Learning to manipulate symbols

by **Wojciech Zaremba**
Tubingen 2014

How to build an intelligent system ?

How to build an intelligent system ?

What tasks should it solve ?

How to build an intelligent system ?

What tasks should it solve ?

Is chess enough ? Or is object recognition enough ?

Few ideas - choose proper tasks

- Atari games as a simplified world
- Learning entire algorithms (requires to deeper understanding / planning)
  - Neural Turing Machine
  - Program Learning
  - Mathematics learning

What can learn our models ?

What can learn our models ?

Can they learn addition ?

What can learn our models ?

Can they learn addition ?

Can they learn arbitrary computation function ?

Examples

Input:

```
i=8827
c=(i-5347)
print((c+8704) if 2641<8500 else
    5308)
```

Target: 12184.

Input:

```
j=8584
for x in range(8):
  j+=920
b=(1500+j)
print((b+7567))
```

Target: 25011.

Sequence of character on the input and on the output.

Why is it important ?

It’s a very hard task that requires:

- modelling long-distance dependencies
- memory (e.g. variable assignment)
- branching (if-statement)
- multiple tasks within one

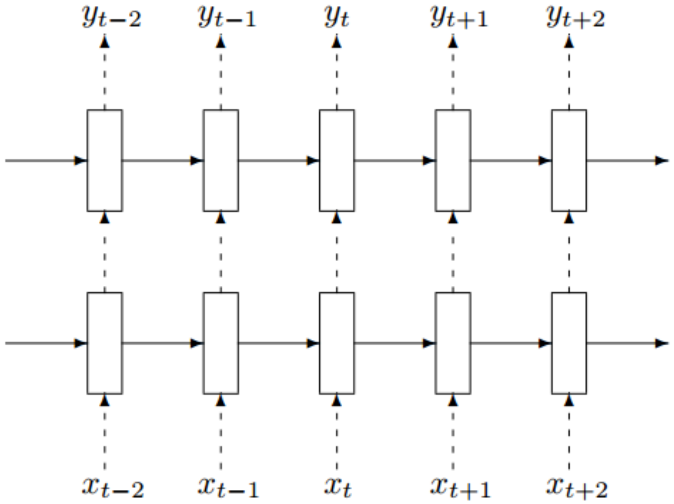
Data consumption

Model reads programs character by character, and tries to predict execution output.

It doesn’t need to predict the next character in every step.

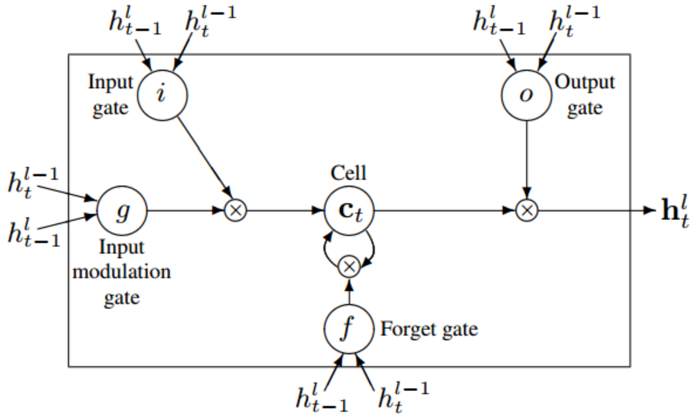
Our model - RNN

- 2 layers
- 400 units each
- trained with SGD
- cross-entropy loss
- Input vocabulary size 42
- Output vocabulary size 11



Our model - RNN with LSTM* cells

- LSTM presumably can model long range dependencies
- Train until there is no improvement on a validation set.



* S Hochreiter, J Schmidhuber, Graves, Long short-term memory

Subclass of programs

- can be evaluated with a single left-to-right pass
- operations: addition, subtraction, multiplication, variable assignment, if-statement, and for-loops
- Problem complexity is defined in terms of the length of numbers and depth of nesting

Why is it difficult ?

RNN’s point of view:

Input:
vqppkn
sqdvfljmnc
y2vxdddsepnimcbvubkomhrpliibtwztbljipcc
Target: hkhpg

Qualitative results. Exact prediction.

Input:

```
f=(8794 if 8887<9713 else (3*8334))
print((f+574))
```

Target: 9368.

Model prediction: 9368.

Properly deals with if statement and addition.

Qualitative results. 1 digit mistake.

Input:

```
j=8584
for x in range(8):
  j+=920
b=(1500+j)
print((b+7567))
```

Target: 25011.

Model prediction: 23011.

Often leading digits and the last digits are correct.

Qualitative results. Exact prediction.

Input:

```
c=445
d=(c-4223)
for x in range(1):
  d+=5272
print((8942 if d<3749 else 2951))
```

Target: 8942.

Model prediction: 8942.

Some very nested examples might be very simple.

Qualitative results. 2 digit mistake.

Input:

```
a=1027
for x in range(2):
  a+=(402 if 6358>8211 else 2158)
print(a)
```

Target: 5343.

Model prediction: 5293.

Again, leading digits and the last digits are correct.

Scheduling strategies

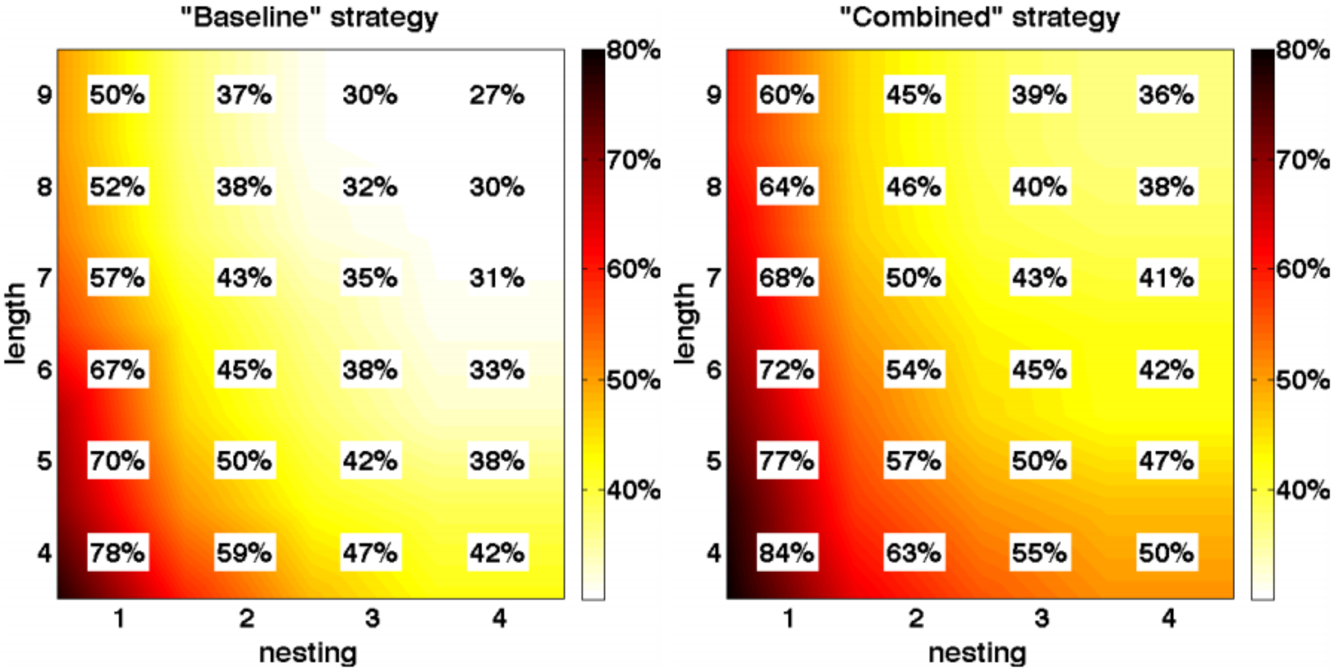
- No curriculum learning (baseline)
  - Learning with target distribution
- Naive curriculum strategy (naive)
  - Making task gradually more difficult

Scheduling strategies

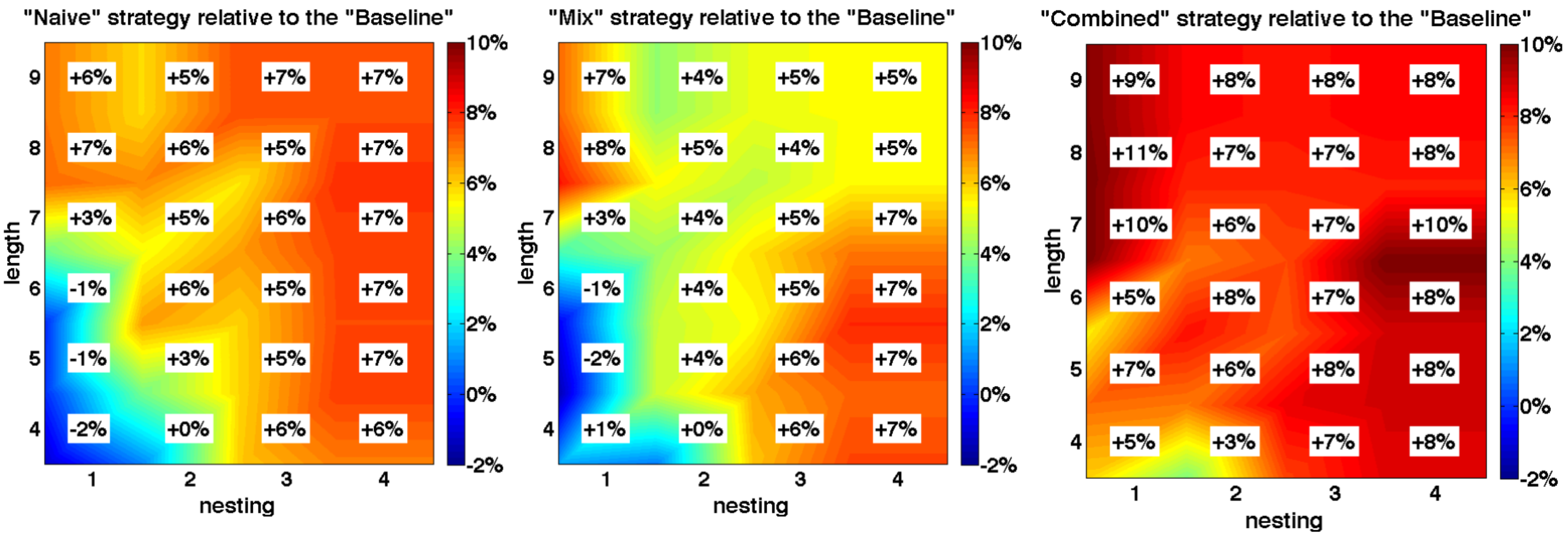
- Mixed strategy (mix)
  - Mix of all levels of hardness. Simplest programs occur as often as hardest one. Distribution rand (10^rand(length)) vs rand(10^length).

- Combined strategy (combined)
  - Combination of mix with naive curriculum learning (so far the best).

Quantitative results. Absolute performance.



Quantitative results.
Relative performance.



Understanding vs. memorizing

- We don’t know how much our networks “understand” the meaning of programs vs how much they memorize.
- Test dataset, validation dataset, and training datasets have no common samples, but are very similar.

Learning identities in mathematics

- Executing computer programs requires learning how to evaluate predefined functions (e.g. addition etc.)
- Proving problems in mathematics is much harder, as we often don’t know proof in advance.
- We can just verify correctness when proof is given.

Mathematics

- Theorem proving
  - Requires search over all possible combinations of operators
  - Intractable for all but simple proofs
- Yet (some) humans are able to do it
  - Have experience of related problems
  - Known math “tricks”

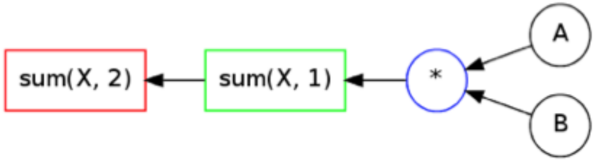
- We focus on simpler problem: discovering identities

Toy Example

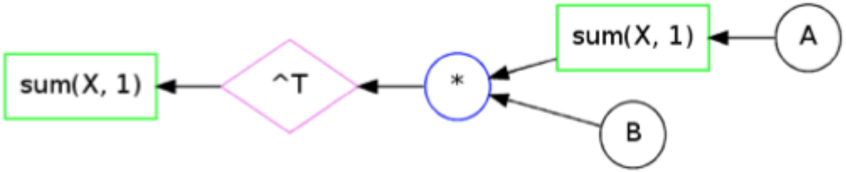
Consider two matrices A and B:

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

Naive computation takes O(n^3):



An equivalent O(n^2) computation:



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

Overview

- Representation of math expressions
- Searching over expressions
- Distributed representation of expressions using a tree neural network (recursive neural networks).

Grammar Rules

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)

Allowable Expressions

- Variables: matrix or vector
- Targets are homogeneous polynomials
  - i.e. only contain terms of same
    - degree (ab + a2 + ac) (all terms are degree 2)
    - but not (a2 +b)

- Still includes many useful expressions

Example: Taylor Series Approximation

Consider RBM partition function:

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

1st term in Taylor series:

$$\sum_{v,h} v^T W h = 2^{n+m-2} \sum_{i,j} W_{i,j}$$
$$v \in \{0,1\}^n$$
$$h \in \{0,1\}^m$$

Example: Taylor Series Approximation

2nd term in Taylor series:

$$\sum_{v,h}(v^T Wh)^2 = 2^{n+m-4}\Big[$$

$$\sum_{i,j} W_{i,j}^2 + (\sum_{i,j} W_{i,j})^2 +$$

$$\sum_i(\sum_j W_{i,j})^2 + \sum_j(\sum_i W_{i,j})^2\Big]$$

$$v \in \{0,1\}^n$$

$$h \in \{0,1\}^m$$

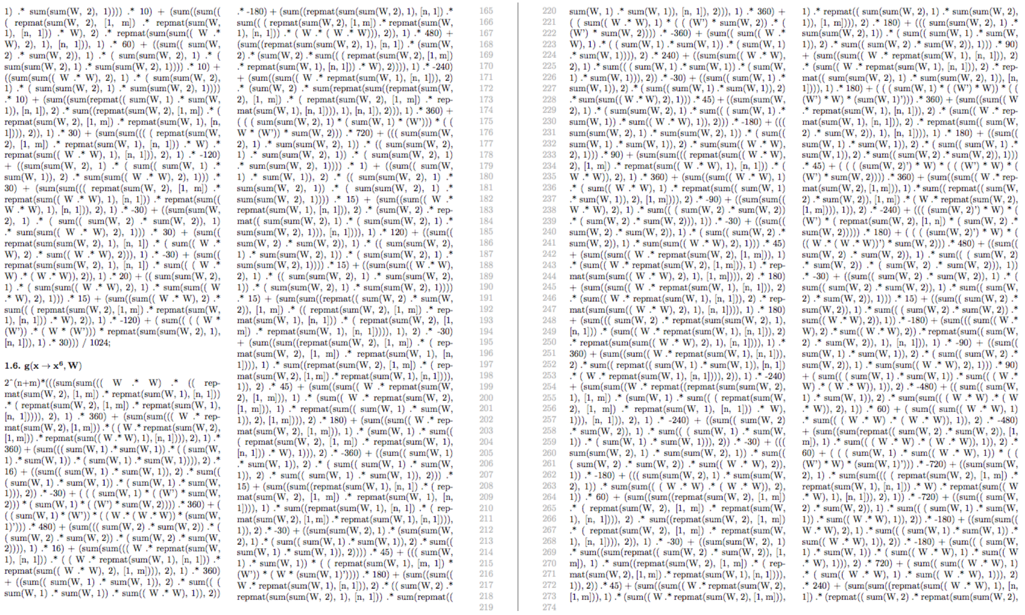
this is a polynomial computation vs exponential computation in the naive algorithm

Example: Taylor Series Approximation

6th order:

Our framework can find it.

O(n^3)



Representing Symbolic Expressions

- Pure symbolic too slow
- Use numerical representation
  - Pick P random numbers (P large) for each element of each variable
  - So for an r x c matrix, we have P copies, each filled with random numbers
- Important detail: we use fixed r and c
  - No definitive guarantee for other dimensions

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 real values is unstable, so use integers modulo large prime.

Overview

- Representation of math expressions
- Searching over expressions
- Distributed representation of expressions using a tree neural network

Combinatorial Explosion

- Polynomials of degree 1:

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

- Polynomials of degree 2:

$$A^2, (A^2)^T, AA^T, A^T A, \sum_i (AA^T)_{i,:}, \sum_{i,j} (AA^T)_{i,j}, \sum_i A^2_{i,:}, \sum_j A^2_{:,j}, (\sum_{i,j} A)^2, \ldots$$

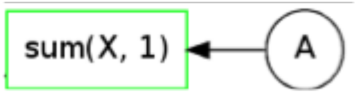
Prior Over Computation Trees

- 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

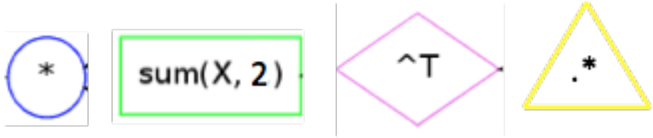
Want to learn a good prior over expressions

Searching over Computation Trees

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

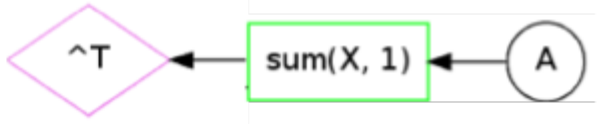
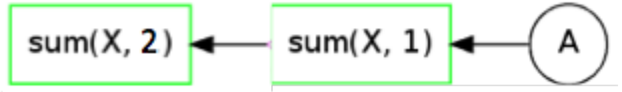
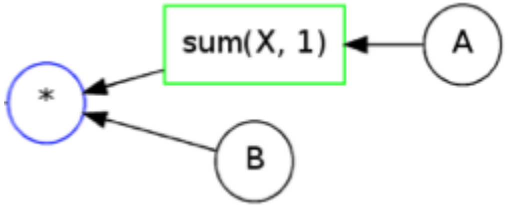


  - Valid operators to append:



Searching over Computation Trees

- 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

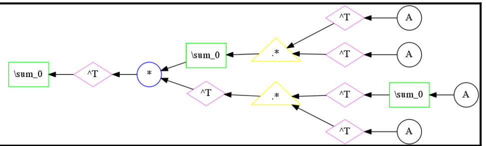
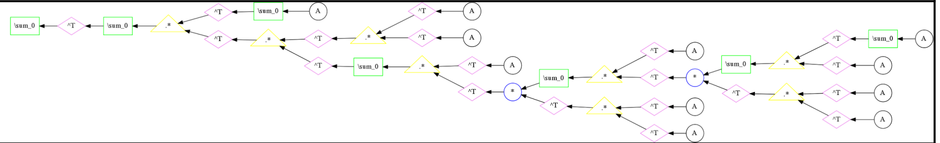
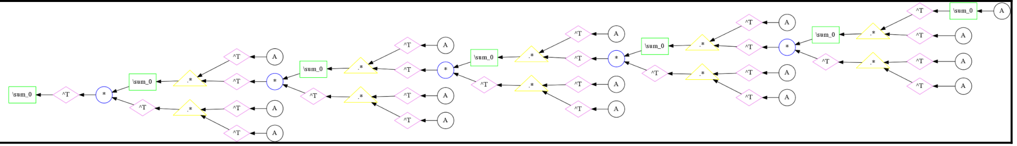
- Naïve:
    - no prior Just select randomly from all valid operators

- n-gram prior

- Tree Neural Network prior

Prior learning

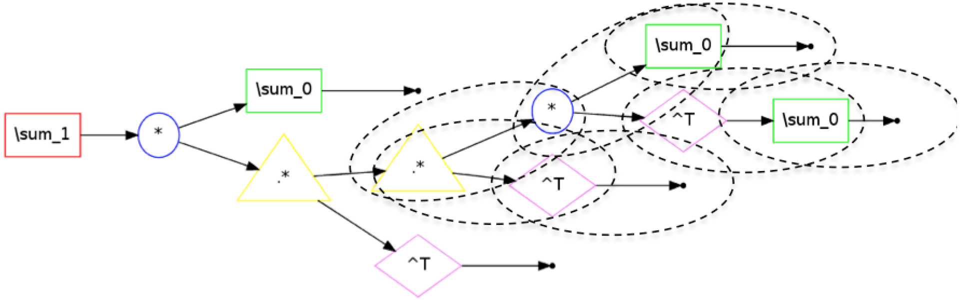
- Use curriculum learning approach
- Start with easy targets (low polynomial degree k)

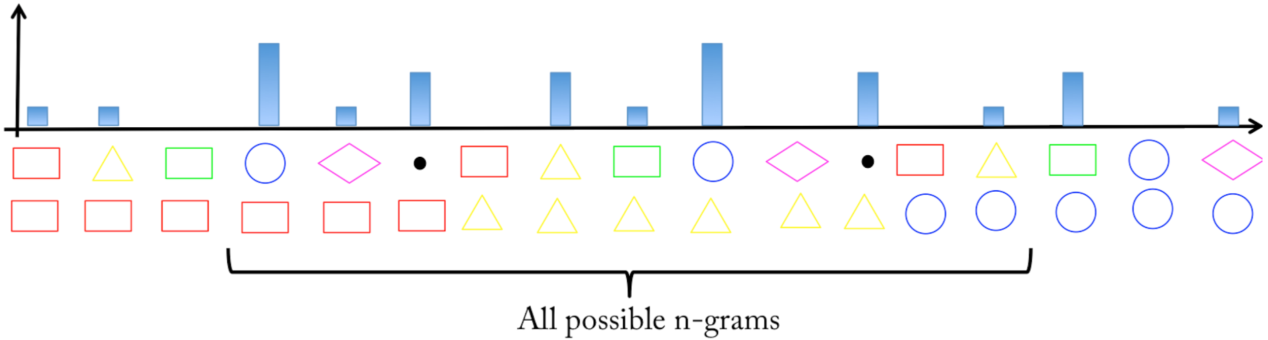
k = 3	
k = 8	
k = 11	

- Build prior from these simple solutions
- Apply to harder target (next degree k)

Building N-gram Prior

- Break solutions into n-grams
- Prior is histogram of grams:

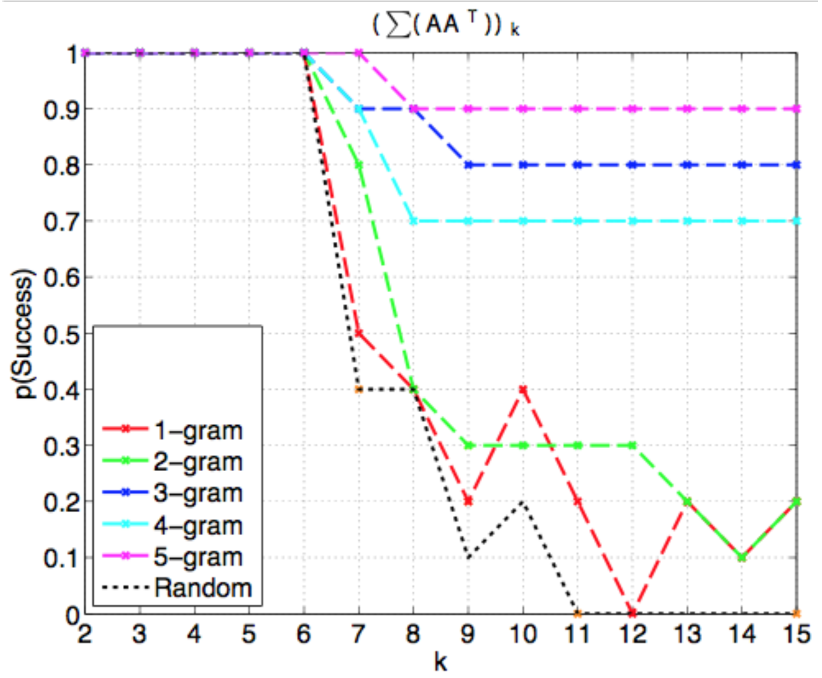




Experiments

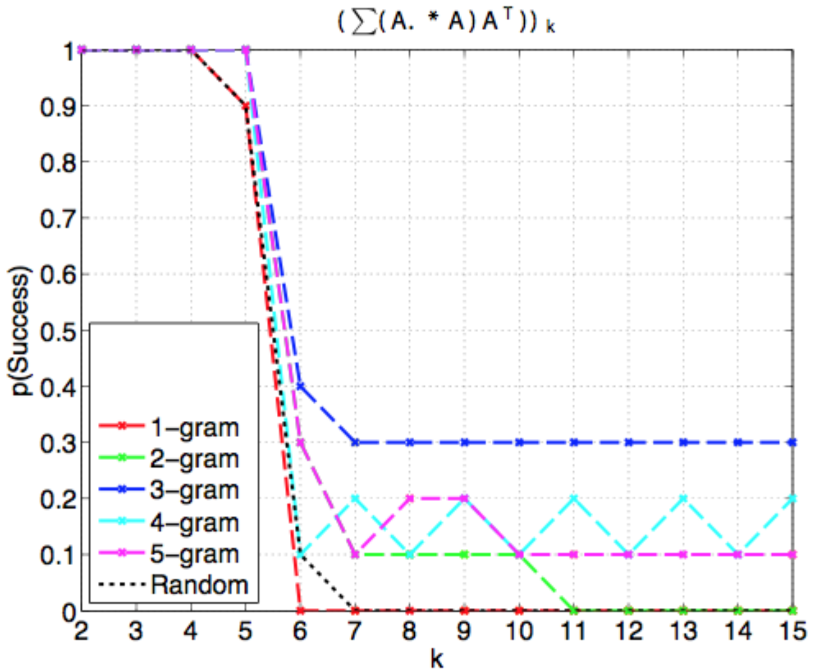
- 5 families of expressions (vary degree k)
  - Multiply-sum: $(\sum \mathbf{A}\mathbf{A}^{\mathbf{T}})_{\mathbf{k}}$
  - Element-wise multiply-sum: $(\sum(\mathbf{A}.*\mathbf{A})\mathbf{A}^{\mathbf{T}})_k$
  - Symmetric polynomials: $\sum_{i<j<k} A_i A_j A_k$
  - RBM-1: $\sum_{v\in\{0,1\}^n}(v^T A)^k$
  - RBM-2: $\sum_{v\in\{0,1\}^n, h\in\{0,1\}^n}(v^T A h)^k$

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



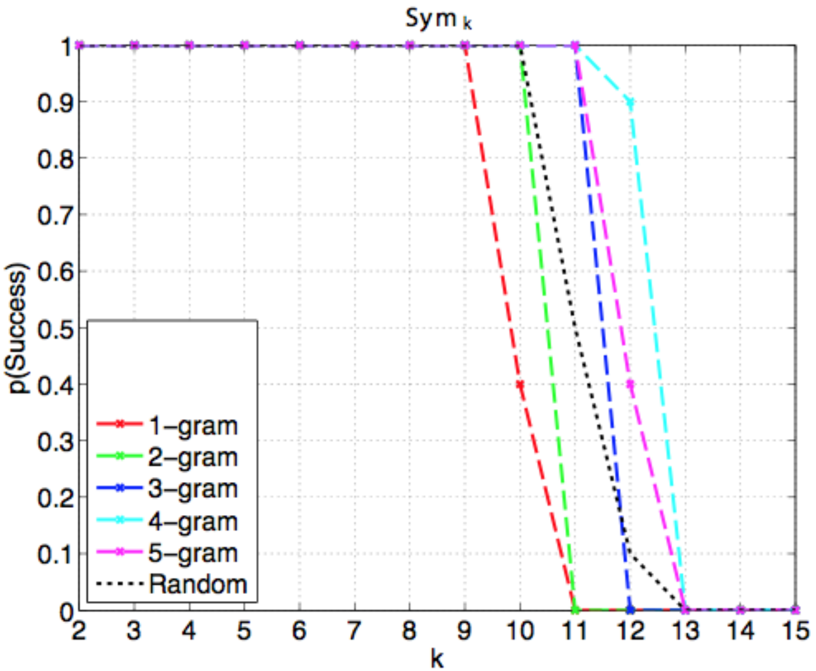
$$\sum_{i,j}(A * A^T),\ \ \sum_{i,j}(A * A^T) * A,\ \ \sum_{i,j}(A * A^T)^2,\ \ \sum_{i,j}(A * A^T)^2 * A, \ldots$$

```
K=2:  sum((A * ((sum(A, 1)) ')) , 1);
K=5:  sum((A * ((((A * (((sum((A'), 1)) * A)'))') * A)')), 1)
K=9:  sum((A * ((((A * ((((A * ((((A * (((sum((A'), 1)) * A)'))') * A)'))') * A)'))') * A)')), 1))
K=14: sum((A * ((((A * ((((A * ((((A * ((((A * ((((A * ((((A * ((sum(A, 1)) ')) ') * A) ')) ') * A) ')) ') * A)'))') *
A)'))') * A)'))') * A)')), 1)
```



$$\sum_{i,j}(A.*A)*A^T,\ \ \sum_{i,j}(A.*A)*A^T*(A.*A),\ \ \sum_{i,j}((A.*A)*A^T)^2,\dots$$

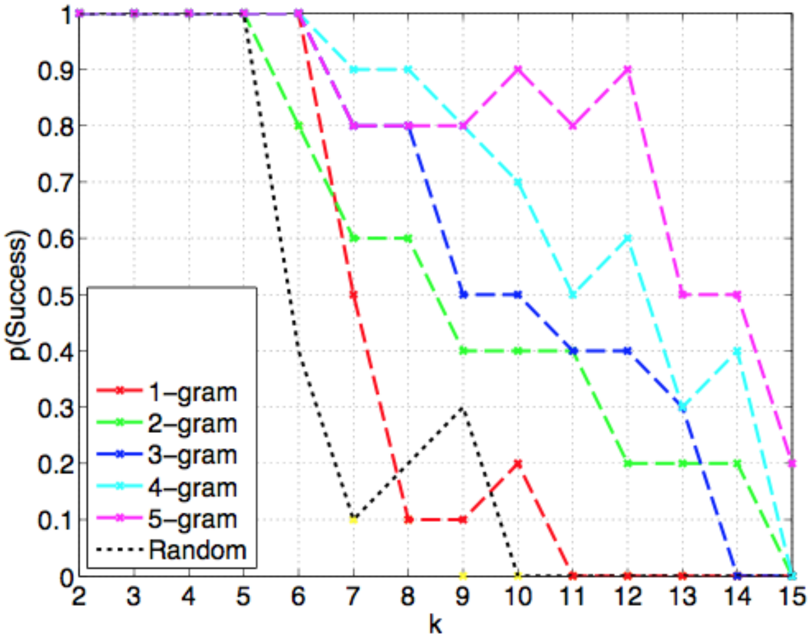
```
K=2:  sum(((sum(A, 1)) .* (sum(A, 1))) , 2)
K=3:  sum((sum(((repmat((sum((repmat((sum(A, 1)), n, 1) *. A), 2)), 1, m) .* A) .* A), 2)), 1)
K=4:  sum((sum((repmat((sum((repmat((sum(((repmat((sum(A, 1)), n, 1) *. A) .* A), 2)), 1, m)
     .* A)  , 1)), n, 1) .* A), 2)), 1)
```



$$\sum_{i<j} A_i A_j, \ \sum_{i<j<k} A_i A_j A_k, \ \sum_{i<j<k<l} A_i A_j A_k A_l, \ldots$$

```
K=2: (1 / 2) * (((sum(A, 2)) * (sum(A, 2)))) + (50 / -100)* ((A * (A')));
K=3: (1 / 6) * ((sum(((sum(A, 2)) * ((sum(A, 2)) * A)) , 2))) + (50 / -100)* ((A * (((sum(A, 2))
      * A) ') )) + (1 / 3)* (((A .* A) * (A')));
K=4: (25 / -100) * ((A * (((sum(A, 2)) * ((sum(A, 2)) * A))')))  + (1 / 8)* ((A * ((A * ((A') * A))
      ')))  + (1 / 3)* (((A * ((A') .* (A'))) * (sum(A, 2)))) + (25 / -100)* (((((A') .* (A'))') *
      ((A') .* (A')))) + (1 / 24)* ((sum(((sum(((sum(A, 2))* ((sum(A, 2)) * A)), 2)) * A), 2)));
```

$$\sum_{v\in\{0,1\}^n}(v^T A)^k$$




```
K=2: 2^(n - 3) * (2 * ((sum(((A .* A) ') , 1))) + 2 * ((sum(((A') .* repmat((sum((A') , 1)) , m, 1)) , 1))))
K=3: 2^(n - 4) * (6 * ((sum((((((A') .* repmat((sum((A') , 1)) , m, 1)) ') * A) ') , 1))) + 2 * ((sum(((A') .* repmat
((sum(((A') .* repmat((sum((A') , 1)) , m, 1)) , 1)) , m, 1)) , 1))));
K=4: 2^(n - 5) * (12 * (((sum(((A .* A) ') , 1)) .* (sum(((A') .* repmat((sum((A') , 1)) , m, 1)) , 1)))) + 6* (((sum
(((A .* A) ') , 1)) .* (sum(((A .* A) ') , 1)))) + 2 * ((sum(((A') .* repmat((sum(((A') .*
 repmat((sum(((A') .* repmat((sum((A'), 1)), m, 1)), 1)), m, 1)), 1)), m, 1)), 1))) +*4 ((sum
 (((((((A .* A)') .* (A'))') .* A)'), 1))));
```

RBM-2

- No scorer strategy able to get beyond k=5
  - However, the k = 5 solution was found by the TNN consistently faster than the random strategy (100 ± 12 vs 438 ± 77 secs).
- Hypothetically, RBM-2 doesn’t have many repetitive structures.

K=5:
```
2^(n+m)*(((sum(sum((repmat(( sum(A, 1) .* sum(A, 1)), [n, 1]) .* ( ( repmat(sum(A, 2), [1, m]) .* repmat(sum(A, 1), [n, 1])) .* A)), 2), 1) .* -40) + (sum((( sum(A, 1)
.* sum(A, 1)) .* sum((repmat(sum(A, 1), [n, 1]) .* ( repmat(sum(A, 2), [1, m]) .* repmat(sum(A, 1) , [n, 1]))) , 1)) , 2) *. -10) + (( sum(sum(A, 2), 1) .* sum(( ( sum(A, 2) .
* sum(A, 2)) .* sum(( A .* A), 2)), + (sum(( repmat (sum(sum(A, 2) , 1) , [n , 1])*.( repmat (sum(sum(A, 2) , 1) , [n , 1]) repmat(sum(A, 2) , [1, m])* .repmat(sum(A, 1) ,
[n, 1])) .* A) , 2))) , 1) + (sum((sum(( A .* repmat(sum(A, 1) , [n, 1])) , 2) .* sum((repmat(sum(A, 1) , [n, 1)) .* -60) .* sum(( (.* 60) ... 1]) .* (repmat(sum(A, 2) , [1,
m]) .* repmat(sum(A, 1) , [n, 1]))) , 2)) , 1) *. 60) + sum(sum( ( ( repmat(sum(A, 2) , [1, m]) *. repmat(sum(A, 1) , [n, 1])) .* ( A .* ( A .* A))) , 2) , 1) .* 80) + (sum
((sum(A, 2) .* (sum(A, 2) .* sum(( ( repmat(sum(A, 2) , [1, m]) .* repmat(sum(A, 1) , [n, 1])) .* A), 2))), 1) .* -40)+ (sum((repmat(sum(( sum(A, 2) .* sum(A, 2)), 1), [n, 1])
.* sum(( ( repmat(sum(A, 2), [1, m]) .* repmat(sum(A, 1), [n, 1])) .* A), 2)), 1) .* 60) + ( ( ( sum(A, 1)* (A’)) * ( ( A * (A’)) * sum(A, 2))) .* 120) + (sum((( sum(A, 2) .*
sum(A, 2)) .* sum((repmat(sum(A, 2), [1, m]) .* ( repmat(sum(A, 2), [1, m]) .* repmat(sum(A, 1) , [n, 1]))) , 2)) , 1) *. -10) + (( sum(sum(A, 2), 1) .* sum(( ( sum(A, 1) .*
sum(A, 1)) .* sum(( A .* A), 1)), 2)) .* -60) + (sum((sum(repmat(( sum(A, 2) .* sum(A, 2)), [1, m]), 1) .* sum((repmat(sum(A, 2), [1, m]) .* ( repmat(sum(A, 2) , [1, m])* .
repmat(sum(A, 1) , [n, 1]))) , 1)) , 2) *. 15) + ( ( ( sum(sum(A, .......
```

Overview

- Representation of math expressions
- Searching over expressions
- Distributed representation of expressions using a tree neural network

Recursive nets why ?

- N-gram can one have a shallow understanding (limited by N).
- Looking for model that can comprehend entire computation tree regardless of its depth.

TNN Pre-Training

RNN $\phi_W(\mathcal{S}) = x$ maps expression S to vector x

- Two examples:

$$\phi(A^T) = x_1 \qquad\qquad \phi((A\,.*\,A)^T) = x_2$$

- But want RNN to "understand" math, i.e.:

$$\phi(((A^T)^T)^T) \approx x_1 \qquad\qquad \phi(A^T.*A^T) \approx x_2$$

TNN Pre-Training

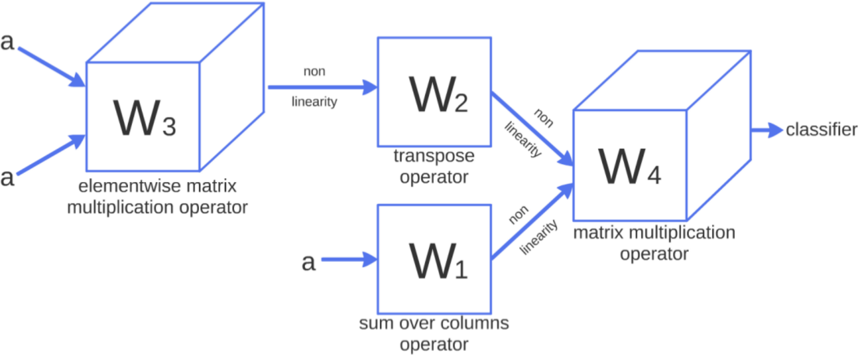
- Train on equivalent mathematical expressions.
- Goal: make it understand entire computation tree.

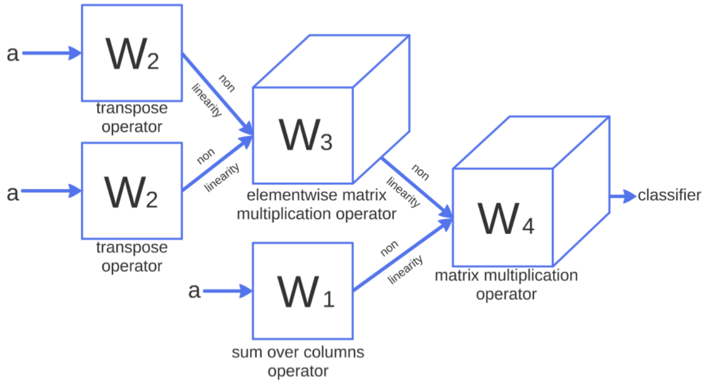
```
(((sum((sum((A * (A')), 1)), 2)) * ((A * (((sum((A'), 1)) * A)'))')) * A)
(sum(((sum((A * (A')), 2)) * ((sum((A'), 1)) * (A * ((A') * A)))), 1))
(((sum(A, 1)) * (((sum(A, 2)) * (sum(A, 1)))')) * (A * ((A') * A)))
((((sum((sum((A * (A')), 1)), 2)) * ((sum((A'), 1)) * (A * ((A') * A))))')')
((sum(A,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





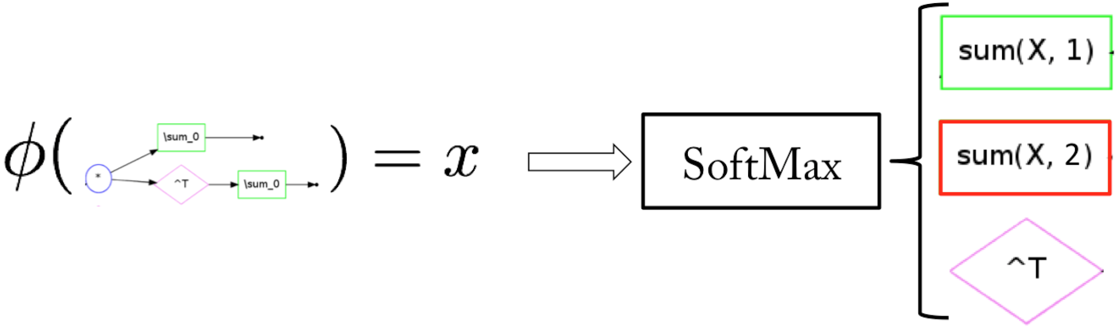
TNN Pre-Training Results

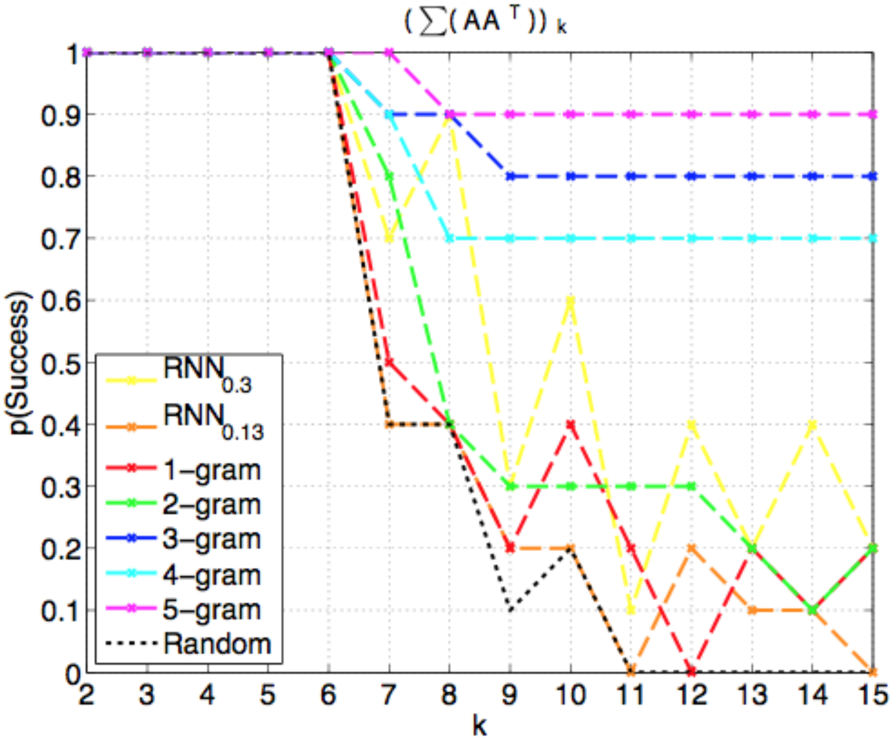
	Degree $k = 3$	Degree $k = 4$	Degree $k = 5$	Degree $k = 6$
Test accuracy	$100\% \pm 0\%$	$96.9\% \pm 1.5\%$	$94.7\% \pm 1.0\%$	$95.3\% \pm 0.7\%$
Number of classes	12	125	970	1687
Number of expressions	126	1520	13038	24210

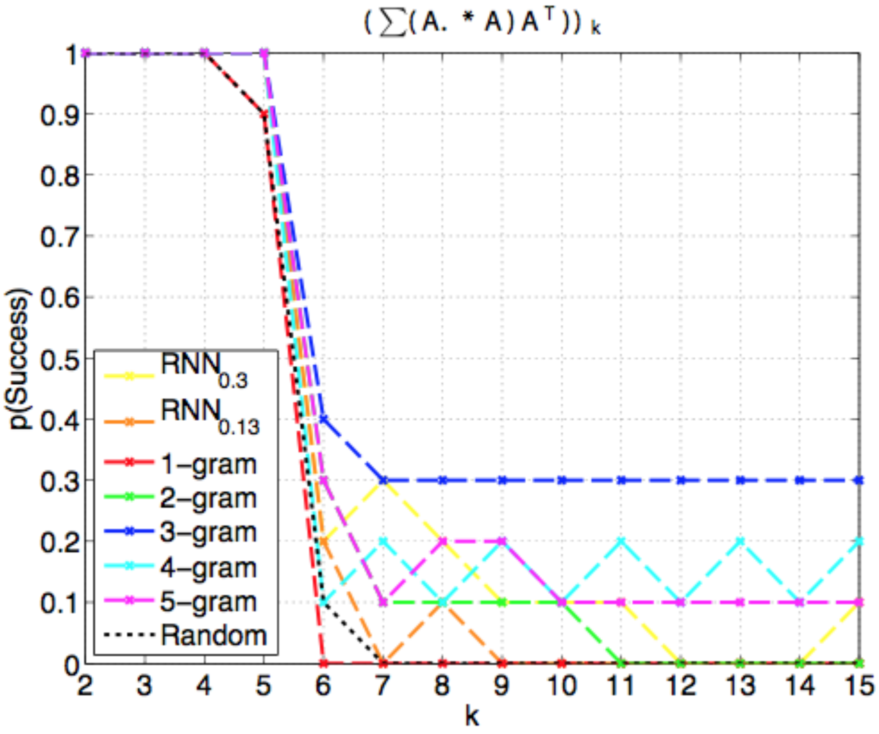
Note: no explicit knowledge of math operators

Building Prior from TNN

- Take solutions from lower degrees within family
- Pass each part through pre-trained TNN

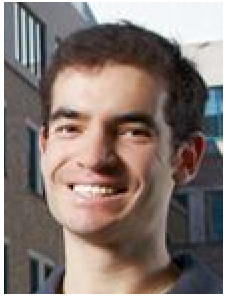



$(\sum(AA^T))_k$
p(Success)
k
RNN$_{0.3}$
RNN$_{0.13}$
1–gram
2–gram
3–gram
4–gram
5–gram
Random


$(\sum(A.*A)A^T))_k$
p(Success)
k
RNN$_{0.3}$
RNN$_{0.13}$
1–gram
2–gram
3–gram
4–gram
5–gram
Random

Thanks to my collaborators

Ilya Sutskever, Karol Kurach, and Rob Fergus

Q&A

- Learning Atari games
- Predicting program execution results
- RNN with LSTMs
- Scheduling strategies (baseline, naive, mix, combined)
- Learning mathematical identities
- Representation of mathematical identities.

Paper: Learning to Execute (arxiv)

https://github.com/wojciechz/learning_to_execute

Paper: Learning to Discover Efficient Mathematical Identities (NIPS 2014 spotlight)

https://github.com/kkurach/math_learning

I am happy to answer any questions.