TS4: Tensorized Structured State Space Sequence Models

Anonymous ACL submission

Abstract

Recently, structured state space sequence (S4) models (Gu et al., 2022) have generated considerable interest due to their simplicity and favorable performance compared to the transformer architecture in certain sequence modeling tasks. A very important property that distinguishes these models from traditional gated RNNs is the linear dependence of the model output on the latent space vector at each time step, even when an input dependent selection mechanism is incorporated, (Gu and Dao, 2023). This means that the computation underlying inference and sequence mapping in these models involves linear time evolution of the latent space vector. Inspired by long standing studies of time evolution of matrix product states in quantum mechanics (Cirac et al., 2021), we study the problem of compressing the latent space of sequence models using tensorization methods. Such tensorized sequence models, we call TS4. Various novel structures on the parameters of S4 models within the tensorization setting are imposed to propose new classes of structured sequence models.

1 Introduction

Machine learning models that take inputs in a fixed sequential order (so-called sequence models) have diverse applications. A limited set of examples of such applications includes natural language processing, time series forecasting, speech recognition, voice recognition and many others. A concrete realization of sequence models is the transformer architecture (Vaswani et al., 2017) which serves as the central backbone of the current generation of large language models (Achiam et al., 2023). The fundamental quadratic scaling problem of the attention module in transformers is well known. Various proposed solutions exist to mitigate this. One class of these solutions, named S4 models, involves using a latent vector space in order to learn an effective representation of context and memory, which can then be used to model the next element of the input sequence. The relationship of the latent vector to the input to be processed at each time step in such models is quite reminiscent of classical Kalman filters. Additionally. these models can also be thought of as a combination of convolutional neural networks (CNN) and recurrent neural networks (RNN). Different classes of these models have been investigated. All of them differ in the manner in which the corresponding Kalman filter state transition matrix (which is denoted by the model parameter A in the relevant literature) is parametrized.

One peculiar property of these models is that the latent vector always evolves *linearly*. This is a marked difference from the generally non-linear evolution of the latent vectors in gated RNNs. All the proposals in this short paper were therefore inspired by the superficial similarities of this linear evolution of the latent vector in S4 models with the linearity inherent in the time evolution dynamics of quantum systems. Noting that quantum mechanics is also afflicted with the curse of dimensionality for large systems, coupled with the observation that larger latent spaces in S4 models will generally lead to better modeling of properties of the input sequence, we sought to borrow some ideas that are used to economically represent complicated quantum states using what are called matrix product states (defined below), to the setting of S4 models.

2 Issues solved by TS4 models

Before we discuss the construction of TS4 models in detail, we would like to provide a motivation in terms of practical issues that are solved by imposing structure on S4 models after tensorizing the latent space.

2.1 Requirement of complex field

Real diagonalizable matrices are *not* dense in $\mathbb{R}^{N\times N}$. This fact implies that the underlying field must be complex for the diagonal choice of A in popular versions of S4 models to work. Thus, we need to use complex numbers as part of the training process. This causes issues with activation functions like the softmax which is strictly ill-behaved for complex inputs. Moreover, various empirical constraints have to be imposed on the real and imaginary parts of the entries of A.

2.2 Numerical Stability

For even moderately large sequence lengths, the corresponding norms of matrix elements in the diagonal representation may blow up. This causes convergence issues during training, and needs to be carefully handled. Further, the real part of the entries in A may become large and positive, and therefore, need to be regulated properly.

3 Preliminaries

3.1 Notation and Terminology

For a vector space V, we denote the set of linear operators acting on that vector space by the notation $L(V)$. For the tensor product of vector spaces,

$$V = V_1 \otimes V_2 \otimes \cdots \otimes V_k,$$

we denote by e_I an element of the standard basis set of V constructed by tensoring the individual basis elements of V_i, namely, $e_I = e_{i_1} \otimes e_{i_2} \cdots \otimes e_{i_k}$, when the multi-index $I = (i_1, \cdots, i_k)$, and $e_j \in V_j$ is a basis element. We further call each V_i above a *local factor* of the full vector space V. An important fact to keep in mind that would be useful subsequently is that if $h_j \in L(V_j), h_k \in L(V_k)$, and $j \neq k$, then,

$$[h_j, h_k] = 0.$$

We denote discrete time sequences using the notation x_t. For any two such sequences x_t, y_t, we denote their convolution as

$$z_t = x_t \star y_t$$

3.2 Technical Background

Consider a general tensor whose components are based on a vector space of dimension N.

$$T(i_1, \cdots, i_k). \qquad (1)$$

If each index i_j here ranges over 1 to N, then this tensor in genral has N^k components. Thus the total number of required components scales exponentially with the number of dimensions of the tensor. Dense tensors in large dimensions therefore, cannot be practically stored as is. To fix this, the notion of a tensor train was introduced. This is a set of third rank tensors B_i (except for B_1, and B_k which are second rank), such that we can write

$$\begin{aligned} &T(i_1 \cdots i_k) \\ &= B_1(i_1)B_2(i_2)\cdots B_{k-1}(i_{k-1})B_k(i_k) \qquad (2) \end{aligned}$$

Writing out the indices for the implicit matrix multiplications in Eq.(2) in order to set our convention, we get

$$\begin{aligned} T(i_1 \cdots i_k) = \sum_{\alpha_i} B_1(i_1, \alpha_1) B_2(\alpha_1, i_2, \alpha_3) \\ \cdots B_{k-1}(\alpha_{k-2}, i_{k-1}, \alpha_{k-1}) B_k(\alpha_{k-1}, i_k). \quad (3) \end{aligned}$$

This representation is useful if the size of each $B_j(k)$ is bounded by a constant $r \times r$. In this case, the total number of scalars in all the B_j put together is simply kNr^2, which is thus linear in both k and N (compared to exponential in dk and polynomial in N before). This is huge savings and implies that tensor trains can be stored practically on a computer. We now briefly look at how to construct such a decomposition for a given tensor. tensor train generation can now be reduced to multiple SVDs. To see this, start with an arbitrary tensor $T_{i_1 \cdots i_k}$ such that index i_j satisfies $1 \leq i_j \leq n_j$. Consider its first *unfolding* A_1, such that

$$A_1(i_1; i_2, \cdots, i_k) = T(i_1, \cdots, i_k)$$

The notation for the A_1 just means that T has been reshaped, such that it has n_1 rows, and $\prod_{j=2}^{k} n_j$ columns, the latter having been structured into a fat index. Perform a singular value decomposition on A_1, so that $A_1 = U_1 V_1$. Writing out the indices explicitly, we get

$$\begin{aligned} &A_1(i_1; i_2, \cdots, i_k) \\ &= \sum_{\alpha_1=1}^{r_1} U_1(i_1, \alpha_1) V_1(\alpha_1, i_2, \cdots, i_k). \end{aligned}$$

The number r_1 here represents the algebraic rank of A_1, which is the number of linearly independent columns and rows. This implies that the matrix $X = U_1^T U_1$ is a full rank square matrix of dimension $r_1 \times r_1$. We can now solve for V_1 as

$V_1 = S_1 A_1$, which in terms of indices can be written out as

$$V_1(\alpha_1, i_2, \cdots, i_k) = \sum_{i_1=1}^{n_1} S_1(\alpha_1, i_1) A(i_1, \cdots, i_k),$$

where we have defined $S_1 = X^{-1} U_1^T$. Next, consider the tensor V_1 obtained above, and reshape it into A_2 to have $r_1 n_2$ rows and $\prod_{j=3}^{d} n_j$ columns, such that

$$A_2(\alpha_1 i_2; i_3, \cdots, i_k) = V_1(\alpha_1, i_2, \cdots, i_k)$$

We can again write $A_2 = U_2 V_2$, with the indices written out explicitly as,

$$A_2(\alpha_i i_2; i_3, \cdots, i_k) = \sum_{\alpha_2=1}^{r_2} U_2(\alpha_1, i_2; \alpha_2) V_2(\alpha_2, i_3, \cdots, i_k).$$

Repeating this process by induction, we obtain the tensors

$$U_1(i_1, \alpha_1), U_2(\alpha_1 i_2; \alpha_2), \cdots, U_k(\alpha_{k-1}, i_k).$$

Finally, the so called core tensors of the decomposition in Eq.(2) are obtained as

$$\begin{aligned} B_1(i_1, \alpha_1) &= U_1(i_1, \alpha_1), \\ B_k(\alpha_{k-1}, i_k) &= U_k(\alpha_{k-1}, i_k), \\ B_j(\alpha_{j-1}, i_j, \alpha_j) &= U_j(\alpha_{j-1} i_j; \alpha_j) \\ 2 \le &j \le k-1 \end{aligned} \tag{4}$$

We note in passing that unlike diagonalization, this tensor train generation is applicable to both real and complex tensors.

4 Structured State Space Sequence Models

The current analysis of S4 models involves creating a mapping between sequences $x_t \to y_t$, as solutions of the equations

$$h'_t = A h_t + B x_t, \qquad y_t = C h_t. \tag{5}$$

Here,

$$h_t \in \mathbb{R}^{N \times 1}, A \in \mathbb{R}^{N \times N}, \\ B \in \mathbb{R}^{N \times 1}, C \in \mathbb{R}^{1 \times N},$$

are vectors and matrices living in the latent space. Note further that, we can consider a discretization of this set of equations, with an implicit time step Δ, and re-write this as an effectively discrete evolution. We define,

$$\overline{A} = \exp(\Delta A), \qquad \overline{B} = (\Delta A)^{-1}(\overline{A} - \mathbb{I})(\Delta B). \tag{6}$$

Then, we the kernel $\overline{K}$ can be written as

$$\overline{K} = (C\overline{B}, C\overline{AB}, \cdots, C\overline{A}^{k-1}\overline{B}, \cdots), \tag{7}$$

and the output is obtained as

$$y_t = x_t \star \overline{K} \tag{8}$$

We note that the convolution in Eq.(8) is an operation that can be implemented quite efficiently in modern hardware using FFTs. Thus, our focus would be on exploring methods for the computation in Eq.(7) in an efficient manner.

5 Our approach: TS4 models

We now focus on a hitherto unexplored way of imposing structure on state space models. The proposal is as follows. Suppose the latent vector space V of dimension N, can be written as the tensor product of k real vector spaces of dimension d, $V = H_1 \otimes H_2 \otimes \cdots \otimes H_k$.This implies that $N = d^k$. Suppose further we choose A to have the form

$$A = \sum_j h_j \\ h_j \in L(H_j \otimes H_{j+1} \otimes \cdots \otimes H_{j+l}). \tag{9}$$

This expression for A is inspired by the local Hamiltonians that one writes for lattice systems in quantum physics. If we take l to be large, that means that the information in the latent state is shared among a larger number of factor spaces. We note that the number of parameters P in A is just $P \in O(kd^{2l})$. If we therefore choose $l << k$, then, the number of parameters in A scales as log$N$, compared to N for a diagonal A. We have therefore effectively reduced the number of parameters that need to be trained.

5.1 Novel classes of TS4 models

The structure on TS4 models is now imposed by specifying l in Eq.(9). We call a TS4 model s-local if, in Eq.(9), $l = s$. Operationally, this means that the encoding information of the input sequence is scrambled to s local factor spaces by the A matrix. While general s-local models are harder to analyze analytically, we impose additional structure on the local operators h_j in Eq.(9) and constrain them to be strings of operators, in the form

$$h_j = \prod_{m=j}^{j+s} f_m, \quad f_m \in L(H_m).$$

This is the simplest form of structure that can be imposed on the h_j, and for small enough s, one can get good closed form approximations (and sometimes, exact expressions) for the exponent $\overline{A}$ in Eq.(6). Since the vector $B \in V$, it needs to similarly be represented in a form that is economical. Accordingly, we choose a tensor train form for its components

$$B(i_1, \cdots, i_k) = B_1(i_1)B_2(i_2)\cdots B_k(i_k) \quad (10)$$

where B_1, B_k are rank-1 tensors, and the others are rank three tensors. A similar expression for C holds.

6 Conclusion and Future Direction

In this short paper, we have discussed novel ways of imposing structure on state space models after tensorization of the latent space. These additional structural constraints are inspired from the behavior of quantum lattice systems with local interactions. Locality is imposed by requiring the degree of scrambling of information by the A parameter is restricted to a pre-determined number of local copies of the latent space. We are in the process of running evaluations of these ideas for sequence modeling tasks. Further, we would like to compare explicitly with the results of (Su et al., 2024).

Limitations

While this work proposes a novel framework for analyzing and potentially compressing S4 models using tensor product spaces, several limitations should be acknowledged. First, the theoretical development is primarily heuristic and lacks rigorous proofs or guarantees about representational capacity or stability under this reformulation. As a result, claims about memory savings or parameter efficiency are based on empirical intuition rather than formal bounds.

Second, our experimental evaluation is still in early stages. While the proposed framework offers a compelling perspective, we have not yet conducted large-scale benchmarks across multiple NLP tasks to quantify improvements or trade-offs in accuracy, training time, or hardware efficiency. This limits the immediate applicability of our findings to practical systems.

Third, the current formulation assumes certain factorization structures in the state space matrices, which may not hold in pre-trained S4 models or may require retraining from scratch. This restricts the generality of the method, especially when attempting to apply it in a plug-and-play fashion to existing SSM-based architectures.

Finally, tensor product representations can grow rapidly in dimensionality if not carefully constrained, potentially negating memory savings in practice. Future work will explore low-rank or decomposed representations to mitigate this overhead.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

J Ignacio Cirac, David Perez-Garcia, Norbert Schuch, and Frank Verstraete. 2021. Matrix product states and projected entangled pair states: Concepts, symmetries, theorems. *Reviews of Modern Physics*, 93(4):045003.

Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv:2312.00752*.

Albert Gu, Karan Goel, and Christopher Re. 2022. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*.

Zhan Su, Yuqin Zhou, Fengran Mo, and Jakob Grue Simonsen. 2024. Language modeling using tensor trains. *arXiv preprint arXiv:2405.04590*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.