Christian B. Mendl, Richard M. Milbradt due: 09 June 2022 (before tutorial)

Tutorial 6 (Solving partial differential equations using a Tucker Ansatz)
Tensor network methods are a powerful tool for solving PDEs which almost "factorize" in coordinate directions. As specific example, we consider Poisson's equation

$$\begin{aligned} -\Delta\phi &= \mathsf{b} \quad \text{on } \Omega = [0,1]^3, \\ \phi|_{\partial\Omega} &= 0 \end{aligned}$$

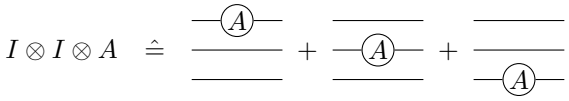
Here $\Delta = \partial_x^2 + \partial_y^2 + \partial_z^2$ is the Laplace operator, $\mathsf{b} : \Omega \to \mathbb{R}$ is a given function, we search for a solution $\phi : \Omega \to \mathbb{R}$, and $\phi|_{\partial\Omega} = 0$ means that ϕ should be zero on the boundary of Ω (i.e., the faces of the unit cube). We will discretize this PDE and approximate ϕ by a Tucker format representation.[1]

(a) Let $\mathsf{f} : [0,1] \to \mathbb{R}$ be a smooth function. Provide a short derivation of the finite difference approximation

$$-\mathsf{f}''(x) = \frac{-\mathsf{f}(x+h) + 2\mathsf{f}(x) - \mathsf{f}(x-h)}{h^2} + \mathcal{O}(h^2).$$

(b) We now discretize the interval $[0,1]$ using the grid points $x_i = ih$ for $i = 0, 1, \ldots, n$ with $h = \frac{1}{n}$ and some large integer n. Together with zero boundary conditions $\mathsf{f}(0) = \mathsf{f}(1) = 0$, f is discretized as vector $f = (\mathsf{f}(x_1), \ldots, \mathsf{f}(x_{n-1})) \in \mathbb{R}^{n-1}$, omitting the points x_0 and x_n. Assemble a matrix $A \in \mathbb{R}^{(n-1)\times(n-1)}$ such that Af is the discretized negative second derivative of f.

(c) Using such a discretization A of $-\partial_x^2$ in one dimension, show that a discretization of $-\Delta$ on $\Omega = [0,1]^3$ with zero boundary conditions is given by

$L = A \otimes I \otimes I + I \otimes A \otimes I + I \otimes I \otimes A \;\;\hat{=}\;$ 

where the identity matrix I has the same size as A. Thus $L \in \mathbb{R}^{(n-1)^3 \times (n-1)^3}$.

Now the problem of solving Poisson's equation amounts to solving the linear system

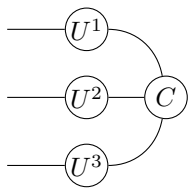
$$L\phi = b. \tag{1}$$

To allow for optimizing a tensor Ansatz for ϕ, we want to convert this to a variational minimization problem. This is achieved by the least squares formulation (using that L is positive semidefinite):

$$\min_{\phi} \frac{1}{2}\langle \phi, L\phi \rangle - \langle \phi, b \rangle \tag{2}$$

(d) Show that (1) and (2) are equivalent when admitting a general vector ϕ, by computing the gradient of (2) with respect to the entries of ϕ. Why must L be positive semidefinite?

To approximate the solution, we use the Tucker format as Ansatz for ϕ, with isometries U^1, U^2, U^3 and a core tensor C:

$\phi =$ 

with $\quad U^\ell \in \mathbb{R}^{(n-1)\times k_\ell}$ for $\ell = 1, \ldots, 3, \quad C \in \mathbb{R}^{k_1 \times k_2 \times k_3}$

and $k_1, k_2, k_3 \ll n$.

(e) Express $\langle \phi, L\phi \rangle$ and $\langle \phi, b \rangle$ in terms of diagrams, assuming that b is likewise stored in Tucker format.

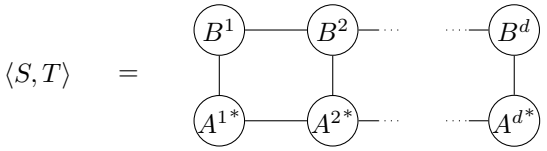
(f) Specify a corresponding "alternating least squares" (ALS) algorithm to obtain an approximate solution.

[1] In this tutorial all functions and tensors are real-valued.

Exercise 6.1 (Exact representation in MPS form)
While matrix product states are typically used for approximating high dimensional tensors using relatively small virtual bond dimensions, the MPS form is not a restriction by itself. For given $n_1, \dots, n_d$, what are the required virtual bond dimensions $D_1, \dots, D_{d-1}$ to represent any tensor $T \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ exactly in MPS form? Justify your answer.

Hint: You can answer this question based on matrix rank considerations. It might be instructive to reconsider the representation $T = T_{\leq \ell} \cdot T_{\geq \ell+1}$ for $\ell \in \{1, \dots, d-1\}$, or the steps of the TT-SVD algorithm. Note that $\operatorname{rank}(A) \leq \min(m, k)$ and $\operatorname{rank}(AB) \leq \min(m, k, n)$ for any $A \in \mathbb{C}^{m \times k}$ and $B \in \mathbb{C}^{k \times n}$.

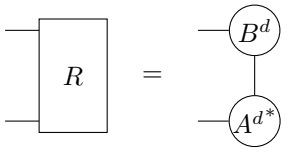
Exercise 6.2 (Inner product of MPS tensors)
The goal of this exercise is an efficient evaluation of the inner product $\langle S, T \rangle$ of two tensors S and T in MPS form, with same degree d. We denote the MPS matrices of S by $A^\ell \in \mathbb{C}^{n_\ell \times D_{\ell-1} \times D_\ell}$ for $\ell = 1, \dots, d$, and likewise the MPS matrices of T by $B^\ell \in \mathbb{C}^{n_\ell \times D'_{\ell-1} \times D'_\ell}$. Note that S and T must have the same logical dimensions $n_1 \times \cdots \times n_\ell$ (such that the inner product is well defined), but that the virtual bond dimensions can be different. By convention, the leading and trailing virtual bond dimensions are (dummy) singleton dimensions, i.e, $D_0 = D'_0 = 1$, $D_d = D'_d = 1$. Graphically, the inner product is the result of the following tensor network contraction (with * denoting element wise complex conjugation, the upward leg of A^ℓ corresponding to its first dimension n_ℓ, and omitting the singleton bonds):



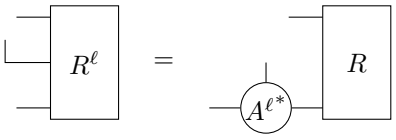
For computational and memory efficiency, this contraction should be performed from "right to left" (or equivalently from "left to right"), which leads to the following algorithm:

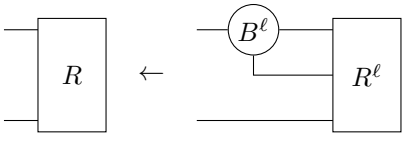
MPS inner product

Input: list of MPS tensors A^ℓ and B^ℓ, $\ell = 1, \dots, d$

1. Initialize $R \in \mathbb{C}^{D'_{d-1} \times D_{d-1}}$ as 

2. `for` $\ell = d-1, \dots, 2, 1$:

   Contract $A^{\ell *}$ with R: 

   Contract B^ℓ with R^ℓ and update R: 

3. `return` the single entry of $R \in \mathbb{C}^{D'_0 \times D_0} = \mathbb{C}^{1 \times 1}$

(a) Implement and run the algorithm using the Jupyter notebook template from the Moodle page.

Hint: `np.tensordot` is useful for each step; be careful with the ordering of dimensions!

(b) Assuming for simplicity that $n_\ell = n$ for $\ell = 1, \dots, d$ and $D_\ell = D'_\ell = D$ for $\ell = 1, \dots, d-1$, what is the asymptotic computational cost (in terms of powers of n, D and d) of this algorithm (when ignoring optimizations as described in Tutorial 4)?

Hint: First identify the most expensive step of the algorithm.