Tensors

and their applications

Ivan Oseledets, work by Gleb Ryzhakov, Andrey Chertkov, Konstantin Sozykin, Roman Schutsky, Andrej Cichocki, Anh-Huy Phan

Skoltech

The task of this part

Our main result is the algorithm of building exact Tensor Train (TT) representation to the multidimensional function knowing its analytical expression.
Existing methods:

- Constructive TT for a restricted number of functions;
- Cross approximation;
- Alternation Least Squares (ALS) method for TT-approximation;
- Gradient Decent methods *à la* machine learning;
- etc.

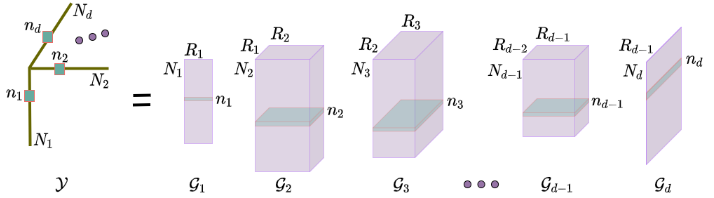
Most of them require many iterations and they are approximate methods, well suited for black box.

Background
Notations

$$\mathcal{K}(i_1, i_2, \ldots, i_d) = \mathcal{G}_1(i_1)\mathcal{G}_2(i_2)\cdots\mathcal{G}_d(i_d) =$$

$$\sum_{\alpha_1=1}^{R_1}\sum_{\alpha_2=1}^{R_2}\cdots\sum_{\alpha_d=1}^{R_{d-1}} \mathcal{G}_1(1, i_1, \alpha_1)\mathcal{G}_2(\alpha_1, i_2, \alpha_2)\cdots\mathcal{G}_d(\alpha_{d-1}, i_d, 1).$$

By $\{R_i\}_{i=0}^{d}$ we denote *ranks* of the TT-decomposition (we let $R_0 = R_d = 1$), and $\mathcal{G}_i \in \mathbb{C}^{R_{i-1}\times N_i \times R_i}$ are TT-*cores*.



Background

constructive TT

In [1] the cores of the TT decomposition for functions H of the following form were explicitly constructed

$$H(x_1, x_2, \dots, x_d) = F(x_1 + x_2 + \dots + x_d)$$

for several functions F. The correspondence between the tensor integer indices and the complex values of x is as follows:
$\mathcal{K}(i_1, i_2, \dots, i_d) = H(x_1[i_1], x_2[i_2], \dots, x_d[i_d])$.
In this case, all the cores are of the same form:

$$\mathcal{G}_k(:, i_k, :) = \mathcal{G}(x_k[i_k]), \quad \text{with } \mathcal{G}(x) \cdot \mathcal{G}(y) = \mathcal{G}(x+y),$$

where $\mathcal{G}(x)$ is some matrix function. But we multiply cores in TT, how to make sum from a product?

[1] I. V. Oseledets. Constructive representation of functions in low-rank tensor formats. *Constructive Approximation*, 37(1):1–18, December 2012.

Background

constructive TT

$$\exp(x_1 A)\cdot\exp(x_2 A)\cdot\exp(x_n A) = \exp\big((x_1 + x_2 + \cdots x_n)A\big).$$

$$\exp\left(x\begin{pmatrix}0 & 1\\ 0 & 0\end{pmatrix}\right) = \begin{pmatrix}1 & x\\ 0 & 1\end{pmatrix} \iff F = x_1 + x_2 + \cdots + x_d,$$

$$\exp\left(x\begin{pmatrix}0 & -1\\ 1 & 0\end{pmatrix}\right) = \begin{pmatrix}\cos(x) & -\sin(x)\\ \sin(x) & \cos(x)\end{pmatrix} \iff F = \sin(x_1 + \cdots x_d),\ (\text{or}\ \cos),$$

$$\exp\left(x\begin{pmatrix}0 & 1 & 0\\ 0 & 0 & 2\\ 0 & 0 & 0\end{pmatrix}\right) = \begin{pmatrix}1 & x & x^2\\ 0 & 1 & 2x\\ 0 & 0 & 1\end{pmatrix} \iff F = (x_1 + \cdots x_d)^2,$$

etc.

Background

constructive TT

$$\exp(x_1 A) \cdot \exp(x_2 A) \cdot \exp(x_n A) = \exp\big((x_1 + x_2 + \cdots x_n) A\big).$$

$$\exp\left(x \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}\right) = \begin{pmatrix} 1 & x \\ 0 & 1 \end{pmatrix} \iff F = x_1 + x_2 + \cdots + x_d,$$

$$\exp\left(x \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}\right) = \begin{pmatrix} \cos(x) & -\sin(x) \\ \sin(x) & \cos(x) \end{pmatrix} \iff F = \sin(x_1 + \cdots x_d),\ (\text{or } \cos),$$

$$\exp\left(x \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{pmatrix}\right) = \begin{pmatrix} 1 & x & x^2 \\ 0 & 1 & 2x \\ 0 & 0 & 1 \end{pmatrix} \iff F = (x_1 + \cdots x_d)^2,$$

etc.

Background

constructive TT

$$\exp(x_1 A)\cdot\exp(x_2 A)\cdot\exp(x_n A) = \exp\big((x_1 + x_2 + \cdots x_n)A\big).$$

$$\exp\left(x\begin{pmatrix}0 & 1\\ 0 & 0\end{pmatrix}\right) = \begin{pmatrix}1 & x\\ 0 & 1\end{pmatrix} \iff F = x_1 + x_2 + \cdots + x_d,$$

$$\exp\left(x\begin{pmatrix}0 & -1\\ 1 & 0\end{pmatrix}\right) = \begin{pmatrix}\cos(x) & -\sin(x)\\ \sin(x) & \cos(x)\end{pmatrix} \iff F = \sin(x_1 + \cdots x_d),\ (\text{or}\ \cos),$$

$$\exp\left(x\begin{pmatrix}0 & 1 & 0\\ 0 & 0 & 2\\ 0 & 0 & 0\end{pmatrix}\right) = \begin{pmatrix}1 & x & x^2\\ 0 & 1 & 2x\\ 0 & 0 & 1\end{pmatrix} \iff F = (x_1 + \cdots x_d)^2,$$

etc.

Background

constructive TT

$$\exp(x_1 A) \cdot \exp(x_2 A) \cdot \exp(x_n A) = \exp\big((x_1 + x_2 + \cdots x_n)A\big).$$

$$\exp\left(x\begin{pmatrix}0 & 1\\ 0 & 0\end{pmatrix}\right) = \begin{pmatrix}1 & x\\ 0 & 1\end{pmatrix} \iff F = x_1 + x_2 + \cdots + x_d,$$

$$\exp\left(x\begin{pmatrix}0 & -1\\ 1 & 0\end{pmatrix}\right) = \begin{pmatrix}\cos(x) & -\sin(x)\\ \sin(x) & \cos(x)\end{pmatrix} \iff F = \sin(x_1 + \cdots x_d),\ (\text{or } \cos),$$

$$\exp\left(x\begin{pmatrix}0 & 1 & 0\\ 0 & 0 & 2\\ 0 & 0 & 0\end{pmatrix}\right) = \begin{pmatrix}1 & x & x^2\\ 0 & 1 & 2x\\ 0 & 0 & 1\end{pmatrix} \iff F = (x_1 + \cdots x_d)^2,$$

etc.

Main result

Is it possible to obtain the cores of the TT decomposition directly, given *any* analytical dependence of tensor values on its indices?

Yes, if this dependency is given as a sequence of functions of two argument, each of which has as its argument the output of the previous function and the current index (and if the images of these functions are not large).

Functional dependence in the form of a sequence of pairwise functions may seem rather narrow, however, this assignment covers quite a large range of functional dependencies of tensor value on its indices if such a set of functions is chosen skillfully.

Main result

Pair-wise functions

To define these pairwise functions, recall what a Reverse Polish Notation is. In this notation, the function arguments are written first, followed by the functions themselves. There are no parentheses or other separating characters.
For example, the expression

$$\sin(x \cdot y) + \ln 5$$

in this notation looks like

$$x \; y * \sin \; 5 \; \ln \; +$$

Main result

Pair-wise functions

In Reverse Polish Notation our condition on the type of analytical dependence is as follows

$$\mathcal{K}(i_1,\, i_2,\, \ldots,\, i_d) = \\ (0 i_1 f^{(1)} i_2 f^{(2)} \cdots i_{l-1} f^{(l-1)})(0 i_d f^{(d)} i_{d-1} f^{(d-1)} \cdots i_{l+1} f^{(l+1)}) i_l f^{(l)}.$$

In other words, One, from to the left of the right, is as follows

$$a_1(i_1) = f^{(1)}(i_1,\, 0), \quad a_2(i_1,\, i_2) = f^{(2)}(i_2,\, a_1), \quad a_3(i_1,\, i_2,\, i_3) = f^{(3)}(i_3,\, a_2),$$
$$\cdots$$
$$a_{l-1}(i_1,\, i_2,\, \cdots,\, i_{l-1}) = f^{(l-1)}(i_{l-1},\, a_{l-2}),$$

then from right to left

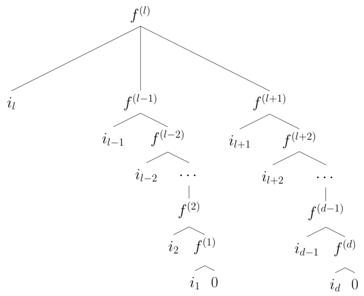
$$a_d(i_d) = f^{(d)}(i_d,\, 0), \quad a_{d-1}(i_d,\, i_{d-1}) = f^{(d-1)}(i_{d-1},\, a_d),$$
$$\cdots$$
$$a_{l+1}(i_d,\, i_{d-1},\, \cdots,\, i_{l+1}) = f^{(l+1)}(i_{l+1},\, a_{l+2}),$$

and, finally,

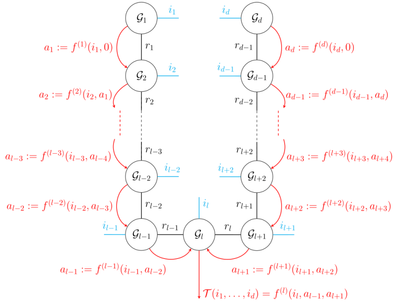
$$\mathcal{K}(i_1,\, i_2,\, \ldots,\, i_d) = f^{(l)}(i_l,\, a_{l-1},\, a_{l+1}).$$

Main result

Pair-wise functions



(a) Computation tree


$\mathcal{G}_1$
i_1
i_d
$\mathcal{G}_d$
$a_1 := f^{(1)}(i_1, 0)$
r_1
r_{d-1}
$a_d := f^{(d)}(i_d, 0)$
$\mathcal{G}_2$
i_2
i_{d-1}
$\mathcal{G}_{d-1}$
$a_2 := f^{(2)}(i_2, a_1)$
r_2
r_{d-2}
$a_{d-1} := f^{(d-1)}(i_{d-1}, a_d)$
r_{l-3}
r_{l+2}
$a_{l-3} := f^{(l-3)}(i_{l-3}, a_{l-4})$
$a_{l+3} := f^{(l+3)}(i_{l+3}, a_{l+4})$
$\mathcal{G}_{l-2}$
i_{l-2}
i_{l+2}
$\mathcal{G}_{l+2}$
$a_{l-2} := f^{(l-2)}(i_{l-2}, a_{l-3})$
r_{l-2}
i_l
r_{l+1}
$a_{l+2} := f^{(l+2)}(i_{l+2}, a_{l+3})$
i_{l-1}
$\mathcal{G}_{l-1}$
r_{l-1}
$\mathcal{G}_l$
r_l
$\mathcal{G}_{l+1}$
i_{l+1}
$a_{l-1} := f^{(l-1)}(i_{l-1}, a_{l-2})$
$a_{l+1} := f^{(l+1)}(i_{l+1}, a_{l+2})$
$\mathcal{T}(i_1, \dots, i_d) = f^{(l)}(i_l, a_{l-1}, a_{l+1})$

(b) Tensor in TT-format, corresponding to the computational tree

Figure 1: Computation tree we can handle and the resulting TT-decomposition

We call l a *middle* index, and the corresponding function $f^{(l)}$ and the core $\mathcal{G}_l$—*middle-function* and *middle-core* respectively.

Pair-wise functions

Trivial example

The sum function.

$$\mathcal{K}(i_1,\, i_2,\, \dots,\, i_d) = x_1[i_1] + x_2[i_2] + \ldots + x_d[i_d].$$

This function naturally evolves into a sequence of pairwise due to

$$\mathcal{K}(i_1,\, i_2,\, \dots,\, i_d) = \Big(\Big(\cdots((x_1 + x_2) + x_3) + \ldots\Big) + x_d\Big).$$

Thus, each function $f^{(k)}$ is just a sum of two arguments

$$f^{(k)}(x,\, y) = x + y, \quad \forall k.$$

Pair-wise functions

Less trivial example

Consider step function in the so-called *Quantized Tensor Train decomposition* (QTT) when the tensor indices are binary $i_k \in \{0, 1\}$ and all together represent a binary representation of some integer from the set $\{0, 1, \ldots, 2^d - 1\}$. The value of the tensor represents the value of some given function P defined on this set,

$$\mathcal{I}(i_1, i_2, \ldots, i_d) = P\left(\sum_{j=0}^{d-1} i_{d-i} 2^j\right).$$

Function P is equal to the step function P_{step} in this example:

$$P_{\text{step}}(x) = \begin{cases} 0, & x \le t, \\ 1, & x > t \end{cases}$$

for the given integer number t, $0 \le t < 2^d$.

Pair-wise functions

Less trivial example

Let the binary representation of t be $t = \sum_{j=0}^{d-1} b_{d-i} 2^j$.

Then the form of the derivative functions for this tensor are depend only on the value of b_k and do not depend on the index k itself. This function are the following[2]:

$$\begin{array}{lll} \text{If } b_k = 0, & \text{then } f^{(k)}(0, x) = x, & f^{(k)}(1, x) = 1; \\ \text{if } b_k = 1, & \text{then } f^{(k)}(0, x) = \begin{cases} 1, & x = 1 \\ \text{None}, & x = 0. \end{cases}, & f^{(k)}(1, x) = x. \end{array}$$

Note that in this example, the original analytic representation for the tensor did not assume pairwise interaction of the indices. On the contrary, the formula for step function is quite integral: its value depends on all variables at once.

[2] the functions f are predefined in the following way. If, in the process of calculating a tensor value, the function $f^{(k)}$ arguments are not in its domain, we assume that it returns an empty value (we denote this value by `None` as in Python language). The next function ($f^{(k-1)}$ or $f^{(k+1)}$), having received `None`, returns also `None`, and so on, up to the "middle" function $f^{(l)}$, which returns 0 if at least one of its arguments is `None`.

Practical example

Cooperative games

As an example, consider several so-called cooperative games Omitting the details of the economic formulation of the problem, let us briefly consider its mathematical model. In general, in the theory of cooperative games it is necessary to calculate the following sum over all subsets of the given set $\mathbb{T}$ of players

$$\pi(k) := \sum_{\mathbb{S} \subseteq \mathbb{T} \setminus \{k\}} p(|\mathbb{S}|)\big(\nu(\mathbb{S} \cup \{k\}) - \nu(\mathbb{S})\big), \quad \text{for all } k \in \mathbb{T}.$$

Here p is some function of the number of players in a *coalition* $\mathbb{S}$. The function of a coalition ν is the function of interest, it depends on the game under consideration. This function denotes the gain that a given coalition receives (*value of the coalition*).

Cooperative games

Set up

We took 4 cooperative games from the article[3] where the author applies tensor methods to fast summation. However, TT-cross is used in this article, which in the case of these games is both slower and less accurate.
As an example for the set up, consider so-called **Shoe sale game.** In this game, participants are divided into two categories—those who sell left boots (indices 1 through L) and those who sell right boots (indices $L+1$ through $2L+1$). As shoes can be sold only in pairs, the value of a coalition is the minimum of the numbers of "left" and "right" players in a coalition.

[3]Rafael Ballester-Ripoll. Tensor approximation of cooperative games and their semivalues. *International Journal of Approximate Reasoning*, 142:94–108, 2022.

Cooperative games

Shoe sale game

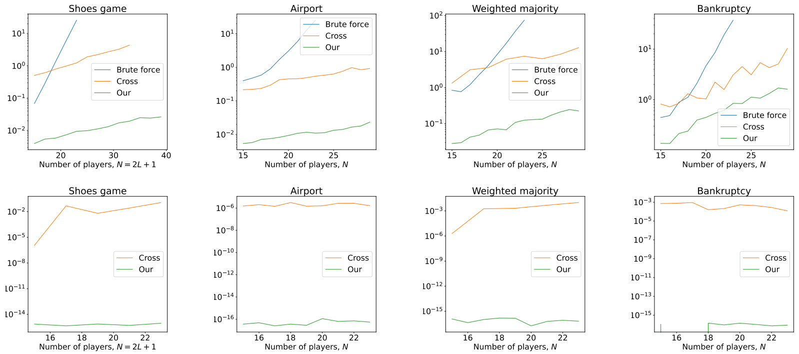
To find the required value π it is convenient to construct tensors that have a dimension equal to the number of players. Each index of this tensor is binary: 1 means a player is a member of a coalition, 0 means he is not.
To construct the TT decomposition of the tensors $p(|\mathbb{S}|)\nu(\mathbb{S})$ using our method, let us take the following derivative functions:

$$f^{(k)}(i, x) = x + i, \;\; 1 \le k \le d, \, k \ne L + 1,$$
$$f^{(L+1)}(i, x, y + i) = \min(x, y) p(x + y + i), \;\; i = 0, 1,$$

thus middle-core is placed on the position $l = L + 1$. The derivative functions for constructing the tensor $p(|\mathbb{S}| - 1)\nu(\mathbb{S})$ are selected in a similar way (we let $p(-1) = p(2L + 1) = 0$).

Results

Cooperative games



Times in seconds and relative accuracy as functions of number of players for four cooperative games. Brute force—calculating the sum directly, Cross—results from the paper of Ballester.

Results

Matrix calculus

Consider a task of calculating permanent[4] of a matrix $\{a_{ij}\} = A \in \mathbb{C}^{d \times d}$. To solve this problem using the presented technique, let us construct two tensors in TT-format. The first tensor $\mathcal{A}$ will represent products of matrix A elements in the form

$$\mathcal{A}(i_1,\, i_2,\, \dots,\, i_d) = a_{i_1,1} a_{i_2,2} \cdots a_{i_d,d}.$$

This is rank-1 tensor and its cores $\{\mathcal{H}_k \in \mathbb{C}^{1 \times d \times 1}\}_{k=1}^{d}$ are $\mathcal{H}_k(1,\, i,\, 1) = a_{ik},\ i = 1,\, \dots,\, d$.

[4]Recall that matrix permanent defined as follows $\operatorname{perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)}$

Results

Matrix calculus

The second tensor is an indicator tensor for such a set of indices, in which all indices are different

$$\mathcal{I}(i_1, i_2, \ldots, i_d) = \begin{cases} 1, & \text{if all } i_1, i_2, \ldots, i_d \text{ are different}, \\ 0, & \text{else}. \end{cases}$$

The cores $\mathcal{G}$ of this tensor are obtained using the following derivative functions

$$f^{(k)}(i, x) = \begin{cases} x + 2^i, & x \,\&\, 2^i = 0, \\ \texttt{None}, & \text{else} \end{cases}, \quad k < d;$$

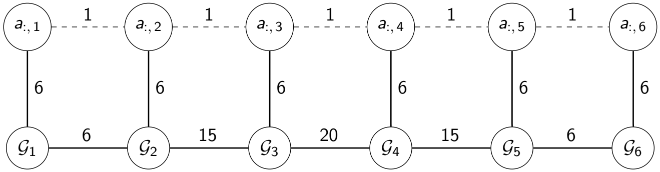
$$f^{(d)}(i, x) = \begin{cases} 1, & x \,\&\, 2^i = 0, \\ 0, & \text{else} \end{cases},$$

where the ampersand sign stands for bitwise AND for integers. In this scheme the middle-core is the last (d-th) core.

Results

Matrix calculus

The permanent value is equal to the convolution of tensor $\mathcal{I}$ and tensor $\mathcal{A}$. Since the tensor $\mathcal{A}$ is one-rank tensor, we can look at the computation of the permanent as an contraction of the tensor $\mathcal{I}$ with weights equal to the corresponding elements of the given matrix A.



Results

Matrix calculus

After the cores are built, total number of operations n_{tot} (both additions and multiplications) required to obtain the result at these ranks has asymotics

$$n_{\text{tot}} \sim 2^N N.$$

This asymptotic is better than the one that can be obtained from the well-known Ryser's formula for calculating the permanent: $P(A) = (-1)^N \sum_{\mathbb{S}\subseteq\{1,2,\dots,N\}} (-1)^{|\mathbb{S}|} \prod_{i=1}^{N} \sum_{j\in\mathbb{S}} a_{ij}$. When applied head-on, this formula requires $O(2^{N-1}N^2)$ operations. It is true that if one uses Hamiltonian walk on $(N-1)$-cube (Gray code) for a more optimal sequence of subset walks, this formula will give asymptotic $n_{\text{tot}} \sim (2^{N-1}N)$ which is only twice as good as ours.

Main contribution

Our main contribution and advantages of our approach

- the exact and fast representation of the tensor in TT-format, which can then, if necessary, be rounded to smaller ranks with a given accuracy. In many of the given examples, this representation is optimal in the sense that the ranks of the TT decomposition cannot be reduced without loss of accuracy;
- highly sparse structure of TT-decomposition cores which leads to a noticeable reduction in calculations;
- an unified approach and a simple algorithmic interface to inherently different tasks and areas including those problems for which it is not immediately obvious the representation of the function specifying the tensor value in the form of consecutive functions;

Main contribution

- the ability to construct an approximate TT-decomposition with a controlled error or/and with the specified maximum ranks of the TT decomposition;
- the possibility to build an iterative algorithm for calculating the value sought in a particular problem, without involving the notion of the tensor or matrix operations;
- the ability to combine different conditions and quickly impose additional conditions when the basic tensor has already been constructed (and rounded).

arXive paper: https://arxiv.org/abs/2206.03832
Github repository: https://github.com/G-Ryzhakov/Constructive-TT

Tensor Cross Approximation in optimization

Classical result: a rank-r matrix can be recovered from r columns and r rows!

$$A = C\hat{A}^{-1}R$$

Similar result holds for tensors (O., Tyrtyshnikov, TT-cross approximation).

A tensor can be recovered from $\mathcal{O}(dnr^2)$ elements!

Tensor Cross Approximation in optimization

Classical result: a rank-r matrix can be recovered from r columns and r rows!

$$A = C\hat{A}^{-1}R$$

Good choice for submatrix $\hat{A}$: maximum-volume submatrix (the one, that maximizes the determinant).

Tensor Cross Approximation in optimization

$$A = C\hat{A}^{-1}R$$

Good choice for submatrix $\hat{A}$:**maximum-volume submatrix** (the one, that maximizes the determinant).

A maximum-volume submatrix has large element:

$$\|\hat{A}\|_C \leq \frac{\|A\|_C}{r^2 + r}, \tag{1}$$

i.e. if we find it, we get dimension-independent bound.

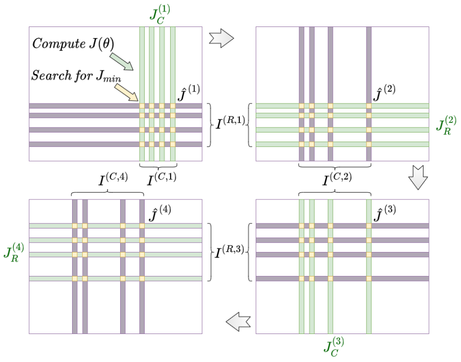
TT-opt

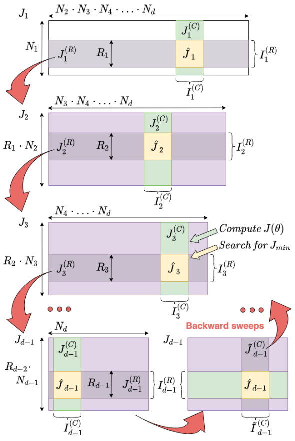
The idea of TT-opt is simple: we start the cross approximation method and record the maximal element on the way.

Several tricks (how to find minimum, how to improve the convergence)

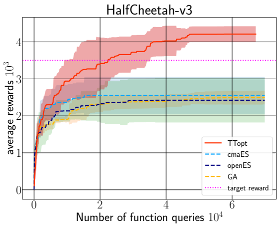
Outperforms several evolutionary algorithms!

TT-opt scheme





Application to RL



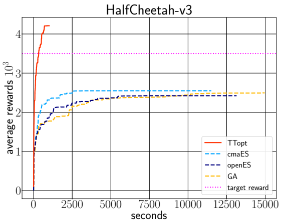


Figure: Training curves of TTOpt and baselines for HalfCheetah-v3 ($N = 3$). The upper plot is the average cumulative reward versus the number of interactions with the environment. The lower plot is the same versus execution time. The reward is averaged for seven seeds. The shaded area shows the difference of one standard deviation around the mean.

Thank you!
Questions?