IFT 6760A - Lecture 12
More on the Tensor-Train decomposition and Tensor Networks

Scribe(s): Tobi Carvallo, Aayushi Kulshrestha, Faruk Ahmed **Instructor:** Guillaume Rabusseau

1 Summary

In the previous lecture, we introduced a third kind of tensor decomposition which is the Tensor Train (TT) decomposition. We covered the definition, tensor diagram notation, as well as basic algebraic operations in a TT-decomposition (addition, point wise multiplication and contraction). We have seen how this decomposition can be advantageous, specially with high-dimensionsional tensors; the number of parameters in a TT representation is $\mathcal{O}(ndR^2)$ compared to the $\mathcal{O}(d^n)$ storage for a naive representation. We did an overview of applications in machine learning where higher dimensions are beneficial for solving certain problems with linear models. Finally, we discussed learning the parameters of a weight tensor, when it is expressed as a tensor train.

In this lecture, we begin by introducing TT-SVD, an algorithm used to transform a dense tensor into a tensor-train (TT) format. The TT-format is advantageous for enabling efficient operations in downstream tasks such as learning with very high-dimensional projections of data, which we shall see. This representation can also be a starting point to finding low-rank approximations by reducing the intermediate ranks of the tensor train. We then show how to perform SVD efficiently in a tensor train, with repeated QR-decompositions of cores and merging operations, which leads to significant improvement in efficiency compared to equivalently performing SVD on a dense matricization of the tensor. Finally, we reprise the discussion of learning weight tensors in their decomposed representation, via the *alternating minimization* and *DMRG-like* algorithms.

2 Introduction to TT-SVD

As seen in the previous class, the tensor network obtained using TT-decomposition scales linearly with the order of the tensor, as opposed to exponentially when using dense tensors or the Tucker decomposition. In this section, we first focus on introducing the algorithm for obtaining TT-format of a tensor network, known as TT-SVD which can be further used to generate quasi-low rank approximations.
To formally define TT-SVD, we first define the TT-rank of a tensor.

Definition 1 (TT Rank)**.** *The TT rank of $\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times \cdots d_n}$ is the smallest $(R_1, R_2, \cdots, R_{p-1})$ such that $\mathcal{T}$ has a tensor train decomposition of rank $(R_1, \cdots, R_{p-1})$.*

For the sake of simplicity we shall assume from this point on that all the dimension of the tensor are of the same dimension (hypercubic). There is no loss of generality since everything extends to tensors of arbitrary dimensions.

Definition 2 (Matricization along several modes)**.** *The mode-$(1, \cdots, k)$ matricization of an order-p tensor $\mathcal{T}$ is defined as $\mathcal{T}_{(1,\cdots,k)} = \text{RESHAPE}(\mathcal{T}, d^k \times d^{p-k})$ for all $k = 1, \cdots, p$.*

See Figure 2 for an illustrative example.

Theorem 3. *Let $R_i = rank(\mathcal{T}_{(1,\cdots,i)})$ for each $i = 1, \ldots, p-1$. Then the TT rank of $\mathcal{T}$ is $(R_1, R_2, \cdots, R_{p-1})$.*

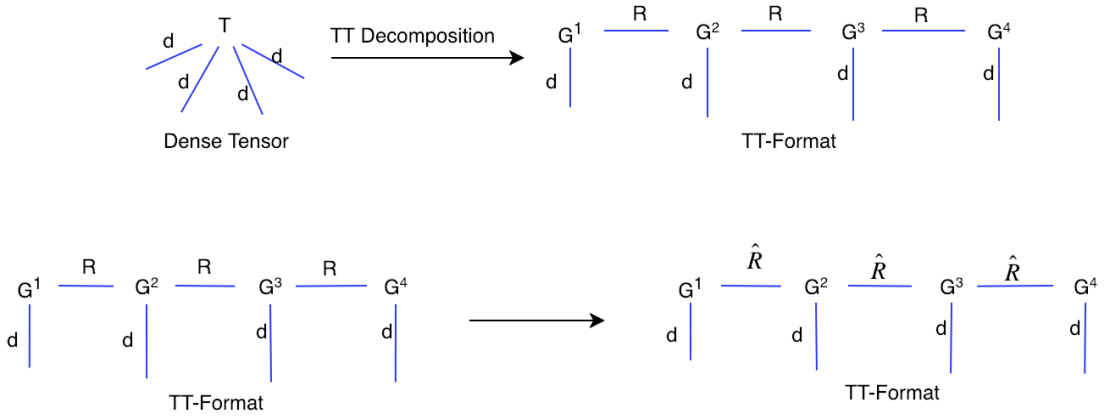


Figure 1: Pictorial Representation of a TT-decomposition using TT-SVD and of the result of TT-rounding.

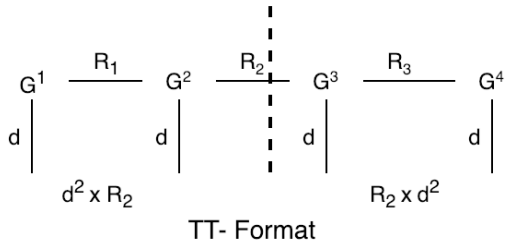


Figure 2: The above picture shows mode-2 matricization where $p = 4, k = 2$. This will thus give us a $d^2 \times d^2$ matrix

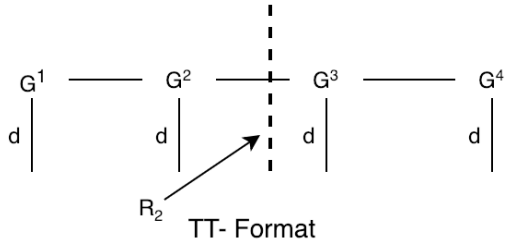


Figure 3: A visual interpretation of Theorem 3

As shown in Figure 3, by splitting the tensor network at R_2 and performing matricization, we see that the resulting matrix is formed by the product of two matrices of dimension $d^2 \times R_2$ over their dimension R_2. The result is then a $d^2 \times d^2$ matrix of maximum rank R_2 (since $d^2 > d > R_2$). This logic can be applied at every R_i to see that the TT rank of $\mathcal{T}$ is indeed $(R_1, R_2, \cdots, R_{p-1})$.

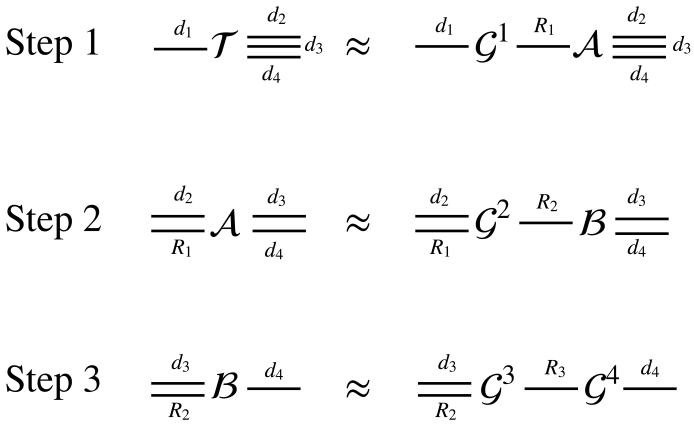
This observation allows us to see that given the TT-rank of a tensor, we can infer the ranks of the matricizations over the p modes. In the following section, where we discuss the TT-SVD algorithm, we shall see the other direction of this theorem: given a set of desired ranks for truncating intermediate SVDs of matricizations, we can design a tensor train with the equivalent rank.

2.1 Performing TT-SVD

We shall provide a high-level overview of the TT-SVD algorithm before describing it more formally. Let us say we are given a 4-way tensor $\mathcal{T}$ that we shall decompose into a tensor train.

We take as input the tensor $\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3 \times d_4}$, and a set of 3 target ranks (R_1, R_2, R_3). Our goal is to compute a tensor train decomposition of $\mathcal{T}$ such that it is decomposed into a train of 4 cores $\mathcal{G}^1, \mathcal{G}^2, \mathcal{G}^3, \mathcal{G}^4$, with dangling dimensions d_1, d_2, d_3, d_4 and bond dimensions R_1, R_2, R_3.

The steps consist of repeated matricizations, and truncated-SVDs of the matrices under rank-R_i approximations, as illustrated in the following diagram.



If it so happens that the TT-rank of $\mathcal{T}$ is actually (R_1, R_2, R_3) then the approximations would be equalities, and the decomposition would be exact.

We have some results for quantifying the loss of precision when performing TT-SVD, and the main theorem is as follows:

Theorem 4. *If $\mathcal{T}_{(1,2,\ldots,k)} = \mathbf{M_k} + \mathbf{E_k}$ for each $k = 1, \cdots, (p-1)$, where $\mathbf{M_k}$ is of rank R_k and $\|\mathbf{E_k}\|_F \leq \varepsilon_k$, then the output of TT-SVD, $\hat{\mathcal{T}}$ satisfies :*

$$\|\mathcal{T} - \hat{\mathcal{T}}\|_F^2 \leq \sum_{k=1}^{p-1} \varepsilon^2$$

This theorem defines the upper bound for loss of information when applying TT-SVD algorithm to a tensor.
Note that instead of specifying a set of ranks $(R_1, R_2, \cdots, R_{p-1})$ as input to TT-SVD algorithm, we can now provide an error budget and discover intermediate ranks accordingly.

Corollary 5. *If $\mathcal{T}^*$ is the best approximation of $\mathcal{T}$ of TT rank $(R_1, R_2, \cdots, R_{p-1})$, then $\|\mathcal{T} - \hat{\mathcal{T}}\|_F \leq \sqrt{p-1}\|\mathcal{T} - \mathcal{T}^*\|_F$*

As we saw, the TT-SVD algorithm consists of alternating steps of matricization and performing low-rank approximated SVD on the matrices. Let us now give a more formal description of this algorithm:

Algorithm 1 Algorithm for generating TT-format using TT-SVD

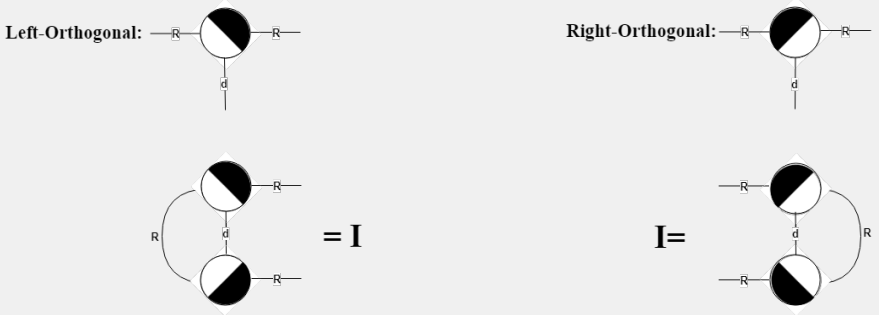
Let $\mathcal{T}_{(1)} \approx \mathbf{UDV^T}$ be a rank R_1 truncated SVD, where $\mathcal{T}_{(1)} \in \mathbb{R}^{d \times d^3}, \mathbf{U} \in \mathbb{R}^{d \times R_1}, \mathbf{DV^T} \in \mathbb{R}^{R_1 \times d^3}$
Let $\mathcal{G}^1$ = $\mathbf{U}$ and $\mathbf{A}$ = RESHAPE$(\mathbf{DV}^\top, R_1 d \times d^2)$
Let $\mathbf{A} \approx \mathbf{UDV}^\top$ be a rank R_2 truncated SVD
Let $\mathcal{G}^2$ = RESHAPE$(\mathbf{U}, R_1 \times d \times R_2)$
$\mathbf{B}$ = RESHAPE$(\mathbf{DV}^\top, R_2 d \times d)$
...so on and so forth for the number of cores required.

Note, that in the above algorithm, the cores $\mathcal{G}^1, \mathcal{G}^2, \cdots, \mathcal{G}^{p-2}$ are all left-orthogonal by construction. That is, $(\mathcal{G}^1_{(2)})^\top \mathcal{G}^1_{(2)} = \mathbf{I}$ and $(\mathcal{G}^i_{(3)})^\top \mathcal{G}^i_{(3)} = \mathbf{I}$ for all $i = 1, \ldots, p-2$.

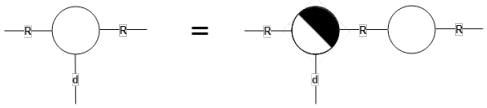
Naively performing SVD is exponential in the dimension of the tensor, $\mathcal{O}(pd^{p+1})$ [1]. In the next section we introduce an efficient technique for the intermediate SVDs of the matricizations which scales linearly with the dimension of the tensor.

3 Efficient SVD in TT format

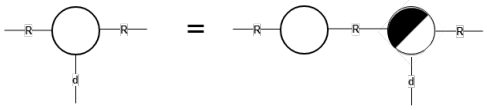
Definition 6 (Left-right Orthogonality)**.** *A core $\mathcal{G} \in \mathbb{R}^{R \times d \times R}$ is said to be left orthogonal if its left matricization $\mathcal{G}_{(3)}$ is orthogonal i.e. $\mathcal{G}_{(3)}\mathcal{G}_{(3)}^\top = I$. Similarly, it is said to be right orthogonal if its right matricization $\mathcal{G}_{(1)}$ is orthogonal i.e. $\mathcal{G}_{(1)}{\mathcal{G}_{(1)}}^\top = I$:*



Any core can be decomposed into two parts, one of which is orthogonal, using QR decomposition as demonstrated below:



QR decomposition of $\mathcal{G}_{(3)}$



QR decomposition of $\mathcal{G}_{(1)}$

As shown in Figure 4, by using QR decomposition on a TT decomposition it's possible to propagate the left/right orthogonality up until the last core by recursively doing the the QR decomposition of a core, merging the R part with the next core which we then QR-decompose and so on.

Using this strategy, an efficient SVD decomposition can be designed. As shown in Figure 5, QR decomposition can be made on either side of the TT decomposition of the tensor $\mathcal{T}$ up to the middle where we do an SVD decomposition of the central matrix (which resulted from the multiplication of last R propagated from the left and right orthogonalization). Taking the result of the decomposition as $\mathbf{U}\mathbf{D}\mathbf{V}^\top$, it can be seen that everything left of $\mathbf{D}$ is left orthogonal and then can be regrouped as $\hat{\mathcal{U}}$ which is still left orthogonal (since the contraction of a left orthogonal tensor stay left orthogonal, and respectively for right orthogonal tensors; see Figure 6). Similarly, the same thing can be done with the right side and regrouped as $\hat{\mathcal{V}}^\top$. We then have an SVD decomposition $\hat{\mathcal{U}}\hat{\mathcal{D}}\hat{\mathcal{V}}^\top$ of $\mathcal{T}$ (where $\hat{\mathcal{D}}$ is simply $\mathcal{D}$).

Looking at the complexity of each step, we have that each QR decomposition takes $\mathcal{O}(R^3 d)$ since the cores are

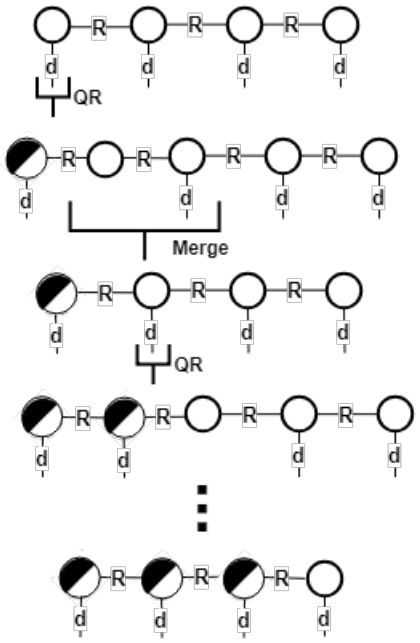


Figure 4: QR propagation

of size $R \times d \times R$, which in total takes $\mathcal{O}(pR^3d)$ (where p is the order of the tensor). As for the SVD decomposition the complexity is in $\mathcal{O}(R^3)$. The overall complexity is then $\mathcal{O}(pR^3d)$. We see that using this strategy is efficient being only linear with the dimension of the tensor.

4 Learning with Tensor-Trains

4.1 Background

Goal: In this section, we are interested in learning a linear function f, which maps from a high-dimensional d^m space to a scalar,

$$f : \mathbb{R}^{d^m} \longmapsto \mathbb{R}. \tag{1}$$

This mapping is learned by fitting a weight matrix $\mathbf{W}$, and predictions are then performed by a product with the input $\mathbf{x}$,

$$f(\mathbf{x}) = \mathbf{W}\mathbf{x}, \tag{2}$$

where $\mathbf{W} \in \mathbb{R}^{1 \times d^m}$.

In order to learn the mapping, we would like to follow the standard procedure of defining a loss function, and performing gradient descent over the parameters, in this case the elements of $\mathbf{W}$, to minimize our loss over a given training set of data.

Standard learning setup: In general, given a set of (lower-)dimensional training data $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \cdots, (\mathbf{x}_N, \mathbf{y}_N)\}$, with $\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^o$, and a specified loss function $\ell(\mathbf{x}, \mathbf{y})$, our objective is to minimize the average loss over the training set,

$$\min \frac{1}{N} \sum_{n=1}^{N} \ell(\mathbf{W}\mathbf{x}, \mathbf{y}), \tag{3}$$

where $\mathbf{W} \in \mathbb{R}^{o \times m}$.

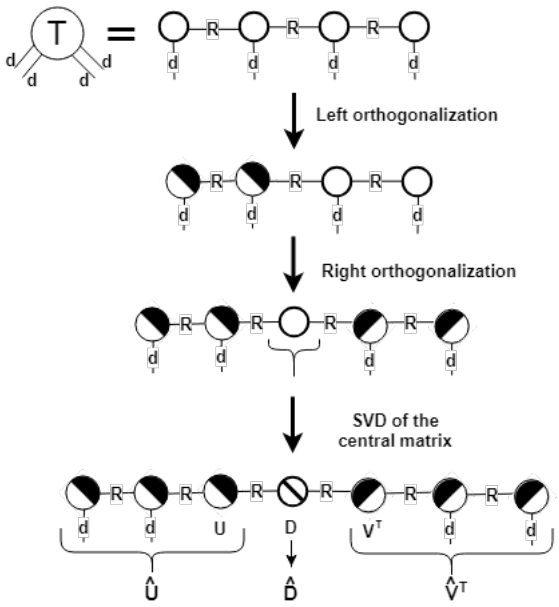


Figure 5: Efficient SVD using QR decomposition

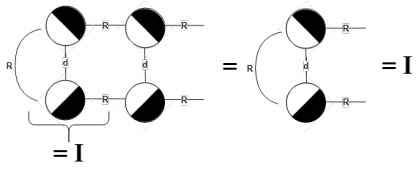


Figure 6: Conservation of left/right orthogonality through contractions of left/right tensors.

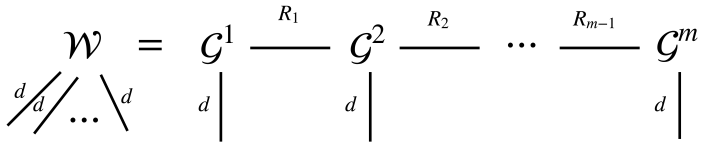
Projecting to higher dimensions: Discriminative tasks become easier in higher-dimensional spaces, with the intuition that it is easier to draw boundaries bewteen data points if they are "pulled apart" through projection into a higher dimensional space. For example, think of trying to draw a linear separator between $\star$s and $\circ$s on a 1-D subspace such that they are placed alternatingly $-\star-\circ-\star-\circ$ (it is impossible). If these points are now projected into a 2-D space such that the $\star$s and $\circ$s are pulled apart in the second dimension, it becomes possible to draw a plane separating the pairs.

4.2 Tensorizing learning

With this motivation for operating in high-dimensional spaces, we shall concern ourselves with the computational challenge of learning in (projected) high dimensions. The solution is, of course, to use tensor factorizations. We shall project $\mathbf{x}$ into a high dimensional space

$$\mathbf{x} \in \mathbb{R}^m \longmapsto \boldsymbol{\mathcal{X}} \in \mathbb{R}^{\overbrace{d\times d\times\cdots\times d}^{m \text{ times}}},$$

and learn $\boldsymbol{\mathcal{W}}$ in the Tensor-Train format, meaning we shall solve (3) under a factorization of $\boldsymbol{\mathcal{W}}$ into a train of cores.



Note: For simplicity, we shall assume here that the predicted variable y is a scalar. All of the discussion extends to the case when it is a vector, with an extra edge added to $\mathcal{W}$.

Projecting $\mathbf{x}$ to $\mathcal{X}$: For projecting $\mathbf{x} \in \mathbb{R}^m$ to $\mathcal{X} \in \mathbb{R}^{d \times d \times \cdots \times d}$, we shall use a dimension-wise projection operator ϕ and compute the final projection with outer-products over the dimension-wise projections as follows,

$$\mathcal{X} = \phi(\mathbf{x}_1) \circ \phi(\mathbf{x}_2) \circ \cdots \circ \phi(\mathbf{x}_m). \tag{4}$$

For example, a $\phi : \mathbb{R} \longmapsto \mathbb{R}^2$ can be constructed as $\phi(a) = \begin{bmatrix} a & 1 \end{bmatrix}^\top$, and the subsequent projection dimension would be 2^m.

Alternating Minimization: Given the tensor factorization of $\mathcal{W}$ (from a choice of ranks $R_1, \cdots, R_{m-1}$), we can perform alternating minimization where we optimize over the subset of parameters in every core at a time until convergence.

Algorithm 2 Alternating Minimization

Require: Initial cores $\mathcal{G}^1, \cdots, \mathcal{G}^m$, loss function ℓ, dataset $\mathcal{D}$, learning rate η
  repeat {over data}
    for each core $\mathcal{G}^i$ **do**
      $\mathcal{G}^i \leftarrow \mathcal{G}^i - \eta \nabla_{\mathcal{G}^i} \ell$
      Optional: Orthogonalize $\mathcal{G}^i$
    end for
  until convergence

DMRG-inspired alternating minimization: Another more flexible algorithm is inspired from the *density matrix renormalization group* in quantum physics, and was presented by Stoudenmire and Schwab in [2].

The basic idea is that alternating style minimization is carried out over coupled pairs of cores. In order to improve computational efficiency further, the update is carried out over the merged pair, but the updated marged-pair is then decomposed back into two cores but under a pre-specified rank-budget and tolerance. This has the added advantage of adaptively learning a low-rank tensor train through training. The algorithm is shown below (red lines denote dimension-edges in the tensor diagram).

Algorithm 3 DMRG-based Alternating Minimization

Require: Initial cores $\mathcal{G}^1, \cdots, \mathcal{G}^m$, loss function ℓ, dataset $\mathcal{D}$, learning rate η, rank budget R_{max}, tolerance ϵ
  repeat {over data}
    for each coupled core pair $\mathcal{G}^i$, $\mathcal{G}^{i+1}$ **do**
      $>\mathcal{B}< = >\mathcal{G}^i - \mathcal{G}^{i+1}<$
      $\hat{\mathcal{B}} \leftarrow \mathcal{B} - \eta \nabla_{\mathcal{B}} \ell$
      $>\hat{\mathcal{B}}< \approx >\hat{\mathcal{G}}^i \underline{R_{max}} \hat{\mathcal{G}}^{i+1}<$ (using truncated-SVD, for example)
      $\mathcal{G}^i \leftarrow \hat{\mathcal{G}}^i; \mathcal{G}^{i+1} \leftarrow \hat{\mathcal{G}}^{i+1}$
    end for
  until convergence

5 Additional utilities

The contraction linear dependency with respect to the tensor dimension of the TT decomposition make it useful in a multitude of situation. Here are a couple of examples:

1. As seen previously, the TT decomposition is an efficient way to linearly solve regression problems by projecting to a higher-dimensional space. The same strategy can be applied in the context of multi-classification algorithms by simply adding an additional dimension to the tensor corresponding the output. By contracting over all the mappings of X, the result is then the output vector where each entry corresponds to one of the classes.

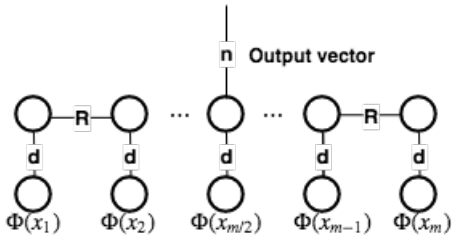


Figure 7: Multi-classification using TT decomposition

2. It can also be used similarly for convolution to connect the pixels of a picture but following a certain pattern. The pattern can be space filling curve, straight up normal convolution like in CNN, or even more complex one like tensor ring which have periodic boundary condition.

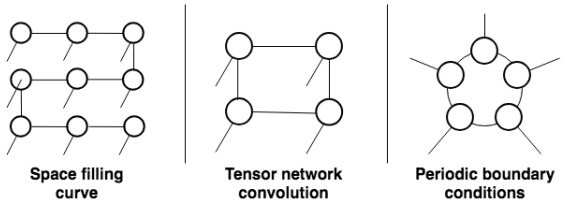


Figure 8: Connection patterns

References

[1] B. Huber, R. Schneider, and S. Wolf. *A Randomized Tensor Train Singular Value Decomposition*. Springer International Publishing, Cham, 2017. ISBN 978-3-319-69802-1. doi: 10.1007/978-3-319-69802-1_9. URL https://doi.org/10.1007/978-3-319-69802-1_9.

[2] E. Stoudenmire and D. J. Schwab. Supervised learning with tensor networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4799–4807. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6211-supervised-learning-with-tensor-networks.pdf.