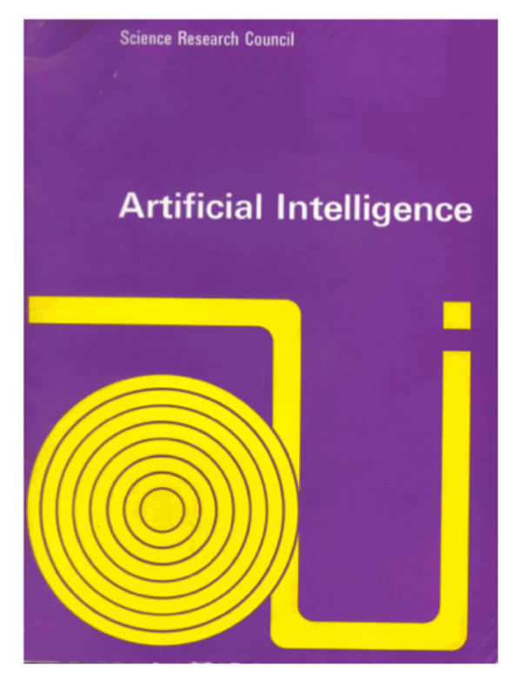
Learning to Discover Efficient Mathematical Identities

Wojciech Zaremba[1,3], Karol Kurach[2,3], Rob Fergus[1,4]

[1]Courant Institute NYU; [2]University of Warsaw; [3]Google; [4]Facebook

The Combinatorial Explosion
Discrete search is a central issue in AI



From "Artificial Intelligence: A General Survey" by Professor Sir James Lighthill, 1973.

"...failure to recognize the implications of the combinatorial explosion. This is a general obstacle to the construction of a self-organising system on a large knowledge base which results from the explosive growth of any combinatorial expression, representing numbers of possible ways of grouping elements of the knowledge base according to particular rules, as the base's size increases."

Math & Theorem Proving

Contemporary theorem provers

- Requires search over all possible combinations of operators
- Intractable for all but simple proofs
- Automated Theorem Provers use heuristics

Yet (some) humans are able to prove theorems

- Have experience of related problems
- Known math "tricks"

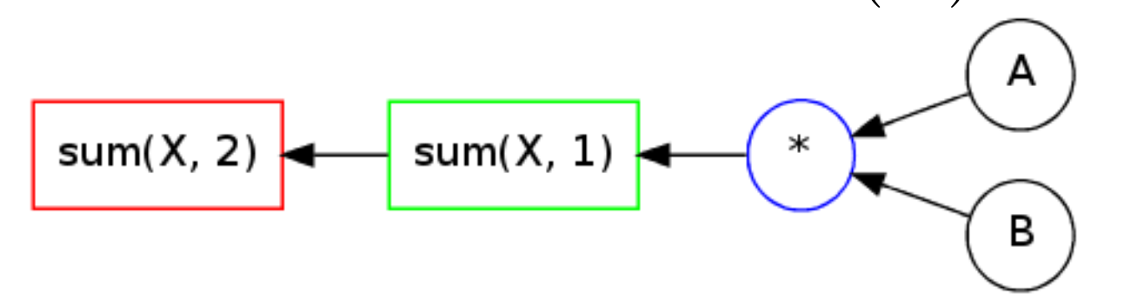
We focus on simpler problem: discovering identities in mathematics.

Toy Example

Consider two matrices A and B:

$$\sum_{i,k}(AB)_{i,k} = \sum_i\sum_j\sum_k a_{i,j}b_{j,k}$$

Naive computation takes $O(n^3)$:



Discovering Efficient Identities

Define a grammar G of operators Given some target expression T within the domain of G e.g. sum(sum(A*B,2),1). Find an identical expression that has lower computational complexity i.e. avoids high complexity operators (like matrix multiplication).

Matrix-matrix multiply	X * Y
Matrix-vector multiply	X * y
Matrix-element multiply	X .* Y
Matrix transpose	X'
Column-sum	sum(X,1)
Row-sum	sum(X,2)
Column-repeat	repmat(X,1,m)
Row-repeat	repmat(X,n,1)

Polynomials of degree 1

$$A, A^T, \sum_i A_{i,:}, \sum_j A_{:,j}, \sum_{i,j} A_{i,j}, \dots$$

Polynomials of degree 2

$$A^2, (A^2)^T, AA^T, A^TA, \sum_i (AA^T)_{i,:}, \dots$$

Space grows exponentially fast.

Representing Symbolic Expressions

Target expression: sum(sum(A*A,1),2) Use P copies of A
Representation of target is descriptor vector (length P)

- Each element is evaluation one copy
- Vector is of length P
- If descriptors match equivalent expressions.

Using is unstable, so use integers modulo q, where q is large prime.

Example: Taylor Series Approximation

Consider RBM partition function:

$$\sum_{v,h}\exp(v^TWh) = \sum_{k=0}^{\infty}\sum_{v,h}\frac{(v^TWh)^k}{k!}$$
$$v \in \{0,1\}^n$$
$$h \in \{0,1\}^m$$

6th term. Higher terms too complex for manual derivation.

1st term in Taylor series:

$$\sum_{k=0}^{\infty}\sum_{v,h} v^TWh = 2^{n+m-2}\sum_{i,j}W_{i,j}$$

2st term in Taylor series:

$$\sum_{k=0}^{\infty}\sum_{v,h}(v^TWh)^2 = 2^{n+m-4}[\sum_{i,j}W_{i,j}^2 + (\sum_{i,j}W_{i,j})^2 + \sum_i(\sum_j W_{i,j})^2 + \sum_j(\sum_i W_{i,j})^2]$$

Note that identity computes in $O(nm)$ versus $O(2^{n+m})$ for original expression.

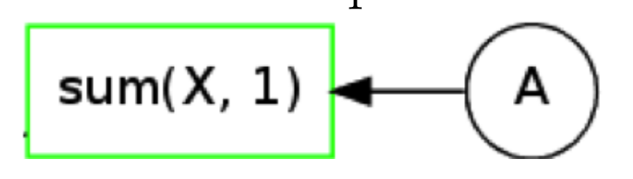
Prior Over Computation Trees

Recall goal: find equivalent expressions to target i.e. descriptors match Restrict grammar to use operators with lower complexity than target If any match found then sure to be efficient w.r.t. target

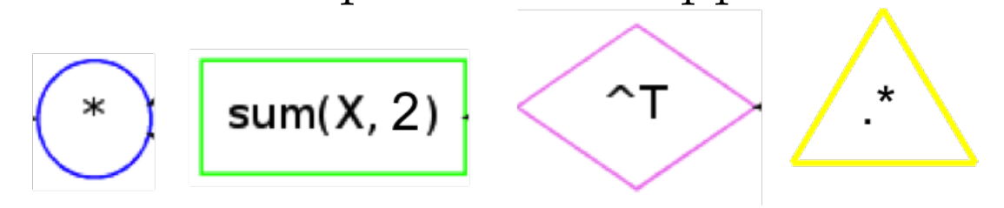
Want to learn a good prior over expressions

Scheduler picks potential new operators to append to current expression(s).
Example:

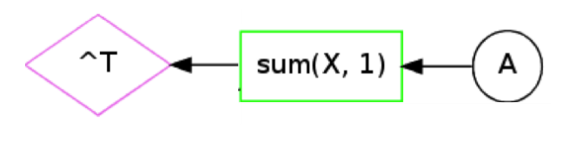
Current expression:



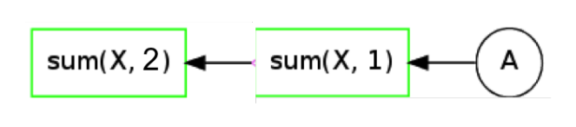
Valid operators to append:



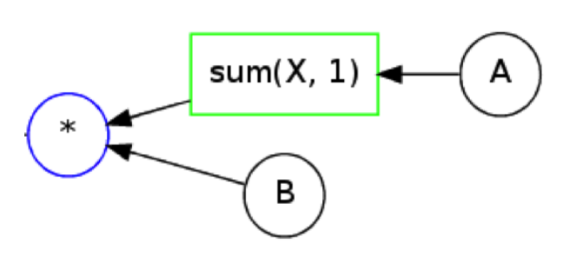
Scorer ranks each possibility (i.e. how likely they are to lead to the solution), using prior.



Score 0.3



Score 0.05



Score 0.65

Sample new operator according to scorer probabilities

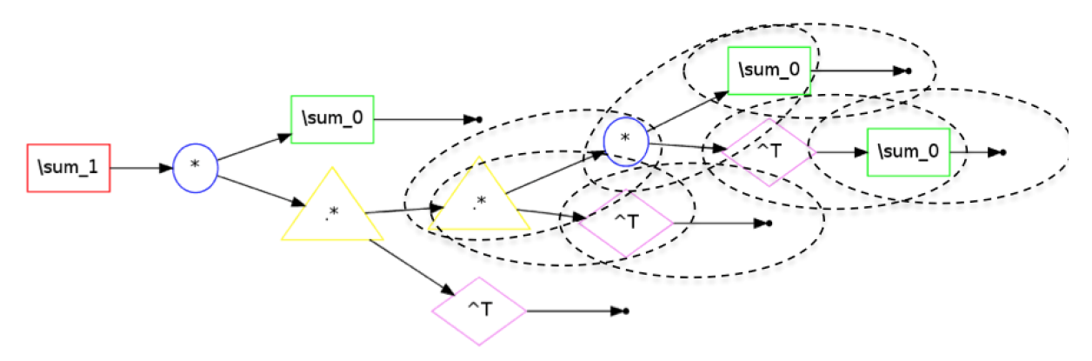
Scorer Strategy

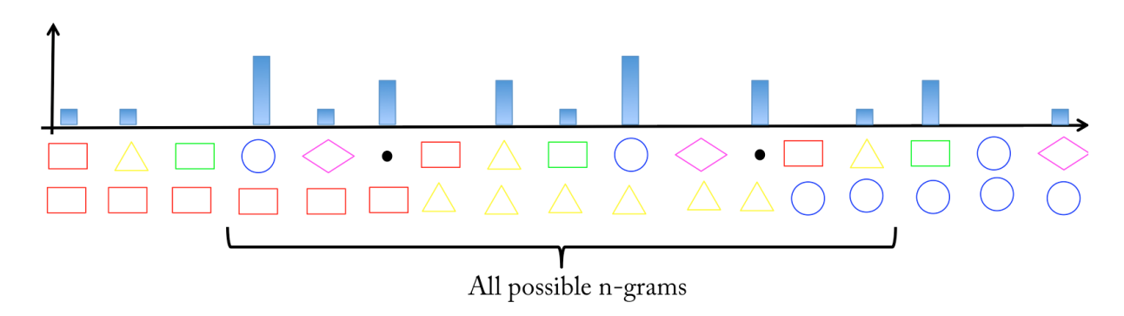
- Naive: no prior, Just select randomly from all valid operators
- n-gram prior
- Recurrent Neural Network prior

Use curriculum learning approach. Start with easy targets, and uniform prior (low polynomial degree k). Refine prior using new solution.

	Target Expression	Efficient expression
K = 1	$\sum_i a_i$	sum(sum(A, 1), 2)
K = 2	$\sum_{i,j} a_i a_j$	0.5 * (sum((repmat(sum(A', 1), 1, m) .* A), 2)) + -0.5 * (sum((A .* A)', 1)))
K = 3	$\sum_{i,j,k} a_i a_j a_k$	(-0.5 * (sum((repmat(sum((A' .* A'), 1), 1, m) .* A), 2)) + 1/3 * (sum(((A' .* A')' .* A), 2)) + 1/6 * (sum((A' .* repmat(sum((repmat(sum(A', 1), 1, m) .* A), 2), m, 1)), 1)))

n-gram



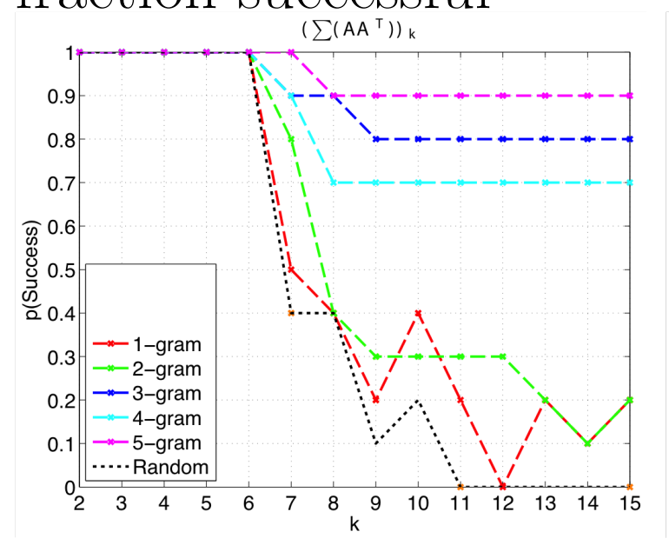
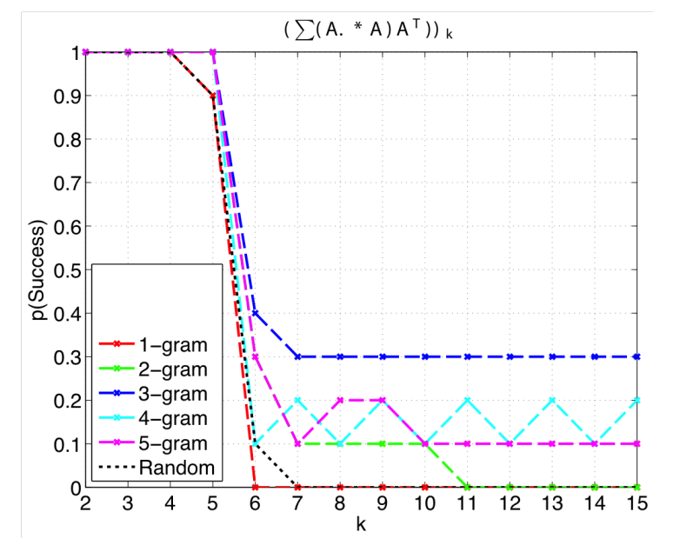
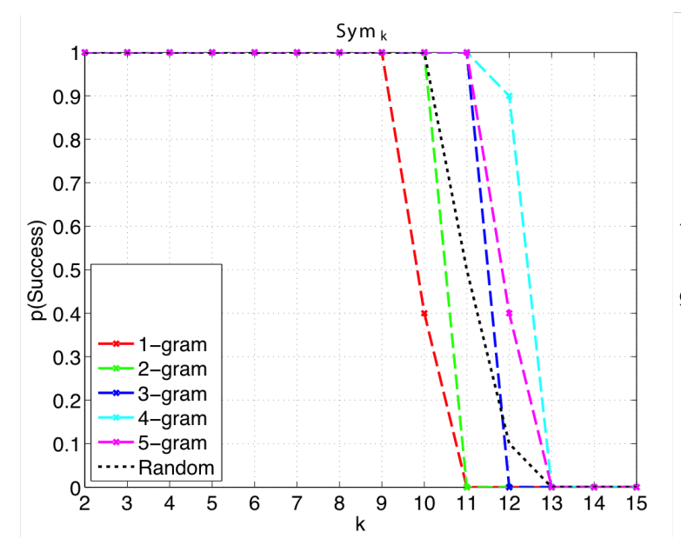
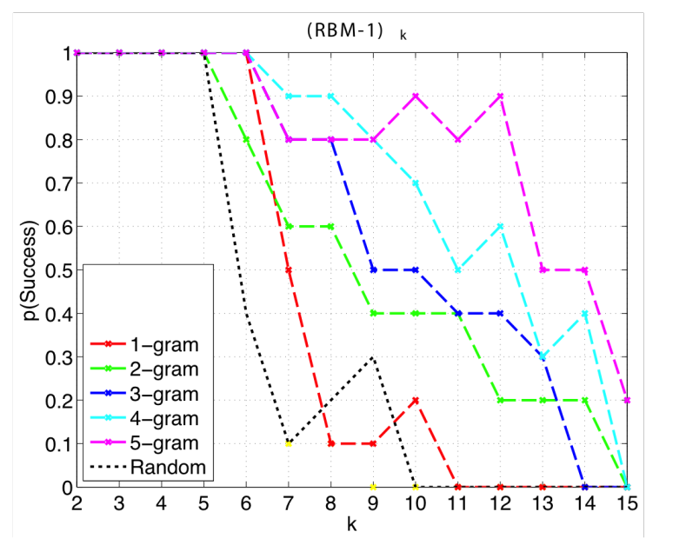


Experiments in discovering identities

5 families of expressions (vary degree k)

- Multiply-sum: $\sum(AA^T)_k$
- Element-wise multiply-sum: $\sum((A.*A)A^T)_k$
- Symmetric polynomials: $\sum_{i,j,k} a_i a_j a_k$
- RBM-1: $\sum_{v\in\{0,1\}^n}(v^tA)^k$
- RBM-2: $\sum_{v\in\{0,1\}^n, h\in\{0,1\}^m}(v^tAh)^k$

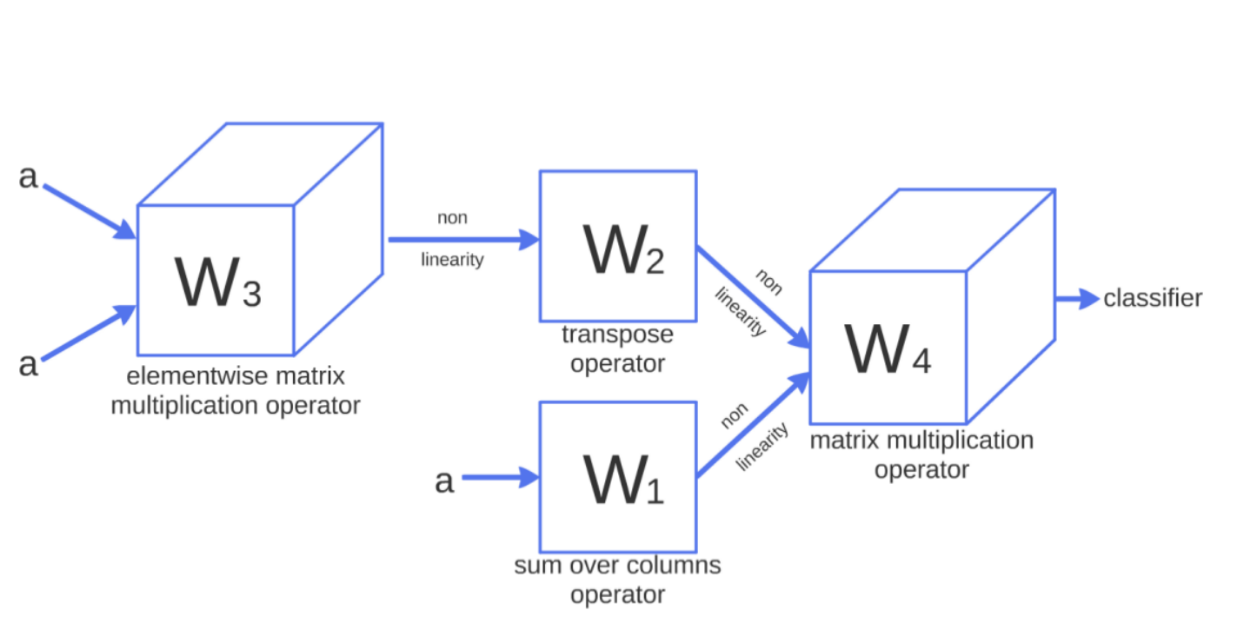
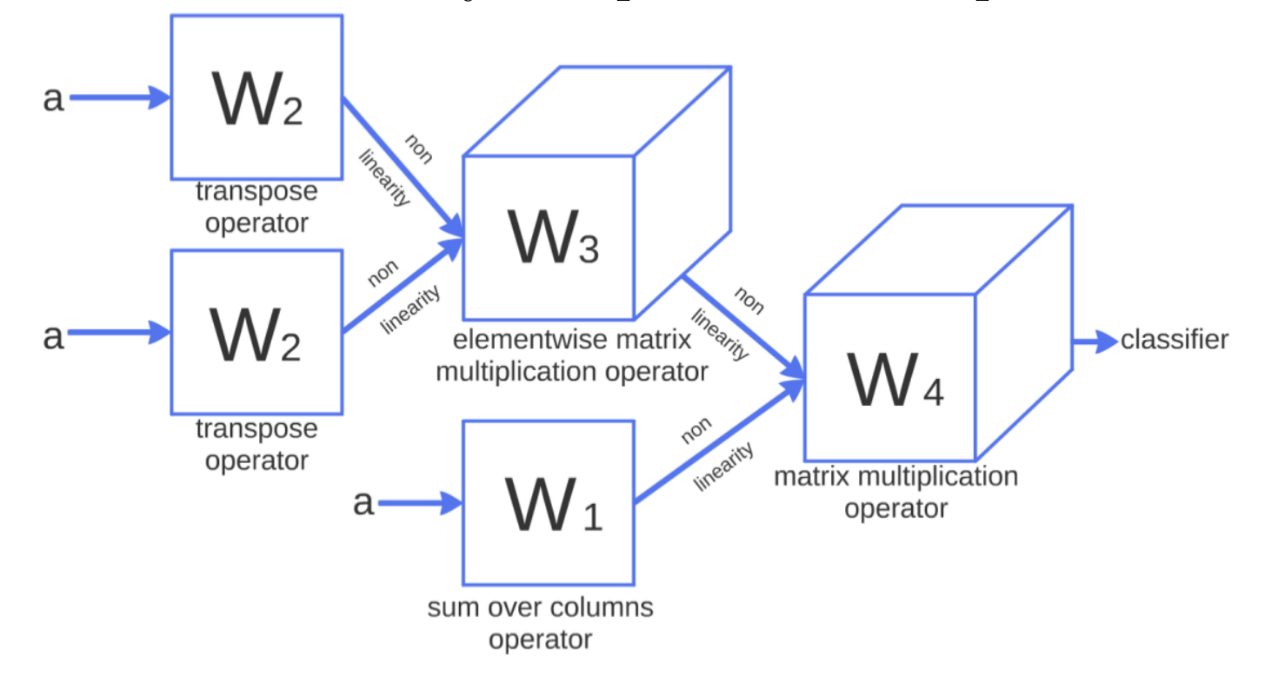
Start with $k = 1$ and work up to $k = 15$. Time cut-off: 600 seconds Repeat 10 times, measure fraction successful

RBM-2
No scorer strategy able to get beyond $k = 5$ However, the $k = 5$ solution was found by the RNN consistently faster than the random strategy (100 +- 12 vs 438 +- 77 secs).

Recurrsive neural network strategy

n-gram has a shallow understanding of mathematical expressions (up to depth n). We trained recurrsive neural network to identify equivalent expressions.



a
W3
elementwise matrix multiplication operator
W2
transpose operator
W1
sum over columns operator
W4
matrix multiplication operator
non linearity
classifier

- Training data: for each degree find groups of equivalent expressions (exhaustive search).
- Label each group as a different class.
- Add softmax classifier to RNN and train with cross-entropy loss.

Training examples for degree k=6:

```
(((sum((sum((A * (A')), 1)), 2)) * ((A * (((sum((A'), 1)) * A)'))')) * A)
(sum(((sum((A * (A')), 2)) * ((sum((A'), 1)) * (A * ((A') * A)))), 1))
(((sum(A, 1)) * (((sum(A, 2)) * (sum(A, 1)))')) * (A * ((A') * A)))
((((sum((sum((A * (A')), 1)), 2)) * ((sum((A'), 1)) * (A * ((A') * A))))')')
((sum(A, 1)) * (((A') * (A * ((A') * ((sum(A, 2)) * (sum(A, 1))))))'))
((sum((sum((A * (A')), 1)), 2)) * ((sum((A'), 1)) * (A * ((A') * A))))
(((sum((sum((A * (A')), 1)), 2)) * ((sum((A'), 1)) * A)) * ((A') * A))
```

(a) Class A

```
((A') * ((sum(A, 2)) * ((sum((A'), 1)) * (A * (((sum((A'), 1)) * A)')))))
(sum(((A') * ((sum(A, 2)) * ((sum((A'), 1)) * (A * ((A') * A))))), 2))
((((sum(A, 2)) * ((sum((A'), 1)) * A))') * (A * (((sum((A'), 1)) * A)')))
(((sum((A'), 1)) * (A * ((A') * ((sum(A, 2)) * ((sum((A'), 1)) * A)))))')
((((sum((A'), 1)) * A)') * ((sum((A'), 1)) * (A * (((sum((A'), 1)) * A)'))))
(((A * ((A') * ((sum(A, 2)) * ((sum((A'), 1)) * A))))') * (sum(A, 2)))
(((A') * ((sum(A, 2)) * ((sum((A'), 1)) * A))) * (sum(((A') * A), 2)))
```

(b) Class B

	Degree $k = 3$	Degree $k = 4$	Degree $k = 5$	Degree $k = 6$
Test accuracy	$100\% \pm 0\%$	$96.9\% \pm 1.5\%$	$94.7\% \pm 1.0\%$	$95.3\% \pm 0.7\%$
Number of classes	12	125	970	1687
Number of expressions	126	1520	13038	24210

Note: no explicit knowledge of math operators

Related work

- Recent efforts to combine ML with theorem provers, e.g. Bridge et al. 2014
- Distilling Free-Form Natural Laws from Experimental Data, Michael Schmidt and Hod Lipson. Science, Vol. 324, April 3, 2009.
- Recursive neural nets in NLP Socher et al. 2013, Luong et al., learning logics Bowman et al. 2014 for logical predicates
- Learning to Execute, Wojciech Zaremba & Ilya Sutskever, arXiv 1410.4615, 2014