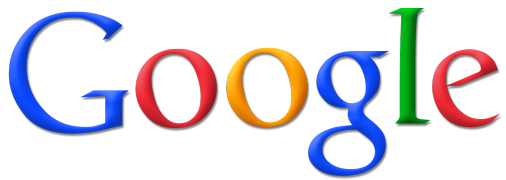
Learning to Discover Efficient Mathematical Identities

by **Wojciech Zaremba**,
Karol Kurach, and Rob Fergus
ref: http://arxiv.org/abs/1406.1584



A toy example

Let’s consider two matrices A, B

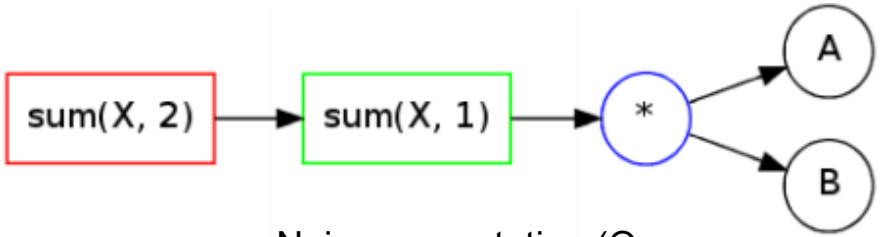
$$\sum_{i,k}(AB)_{i,k} = \sum_i \sum_j \sum_k a_{i,j} b_{j,k}$$

A toy example

Let's consider two matrices A, 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).



Naive computation (O (n^3) time)

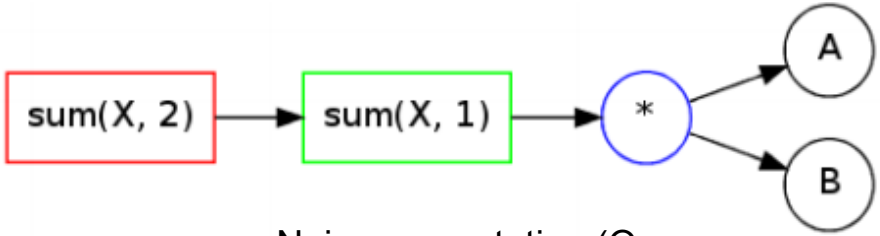
A toy example

Let's consider two matrices A, B

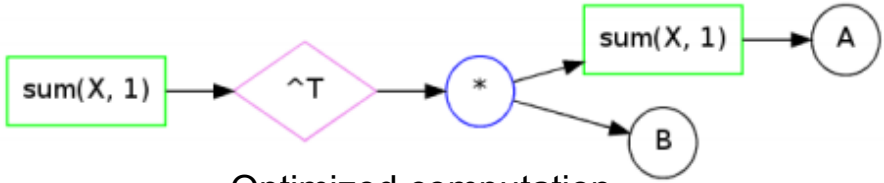
$$\sum_{i,k}(AB)_{i,k} = \sum_i \sum_j \sum_k a_{i,j} b_{j,k}$$

Naive computation takes O(n^3).

Our framework found O(n^2) computation



Naive computation (O(n^3) time)



Optimized computation (O(n^2) time)

Overview

- How to represent computation
- How to search over computations
- Distributed representation of computation

Computation encoding A*B

$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} = \begin{bmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & a_{1,1}b_{1,2} + a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} + a_{2,2}b_{2,1} & a_{2,1}b_{1,2} + a_{2,2}b_{2,2} \end{bmatrix} = \left\langle \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \end{bmatrix}, \begin{bmatrix} a_{1,1}b_{1,1} \\ a_{1,2}b_{2,1} \\ a_{1,1}b_{1,2} \\ a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} \\ a_{2,2}b_{2,1} \\ a_{2,1}b_{1,2} \\ a_{2,2}b_{2,2} \end{bmatrix} \right\rangle$$

Symbolic representation A*B based on monomials

Computation encoding sum(A*B). Takes O(n^3) time.

$$\sum\left(\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix}\right) = \left\langle \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} a_{1,1}b_{1,1} \\ a_{1,2}b_{2,1} \\ a_{1,1}b_{1,2} \\ a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} \\ a_{2,2}b_{2,1} \\ a_{2,1}b_{1,2} \\ a_{2,2}b_{2,2} \end{bmatrix} \right\rangle$$

Symbolic representation sum(A*B) based on monomials

Computation encoding sum(sum(A, 1)*B). Takes O(n^2) time

$$\sum \left(\begin{bmatrix} a_{1,1} + a_{2,1} & a_{1,2} + a_{2,2} \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \right) = \left\langle \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} a_{1,1}b_{1,1} \\ a_{1,2}b_{2,1} \\ a_{1,1}b_{1,2} \\ a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} \\ a_{2,2}b_{2,1} \\ a_{2,1}b_{1,2} \\ a_{2,2}b_{2,2} \end{bmatrix} \right\rangle$$

Symbolic representation sum(sum(A, 1)*B) based on monomials

Allowed computations

Grammar rules :

- matrix multiplication
- elementwise multiplication
- transposition
- sum, over columns and rows
- addition, multiplication by constant
- we can consider arbitrary bigger grammar ….

e.g. : (((sum((sum((A * (A')), 1)), 2)) * ((A * (((sum((A'), 1)) * A)'))')) * A)

Many computations are in this family

- E.g. finite Taylor expansion of any function ….

Many computations are in this family

- E.g. finite Taylor expansion of any function ….

for instance, partition function of Restricted Boltzmann Machine (RBM)

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

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

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

Exact solution for k=1 (first term in Taylor series)

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

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

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

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

Exact solution for k=2 (second term in Taylor series)

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

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

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

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

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

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

How to find equivalent computations ?

How to find equivalent computations ?

Manual methods fail
(I have spend half a year on it).

Exact solution for k=6 (sneak preview) derived by our framework

Maybe machines should be searching for patterns in computation

Overview

- How to represent computation
- How to search over computations
- Distributed representation of computation

Explosion of computation space

Polynomials of degree one in a matrix A:

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

Polynomials of degree two :

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

Space grows super-exponentially fast.

Prior over computation trees

- Explore space of computation efficiently
- Find equivalent expressions to the target one
  - But using operations with lower complexity

- Want to learn prior over sensible computations
  - Humans learn prior over proofs in mathematics

Searching over computation trees

Scheduler picks potential new expressions to append to current expressions

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

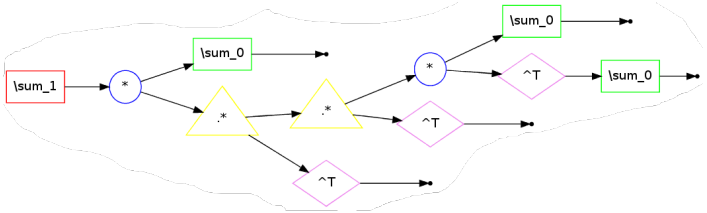
We want to learn a good scorer.

Scoring strategies

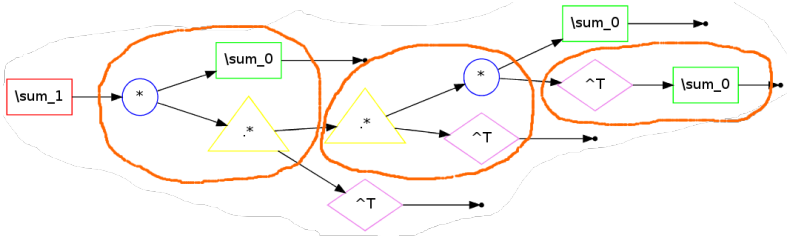
- naive scorer don’t use any prior. All computations are equally probable
- n-gram models
- learnt scorer (little bit about it at the end)

n-gram prior over trees

Exemplary intermediate solution:



Bi-grams:



Build n-grams distribution from solutions of simpler expressions

- Patterns that worked before might be useful

Experiments:
5 families of related problems

- $(\sum \mathbf{A}\mathbf{A}^{\mathbf{T}})_{\mathbf{k}}$
- $(\sum(\mathbf{A}. * \mathbf{A})\mathbf{A}^{\mathbf{T}})_k$
- Symmetric polynomials, e.g. $\sum_{i<j<k} A_i A_j A_k$
- RBM-1 $\sum_{v\in\{0,1\}^n}(v^T A)^k$
- RBM-2 $\sum_{v\in\{0,1\}^n, h\in\{0,1\}^n}(v^T A h)^k$

Family sum(AA^T)_k

Targets → Exemplary solution:

- sum(A*A’) → (sum(((sum(A, 1)) .* (sum(A, 1))), 2))
- sum(A*A’*A) → (sum((sum((A .* (sum(((sum(A, 2)) .* A), 1))), 2)), 1))
- sum(A*A’*A*A’) → (sum((sum((A .* (sum(((sum((A .* (sum(A, 1))), 2)) .* A), 1))), 2)), 1))
- sum(A*A’*A*A’*A) → (sum((sum((A .* (sum(((sum((A .* (sum(((sum(A, 2)) .* A), 1))), 2)) .* A), 1))), 2)), 1))
- …

	naive	1-gram	2-gram	3-gram	4-gram	5-gram
Hardest possible example to solve	10	11	>15	>15	>15	>15

Family (RBM-1)_k

Targets → Exemplary solution:

- $\sum_{v \in \{0,1\}^n} \langle v, A \rangle$ → 16.0 * (sum((sum(A, 2)), 1))
- $\sum_{v \in \{0,1\}^n} \langle v, A \rangle^2$ → 8.0 * (sum(((sum(A, 1)) .* (sum(A, 1))), 2)) + 8.0 * ((sum((sum(A, 1)), 2)) .* (sum((sum(A, 1)), 2)))
- $\sum_{v \in \{0,1\}^n} \langle v, A \rangle^3$ → 12.0 * (sum((sum((A .* (sum(((sum((sum(A, 2)), 1)) .* A), 1))), 2)), 1)) + 4.0 * (sum(((sum((sum(A, 2)), 1)) .* ((sum((sum(A, 2)), 1)) .* (sum(A, 2)))), 1))

	naive	1-gram	2-gram	3-gram	4-gram	5-gram
Hardest possible example to solve	9	10	14	13	14	>15

Overview

- How to represent computation
- How to search over computations
- Distributed representation of computation

The meaning of a ~~word~~ computation is described by the ~~words~~ computations accompanying it

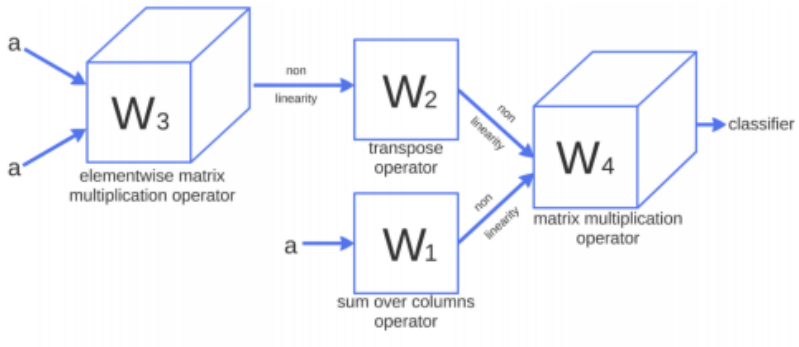
How we can represent a computation?

- Vector representation for every computation
  - e.g. A^T = vector_1 , \sum(A^T, 1) = vector_2,

- Want to learn how to compose their vector representations
  - i.e. ((A^T)^T)^T ~ vector_1, \sum(A, 2)^T ~ vector_2

Learnt representation with neural net

Recursive Neural network → RNN

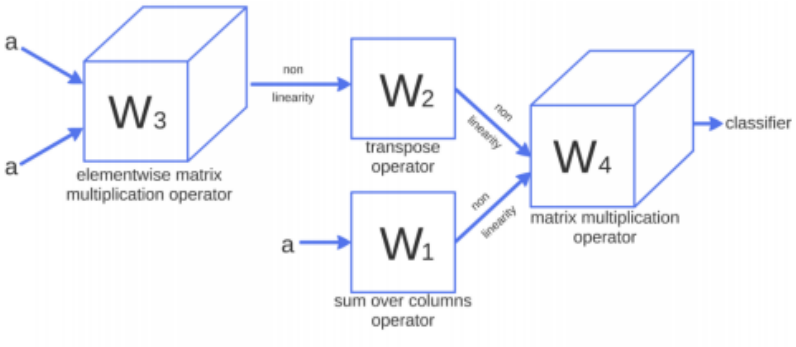


(a) $(A. * A)' * \text{sum}(A, 2)$

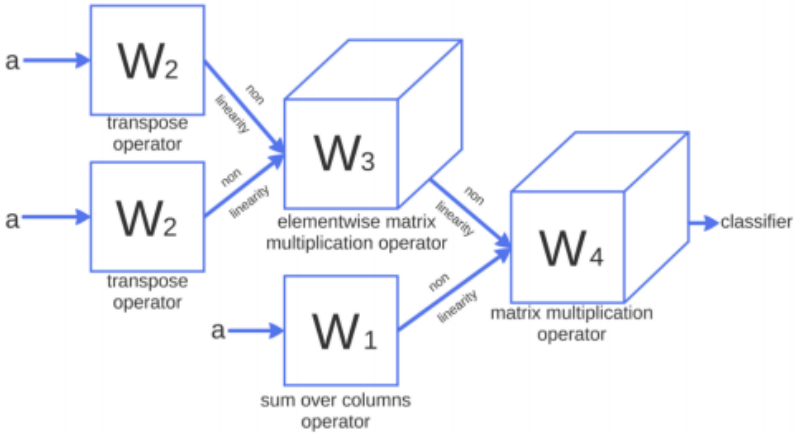
No understanding of underlying mathematical operators (no grounding)

Learnt representation with RNN

Recursive Neural network → RNN



(a) $(A. * A)' * \text{sum}(A, 2)$



(b) $(A'. * A') * \text{sum}(A, 2)$

No understanding of underlying mathematical operators (no grounding)

Task - classify expressions

Example from A class:

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

Example from B class:

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

From which class is this example ?

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

Performance - expression classification

```
(((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 = 6
Test accuracy	100%	95.3%
Number of classes	12	1687

Learnt representation - tricks

- Initialization as identity + noise (critical)
- ReLU (previously people used tanh)
- Curriculum learning
- Prediction matrix has x100 learning rate
- We update initial random vector of symbol

RNNs for a better discovery learning

- We have a real vector representation for any computations

- We use a linear classifier on such representation to train scorer

Family sum(AA^T)_k with RNN

RNN gives more diversified solutions (doesn't just copy them), but it doesn't perform as good as n-gram.

Targets → Exemplary solution of RNN:

- sum(A*A') → (sum((A * ((sum(A, 1))')), 1))
- sum(A*A'*A) → ((sum(A, 1)) * ((A') * (sum(A, 2))))
- sum(A*A'*A*A') → ((((sum(A, 1)) * (A')) * A) * ((sum(A, 1))'))
- sum(A*A'*A*A'*A) → ((sum(A, 1)) * ((A') * (A * ((A') * (sum(A, 2))))))

	naive	5-gram	RNN
Hardest possible example to solve	10	>15	~15

Summary

- Simple statistical priors over computations like n-gram allows the discovery of many new math formulae

- Use neural nets to map computational expressions to continuous vectors
  - Also use for formulae discovery

Future work

- Computations = Knowledge representation = Mathematical proofs = Programs = etc.
  - predictions on programs / program induction
  - explore space of mathematical proofs

- Replace recursive neural network with recurrent ?

Q&A

- How to represent computation
  - symbolic representation
- How to search over computations
- Distributed representation of computation
  - recursive networks
  - training tricks

Thank you. I am happy to take any question.