An introduction to tensor trains

Stefano Crotti - stefano.crotti@polito.it

February 28, 2024

Tensor trains are a mathematical tool. They are mostly used in physics as a parametrization for variational quantum states under the name matrix product states. They are the simplest form of tensor network.

Here we will introduce tensor trains as a parametrization for probability distributions over discrete variables. We will first explore why a distribution parametrized by a tensor train is "tractable", i.e. many useful quantities can be computed efficiently. Then we will show how any distribution can be written as a tensor train. The natural next step, not covered here, is to explore how the parameters of a tensor train can be optimized in order to fit some target distribution, whether analytical or empirical.

As a measure of the running time of an algorithm we will be using the big O notation.

Definition

Given n discrete variables $\underline{x} = x^1, x^2, \dots, x^n$ each taking values in $\mathcal{X} = \{1, 2, \dots, q\}$ and a set of qn complex-valued matrices $\left\{A^i(x^i)\right\}_{i=1}^n$, a tensor train is a function $\psi : \mathcal{X}^n \to \mathbb{C}$

$$\psi(\underline{x}) = A^1(x^1)A^2(x^2)\cdots A^n(x^n)$$

where $A^i(x^i)A^{i+1}(x^{i+1})$ is intended as a matrix product and the sizes of the matrices are compatible with each other. Moreover we require A^1 to have one row and A^n to have one column, such that the overall product gives a scalar. Notice that in principle the domain $\mathcal{X}$ could be different for each variable. Moreover, here we work in a general setting with a complex-valued ψ, however when it parametrizes a probability distribution one might (and should!) require real and non-negative values.

Problem 1.

Given a tensor train $\psi(\underline{x}) = A^1(x^1)A^2(x^2)\cdots A^n(x^n)$ with matrices of size $d \times d$ (except the first and last one which are $1 \times d$ and $d \times 1$, respectively), devise a strategy polynomial in n, q, d to compute:

1. $Z_1 = \sum_{\underline{x}} \psi(\underline{x})$ (Hint: one line of algebra) ($\Leftarrow$ this is how you normalize ψ if it is a probability distribution)
2. $Z_2 = \sum_{\underline{x}} |\psi(\underline{x})|^2 = \sum_{\underline{x}} \psi(\underline{x})^* \psi(\underline{x})$ (Hint: if A is a scalar, then $A^* = A^\dagger$) ($\Leftarrow$ this is how you normalize ψ if it is a wavefunction)
3. $p^i(\tilde{x}^i) = \sum_{\underline{x}} \frac{1}{Z} \psi(\underline{x}) \delta_{x^i, \tilde{x}^i}$

Bonus:

1. Can you compute all marginals $p^i(\tilde{x}^i)\ \forall i$ in time $\mathcal{O}(n)$?
2. Find a polynomial strategy to draw samples from $p(\underline{x}) = \frac{1}{Z}\psi(\underline{x})$ (Hint: any multivariate distribution can be decomposed as the product of conditional distributions via repeated application of Bayes' formula, then...) (This stuff is actually useful! See [HWF+18])
3. Repeat all of the above for a tensor train with periodic boundary conditions: $\psi(\underline{x}) = \mathrm{Tr}\left[A^1(x^1)A^2(x^2)\cdots A^n(x^n)\right]$, where Tr is the matrix trace $\mathrm{Tr}A = \sum_i A_{ii}$ and all matrices have size $d \times d$.
4. The gradient $\frac{\partial \log Z_2}{\partial [A^i(y)]_{j,k}}$. (This stuff is actually useful! See [HWF+18])

Solution (1.) By linearity of matrix product, each sum slides in front of the corresponding matrix

$$Z_1 = \prod_{i=1}^{n} \sum_{x_i} A^i(x^i) \tag{1}$$

which takes computation time $\mathcal{O}(nqd^2)$.

(2.) Use the hint $\psi^* = \psi^\dagger$

$$Z_2 = \sum_{\underline{x}} \psi(\underline{x})^\dagger \psi(\underline{x}) \tag{2}$$

$$= \sum_{\underline{x}} [A^n(x^n)]^\dagger \cdots \left[A^2(x^2)\right]^\dagger \left[A^1(x^1)\right]^\dagger A^1(x^1)A^2(x^2)\cdots A^n(x^n) \tag{3}$$

then define

$$B^i = \sum_{x^1,x^2,\ldots,x^i} \left[A^i(x^i)\right]^\dagger \cdots \left[A^2(x^2)\right]^\dagger \left[A^1(x^1)\right]^\dagger A^1(x^1)A^2(x^2)\cdots A^i(x^i) \tag{4}$$

where now $B^n = Z_2$. The B^i's can be computed recursively as

$$B^i = \sum_{x^i} \left[A^i(x^i)\right]^\dagger B^{i-1} A^i(x^i) \tag{5}$$

starting from the initial condition $B^0 = 1$. This takes computation time $\mathcal{O}(nqd^3)$.

(3.) Repeat (1.) but insert the Kroenecker delta while computing B^i. This takes computation time $\mathcal{O}(n^2qd^2)$.

Observation 1 A factorized ("mean-field" in the language of physicists) function

$$f(\underline{x}) = f^1(x^1)f^2(x^2)\cdots f^n(x^n)$$

(where each f^i is scalar-valued) is a tensor train with matrix sizes all equal to 1.

Universality of tensor trains

Let's show that any function of discrete variables $f(x^1, x^2, \ldots, x^n)$ (thus, in particular, any probability distribution) can be written in tensor train format, loosely following the derivation in [OT10]. **Warning**: this fact doesn't have any immediate useful implication in practice: as we will see, the matrices in the resulting tensor train are huge in size.

The reshape operation Any d-dimensional array $A_{i_1,i_2,\ldots,i_d}$ can be transformed into a matrix by collecting its indices into two groups. This operation takes its name from the `reshape` function in scientific programming languages. We write it as

$$\tilde{A}_{(i_1,i_2,\ldots,i_k),(i_k,i_{k+1},\ldots,i_d)} = A_{i_1,i_2,\ldots,i_d}$$

where the first k indices have been grouped together to form the row index of $\tilde{A}$, the remaining $d-k$ forming the columns. In the following we will sometimes drop the $\tilde{}$ and just call both objects A; a little abuse of notation never hurt anybody.

Recall: singular value decomposition (SVD) Any matrix $A \in \mathbb{C}^{m,n}$ can be decomposed as $A = U\Sigma V^\dagger$ with Σ a diagonal matrix whose elements, the singular values, are non-negative, and $U^\dagger U = VV^\dagger = \mathbb{1}$.

We will repeatedly use reshaping and SVD to give a constructive proof of the universality property of tensor trains. We start by realizing that ψ, being a function of discrete variables, can be thought of as an array ($\simeq$ tensor) with n indices $\psi_{x^1,x^2,\ldots,x^n} \equiv \psi(x^1,x^2,\ldots,x^n)$. Now we reshape the array into $\psi_{x^1,(x^2,\ldots,x^n)}$ and decompose it via SVD as $\psi_{x^1,(x^2,\ldots,x^n)} = \sum_k A^1_{x^1,k} B_{k,(x^2,\ldots,x^n)}$ (we called $A^1 = U, B = \Sigma V^\dagger$). Now we reshape $A^1_{x^1,k}$ as $\left[A^1(x^1)\right]_k$, i.e. a row vector indexed by k, and we are left with

$$\psi(x^1,x^2,\ldots,x^n) = A^1(x^1)B(x^2,\ldots,x^n).$$

Now we only need to repeat the same operation on B: we reshape it into $B_{(k,x^2),(x^3,\ldots,x^n)}$, then decompose it as $B_{(k,x^2),(x^3,\ldots,x^n)} = \sum_m A^2_{(k,x^2,m)} C_{(m,x^3,\ldots,x^n)}$, reshape $A^2_{(k,x^2,m)} = \left[A^2(x^2)\right]_{k,m}$ and we are left with

$$\psi(x^1,x^2,\ldots,x^n) = A^1(x^1)A^2(x^2)C(x^3,\ldots,x^n).$$

The procedure is iterated n times until the tensor train format is found.

Here we didn't bother to write down the range of each index in the SVD's, but it's not difficult to convince oneself that the sizes of the A^i matrices grow exponentially with i. In the end this is no surprise: we started with an object $(\psi_{x^1,x^2,\ldots,x^n})$ with q^n entries and wrote it in an equivalent form which still requires an exponential number of parameters to be expressed. The reason we show this is that the tensor train format allows for clever ways of compressing the information into smaller matrices. Through the machinery of SVD-based truncations, one can sometimes find good approximations using small enough matrices. Once the matrix size d is brought down to a manageable number, all the quantities mentioned in problem 1 can be computed efficiently.

References and reading material

- https://tensornetwork.org/ is a website with both introductory level explanations and pointers to applications
- Tensor-Train Decomposition introduces the concept from a mathematical point of view
- The density-matrix renormalization group in the age of matrix product states is a standard reference for the use in quantum mechanics

Applications in various fields:

- Tropical Tensor Network for Ground States of Spin Glasses
- Supervised Learning with Quantum-Inspired Tensor Networks
- Equivalence of restricted Boltzmann machines and tensor network states
- Using matrix product states to study the dynamical large deviations of kinetically constrained models
- The matrix product approximation for the dynamic cavity method
- Exact solution of a 1D asymmetric exclusion model using a matrix formulation

References

[HWF+18] Zhao-Yu Han, Jun Wang, Heng Fan, Lei Wang, and Pan Zhang. Unsupervised generative modeling using matrix product states. *Physical Review X*, 8(3):031012, 2018.

[OT10] Ivan Oseledets and Eugene Tyrtyshnikov. Tt-cross approximation for multidimensional arrays. *Linear Algebra and its Applications*, 432(1):70–88, 2010.