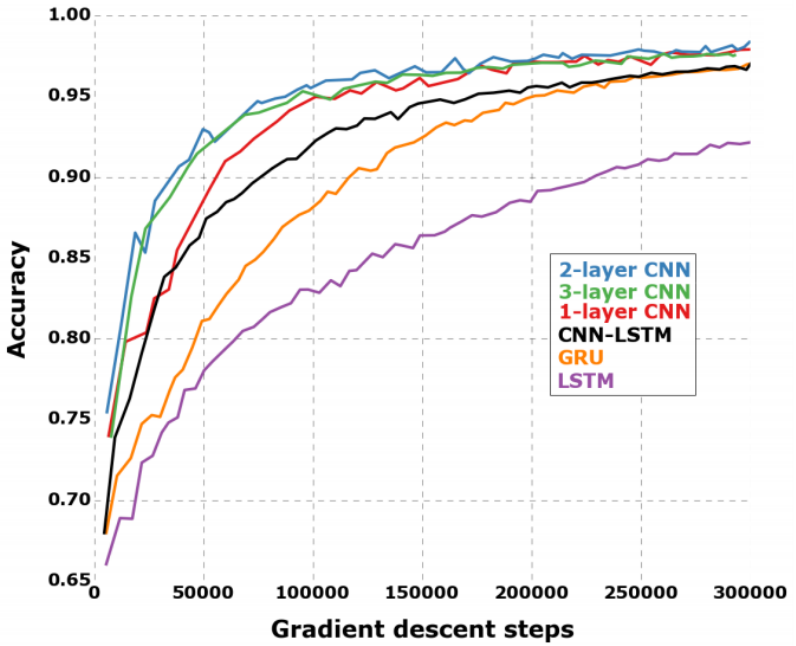
DeepMath: Results

Cutoff	k-NN Baseline (%)	char-CNN (%)	word-CNN (%)	def-CNN-LSTM (%)	def-CNN (%)	def+char-CNN (%)
16	674 (24.6)	687 (25.1)	709 (25.9)	644 (23.5)	734 (26.8)	835 (30.5)
32	1081 (39.4)	1028 (37.5)	1063 (38.8)	924 (33.7)	1093 (39.9)	1218 (44.4)
64	1399 (51)	1295 (47.2)	1355 (49.4)	1196 (43.6)	1381 (50.4)	1470 (53.6)
128	1612 (58.8)	1534 (55.9)	1552 (56.6)	1401 (51.1)	1617 (59)	1695 (61.8)
256	1709 (62.3)	1656 (60.4)	1635 (59.6)	1519 (55.4)	1708 (62.3)	1780 (64.9)
512	1762 (64.3)	1711 (62.4)	1712 (62.4)	1593 (58.1)	1780 (64.9)	1830 (66.7)
1024	1786 (65.1)	1762 (64.3)	1755 (64)	1647 (60.1)	1822 (66.4)	1862 (67.9)

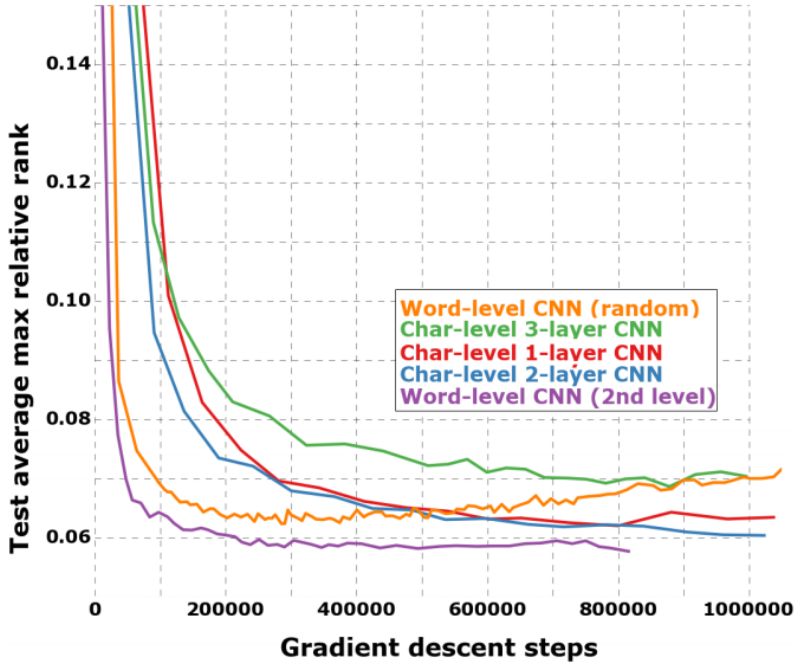
Table 1: Results of ATP premise selection experiments with hard negative mining on a test set of 2,742 theorems.

- E-prover proved theorem percentages
- Union of all methods: 80.9%
- Union of deep network methods: 78.4%

DeepMath: Accuracy

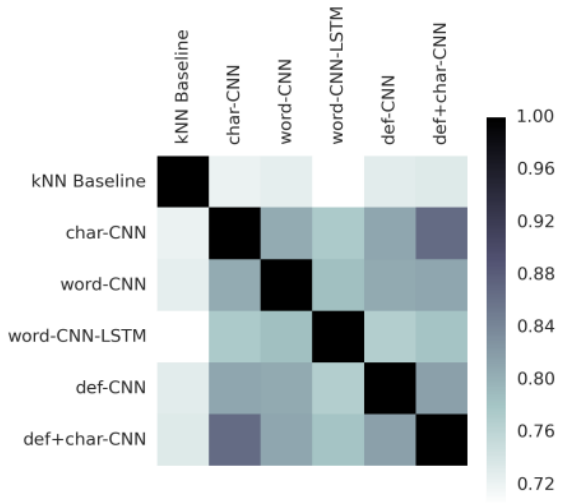


(a) Training accuracy for different character-level models without hard negative mining. Recurrent models seem underperform, while pure convolutional models yield the best results. For each architecture, we trained three models with different random initialization seeds. Only the best runs are shown on this graph; we did not see much variance between runs on the same architecture.



(b) Test average max relative rank for different models without hard negative mining. The best is a word-level CNN using definition embeddings from a character-level 2-layer CNN. An identical word-embedding model with random starting embedding overfits after only 250,000 iterations and underperforms the best character-level model.

DeepMath: Statistics



(a) Jaccard similarities between proved sets of conjectures across models. Each of the neural network model prediction are more like each other than those of the k-NN baseline.

Model	Test min average relative rank
char-CNN	0.0585
word-CNN	0.06
def-CNN-LSTM	0.0605
def-CNN	**0.0575**

(b) Best sustained test results obtained by the above models. Lower values are better. This was monitored continuously during training on a holdout set with 400 theorems, using all true positive premises and 128 randomly selected negatives. In this setup, the lowest attainable max average relative rank with perfect predictions is 0.051.

Hard Negatives

Learning Lemma Usefulness

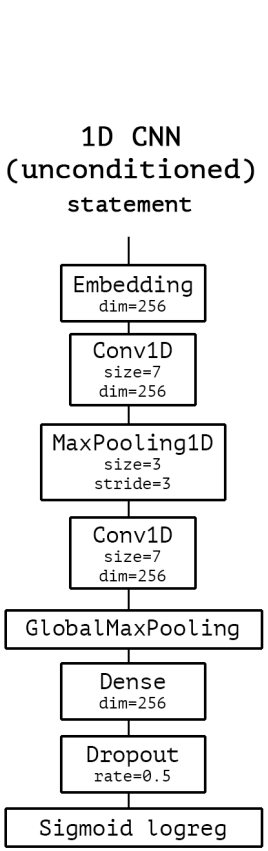
HOLStep Dataset

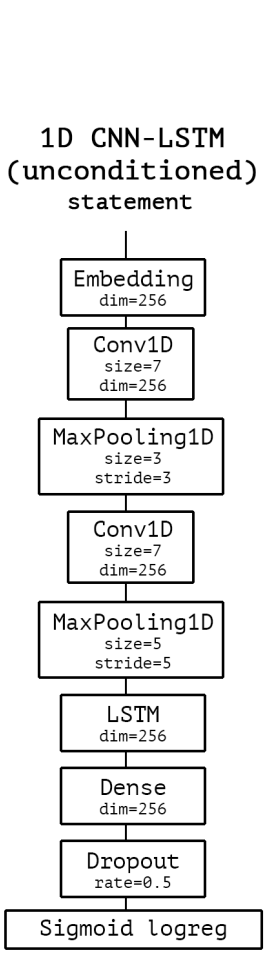
- Intermediate steps of the Kepler proof
- Only relevant proofs of reasonable size
- Annotate steps as useful and unused
  - Same number of positive and negative
- Tokenization and normalization of statements

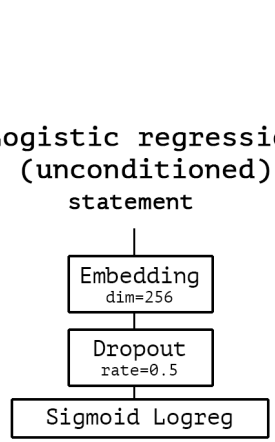
Statistics

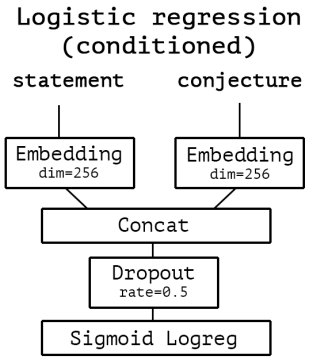
	Train	Test	Positive	Negative
Examples	2013046	196030	1104538	1104538
Avg. length	503.18	440.20	535.52	459.66
Avg. tokens	87.01	80.62	95.48	77.40
Conjectures	9999	1411	-	-
Avg. deps	29.58	22.82	-	-

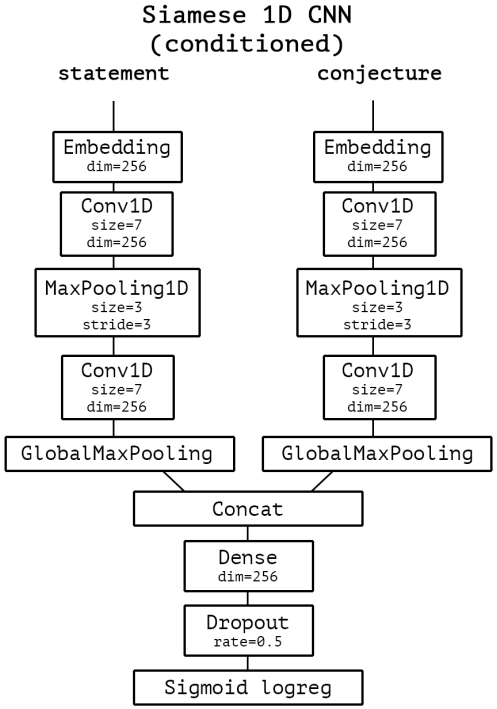
Considered Models

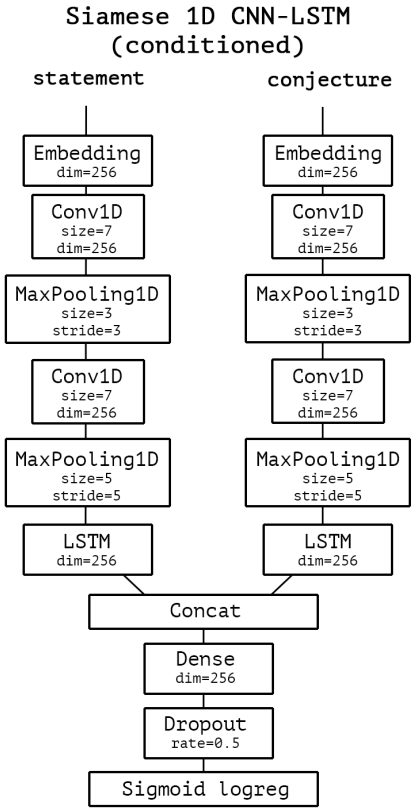




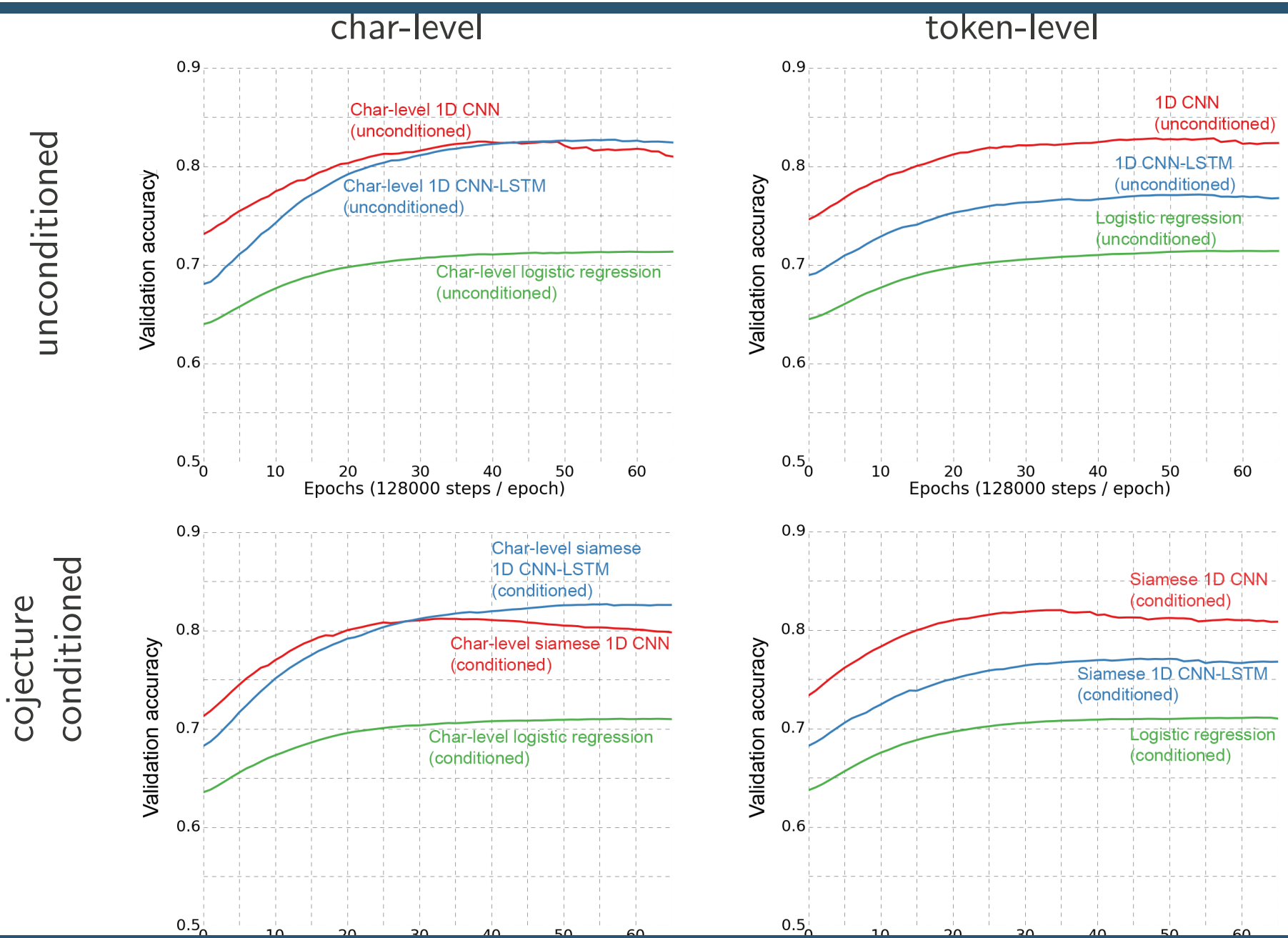








Baselines (Training Profiles)



What about full automated proofs?

Proof by contradiction

- Assume that the conjecture does not hold
- Derive that axioms and negated conjecture imply $\bot$

Saturation

- Convert problem to CNF
- Enumerate the consequences of the available clauses
- Goal: get to the empty clause

Redundancies

Simplify or eliminate some clauses (contract)

Summary

Today

- Theorem proving systems
- Machine learning problems
- Lemma relevance
- Deep learning for theorem proving

Tomorrow

- Guided Automated Reasoning
- More human-like proof
- Logical translations
- Unsupervised methods