Title

Interrogating the Tensor Network Regression Model

Permalink

https://escholarship.org/uc/item/0cf9s4n6

Author

Convy, Ian Patrick

Publication Date

2023

Peer reviewed|Thesis/dissertation

Interrogating the Tensor Network Regression Model

By

Ian P. Convy

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Chemistry

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor K. Birgitta Whaley, Chair
Professor Teresa Head-Gordon
Professor Michael Zaletel

Spring 2023

Interrogating the Tensor Network Regression Model

Abstract

Interrogating the Tensor Network Regression Model

by

Ian P. Convy

Doctor of Philosophy in Chemistry

University of California, Berkeley

Professor K. Birgitta Whaley, Chair

There has been growing interest in using tensor networks as machine learning models, inspired by their successes in quantum many-body physics and tensor analysis. These models operate by first mapping input data into an exponentially-large vector space, and then performing linear regression on the resulting feature set. It is well-known that the expressive power of a tensor network regression algorithm originates from its tensor-product featurization, but it is unclear how the tensors in the network are able to convert such a high-dimensional and unstructured intermediate into a useful output. We explore this question by probing the properties of tensor network models on three fronts. First, we assess how the performance of a tensor network classifier degrades when the the size and complexity of the expanded feature space is reduced, and find that most of the space is not effectively utilized by the model. Next, we characterize how the rank of a tensor network impacts the class of regression functions that it can represent, demonstrating that even quadratic polynomials can be impossible to fully realize in most cases. Finally, we use a novel neural network algorithm to determine whether classical images possess correlation structures that mirror those found in quantum wavefunctions, and find evidence of area law scaling in the MNIST and Tiny Images datasets. Taken together, these results demonstrate how mathematical tools from tensor analysis and quantum physics can be leveraged to gain deep insight into the inner workings of tensor network machine learning models.

Contents

Chapter 1

Introduction

Over the last decade, machine learning has seen an explosion in popularity and application, driven largely by the increased availability of massive datasets and the development of powerful graphics processing units (GPUs). The algorithm of choice for modern ML has been the neural network, which is a biology-inspired architecture that can be easily composed end-to-end to create various types of multilayered *deep learning* models [1][2]. Popular examples include the dense feed-forward neural network [3], the convolutional neural network [4][5], and the recurrent neural network [6]. While these techniques have shown remarkable success, solving previously intractable problems in areas such as image classification [7][8], speech recognition [9], and generative modeling [10] among many others, they are hampered by poor interpretability and opaque operation [11]. We know that these models work, but how and why remains elusive. There is demand, therefore, for an alternative machine learning framework that can solve sophisticated problems while also being more amenable to mathematical analysis.

Far away from the field of machine learning[1], research in quantum many-body physics has also seen a boon from the increasing availability of powerful computers [15][16]. While analytical solutions are often a physicist's highest aspiration, there are a large number of problems for which numerical experiments are the main avenue of study. This is especially true in quantum physics due to the intractability of the many-body Schrödinger equation, and the field has therefore amassed an impressive toolbox of computational techniques [17][18]. Many methods revolve around the selection of a so-called *ansatz*, which is effectively just a guess for the mathematical form of the wavefunction that is chosen based on knowledge of the underlying system. An effective ansatz must be flexible enough to capture the phenomena of interest while also being restrictive enough to be simulated efficiently. As with machine learning models, there is a desire for ansatze which are both powerful and interpretable, so that scientific insights can be extracted from the numerical experiments.

An ansatz that fulfills both of these requirements, and that will serve as the object of interest in this thesis, is the tensor network [19]. Tensors, which can be understood as higher-dimensional generalizations of vectors and matrices, occur naturally in quantum physics due to the tensor-product structure of the multi-body Hilbert space [20]. The key

[1] Or not-so-far-away in recent times, as methods from deep learning are increasingly being applied to quantum physics problems [12][13] and vice versa [14].

feature of tensor *networks*, however, is that they can represent extremely large tensors as the contraction of a set of much smaller component tensors. This substitution is critical, as the number of elements in a wavefunction tensor grows exponentially with the number of components in the system, while the number of elements in a tensor network ansatz for that same system grows only linearly. This improvement in scaling necessarily comes at a representational cost, as the ansatz can only reproduce a small portion of the total Hilbert space. That said, a major advantage of tensor networks is that their limitations and modes of operation can be understood rigorously using techniques from tensor analysis, which opens the door for mathematical insights that are much deeper than might be possible for, say, a neural network.

Indeed, the potential benefits of using tensor networks for machine learning in place of neural networks has become an active area of research in recent years. Early efforts starting in 2016 concentrated on the matrix product state [21][22][23], which is one of the most commonly-used tensor networks in quantum physics, but focus has since expanded to a variety of other network types [24][25][26]. Underlying all of these algorithms is the same basic objective: efficiently model the properties of an exponentially-large feature space. Rather than representing multi-body wavefunctions, the tensor network ansatz is instead used to represent a tensor of parameters which are optimized with respect to a specific dataset and machine learning task. The number of elements in this higher-order tensor will in general scale exponentially with the number of features. However, as in quantum physics, we can use the tensor network ansatz to reduce this down to linear scaling.

Although tensor network machine learning is still in its early stages, there have already been impressive results shown on many benchmark machine learning tasks, from image classification [22] to generative modeling [27]. With the increasing interest in tensor network methods, there is little doubt that many more domains will be explored. The purpose of this thesis, however, is not to push on the frontiers of application, but instead to take a step back and ask what is going on “under-the-hood” of these models, so to speak. One of the most attractive aspects of tensor networks is their grounding in well-studied areas of tensor mathematics, as well as the existing knowledge base that has been built around them in the field of quantum physics. We seek to leverage this information in order to probe the operation of tensor networks in a regression setting, with a specific interest in how the models utilize their exponential feature space, and how the right kind of network can be chosen for a given machine learning task.

The body of this thesis is separated into five chapters. In Chapter 2, we introduce the background information on tensors, tensor networks, and regression that will be necessary to understand the rest of the work. Our presentation here will only provide an overview of these three areas, each of which are entire fields of study unto themselves. In Chapter 3, we dive into the practical aspects of constructing an effective tensor network machine learning model, with emphasis on different network architectures, optimization methods, and initialization schemes. This chapter synthesizes knowledge drawn from the existing literature as well from years of experience working with these models.

Chapters 4-6 represent our novel contributions to the field. In Chapter 4, we propose a new method for contracting tensor networks called the *interaction decomposition*. This decomposition breaks up the exponential feature space into subspaces based on the degree of the corresponding feature-product regressors, which allows us to quantify how extensively the model is utilizing the feature space. We also use the interaction decomposition as the basis for a new kind of tensor network model that only performs regression on a portion of the feature subspaces, and demonstrate that these constrained models achieve similar performance to the full models.

In Chapter 5, we establish upper-bounds on the bond dimension necessary to represent multilinear regression up to a given degree, and explore the implications of these bounds on tensor network design. We find that even low-degree regression requires a bond dimension that is impractically large when the coefficients are set arbitrarily, and that the size of a given virtual index is driven almost entirely by feature-products that span the index. We also show that embedding a lower-degree regression function inside a higher-degree function, as can be done using the interaction decomposition, provides a modest decrease in the corresponding bond dimension.

In Chapter 6, we review the entanglement scaling analysis that is used to design tensor network ansatzes in quantum physics, and then adapt it for use on classical machine learning data. In order to quantify the correlations in classical data, we develop a novel machine learning algorithm that uses neural networks to estimate the mutual information between sets of features. We show through numerical experiments that the MNIST and Tiny Images datasets manifest an area-law pattern in their correlations, and discuss the significance of this for tensor network machine learning models. Finally, in Chapter 7, we offer some concluding remarks on our work and consider promising directions for future studies.

The content of this thesis focuses on my work with tensor network machine learning models, and is drawn from the following published papers

- “Interaction decompositions for tensor network regression” [28] (Chapter 4)
- “Mutual information scaling for tensor network machine learning” [29] (Chapter 6)

as well as unpublished material on the tensor rank of multilinear regression models (Chapter 5). In the interest of maintaining a focused and cohesive presentation, I have omitted my work on Bayesian algorithms for quantum error correction, which is published in

- “Machine learning for continuous quantum error correction on superconducting qubits” [30]
- “A logarithmic Bayesian approach to quantum error detection” [31].

Chapter 2

Tensor Network Regression

In this chapter, we provide background information on tensor network regression, which will be key to understanding subsequent chapters. Sec. 2.1 provides an overview of tensors as mathematical objects, with particular focus on tensor operations and tensor rank. Sec. 2.2 introduces the tensor network and its properties, along with a diagrammatic notation that we will use to represent tensor operations throughout this work. Sec. 2.3 describes how tensors can be used for supervised machine learning, and how the tensor network ansatz is crucial to building an effective model.

2.1 Tensors

2.1.1 Perspectives and notation

Throughout this work, we will consider machine learning models that are constructed using *tensors* [32]. We will generally view a tensor as simply a multidimensional array of real numbers, which are referred to as the *elements*, such that each number is indexed by a non-negative integer along every dimension. The *order* of a tensor is equal to the number of dimensions that it has, or equivalently the number of integers needed to specify the position of one of its elements. Under this construction, vectors and matrices are first-order and second-order tensors respectively. We favor this array-based view of tensors because of the emphasis that it places on the tensor elements, which will ultimately serve as the parameters of our regression models.

From a more abstract perspective, a tensor can also be understood as an element of a *tensor product space* [33], which is a composite space formed from the product of multiple vector spaces. Operationally, the product structure of the underlying tensor space manifests in the number and size of the indices, with each vector space being assigned its own index of a size equal to the dimension of the space. This product-space view of tensors lends itself to clearer interpretations of many tensor operations, and we will occasionally make use of it when more theoretical properties are of interest.

One aspect of tensors which we do not consider here, but which is of great significance in other fields such as physics, is their behavior under basis transformations. In physics, tensors are often used to model mathematical relationships between vectors which have been assigned physical meaning outside of a particular coordinate representation. In this context, an elementwise representation of a tensor is meaningful only in conjunction with a specified basis set, which means that two tensors with the same elements can have different meanings if their implied bases differ. From our perspective, however, two tensors with the same elements are always viewed identically, with the understanding that the same orthonormal basis is being used in all cases.

Throughout this work, we will denote tensors with order greater than one using uppercase letters $(A, B, C, ...)$, while vectors will be denoted using a lower case letter under an arrow $(\vec{a}, \vec{b}, \vec{c}, ...)$. Elements of a tensor are specified using subscripts, so that an element of the third-order tensor A is given by A_{ijk}, where i, j, k are non-negative integers. When referring to elements of a vector, the arrow symbol is dropped. We will index starting from one by default, but some dimensions may be indexed from zero for convenience. When indexing using numbers rather than letters, we will place commas between the subscripts for clarity, e.g., $A_{1,13,7}$. Vector spaces will be denoted using double struck letters (generically as $\mathbb{V}$), and sets will be written using calligraphic letters ($\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, ...). To specify the ith member of a set (be it a tensor, vector space, etc), we use a superscript with parentheses, e.g., $A^{(i)}$.

2.1.2 Tensor operations

There are a wide variety of operations that can be defined between tensors, with most simply being generalizations of the familiar matrix and vector operations found in linear algebra. Arithmetic manipulations such as addition and scalar multiplications on tensors can be performed elementwise in the same manner as is done with vectors. For the mth-order tensor A, B, and C, we have

$$C = A + B \;\rightarrow\; C_{i_1 \ldots i_m} = A_{i_1 \ldots i_m} + B_{i_1 \ldots i_m} \tag{2.1}$$

$$C = aA \;\rightarrow\; C_{i_1 \ldots i_m} = aA_{i_1 \ldots i_m}, \tag{2.2}$$

where a is any scalar (we consider only the field of real numbers in this work). These operations do not alter the number or size of the indices, just as the sum two vectors always yields another vector of the same dimension. Note that the addition of two tensors is only meaningfully defined between tensors of the same shape, which means that they must have the same order and have indices of matching size.

Two other important operations, which we will make use of repeatedly throughout this work, are the *tensor product* and the *tensor contraction.* The tensor product $C = A \otimes B$ constructs a new tensor C from every pairwise product between elements of tensor A and elements of tensor B. Each element of the resulting tensor C is given by

$$C_{i_1 \ldots i_m j_1 \ldots j_n} = A_{i_1 \ldots i_m} B_{j_1 \ldots j_n}, \tag{2.3}$$

where the $m + n$ order of C is the sum of the orders of A and B. The tensor product can be understood as a higher-order generalization of the *outer product* from linear algebra, where two vectors (order-1 tensors) are multiplied to generate a matrix (order-2 tensor). Note that if A and B are elements of vector spaces $\mathbb{V}_A$ and $\mathbb{V}_B$ respectively, then C will be an element of the tensor-product space $\mathbb{V}_A \otimes \mathbb{V}_B$.

The tensor contraction between A and B is similar to the corresponding tensor product, except that it generates a new tensor C by taking elements of $A \otimes B$ and summing them along a set of specified dimensions. As an example, if A and B are both third order, then a contraction between the second dimension of A and the third dimension of B is written as

$$C_{i_1 i_3 j_1 j_2} = \sum_k A_{i_1 k i_3} B_{j_1 j_2 k}. \tag{2.4}$$

Note that C is fourth order rather than sixth order, since two of the dimensions of $A \otimes B$ were summed together. The dot product, matrix product, and matrix-vector product can all be understood as contractions between tensors of specific orders.

The final class of operation that we will consider are *reshaping* operations, the most notable of which are *vectorization* and *matricization.* Generically, a reshaping operation takes a tensor and rearranges its elements without changing their values, such that the order of the tensor is altered. This is done by merging multiple indices from the tensor into a single index or, conversely, splitting a single index into multiple indices. In vectorization, all indices of a tensor are merged into a single index, and thus the elements are reshaped into a vector (hence the name). In terms of vector spaces, the vectorization of tensor $A \in \bigotimes_{i=1}^m \mathbb{V}^{(i)}$ can be understood as

$$\mathrm{Vec}[A] = \vec{a}, \quad \text{where} \quad \vec{a} \in \tilde{\mathbb{V}},\ |\tilde{\mathbb{V}}| = \prod_{i=1}^{m} |\mathbb{V}^{(i)}|. \tag{2.5}$$

The dimension $|\tilde{\mathbb{V}}|$ of the new, compound vector space is equal to the product of the dimensions of the merged indices, thus preserving the element number. The mapping between values of the new index and values of the original indices is arbitrary, with a common convention being lexicographic ordering based on the arrangement of the original indices. Matricization is similar to vectorization, except that the tensor indices are first gathered into two disjoint sets, with one group being merged into the row index and the other into the column index. If we let $\mathcal{R}$ and $\mathcal{C}$ denote the sets of vector spaces corresponding to the row indices and column indices respectively, then the matricization of tensor $A \in \bigotimes_{i=1}^m \mathbb{V}^{(i)}$ is

$$\mathrm{Mat}[A] = \tilde{A}, \quad \text{where} \quad \tilde{A} \in \mathbb{V}_{\mathcal{R}} \otimes \mathbb{V}_{\mathcal{C}},\ |\mathbb{V}_{\mathcal{R}}| = \prod_{\mathbb{V}^{(i)} \in \mathcal{R}} |\mathbb{V}^{(i)}|,\ |\mathbb{V}_{\mathcal{C}}| = \prod_{\mathbb{V}^{(i)} \in \mathcal{C}} |\mathbb{V}^{(i)}|. \tag{2.6}$$

Similar to vectorization, the dimensions of the row and column spaces are equal to the product of the dimensions of the spaces in $\mathcal{R}$ and $\mathcal{C}$ respectively.

The utility of vectorization and matricization operations is that they repackage the elements of a tensor in a form that is amenable to analysis and manipulation using tools from linear algebra. Procedures such as the singular-value decomposition can be easily applied to matricized tensors using existing linear algebra software libraries, and interpreted using standard principles from matrix analysis. Properties of the row and column spaces from a matricized tensor can then be mapped back to the tensor product spaces that were associated with the tensor prior to reshaping. When used together, vectorization and matriciziation can be particularly effective at analyzing the contraction of two tensors, as one tensor can be cast as a vector while the other is cast as the matrix which maps it to a new (vectorized) tensor.

2.1.3 Tensor Rank

A key concept in tensor analysis is *tensor rank*, which can be understood as a generalization of the matrix rank (although many properties are not preserved). A rank-one tensor, also referred to as an *elementary* tensor, is a tensor that can be written as the tensor product of order-one tensors (vectors). For example, a third-order elementary tensor D must be writable as

$$D = \vec{a} \otimes \vec{b} \otimes \vec{c} \ \rightarrow \ D_{ijk} = a_i b_j c_k, \tag{2.7}$$

for some vectors $\vec{a}$, $\vec{b}$, and $\vec{c}$. All non-zero vectors have a rank of one trivially, while rank-one matrices and higher-order tensors form sets of measure zero in their respective tensor spaces. From this definition of an elementary tensor, a *rank-r* tensor is a tensor that can written exactly as the sum of r elementary tensors. For example, if the tensor D is an mth-order tensor of rank r, then it must be writable as

$$D = \sum_{i=1}^{r} \vec{v}^{\,(i,1)} \otimes \vec{v}^{\,(i,2)} \otimes ... \otimes \vec{v}^{\,(i,m)}, \tag{2.8}$$

where $\vec{v}^{\,(i,j)}$ is the jth vector component of the ith elementary tensor. This summation is known as the *rank decomposition* of the tensor D. We can see that the definition of rank-one tensors from Eq. (2.7) is consistent with Eq. (2.8), since $r = 1$ sets D equal to a single elementary tensor.

The rank of a tensor provides a measure of its computational and representational complexity. Among third-order tensors, for example, the rank sets the minimum number of vector product operations needed to compute the non-commutative bilinear forms parameterized by a given tensor [34], which in turn sets limits on the asymptotic computational costs associated with matrix multiplication [35]. In the context of numerical representation and storage, the tensor rank places limits on how accurately the elements of a tensor can be compressed into a smaller number of parameters.

While the tensor rank has the appealing attribute of being a direct generalization of the matrix rank, it has number of practical drawbacks. The most significant issue is that there does not exist any systematic algorithm that can solve for the rank of an arbitrary tensor. Similarly, there does not exist any general method to find the best rank-k approximation for a tensor of rank $r > k$, and for some tensors a best approximation may not even exist [36]. Furthermore, numerical methods for computing low-rank approximations can often be fail to converge to the optimal solution, since the optimization problem is inherently non-convex.

2.2 Tensor Networks

2.2.1 The "curse of dimensionality"

The most significant obstacle to working with higher-order tensors is the so-called "curse of dimmensionality" [37], where the size and complexity of a tensor-based task scales exponentially with the order of the tensor. This scaling occurs because the number of elements in a tensor is given by the product of each index dimension, so a tensor of order m and fixed index dimension t will have t^m elements. Any task which involves modeling interactions or relationships among a large number of component spaces, such as the multibody systems in quantum physics or high-dimensional partial differential equations [38], will suffer from this curse.

In practice, we are afflicted with the curse of dimensionality whenever we attempt to construct or manipulate an explicit representation of the elements in a higher-order tensor. If, however, the elements are instead represented as functions of some smaller set of parameters, then the curse can be avoided at the cost of working with a more constrained class of tensors [39]. We have already seen an example of this in the rank decomposition from Eq. (2.8), where a tensor of order m, rank r, and index dimension t is represented using $r \cdot m \cdot t$ vector parameters. As long as the rank of the tensor does not scale exponentially with the order, then $r \cdot m \cdot t \ll t^m$ for large m and the curse of dimmensionality is lifted. Note that the rank of a typical (i.e. randomly sampled) tensor *does* scale exponentially with the order [40], so we avoid exponential scaling at the cost of working with a set of tensors that is usually measure zero in the tensor space. This is often a worthwhile trade off, as many systems of practical interest can be accurately represented using low-rank tensors.

The construction of low-rank tensor representations is the primary purpose of *tensor networks*, which have been leveraged in a variety of different fields as a solution to the curse of dimensionality [41][42][43]. At its most fundamental level, a tensor network is simply a collection of tensors, which we call *component tensors*, along with a set of instructions that describes how to contract those tensors into a new *output tensor*. The output tensor can in principle be of any size, but the utility of tensor networks comes from their ability to build higher-order tensors out of lower-order components. If the number of components in the network is proportional to the order of the output tensor[1], which is typical, the component tensors will collectively contain exponentially fewer elements than the output tensor.

As an introductory example, we can consider a very simple tensor network, consisting of only two component tensors A and B, which is contracted together to generate the output tensor C:

$$C_{i_1 i_1 i_3 j_1 j_3 j_4} = \sum_{k=1}^{t} A_{i_1 i_2 i_3 k} B_{j_1 k j_3 j_4}. \tag{2.9}$$

An expression this simple would almost never be referred to as a “tensor network” in practice, but we can see that it does fullfill both of the previously stated requirements: it has a collection of (lower-order) component tensors $\{A, B\}$, and it instructs us to contract the fourth index of A with the third index of B to construct a sixth-order tensor C. Assuming that each index in Eq. (2.9) is of size $t > 2$, it is clear that tensor C has significantly more elements (t^6) than are contained in A and B combined ($2t^4$). As the number of component tensors and the order of the output tensor increases, the disparity between component elements and output elements becomes far more dramatic.

[1] Strictly speaking we would only need the scaling to be sub-exponential, but virtually all tensor networks used in practice have a number of components that is directly proportional to the order of the output

2.2.2 Tensor network notation

Before considering more complicated networks, it is helpful to introduce a type of index-free tensor notation that can be used to easily represent contractions involving many tensors. In this new notation, which is often called *Penrose notation* or simply referred to as *tensor diagrams*, each tensor is denoted using a geometric shape, while each index is represented by a line or *leg* protruding outward from the shape. A tensor product is implied by placing two tensors next to one another, and a contraction is indicated by having those tensors share one or more legs. A significant advantage of this approach is that there is no need to keep track of so-called "dummy" or *virtual* indices, such as k in Eq. (2.9), which serve only to coordinate the different contractions. We typically do not label the uncontracted or *physical* indices of the output tensor either, since whatever meaning they have is usually clear from the surrounding context.

To practice using this new notation, we can see how familiar tensors and tensor operations from linear algebra are written using both explicit summations and tensor diagrams:

$$\text{Vector:} \quad \vec{v} \to \boxed{v}\!-\qquad\qquad \text{Matrix:} \quad A \to -\!\boxed{A}\!-$$

$$\text{Dot product:} \ \sum_i v_i w_i \to \boxed{v}\!-\!\boxed{w} \qquad \text{Matrix product:} \ \sum_j A_{ij} B_{jk} \to -\!\boxed{A}\!-\!\boxed{B}\!-$$

Notice that the order of the output tensor can be determined by simply looking at the number of unpaired legs in the diagram: zero for the scalar-valued dot product and two for the matrix-valued matrix product.

The utility of tensor diagram notation for depicting tensor networks can seen clearly in Figure 2.1, which shows a network with five component tensors. Even with a relatively simple contraction pattern, the explicit index notation is highly obtuse, requiring that we laboriously trace through and remember all nine index labels, including those of the four virtual indices. The diagram, by contrast, reveals the contraction pattern immediately by tapping into our natural ability to parse spatial arrangements. We will utilize tensor diagrams, along with some explicit index expressions, throughout the remainder of this work to help illustrate the relevant tensor operations.

2.2.3 Tensor network contractions

Our discussion of tensor networks so far has centered on their ability to represent the elements of a higher-order tensor using a much smaller number of component tensor elements. But it is worth considering further what it means to "represent" the elements of a tensor. If nothing else, accurate representation should necessarily mean that we can contract the component tensors of the network and retrieve precisely the elements that we would expect in the output tensor. For small output tensors we could indeed carry out this operation in practice, but for large, higher-order tensors—fully afflicted with the curse of dimensionality—a full contraction of the network can only ever be imagined theoretically.

$$H_{ijkop} = \sum_{l,m,n,r=1}^{t} D_{lim} E_{jmrk} F_{lrn} G_{nop}$$

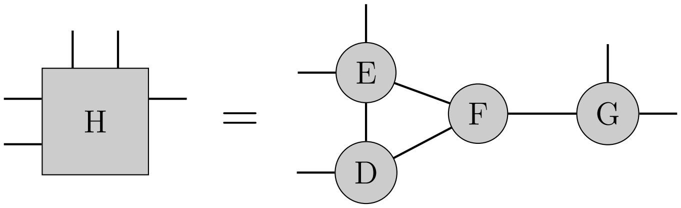
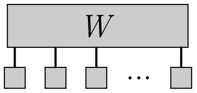


Figure 2.1: Tensor network representing fifth-order tensor H, depicted using an explicit summation (top) and a tensor diagram (bottom). The network is generated by four different contractions, each of which is indicated in the diagram by a shared leg. While it is possible to discern the contraction pattern of the component tensors by studying the index notation, the diagram makes it obvious at a glance.

So what then is the computational utility of a tensor network? The answer lies in considering contraction operations between the higher-order tensor and one or more low-order tensors. To give a concrete example that will come up repeatedly throughout this work, consider the contraction between an mth-order tensor W and a set of m vectors $\{\vec{x}^{\ (i)}\}_{i=0}^{m-1}$ each of dimension t:

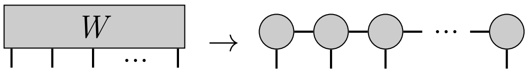
$$\sum_{i_1,\ldots,i_m=1}^{t} W_{i_1\ldots i_m} x_{i_1}^{(1)} x_{i_2}^{(2)} \cdots x_{i_m}^{(m)} \ \rightarrow$$



. (2.10)

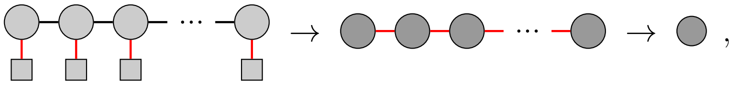
For small m and modest t this calculation is manageable, but as the order of W increases the curse of dimensionality sets in and we will rapidly exhaust our computational resources. Even if we had some algorithm for generating each element $W_{i_1\ldots i_m}$ on the fly, there would still be t^m terms in the sum that would need to be evaluated. Without further information about the structure of W, we cannot make any progress in evaluating Eq. (2.10).

However, let us now say that W can be represented exactly by a tensor network with a reasonable number of component tensor elements. To be more concrete, we will suppose that W can be represented by the contraction of a sequence of matrices and third-order tensors according to the following tensor diagram:



. (2.11)

The total number of elements across all component tensors scales as $\mathcal{O}(mtr^2)$, where r is the size of the virtual indices. These kind of linear tensor networks are called matrix product states (MPS) or tensor train decompositions, and we discuss them further in Sec. 3.1.2. For now, the important thing to notice is that each of the physical indices on W is has been localized to a single low-rank tensor. If we were to first attempt a complete contraction of the network components to recover W, we would run into precisely the same curse of dimensionality that stopped us before. Alternatively, we could first contract each $\vec{x}^{\,(i)}$ with its corresponding component tensor, which would generate a new network consisting of matrices and vectors that could be easily contracted. We can illustrate this procedure using tensor diagrams:

 (2.12)

where the red legs are contracted in each step, the darker shapes indicate tensors generated from the contractions in the previous step (which we refer to as *intermediate tensors*), and the circle at the end with no legs represents the scalar output.

The key takeaway from Eq. (2.12) is that we were able to preform a previously intractable calculation involving a higher-order tensor by utilizing its tensor network representation. The reason for this success is that the component tensors of the MPS were localized to individual indices, which allowed us to contract all of the vectors individually with their corresponding component tensors *before* contracting the remainder of the network. This prevented the order of the intermediate tensors from growing too large, which is recurring theme when finding effective contraction strategies for tensor network operations.

2.2.4 Bond dimension and multiplex rank

When representing a higher-order tensor using a tensor network, there are certain constraints that need to be considered. The most obvious of these is that the number and size of the physical indices in a tensor network must match the number and size of the indices in the represented tensor. Beyond this, there is also a set of more subtle constraints that are imposed on the size of the virtual indices, which is often referred to as the *bond dimension* from the use of tensor networks in quantum physics. In Sec. 2.2.3 we made passing reference to the bond dimension r of the MPS when describing the number of elements in the network, and it plays a similar role in other networks by determining the number of elements in the component tensors.

Since a larger bond dimension implies a larger and thus more computationally-intensive tensor network, it is natural to look for the smallest possible bond dimension that still allows for an exact representation of a given tensor. This bound is given by the *mulitplex rank* of the tensor [44], which is related to but distinct from the tensor rank discussed in Sec. 2.1.3. The multiplex rank is based on the matricization operation introduced in Sec. 2.1.2, and is defined to be the rank of the matrix generated by matricizing a tensor with respect to a specified bipartition of the indices. The number of distinct multiplex ranks for an mth order tensor is given by $\sum_{i=1}^{\lfloor m/2 \rfloor} \binom{m}{i}$, which is the number of possible bipartition schemes for the m indices.

The multiplex rank establishes a lower bound on the size of the virtual indices in a tensor network via its relationship with the singular value decomposition (SVD) [45]. In an SVD, an $\ell \times w$ matrix M is decomposed into the product of an $\ell \times \ell$ unitary matrix U, a $w \times w$ unitary matrix V, and an $\ell \times w$ diagonal matrix Σ with non-negative elements:

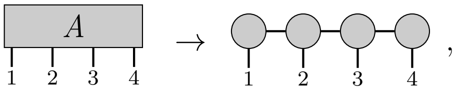
$$M_{ij} = \sum_{q,l} U_{iq} \Sigma_{ql} V_{lj}, \quad \mathrm{diag}(\Sigma) = [\sigma_1, \sigma_2, ..., \sigma_{\min(\ell,w)}], \tag{2.13}$$

where σ_i is the ith singular value of M. A key property of the SVD is that the number of non-zero singular values is equal to the rank r of the matrix, which can be seen by rewriting Eq. (2.13) as

$$M_{ij} = \sum_{q,l} U_{iq}\Sigma_{ql}V_{lj} \;\rightarrow\; M_{ij} = \sum_{n=1}^{r} \sigma_n U_{in}V_{nj} \;\rightarrow\; M = \sum_{n=1}^{r} \sigma_n \vec{u}^{\,(n)} \otimes \vec{v}^{\,(n)}, \tag{2.14}$$

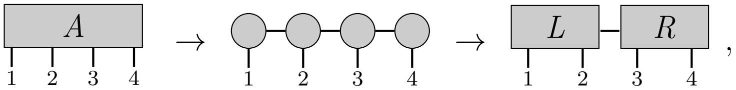
where the final expression matches the form of the rank decomposition from Eq. (2.8) with $\vec{u}^{\,(n)}$ as the nth column of U and $\vec{v}^{\,(n)}$ as the nth row of V.

To make the connection between tensor networks and SVDs, we can consider as an example the fourth-order tensor A being represented by an MPS of bond dimension k:



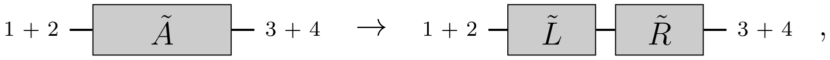
(2.15)

where the physical indices have been labeled for clarity. If we partition the indices into two disjoint sets $\{1, 2\}$ and $\{3, 4\}$ and then contract together components *within* the two partitions, we will have



(2.16)

where there is now a pair of tensors L and R corresponding to the two partitions, and a virtual index connecting them together. If we then matricize A by merging indices 1, 2 into the row index and indices 2, 3 into the column index, and similarly matricize L and R, we get



(2.17)

where $\tilde{A}$, $\tilde{L}$, and $\tilde{R}$ are the matricized versions of A, L, and R respectively. The matrix product in Eq. (2.17) can be written out using explicit index summations in a form very similar to the SVD expression from Eq. (2.14):

$$\tilde{A}_{ij} = \sum_{n=1}^{k} \tilde{L}_{in}\tilde{R}_{nj} \;\rightarrow\; \tilde{A} = \sum_{n=1}^{k} \vec{l}^{\,(n)} \otimes \vec{r}^{\,(n)}, \tag{2.18}$$

where $\vec{l}^{\,(n)}$ is the nth column of $\tilde{L}$ and $\vec{r}^{\,(n)}$ is the nth row of $\tilde{R}$. Since the matrix rank r of $\tilde{A}$ is the smallest number of elementary tensors that can be added together to generate $\tilde{A}$, we know that $k \geq r$ must hold for the MPS from Eq. (2.15) to exactly represent $\tilde{A}$. The matrix rank of $\tilde{A}$ is just the multiplex rank of A with respect to the $\{1, 2\}$ and $\{3, 4\}$ partition of the physical indices, so therefore the multiplex rank lower bounds the bond dimension of the virtual index.

While the steps that we worked through in Eqs. (2.15 - 2.18) were built around a specific example, the broader concept is the same across all tensor networks. For a given partitioning of the physical indices, we can always identify component tensors that either themselves contain physical indices from a single partition or are connected to such components[2]. By contracting these tensors together, we can generate the L and R tensors from Eq. (2.16), and then matricize them with respect to the physical and virtual indices as we did in Eq. (2.17). For a generic network and index partitioning, we can expect to merge multiple virtual indices into the column index of L and row index of R, in which case the multiplex rank bound applies collectively to the product of their bond dimensions. Bounds for different sets of virtual indices can be generated by simply varying the choice of physical index partitions.

2.2.5 Approximate representations

The discussion of multiplex rank and bond dimension bounds in Sec. 2.2.4 is of significant theoretical value, and we will make use of it extensively in Chapter 5, but in practice we are generally not interested in *exact* tensor network representations. For a typical tensor, the multiplex rank grows exponentially with the order, so exact representations are computationally intractable and largely defeat the purpose of using tensor networks in the first place. Instead, our approach will be to *approximate* some tensor of interest using a reasonably-sized network, with the goal of capturing its relevant properties. This is usually done by fixing the form of the network in advance, and then choosing its components based on some measure of similarity with the target tensor.

A common method for constructing tensor network approximations is the *truncated SVD*, which involves computing the SVD for a matricizied tensor as in Eq. (2.13), but then setting some of the singular values to zero. This leads to a truncation in the rank decomposition of the matrix, and therefore to a decrease in the multiplex rank of the resulting tensor. The number of discarded singular values is often chosen such that only the k largest are preserved, where k is the desired bond dimension of the tensor network. The tensor generated via this truncation procedure is the best approximation (with respect to the ℓ^2 norm) of the original tensor among all tensors with a multiplex rank of k. The truncated SVD has been used to generate tensor networks used in compression [46][47] and quantum physics [48], and has the advantage of provable error bounds for many network types [49][50].

Another approach, which we will utilize in this thesis, is to optimize the elements of the network via gradient descent to minimize a chosen cost function. This cost function could simply be the norm of the difference between the output tensor of the network and a target tensor, but more often it is tied directly to the solution of some task. In this latter case, the tensor we are trying to represent is defined only implicitly as the minimizer of the cost function, with our objective being to find the best low-rank approximation that can be generated by our network. There are many different gradient-based methods that can be used for optimization, and we survey three of them in Sec. 3.2.

[2] It is possible that a single component may have physical indices from *both* partitions. In that case, the partitions should be modified so that all of those indices are in the same set, as otherwise a bound cannot be determined using this method.

2.3 Tensor Regression

2.3.1 Image classification and supervised machine learning

The machine learning task that will be considered throughout this thesis is *image classification*, in which we seek a model that can correctly categorize the content of pixel-valued images. We focus specifically on the classification of black and white images, which means that each image is taken to be a matrix of positive numbers in the range $[0, 1]$, with each element representing a grayscale pixel that lies somewhere between pure black (value 0) and pure white (value 1).

From a mathematical perspective, the image classification task can be distilled down into a *regression* problem. In regression, the goal is to learn (or estimate) the relationship between a set of m independent variables $\{x_i\}_{i=1}^{m}$ called *features* and a set of n dependent variables $\{y_i\}_{i=1}^{n}$ called *labels.* For image classification, the features are the pixel values and the labels are the conditional probabilities of each category. We denote a joint sample of these $m+n$ variables as $(\vec{x}, \vec{y})$, where $\vec{x} \in \mathbb{R}^m$ is a vector containing the values of the features and $\vec{y} \in \mathbb{R}^n$ is a vector containing the values of the labels. In the case of *parametric* regression, the relationship between $\vec{x}$ and $\vec{y}$ is modeled by a function $\vec{f}$ such that

$$\vec{y} \approx \vec{f}(\vec{x}; \mathcal{W}), \tag{2.19}$$

where $\mathcal{W}$ is a set of parameters which determines the behavior of the function. We do not expect the relationship in Eq. (2.19) to be exact for any intuitive function $\vec{f}$, except when the data is generated artificially. Indeed, an exact reconstruction will generally be undesirable for real-world data, since the labels often contain noise that should not be directly copied into the model.

To find an effective set of parameters $\mathcal{W}$, we perform optimization under the *supervised learning* paradigm [51]. In supervised learning, the model is presented with a set of *training data* that consists of η samples $\{(\vec{x}^{\,(i)}, \vec{y}^{\,(i)})\}_{i=1}^{\eta}$, where the labels $\vec{y}^{\,(i)}$ have been associated with the features $\vec{x}^{\,(i)}$ through an accurate but often resource-intensive process. During optimization, each feature vector is fed into the model, generating a prediction $\hat{y}^{(i)} \equiv \vec{f}(\vec{x}^{\,(i)}; \mathcal{W})$ for the associated label vector. The similarity of the prediction $\hat{y}^{(i)}$ to ground-truth $\vec{y}^{\,(i)}$ is evaluated using a loss function, and then the parameters $\mathcal{W}$ are altered via an optimization algorithm so that the value of $\hat{y}^{(i)}$ more closely matches that of $\vec{y}^{\,(i)}$. Generally this optimization procedure is performed multiple times across the entire training set, in steps known as *epochs.* We discuss a specific loss function and set of optimization algorithms for tensor-based machine learning models in Sec. 3.2.

Once a regression model has been trained, the performance of the optimized parameters can be evaluated using a set of *test data*, which consists of labeled samples that were not used during the training of the model. It is common for the average loss value of the test data to be significantly worse than the loss value for the training data, a phenomenon known as *overfitting.* The degree of overfitting is tied to both the size η of the training set and the number of parameters $\mathcal{W}$ in the model, with smaller η and larger $\mathcal{W}$ leading to worse overfitting. Overfitting can be mitigated to some extent by placing regularization conditions

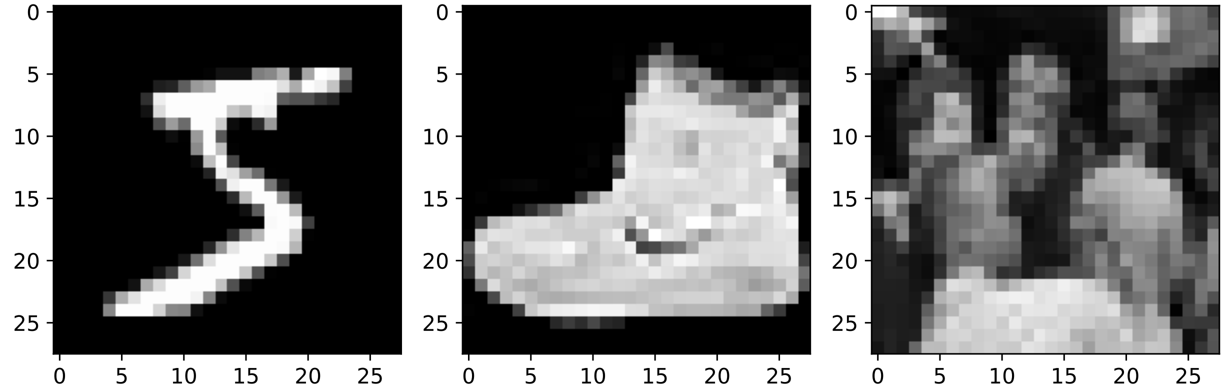


Figure 2.2: Left: The handwritten digit "5" taken from MNIST. Center: An ankle boot image taken from Fashion MNIST. Right: Low-resolution image taken from the Tiny Images dataset, after having been cropped and converted to grayscale. The axes give the height and width of each image in pixels.

on the parameters—which reduce the effective degrees of freedom of the model—but it is difficult to eliminate without an incredibly large training set or a very simple model.

Throughout this thesis we train and evaluate our models on the MNIST [52], Fashion MNIST [53], and Tiny Images datasets [54], which are well-known benchmarks in the field of computer vision. MNIST contains seventy-thousand 28×28 images of handwritten digits 0 - 9, and is one of the most well-studied image datasets. Fashion MNIST was designed to be a more challenging replacement for MNIST, and therefore also has seventy-thousand 28×28 images but with ten different articles of clothing instead of digits. The Tiny Images dataset[3] contains 80 million 32×32 images scraped from a wide variety of web sources, and has the most complex images of the three datasets. Example images from the three datasets are given in Figure 2.2.

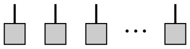
2.3.2 Tensor models

Tensor network regression, which is our ultimate aim, can be understood as a specific form of *tensor regression* [55]. In tensor regression, the function $\vec{f}$ of Eq. (2.19) is expressed as the contraction of a data tensor $X(\vec{x})$, which is a function of the features in a given sample, and a weight tensor W whose elements make up the set of parameters $\mathcal{W}$. The data tensor can, in principle, take on any form, but it is usually constructed from the tensor product of

[3] This dataset was withdrawn by its creators in June of 2020 due to concerns about prejudicial biases in the images. We have no reason to believe that those issues are relevant to our specific use of it here.

a set of m vector-valued functions $\{\vec{h}^{(i)}\}_{i=1}^{m}$ that each take as input a single feature:

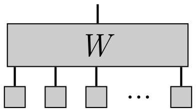
$$X(\vec{x}) = \bigotimes_{i=1}^{m} \vec{h}^{(i)}(x_i) \;\; \rightarrow \tag{2.20}$$



,

where x_i is the ith element of $\vec{x}$ and thus the ith feature out of m. Note that the sequence of tensor products in Eq. (2.20) is similar to a tensor network, insofar as it expresses a higher-order tensor X using a set of order-one components which collectively contain exponentially fewer elements. The regression output $\vec{f}(\vec{x}; \mathcal{W})$ is computed by contracting the weight tensor W with X:

$$f_k(\vec{x}; \mathcal{W}) = \sum_{i_1,\ldots,i_m} W_{k i_1 \ldots i_m} X_{i_1 \ldots i_m}(\vec{x}) \;\; \rightarrow \tag{2.21}$$



,

where W contains an additional dimension k that indexes the output vector of the model. This form of regression is quite distinct from the standard deep learning paradigm, in that the transformation of the data is effectively set in advance here via the data tensor featurization $X(\vec{x})$. All that is then left to optimize are the coefficients $W_{k i_1 \ldots i_m}$ that should be assigned to each of the new regressors. The same delineation cannot in general be made for deep learning models, since they are formed from a composition of non-linear functions that has no clear relation to any series expansion.

The precise role that the original features $\{x_i\}_{i=1}^{m}$ play in Eq. (2.21) depends on the form of X and therefore on the set of functions $\{\vec{h}^{(i)}\}_{i=1}^{m}$ that were chosen. Conventional wisdom regarding this choice can be traced back to the parallel works of Stoudenmire and Schwab [22] and Novikov et al. [21], who each proposed a different transformation scheme. The method from [22] was inspired by techniques in quantum many-body physics, and mapped each feature $x \in [0, 1]$ into the L2-normalized vector $[\cos(\frac{\pi}{2}x),\ \sin(\frac{\pi}{2}x)]$. The approach in [21], by contrast, was motivated by a desire to characterize interactions within categorical (discrete) data, and therefore had each feature mapped to the vector $[1,\ x]$. The advantage of this latter mapping is that every element of the transformed feature space is a product of some subset of the original features, which makes the resulting regression output easier to interpret.

For our work here, we follow Novikov et al. and use functions of the form

$$\vec{h}^{(i)}(x_i) = \begin{bmatrix} 1 \\ x_i \end{bmatrix}, \tag{2.22}$$

which have been used in other implementations of tensor network regression [56][57][58]. When Eq. (2.22) is used to construct X, the regression function $\vec{f}(\vec{x}; \mathcal{W})$ from Eq. (2.21) becomes

$$f_k(\vec{x}; \mathcal{W}) = \sum_{i_1,\ldots,i_m=0}^{1} W_{k i_1 \ldots i_m} x_1^{i_1} x_2^{i_2} \cdots x_m^{i_m}, \tag{2.23}$$

where indices $i_1, \ldots, i_m$ are zero-indexed and $0^0 = 1$ is assumed. Eq. (2.23) shows that tensor regression, when using the definition of $\vec{h}^{(i)}(x_i)$ from Eq. (2.22), is equivalent to linear regression on all possible products formed between the original features, plus a bias term when all of the indices are zero. These feature products are referred to as *interactions*, with the *degree* of an interaction defined to be the number of features that are multiplied together (e.g. interactions of degree three take the form $x_i x_j x_k$, with $i \neq j \neq k$). Regression on the elements of X can generate functions $\vec{f}$ which have non-zero mixed derivatives with respect to the original features. For example,

$$\frac{\partial^2}{\partial x_1 \partial x_2} f_k(\vec{x};\mathcal{W}) = \sum_{i_3,\ldots,i_m=0}^{1} W_{k,1,1,i_3,\ldots,i_m} x_3^{i_3} x_4^{i_4} \cdots x_m^{i_m}. \tag{2.24}$$

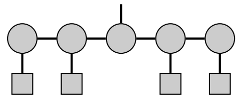
These non-zero derivatives make $\vec{f}$ significantly more expressive than functions generated by linear regression directly on the original features in $\vec{x}$, for which all mixed derivatives must vanish.

2.3.3 The tensor network ansatz

The most immediate challenge to implementing a tensor-based regression model is, as always, the curse of dimensionality. Inspection of Eq. (2.23) reveals that there are 2^m parameters in W, which is exponential in the number of features. Given that our benchmark image datasets contain 784 features, this method of parameterization is simply a non-starter. However, as discussed in Sec. 2.2, we can avoid the exponential scaling of W by representing its elements implicity via a tensor network. We refer to this as the *tensor network ansatz*, and it will allow us to construct machine learning models that follow the same tensor regression formalism introduced in Sec. 2.3.2 while using a more realistic amount of computational resources.

Under a tensor network ansatz, the parameters $\mathcal{W}$ of the regression function $\vec{f}$ shift from being elements of the weight tensor W to instead being elements of the component tensors in the network. This does indeed have the benefit of dramatically reducing the cost of contraction, but it comes with a significant loss of element-wise interpretability. As an example, if an MPS is used to represent W for data with four features, then the output vector $\vec{f}$ has the form

$$f_k(\vec{x};\mathcal{W}) = \sum_{i_1,\ldots,i_4=0}^{1} \sum_{j_1,\ldots,j_4=1}^{r} A_{i_1 j_1} A_{j_1 i_2 j_2} A_{j_2 k j_3} A_{j_3 i_3 j_4} A_{j_4 i_4} x_1^{i_1} x_2^{i_2} x_3^{i_3} x_4^{i_4} \;\rightarrow$$



$$, \tag{2.25}$$

where we have simply taken Eq. (2.23) for $m = 4$ and replaced W with an explicit index expression for the MPS. Note that each feature-product coefficient is now a function of multiple parameters from the tensor network model, as opposed to just a single element of W. This parameterization structure has a significant impact on model optimization, which we discuss in Sec. 3.2, and also serves as a motivation for the interaction decomposition that we explore in Chapter 4.

Thus far, we have argued that the tensor network ansatz is necessary for tensor regression due to the fact that we cannot explicitly represent or manipulate the exponentially large weight tensor W. While this is certainly true, such an argument can leave the mistaken impression that we would voluntarily choose to work with W if only we had a computer big enough to hold its elements. Under this view, the tensor network ansatz is a kind of necessary evil that we are forced to make do with given our computational constraints, and we could hope that perhaps some future advancement in technology will allow us to do away with it as a representional crutch.

In reality, however, the tensor network ansatz also plays a critical role in the *generalizeability* of the tensor regression model. Generalization in machine learning describes the ability of a model to make accurate predictions for data that it was not trained on. This is precisely what the test dataset is designed to evaluate, and it is of critical importance for any task where one expects to see novel data samples. To illustrate how a model built directly on W can be very poor at generalization, we will consider the task of classifying images which have pixels that take on binary values, i.e., $x_i \in \{0, 1\}$. For grayscale images, this translates into each pixel being either pure white or pure black, which is fairly close to how images appear in the MNIST dataset (see Figure 2.2). If we were to use the $[\cos(\frac{\pi}{2}x),\ \sin(\frac{\pi}{2}x)]$ featurization from Stoudenmire and Schwab [22], then Eq. (2.20) would become

$$X(\vec{x}) = \bigotimes_{i=0}^{m-1} \begin{bmatrix} 1 - x_i \\ x_i \end{bmatrix}, \qquad \text{e.g.,}\ \ X(\vec{0}) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes ... \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \tag{2.26}$$

where $\vec{0}$ would represent a completely black image. Note that $X(\vec{x})$ always maps the binary feature vector $\vec{x}$ to a standard basis vector, which means that the contraction of W with $X(\vec{x})$ simply pulls out the element of W corresponding to the binary string encoded in $\vec{x}$. The regression function is therefore given by

$$f_k(\vec{x}; \mathcal{W}) = \sum_{i_1,\ldots,i_m=0}^{1} W_{ki_1\ldots i_m} X_{i_1\ldots i_m}(\vec{x}) = W_{kx_1\ldots x_m}. \tag{2.27}$$

From Eq. (2.27), we can begin to see the problem: the output for each image is determined by only a single parameter, and that parameter is different for each distinct image. During training, parameters will only be optimized if they play a role in determining the output for samples in the training data, so any parameter associated with images outside of the training set will not be optimized at all. This eliminates any capacity for generalization, and thus renders the model ineffective at anything other than memorization.

By contrast, Eq. (2.25) shows that each element of W is associated with a large number of parameters in the MPS model, and, conversely, that each model parameter appears in the representation of many different elements of W. This means that optimizing the model with respect to one binary image will update a significant fraction (exactly half in this case) of the network parameters, and implies that the prediction for a test-set image will be generated using parameters that also played a role in generating predictions for the training-set images. Through this sharing of parameters, tensor network models are able to easily learn on binary images or on any other near-orthogonal dataset.

Chapter 3

Building a Tensor Network Regression Model

In this chapter, we discuss different practical considerations that go into constructing a tensor network regression model. Sec. 3.1 surveys the properties of four different tensor network architectures, starting with the popular MPS model and then moving on to tree tensor networks, projected entangled pair states, and the newly-proposed spanning tree networks. Sec. 3.2 describes the how tensor network models can be trained using Riemannian optimization, DMRG-style sweeps, and stochastic gradient descent, all of which have been employed in the existing literature. In Sec. 3.3, we provide some novel discussion about the challenges of initializing component elements in a tensor network, and outline methods for initialization that we have used successfully.

3.1 Network Architecture

3.1.1 General considerations

When constructing a tensor network regression model, the most immediate question is what type of network design to use for the representation of W. Our discussion of the tensor network ansatz in Sec. 2.3.3 was deliberately vague on this point, but here we seek to provide more practical guidance on the advantages and disadvantages of different architectures. There exist many axes upon which we can compare different tensor networks, but we will focus primarily on the number of model parameters, the ease of contraction, and the pattern of connectivity between component tensors.

An analysis of model parameters is the most straightforward method of comparing different tensor network models, as we can simply count the number of elements in the component tensors. As a general rule, machine learning models with more parameters can learn more sophisticated mappings between features and labels, but are also more susceptible to overfitting on smaller datasets. Note that this is more of a useful heuristic than a mathematical law, as the precise relationship between parameter number and model degrees of freedom can be complicated [59]. In a tensor network regression model, the number of parameters scales linearly with number of component tensors and polynomially with the bond dimension. Since the number of components is intimately tied to the overall architecture of the network, this leaves the bond dimension as the main hyperparameter[1] that can be used to tune the representational capacity of the model.

The next comparison point that we will consider is how easily the tensor network can be contracted to generate a prediction. Unlike the number of parameters, this property of a tensor network model is less about the model's representational capabilities and more about the feasibility and efficiency of actually running it on a computer. The two largest factors in determining the computational burden of a tensor network are the size of the intermediate tensors generated during a contraction and the degree to which the contractions can be parallelized. With respect to the intermediate tensors, the key requirement is that their orders must not scale with the order of W (i.e. with the number of features). This can be most easily enforced by ensuring that the network is acyclic, which allows us to start at the lowest-order tensor components and then contract up through the network tree. The contractions in a model can be easily parallelized if the underlying network is highly symmetric, since this allows for straightforward batched calculations using standard linear algebra software libraries.

Our final axis of comparison is the pattern of connectivity between component tensors in the network, and it is the most challenging to draw conclusions from. By "pattern of connectivity", we mean the number of connections that exist between sets of component tensors. This is closely connected to our discussion of multiplex rank in Sec. 2.2.4, where the rank of the output tensor placed constraints on the bond dimension of the tensor network. In the field of quantum many-body physics, where tensor networks are used extensively, patterns of connectivity are frequently understood in terms of *entanglement scaling* [60], which relates properties of the network to known properties of the physical system that it is trying to represent. A challenge that we face when using tensor networks for regression is that the desired rank properties of W are highly task-dependent and generally not known to us. In Chapters 5 and 6 we describe different methods of probing the rank structure of W, using feature interactions and correlation scaling respectively.

In the following subsections we introduce several well-known tensor network models, and evaluate their use for tensor regression based on three factors that we have outlined above. A diagram showing all of the models and a summary of their properties is given in Table 3.1.

3.1.2 Matrix product states and tensor rings

One of the most popular and well-studied tensor networks is the *matrix product state* (MPS), which we have already introduced obliquely for illustrative purposes in Chapter 2. This network type is characterized by its linear arrangement of component tensors, with third-order tensors occupying the inner positions of the line and second-order tensors placed at each end. The component tensors are connected together by virtual indices that contract neighboring components. The number of components in an MPS is equal to the order of the

[1] A hyperparameter is a parameter that is not directly optimized during training but instead set beforehand.

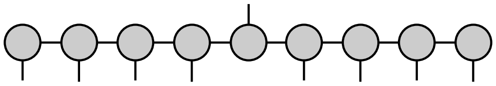
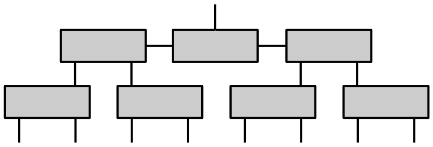
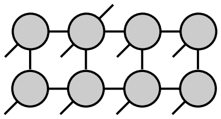
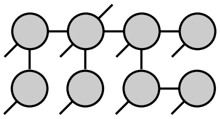
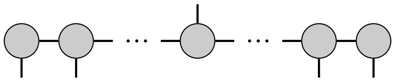
Name	Architecture	Parameters	Contraction
Matrix product state (MPS)		$2dr + nr^2$ $+ (m - 2)dr^2$	Efficient
Tree tensor network (TTN)		$(\frac{m}{2} - 2)r^3 + \frac{m}{2}rd^2$ $+ nr^2$	Efficient
Projected entangled pair state (PEPS)		$(p - 2)(q - 2)dr^4$ $+ 2(p + q - 2)dr^3$ $+ 3dr^2 + ndr^2$	Inefficient
Spanning tree network (STN)		Varies with tree pattern	Efficient

Table 3.1: This table shows the four different network types that we consider in this section, along with a summary of their relevant properties. In the parameter column, r is the bond dimension, m is the number of features (which is eight for each of the diagrams), p and q are the height and width of the feature grid, d is the dimension of the featurization functions, and n is the dimension of the output.

tensor that it is representing, so the MPS representing a weight tensor W with m features is given by

$$W_{ki_1\dots i_m} = \sum_{j_1,\dots,j_m=1}^{r} A^{(1)}_{i_1 j_1} A^{(2)}_{j_1 i_2 j_2} \cdots A^{(o)}_{j_{o-1} k j_o} \cdots A^{(m)}_{j_{m-1} j_m i_{m-1}} A^{(m+1)}_{j_m i_m} \tag{3.1}$$

$\rightarrow$  ,

where $\{A^{(i)}\}_{i=1}^{m+1}$ are the component tensors of the MPS, $\{i_j\}_{j=1}^{m}$ are the physical indices from W which contract with the featurization vectors, k is the index for the output vector, and $\{j_l\}_{l=1}^{m}$ are the virtual indices of the network with bond dimension r. Note that $A^{(1)}$ and $A^{(m+1)}$ are second-order, while the other component tensors are all third-order. The tensor $A^{(o)}$ holds the physical index for the prediction, and we refer to such components as the *output component* of the network. The position of the output component in the MPS is arbitrary, with $o = \frac{m}{2}$ being a common choice.

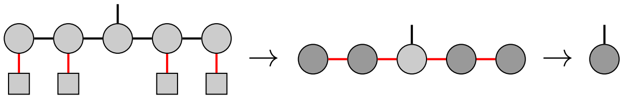
The name "matrix product state" was originally given to this network architecture by the quantum physics community, who use it extensively as an ansatz for many-body wavefunctions [61][62][63]. Indeed, the usage of MPS by physicists goes back farther than even they were originally aware of, as the highly successful density matrix renormalization group (DMRG) variational algorithm was eventually found to possess an MPS structure [64]. We will have more to say about DMRG in Sec. 3.2.3, as it has been used as inspiration for an optimization method in tensor network regression. The MPS network was also independently developed in the field of tensor analysis, where it is called the *tensor train decomposition* [49]. There it is most commonly used for tensor compression [65][66] and data reconstruction [67][68].

When used for regression, an MPS has a number of component tensors equal to the number of features m plus one, with the extra component used to hold the index for the prediction vector $\vec{f}(\vec{x}; \mathcal{W})$. The other component tensors are each paired off with one of the featurization vectors $\vec{h}^{(i)}(x_i)$ from the data tensor $X(\vec{x})$ (see Eq. (2.25) for an example with four features). Given that two of the component tensors are second-orer and the rest are third-order, the total number of parameters in an MPS regression model with bond dimension r is $2dr + nr^2 + (m-2)dr^2$, where d is the dimension of the featurization vectors and n is the dimension of $\vec{f}$. For a fixed n value[2], the number of parameters scales as $\mathcal{O}(mdr^2)$, which is far below the d^m scaling of W. The quadratic dependence of the parameter number on r is among the lowest of all tensor networks, which means that a fairly high bond dimension of 10 - 100 can be used on MPS models while remaining computationally tractable.

One of the most appealing properties of MPS models is that they are very simple and efficient to contract. As discussed previously in Sec. 2.2.3, the intermediate tensors of an MPS can be kept at a low order by first contracting each $\vec{h}^{(i)}(x_i)$ with its corresponding component tensor $A^{(i)}$, which generates a set of first- and second-order intermediate tensors. These tensors can then be contracted together, along with the component tensor associated

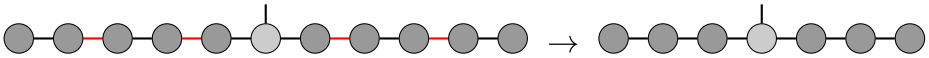
[2] The value of n is task-dependent and usually small. For the image classification tasks considered in this thesis, $n = 10$.

with output vector, to generate $\vec{f}(\vec{x};\mathcal{W})$. For $m = 4$, this process can be represented using tensor diagrams as

 $= \vec{f}(\vec{x};\mathcal{W})$ (3.2)

where the legs colored in red are those that are contracted from one step to the next.

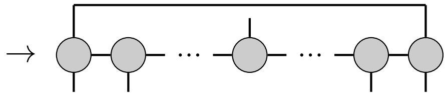
Since the intermediate tensors in the second step of Eq. (3.2) are all either matrices or vectors, we can contract the virtual indices in any order without worrying about generating higher-order tensors. Indeed, by starting with the first-order tensors at each end and contracting down the line, we could carry out the remaining contractions as a sequence of inexpensive matrix-vector products. However, in order to maximize the speed with which we contract the network, it is desirable to parallelize the operations as much as possible. This can be done by taking non-overlapping pairs of intermediate matrices and contracting them together simultaneously [57], ignoring the two end vectors at first. For $m = 10$, this looks like

 , (3.3)

where the number of number of matrices has been halved. This process repeats $\mathcal{O}(\log_2 m)$ times, after which the end vectors and third-order tensor can be contracted with the two remaining matrices to generate the output.

The presence of the two end vectors can be a nuisance when creating a contraction algorithm, since they are differently shaped than the other intermediate tensors. To create a more symmetric network, we can tweak the MPS architecture by adding an extra virtual index that connects the second-order tensors at the ends of the network, thereby turning them into third-order tensors:

$$W_{ki_1\dots i_m} = \sum_{j_1,\dots,j_{m+1}=1}^{r} A^{(1)}_{j_{m+1}i_1j_1} A^{(2)}_{j_1i_2j_2} \cdots A^{(o)}_{j_{o-1}kj_o} \cdots A^{(m)}_{j_{m-1}j_mi_{m-1}} A^{(m+1)}_{j_mi_mj_{m+1}} \tag{3.4}$$

 ,

This is known as a *tensor ring* (TR) [69] or a *periodic* MPS [70], and it shares many (though not all) of the same properties as the standard MPS, such as the $\mathcal{O}(mdr^2)$ parameter scaling. When using a TR, the end tensors can be paired off with other intermediate matrices in the parallelized contraction scheme showin in Eq. (3.3), which makes for a simpler algorithm.

Finally, we consider the pattern of connectivity imposed by the MPS/TR structure. It is well known in quantum many-body physics that wavefunctions generated though an MPS ansatz must obey an *area law* with respect to the entanglement entropy between components of the system [71], and that correlation functions between components must decay exponentially with distance [60]. This is an advantageous property in many applications, but it does limit the set of states which the MPS is capable of representing. These observations have lead to much speculation that machine learning models built using MPS/TR architectures must also exhibit some form of decaying correlation or "entanglement" with respect to the data features [22][72][73], but no rigorous mathematical argument has been put forward. We can see from the linear arrangement of the component tensors that the multiplex rank of W can be at most r for an MPS and r^2 for a TR with respect to any two contiguous bpartitions of the physical indices, but it is unclear what operational significance this has for the regression model. We explore these questions further in Chapter 5.

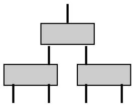
3.1.3 Tree tensor networks

A common alternative to MPS/TR networks is the tree tensor network (TTN), which has seen a wide variety of uses across different fields, from quantum simulation [74][75] to efficient tensor representation [50] (where it is known as the *hierarchical Tucker decomposition*). In a TTN, the component tensors are arranged, as the name suggests, in a tree pattern, which in this work we will assume is binary. When used to represent a weight tensor with m features, the TNN will consist of $m-1$ component tensors, organized into $\ell = \lceil \log_2(m) \rceil$ layers. Each component is a third-order tensor, composed of two physical indices and one virtual index if it is in the bottom layer, two virtual indices and one physical index if it is the output component at the top, and three virtual indices otherwise. An explicit expression for the contractions in a TTN representing a weight tensor W with m features is given (obtusely) by

$$W_{ki_1\dots i_m} = \sum_{j^1_1,\dots,j^1_{\frac{m}{2}}=1}^{r} L^{(1)}_{i_1\dots i_m j^1_1\dots j^1_{\frac{m}{2}}} \sum_{j^2_1,\dots,j^2_{\frac{m}{4}}=1}^{r} L^{(2)}_{j^1_1\dots j^1_{\frac{m}{2}} j^2_1\dots j^2_{\frac{m}{4}}} \sum_{j^3_1,\dots,j^3_{\frac{m}{8}}=1}^{r} \cdots L^{(\ell)}_{j^\ell_1 j^\ell_2 k} \tag{3.5}$$

$$L^{(k)} = \begin{cases} A^{(1,1)}_{i_1 i_2 j^1_1} A^{(1,2)}_{i_3 i_4 j^1_2} \cdots A^{(1,\frac{m}{2})}_{i_{m-1} i_m j^1_{\frac{m}{2}}} & k = 1 \\ A^{(k,1)}_{j^{k-1}_1 j^{k-1}_2 j^k_1} A^{(k,2)}_{j^{k-1}_3 j^{k-1}_4 j^k_2} \cdots A^{(k,\frac{m}{2^k})}_{j^{k-1}_{\frac{m}{2^{k-1}}-1} j^{k-1}_{\frac{m}{2^{k-1}}} j^k_{\frac{m}{2^k}}} & 1 < k < \ell \\ A^{(\ell,1)}_{j^\ell_1 j^\ell_2 k} & k = \ell, \end{cases} \tag{3.6}$$

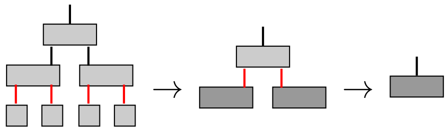
where $A^{(k,p)}$ is the pth component tensor of the kth layer tensor $L^{(k)}$, and j^k_i is the virtual index with bond dimension r that connects the ith tensor of the kth row to the $\lfloor \frac{i}{2} \rfloor$th tensor of the $k+1$th row. While it is awkward to give a generic, mth-order tensor diagram for a TTN, an example for $m = 4$ can easily show the basic binary tree pattern:

$$W_{ki_1\dots i_4} = \sum_{j^1_1, j^1_2=1}^{r} A^{(1,1)}_{i_1 i_2 j^1_1} A^{(1,2)}_{i_1 i_2 j^1_2} A^{(2,1)}_{j^1_1 j^1_2 k} \rightarrow$$  , (3.7)

where the top tensor is the output component. It is clear from their structure that (binary) TTN regression models are intended for situations where the number of features is given by $m = 2^\ell$ for non-negative integer ℓ. Although the architecture can be modified to handle other values of m, we will not consider those networks here and will always work with values of m that are powers of two.

When used for tensor regression, the $\frac{m}{2}$ components in the bottom layer of the TTN are each matched up with a pair of featurization vectors $\{\vec{h}^{(i)}(x_i), \vec{h}^{(i+1)}(x_{i+1})\}$, and then matched up to $\frac{m}{4}$ components in the next layer and so one. The total number of parameters in a TTN regression model with m features and bond dimension r is $(\frac{m}{2}-2)r^3+\frac{m}{2}rd^2+nr^2$, where d is the dimension of the featurization vectors and n is the dimension of $\vec{f}$. For a fixed n, this scales as $\mathcal{O}(mr(r^2+d^2))$, which is cubic in the bond dimension r and quadratic in the featurization dimension d. Note the the scaling orders of the TTN are larger than those of MPS/TR for both r (cubic vs quadratic) and d (quadatic vs linear). The cubic dependence on r means that TTN models are generally used with bond dimensions in the range of 10 - 20, while the quadratic dependence on d is usually not that significant since d is taken to be a small value ($d = 2$ for the polynomial featurization from Sec. 2.3.2).

As was the case for MPS/TR networks, a TNN can be efficiently contracted using a very straightforward procedure. First, the bottom component tensors are contracted with their corresponding featurization vectors, which generates a layer of $\frac{m}{2}$ first-order intermediate tensors. Those intermediate tensors are then contracted with the next layer of component tensors, generating a layer of $\frac{m}{4}$ first-order tensors. These layer-to-layer contractions occur $\log_2(m)$ times, and end with the the last pair of intermediate tensors being contracted with the output component to generate a prediction. For $m = 4$, these steps can be represented using tensor diagrams as



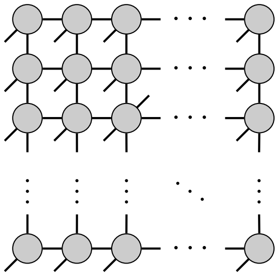
$$= \vec{f}(\vec{x};\mathcal{W}), \tag{3.8}$$

where the legs shown in red are contracted between each step. Note that this contraction scheme lends itself naturally to parallelization, as the contractions for each component in a given layer can be done simultaneously.

The pattern of connectivity in a TTN is similar to that of an MPS/TR, despite their apparent differences in structure. For any two contiguous bipartitions, there can be at most two virtual indices which span the partitions, regardless of the size of either partitions. This places an upper bound of r^2 on the multiplex rank of W with respect to any contiguous bipartitioning. In quantum physics, this bound gives rise to a boundary law scaling for the entanglement entropy, and as with the MPS this may suggest that similar correlation scaling could play a role in the behavior of a TTN regression model. In Chapter 5, we explore how the multiplex rank could be used to more rigorously tie the connectivity patterns of a tensor network to its regression characteristics.

3.1.4 Projected entangled pair states

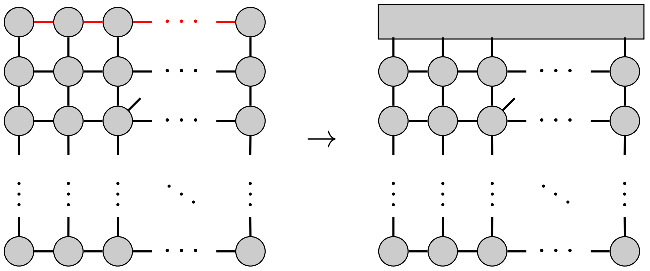
Another important tensor network model used in quantum physics is the projected entangled pair state (PEPS) [76], which consists of third-order, fourth-order and fifth-order component tensors arranged in a two-dimensional rectangular grid. When used for regression, there can also be a sixth-order tensor serving as the output component. Similar to an MPS, a PEPS network representing a weight tensor with m features will contain m component tensors, with virtual indices connecting each tensor to its neighbors on the grid. This means that tensors in the corners of the grid will be third-order, tensors on other parts of the edge will be fourth-order, and tensors internal to the grid will be fifth-order. The explicit index expression for a PEPS model is both messy and unenlightening, but it can be expressed clearly in a tensor diagram as


, (3.9)

where the orders and connectivity of the component tensors are easily visible. Note that we have placed the output component in the third row and third column simply for illustrative purposes, and its location can be chosen arbitrarily.

When PEPS is used for tensor regression (which is rare, for reasons that we will discuss shortly), each component of the network is paired with a featurization vector $\vec{h}^{(i)}(x_i)$ associated with one of the m features. Unlike for an MPS, it is not easy to simply insert an extra tensor into the network to serve as the output component, so usually this extra physical index is attached to one of the existing component tensors (thus increasing its order by one). The number of parameters in a PEPS regression model with p rows, q columns (where $m = p \times q$ features), and a bond dimension r is $(p-2)(q-2)dr^4 + 2(p+q-2)dr^3 + 3dr^2 + ndr^2$, where d is the dimension of the featurization and n is the dimension of the output index which has been placed on one of the corner tensors. For a fixed n, this scales as $\mathcal{O}(mdr^4)$, which is quartic in the bond-dimension. This steeper dependence on r means that a bond dimension of only 5 can easily produce a model with millions of parameters.

The biggest problem with PEPS networks, and something that sets them apart from MPS/TR and TTN, is that they cannot be exactly contracted in an efficient manner. This limitation is a result of the network's cyclic structure, which causes the order of the intermediate tensor to grow when each component is contracted. As an example, consider an $p \times q$ PEPS models. We can begin by contracting each component tensor with its associated featurization vector, as we did for the MPS/TR and TTN models, but difficulties arise in the next step. If we start at the top of the network and begin contracting across the row, we find that the order of the intermediate tensor scales with q:

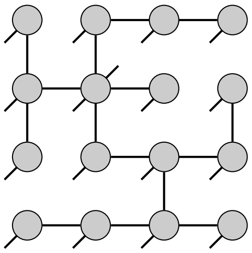

. (3.10)

Other contraction schemes, such as starting from a corner and expanding outward in a sqaure, will face similar scaling issues. Methods have been developed to contract PEPS models in an approximate manner [26], but these are computationally expensive and not conducive to the large number of contractions needed to train and deploy a regression model.

The pattern of connectivity in a PEPS model differs significantly from that of the MPS/TR and TTN models, in that the multiplex rank associated with a contiguous bipartition scales with the size of the partitions. In quantum physics, a PEPS-based wavefunction will manifest an area law scaling in its entanglement entropy [77], which allows it to represent states of two-dimensional spin systems better than an MPS [78]. This specificity toward grid-like configurations has fueled interest in using PEPS for regression tasks on data which is inherently two-dimensional, such as grayscale images. Unfortunately, the difficulties inherent in contracting PEPS have made it difficult to realize this potential advantage.

3.1.5 Spanning trees

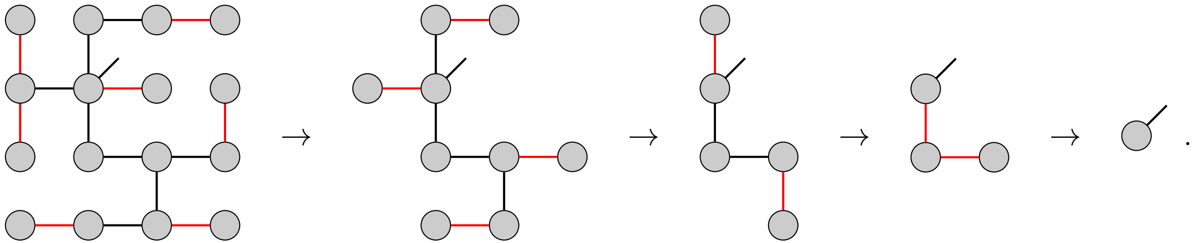
The final type of network that we that we will consider here is what we refer to as the *spanning tree network* (STN), which does not appear to have been explored in any prior work. STNs can be understood as PEPS models which have had virtual indices removed such that all cycles within the network are broken, subject to the constraint that all component tensors must be joined together in the same continous tree. This is the motivation behind the "spanning tree" nomenclature, which refers to a tree that connects all vertices in a graph [79]. An example STN on a 4 × 4 grid is given in the following diagram:

 . (3.11)

Note that there are no cycles present in the graph, and that all components are part of a single tree. As in PEPS, each component tensor has one physical index (except for the output component, which has two), but the number of virtual indices can range anywhere from one to four. The special case where every component tensor is either second or third order is equivalent to an MPS that has been threaded through the grid.

When used for regression, an STN has a number of components equal to the number of features m in the dataset. Although the placement of virtual indices can vary widely between different STNs, the underlying spanning tree always contains $m-1$ edges and thus the network will always possess $m-1$ virtual indices. The number of parameters in an STN depends on the orders of the component tensors, and therefore cannot be enumerated without first specifying the structure of the tree.

The primary advantage of an STN over a PEPS is that it an be exactly contracted in a (relatively) efficient manner. To contract an STN, the featurization vectors $\vec{h}^{(i)}(x_i)$ can first be contracted in parallel with their associated component tensors, leaving a set of first, second, third, and fourth-order tensors (and perhaps a single fifth-order tensor holding the output index). Then, all of the first-order tensors can be contracted with their neighbors on the graph, generating a set of intermediate tensors that are at most fifth-order. Some of these intermediate tensor will themselves be first-order, and all of them can then be contracted with their neighbors to generate a new set of intermediate tensors, some which will again be first-order. This process repeats until every tensor has been contracted. Using the example tensor from Eq. (3.11), this procedure (skipping the contraction of the featurization vectors) can be diagrammed as



(3.12)

One drawback of STN models is that the first-order tensor contractions can be difficult to parallelize using standard linear algebra libraries, since they are contracted with tensors of many different shapes.

When considering the connectivity patterns inherent to a STN model, we will focus on the average or typical behavior of networks drawn randomly from the set of spanning trees, rather than on a specific configuration. With respect to multiplex rank, a typical STN model shows the same area law behavior present in PEPS models for contiguous bipartitions, due to the fact that virtual indices can connect neighboring tensors in all four directions of the grid (even if they do not do so for every component). This may make them well adapted to two-dimensional data in the same manner as PEPS, while still being efficient to contract.

3.2 Optimization

3.2.1 General considerations

Optimizations algorithms lie at the heart of machine learning, as they are what allow the algorithm to "learn" the solution for a given task. Most optimization algorithms, and all of the specific algorithms that we will consider here, utilize the gradients of the model parameters $\mathcal{W}$ with respect to a chosen loss function $\mathcal{L}$. During each optimization step, all or some subset of the parameters are updated so as to minimize the average value of $\mathcal{L}$ across the training dataset. In this work we use the mean square error as our loss function [51], which has the form

$$\mathcal{L}[\{(\vec{x}^{\,(i)},\vec{y}^{\,(i)})\}_{i=1}^{\eta};\ \mathcal{W}] = \frac{1}{\eta}\sum_{i=1}^{\eta}|\vec{y}^{\,(i)} - \vec{f}(\vec{x}^{\,(i)};\mathcal{W})|^2, \tag{3.13}$$

where $\{(\vec{x}^{\,(i)},\vec{y}^{\,(i)})\}_{i=1}^{\eta}$ is the training set. The mean square error loss function is used widely in machine learning and statistics due to its smoothness and convexity with respect to the model prediction $\vec{f}(\vec{x}^{\,(i)};\mathcal{W})$.

To tailor Eq. (3.13) for the specific case of tensor regression, we substitute in the contraction between the weight tensor W and data tensor X from Eq. (2.21) and then evaluate the derivative of $\mathcal{L}$ with respect to elements of W:

$$\mathcal{L} = \frac{1}{\eta}\sum_{j=1}^{\eta}\sum_{k=1}^{n}\left(y_k^{(j)} - \sum_{i_0\ldots i_{m-1}} W_{ki_0\ldots i_{m-1}} X_{i_0\ldots i_{m-1}}(\vec{x}^{\ (j)})\right)^2 \tag{3.14}$$

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{2}{\eta}\sum_{j=1}^{\eta}\sum_{k=1}^{n}(y_k^{(j)} - \hat{y}_k^{(j)})X(\vec{x}^{\ (j)}), \tag{3.15}$$

where $\hat{y}_k^{(j)}$ is the the kth element of the model prediction for $\vec{x}^{\ (j)}$. From Eq. (3.15), we can see that parameterizing the regression model using elements of the weight tensor directly gives a very simple expression for the gradient, since each model parameter acts on only a single element of X.

Of course, for reasons discussed in Sec. 2.3.3, our model parameters are not elements of W but rather elements of the tensor network components used to represent W, which we take to be elements of a generic parameter vector $\vec{\theta}$. From the chain rule, we can compute the derivative of the loss function with respect to $\vec{\theta}$ as

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \vec{\theta}} &= \sum_{i_0,\ldots,i_{m-1}} \frac{\partial \mathcal{L}}{\partial W_{i_0\ldots i_{m-1}}}\frac{\partial W_{i_0\ldots i_{m-1}}}{\partial \vec{\theta}} \\ &= \frac{2}{\eta}\sum_{j=0}^{\eta-1}\sum_{k}(y_k^{(j)} - \hat{y}_k^{(j)})\sum_{i_0,\ldots,i_{m-1}}\frac{\partial W_{i_0\ldots i_{m-1}}}{\partial \vec{\theta}} X_{i_0\ldots i_{m-1}}(\vec{x}^{\ (j)}),\end{aligned} \tag{3.16}$$

which is a less pleasant function than Eq. (3.15). While the precise form of $\frac{\partial W_{i_0\ldots i_{m-1}}}{\partial \vec{\theta}}$ depends on the specific architecture chosen for the tensor network, it will virtually always involve sums and products of multiple parameters and thus be a highly non-convex function of $\vec{\theta}$.

In the following subsections, we review three different optimization algorithms that have been used in the literature to train tensor network regression models. Each of them can be understood as taking a different approach to handling the unpleasantness of Eq. (3.16). Riemannian optimization, which we cover first, seeks to bypass Eq. (3.16) entirely by taking the gradient of W from Eq. (3.15) directly and then projecting it into a direction tangent to the manifold of tensor networks. The DMRG sweep algorithm, by contrast, draws inspiration from methods in quantum physics and preforms convex optimization on the parameters of each component tensor sequentially. Lasty, the stochastic gradient descent algorithm from traditional machine learning uses back-propagation to simply compute and then follow Eq. (3.16) until the loss value converges, with the hope that it ends up close enough to the global minimum to yield an effective model.

3.2.2 Riemannian optimization

One of the first algorithms proposed for the optimization of tensor network regression models was *Riemannian optimization* [80], where the gradient of the weight tensor (not the component tensors) is computed and then projected into the tangent space associated with the network. This approach views the network-represented weight tensor as lying on a Riemannian manifold in tensor space, defined by the architecture of the underlying tensor network. The algorithm can be broken up into three steps:

1. Compute $\frac{\partial \mathcal{L}}{\partial W}$, the gradient of the loss function with respect to the weight tensor W.

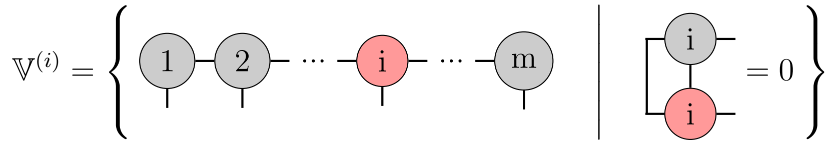
2. Project $\frac{\partial \mathcal{L}}{\partial W}$ into the the tensor space $\mathbb{M}$ tangent to W.

3. Subtract the projected gradient from W and retract back to the tensor network manifold.

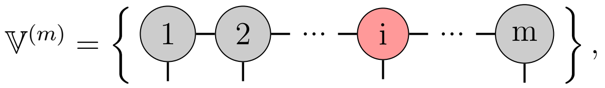
These three steps can be done efficiently when the weight tensor is represented by a low-rank tensor network.

Riemannian optimization was first used for the training of tensor network regression models by Novikov et al. [21] for their MPS-based Exponential Machine. Note that the network architecture does not enter into the first step listed above, as the gradient with respect to elements of W is independent of any particular component representation. It is, however, relevant when defining the tangent space for the projection, and in determining the retraction algorithm that needs to be used to return to the network manifold.

To provide an example of how Riemannian optimization algorithms are constructed, we will outline the algorithm used to train MPS models, following the work of Novikov et al. and Lunbich et al. [81]. Given that we already have an expression for $\frac{\partial \mathcal{L}}{\partial W}$ in Eq. (3.15), the next step is to construct an operator that will project the gradient into the space tangent to W. As shown by Holtz et al. [82], the tangent space for an mth-order MPS can be represented as the direct sum of m orthogonal subspaces $\{\mathbb{V}^{(i)}\}_{i=1}^{m}$, where $\mathbb{V}^{(i)}$ is the set of all MPS which are identical to the MPS representing W except in the ith component. For $i = m$ this component can be set arbitrarily, but for $i < m$ it must be orthogonal to the ith component of model network. Using tensor diagram notation, the subspaces can be written as



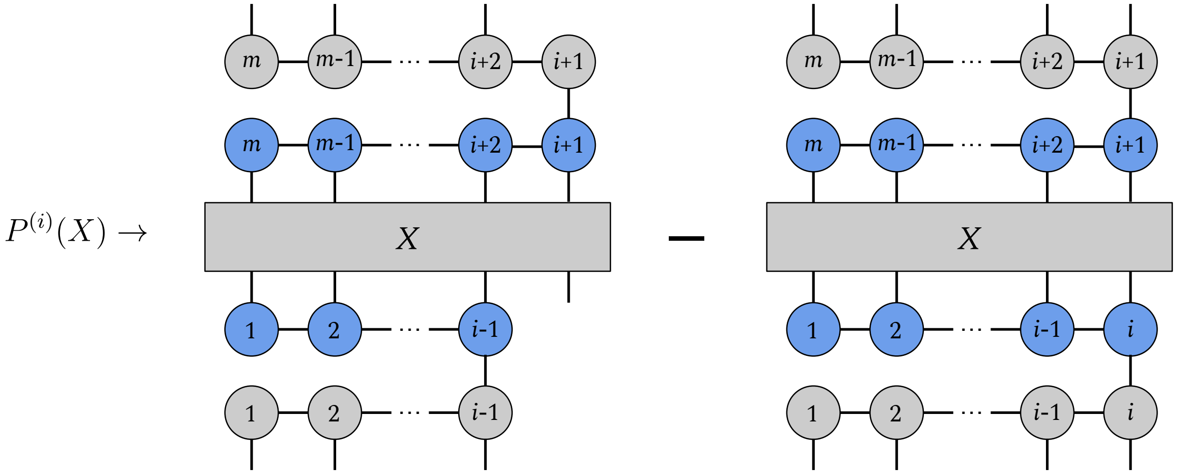
for $i < m$ (3.17)



(3.18)

where the tensors in gray are component of the MPS model and the tensors in red are free to vary in the set. For simplicity we will omit the output component from our diagrams, as its presence does not fundamentally change the algorithm.

To project $\frac{\partial \mathcal{L}}{\partial W}$ into the tangent space $\mathbb{M}$, we need the components of the gradient along each of the $\mathbb{V}^{(i)}$. The form of the ith projector $P^{(i)}$ is given in [81], and its action on the data tensor X tensor can be written out using tensor diagrams as



(3.19)

where the the gray circle tensors are components of the MPS (numbered with respect to their corresponding features) in canonical form [83], and the blue circular tensors are their adjoints. In words, Eq. (3.19) states that $P^{(i)}(X)$ is given by the projection of X into the r-dimensional subspace spanned by the MPS without the ith component, minus the projection which includes the ith component (this second term is not included for $P^{(m)}(X)$). Note that the result of each projection is an MPS of the same size and bond dimension as the original network. The overall projection $P_{\mathbb{M}}$ of the gradient into the tangent space $\mathbb{M}$ is given the sum of the projections into each of the $\mathbb{V}^{(i)}$:

$$\begin{aligned} P_{\mathbb{M}}\left(\frac{\partial \mathcal{L}}{\partial W}\right) &= \sum_{i=1}^{m} P^{(i)}\left(\frac{\partial \mathcal{L}}{\partial W}\right) \\ &= \frac{2}{\eta}\sum_{j=0}^{\eta-1}\sum_{k}(y_k^{(j)} - \hat{y}_k^{(j)})P^{(i)}(X(\vec{x}^{\,(j)})), \end{aligned} \tag{3.20}$$

which is just a linear combination of the projections of X.

With the gradient projection from Eq. (3.20) in hand, we can then scale it by an appropriate step-size α and subtract it from W to generate the new weight tensor $\tilde{W}$:

$$\tilde{W} = W - \alpha P_{\mathbb{M}}\left(\frac{\partial \mathcal{L}}{\partial W}\right). \tag{3.21}$$

Since $\tilde{W}$ is equal to the sum of tensors with low-rank MPS representations, we can efficiently perform the sum in Eq. (3.21) to yield an MPS representation that has a bond-dimension at most $3r$, where r is the bond dimension of the MPS model [49]. The final step in Riemannian optimization is known as *retraction*, and it involves truncating the bond dimension of $\tilde{W}$ so that it is equal to r and thus rejoins the original manifold occupied by W. This can be carried out in an optimal fashion using the sequential-SVD rounding algorithm developed by Oseledets [49]. Retraction marks the end of one round of optimization, after which the process is repeated with a new batch of data.

Despite being one of the pioneering tensor network optimization methods, Riemannian optimization is rarely used in practice, most likely due to its complexity and the lack of implementation in most existing machine learning libraries. The method does have other applications, however, such matrix completion [84] and the optimization of quantum circuit networks [85].

3.2.3 DMRG-style sweeps

The second algorithm that we consider was proposed by Stoudenmire and Schwab [22] in parallel with the work of Novikov et al., and is based on the popular DMRG algorithm employed in quantum physics. The main thrust of the algorithm is that each component tensor in the tensor network regression model is optimized individually, while the parameters in the other components are held fixed. This transforms the non-convex optimization problem described by Eq. (3.16) into a a simple quadratic form, which can be easily minimized.

The DMRG sweep algorithm starts at a specified component tensor, which we will call the *system tensor* S, that can be chosen arbitrarily. All other component tensors and associated featurization vectors are then contracted together, using one of the standard contraction scheme associated with the network (see Sec. 3.1 for examples). The intermediate tensors resulting from these contractions jointly form what we will call the *environment tensor* E. The environment tensor may be a single intermediate tensor or the tensor product of multiple such tensors, depending on the structure of the network and the position of the system tensor. With the network reduced to only the system and environment tensors, the loss function from Eq. (3.14) becomes

$$\mathcal{L} = \frac{1}{\eta} \sum_{j=1}^{\eta} \sum_{k=1}^{n} \left(y_k^{(j)} - S_k \cdot E^{(j)} \right)^2 , \tag{3.22}$$

where $S_k \cdot E$ is a generic expression denoting the element from the contraction of S and E at the kth value of the output index. Note that the environment tensor is formed by the contraction of the data tensor, and therefore varies as a function of the sample.

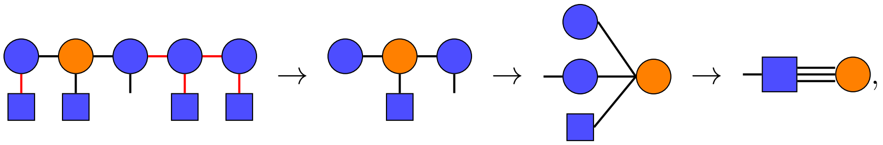
The form of Eq. (3.22) is highly reminiscent of multi-output ordinary least squares regression [51], and it can be put in precisely that form by matricizing S and vectorizing E, such that the virtual indices connecting S and E are grouped into the column index of S. With the reshaped versions of S and E denoted as $\tilde{S}$ and $\vec{e}$ respectively, we have

$$\mathcal{L} = \frac{1}{\eta} \sum_{j=0}^{\eta-1} |\vec{y}^{\,(j)} - \tilde{S}\vec{e}^{\,(j)}\,|^2, \tag{3.23}$$

with $\tilde{S}$ holding the regression parameters and $\vec{e}^{\,(j)}$ serving as the data vector. This loss function can be minimized using any of the techniques developed for least-squares regression of linear models, such as the QR decomposition, normal equations, or gradient descent. After optimization, the parameters are reshaped back into a tensor, a new component tensor is selected to be the system tensor, and the process is repeated.

To provide a concrete example, we can consider DMRG-style sweeps on an MPS regression model. For simplicity, let us assume that $m = 4$ and that we have chosen the second component tensor (i.e. the component associated with feature x_2) to be the system tensor S. The environment tensor E is constructed by contracting the third, fourth, and fifth components together, and then taking the tensor product of that result with the first component tensor and the second featurization vector $\vec{h}^{\,(2)}(x_2)$:

$$\vec{f}(\vec{x};\mathcal{W}) \;=$$



(3.24)

where the environmment tensor is colored in blue and the system tensor is colored in orange. In the last step we show the reshaping of S and E into a matrix $\tilde{S}$ and vector $\vec{e}$ respectively, where $\vec{e}$ has r^3 elements and $\tilde{S}$ has nr^3 elements (with n being the dimension of $\vec{y}$).

Unlike the Riemannian optimization from Sec. 3.2.2, the DMRG-sweep optimization algorithm has continued to see use since its introduction in [22]. This likely because it utilizes more familiar concepts from standard machine learning, such as the final least-squares regression step. That said, it is still less popular than stochastic gradient descent, which we cover next.

3.2.4 Stochastic gradient descent

Stochastic gradient descent (SGD) is by far the most popular class of optimization algorithm for machine learning [86], having been used in various forms since the 1950s [87]. When paired with backpropogation, which we describe in more detail below, SGD serves as the workhorse of deep learning, where models with millions and even billions of parameters are trained to solve a myriad of different tasks via highly non-convex loss functions. Given its entrenched positin in machine learning, it is not surprise that SGD has also been applied to the training of tensor network regression models.

Unlike the Riemannian and DMRG-style optimization methods that we discussed previously, SGD does not attempt to bypass or simplify the gradient expression from Eq. (3.16). Instead, it directly evaluates the gradient of each element using *automatic differentiation* [88], which computes the derivative of a complicated composite function using the known derivatives of its simpler components. More formally, automatic differentiation divides the function $\vec{f}(\vec{x};\vec{\theta})$ into a set of intermediate components $\{\vec{g}^{\,(i)}\}_{i=1}^{l}$, where $\vec{g}^{\,(i)}$ is a function of $\vec{g}^{\,(i-1)}$ (which may include elements of both $\vec{x}$ and $\vec{\theta}$), $\vec{g}^{\,(1)}$ consists of all elements of $\vec{x}$ and $\vec{\theta}$ not included in any of the other components, and $\vec{f}(\vec{x};\vec{\theta}) \equiv \vec{g}^{\,(l)}(\,\vec{g}^{\,(l-1)}(\ldots\,\vec{g}^{\,(1)}))$. The Jacobian of $\vec{f}(\vec{x};\vec{\theta})$ with respect to the model parameter $\vec{\theta}$ is then computed via the chain rule as the matrix product of Jacobians for each $\vec{g}^{\,(i)}$:

$$\frac{\partial f_k}{\partial \theta_j} = \sum_{i_l} \frac{\partial g^l_k}{\partial g^{l-1}_{i_l}} \sum_{i_{l-1}} \frac{\partial g^{l-1}_{i_l}}{\partial g^{l-2}_{i_{l-1}}} \sum_{i_{l-2}} \cdots \sum_{i_2} \frac{\partial g^2_{i_3}}{\partial g^1_{i_2}} \frac{\partial g^1_{i_2}}{\partial \theta_j}. \tag{3.25}$$

This sequence of matrix multiplications can be performed from the left (referred to as backpropogation or *reverse accumulation*) or from the right (referred to as *forward accumulation*), with backpropogation typically being more efficient since the number of parameters is usually much larger than the number of labels.

The term "stochastic" in SGD references the fact that optimization is not usually performed on the entire training dataset at each step, but instead on a small subset called a *mini-batch*. Mini-batch sizes typically range from 32 - 1024 in powers of two, with an optimization cycle over all mini-batches referred to as an *epoch*. In a given mini-batch, the gradient in Eq. (3.25) is computed for each of the samples and then averaged together. Since each element of $\vec{f}$ has its own set of gradients, a single, combined gradient is generated by adding together the different element-wise averages. Finally, this value is subtracted from the parameters $\vec{\theta}$ using a chosen step size α to give the new parameters $\vec{\theta}'$. This procedure can be written out succinctly as

$$\vec{\theta}' = \vec{\theta} - \alpha \cdot \frac{1}{b} \sum_{i=1}^{b} \sum_{k=1}^{n} \frac{\partial f_k(\vec{x}^{\ (i)}; \theta)}{\partial \vec{\theta}}, \tag{3.26}$$

where b is the minibatch size. There exist a number of sophisticated algorithms for determining the ideal value(s) for α, such as the popular Adam [89] and RMSprop[3] optimizers.

For tensor network regression, we use SGD to evaluate the complicated gradient of the loss function in Eq. (3.16), or more specifically the gradient of the contracted network with respect to the elements $\vec{\theta}$ of the component tensors. In this context, the component tensors of the network provide a natural delineation for the $\{\vec{g}^{\ (i)}\}_{i=1}^{l}$ in Eq. (3.25), since each tensor behaves like a multilinear function during the contraction. Taking an MPS model as an example, $\vec{g}^{\ (l)}$ would be given by the contraction of the output component with the two intermediate tensors generated by contracting all of the other component tensors and featurization vectors. These two intermediate tensors would then jointly become the elements of $\vec{g}^{\ (l-1)}$, and take as input the intermediate tensors generated by all components to the left and right of them in the network. This process can be repeated from the inside out until all tensors have been included in one of the $\vec{g}^{\ (i)}$.

From a practitioner's perspective, the most attractive aspect of SGD is that automatic differentiation is robustly supported in virtually all machine learning software packages, which cannot be said for Riemannian optimization or DMRG-style sweeps. SGD is also a very inexpensive algorithm, requiring roughly the same amount of computational resources as simply contracting the network. It is these properties that explain the popularity of SGD-based training for tensor network models in the literature, a popularity that is likely to grow as more machine learning practitioners experiment with tensor-based models.

[3] This widely-used algorithm was never published, having originated in the sixth lecture of Geoffrey Hinton's Coursera course on machine learning. As of May 2023, the slides can be found at the following URL: `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`.

3.3 Initialization

3.3.1 General considerations

The final aspect of model design that we consider here is parameter initialization. Our focus will specifically be on initialization that is done prior to any optimization, and we will therefore not consider topics such as pre-training [90] or transfer learning [91]. In the neural network models used for deep learning, parameter initialization of an $a \times b$ weight matrix is most frequently done by sampling values from a uniform distribution on the range $[-\sqrt{6}(a+b)^{-\frac{1}{2}},\ \sqrt{6}(a+b)^{-\frac{1}{2}}]$ (referred to as *Glorot initialization* [92]). This scheme seeks to stabilize the gradient of the model such that its variance is constant between layers, thus improving the flow of information back through the model during optimization.

Compared to neural networks, the need for precise initialization is much more acute in tensor network models, as a poor choice of values can prevent the model output from even being evaluated, let alone optimized. This precariousness stems from the fact that network contraction occurs via the composition of multilinear maps, and each of these maps can increase or decrease the norm of their input. While this is fairly innocuous for a single tensor contraction, tensor network models tend to have a number of components roughly equal to the number of features m, which can be in the hundreds or thousands. Even if each tensor contraction only scaled the norm by $(1+\epsilon)$, $|\epsilon| < 1$, the composition of m such scalings is

$$(1+\epsilon)^m = \sum_{i=0}^{m} \binom{m}{i} \epsilon^i \tag{3.27}$$

by the binomial theorem. For m in the range 100 - 1000, this series can easily explode toward infinity or vanish to zero, depending on the sign of ϵ.

In order to initialize a tensor network model with a numerically stable contraction, the scaling of Eq. (3.27) needs to be tamed for *all* plausible inputs $\vec{x}$ to the model. The most straightforward way to achieve this is to choose component tensors that preserve some extensive property of their inputs, such that the amount of magnitude scaling is necessarily bounded at a reasonable value. An ideal candidate for this (although other possibilities exist) is a "stochastic tensor", which is simply the reshaping of a *stochastic matrix* [93] into a form that matches the desired shape of the component tensor. A stochastic matrix is a matrix with non-negative elements constrained such that the columns or rows (or both) must sum to one. These matrices are most commonly used to represent conditional probabilities for e.g., a Markov process, though we will not employ any particular probabilistic interpretation here.

To give a simple example of how stochastic matrices can solve our initialization problem, consider a third order tensor A that maps vectors $\vec{b}$ and $\vec{c}$ to a new vector $\vec{d}$:

$$d_k = \sum_{i,j} A_{ijk} b_i c_j. \tag{3.28}$$

Now let us initialize A as a stochastic tensor, such that the summation over index k (the "columns") gives a value of 1 for all i, j pairs. Then the sum of all elements in $\vec{d}$ must be

$$\sum_k d_k = \sum_{i,j,k} A_{ijk} b_i c_j = \left(\sum_i b_i\right)\left(\sum_j c_j\right) \tag{3.29}$$

since $\sum_k A_{ijk} = 1$ by construction. Note that the sum of elements in $\vec{d}$ is given by the products of the sums for $\vec{b}$ and $\vec{c}$. If we were to constrain each of these sums to be 1, and have each of the elements in $\vec{b}$ and $\vec{c}$ be non-negative, then the sum over $\vec{d}$ would also be 1 and its elements would also be non-negative. This would necessarily constrain the ℓ^1 norms of the output $\vec{d}$ and the inputs $\vec{b}$ and $\vec{c}$ to all be 1, which means that ϵ in Eq. (3.27) would vanish with respect to the ℓ^1 norm.

For a more complicated tensor network model, the significance of Eq. (3.29) is that we can, in principle, compose an infinite number of tensor contractions together without increasing the ℓ^1 norm of the output. For a tensor regression model, this would requre that the featurization vectors $\vec{h}^{\,(i)}(x_i)$ be constrained to have non-negative elements that sum to 1. The $[1,\ x_i]$ polynomial featurization from Eq. (2.22) does not fulfill this requirement in general, but we can approximately obey the rule by shrinking the elements of the component tensors associated with each x_i, thereby reducing their initial effect on the contraction. This trick is discussed in more detail in Sec. 3.3.2.

In the following subsections, we consider initialization strategies for the MPS/TR, TTN and STN regression models. Each of these network architectures are amenable to initialization with stochastic matrices, although the forms of these matrices will vary based on the shapes and contractions patterns of the component tensors.

3.3.2 MPS/tensor ring initialization

In an MPS model operating on data with m features, the component tensors $\{A^{(i)}\}_{i=1}^{m+1}$ consist of two second-order tensors $A^{(1)}$, $A^{(m+1)}$ and $m-1$ third-order tensors connected together along a line of virtual indices. If we consider a contraction of the network starting from the matrix $A^{(1)}$ and then proceeding through the other components in sequence, then the first contraction is simply a matrix-vector product between $A^{(1)}$ and the first featurization vector $\vec{h}^{\,(1)}(x_1) = [1,\ x_1]$:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ \vdots & \vdots \\ a_{r1} & a_{r2} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \end{bmatrix} = \begin{bmatrix} a_{11} + a_{12}x_1 \\ a_{21} + a_{22}x_1 \\ \vdots \\ a_{r1} + a_{r2}x_1 \end{bmatrix}, \tag{3.30}$$

where r is the bond dimension of the MPS. When $A^{(1)}$ is initialized as a stochastic matrix with columns that sum to 1, the sum of the elements of the resulting vector is $(a_{11} + ... + a_{1r}) + (a_{11} + ... + a_{1r})x_1 = 1 + x_1$, which is not the desired value of 1 discussed in Sec. 3.3. While this difference is not fatal at first, each of the featurization tensors will contribute their own factor of $(1 + x_i)$ to the sum, which will quickly lead to numerical overflow or underflow akin to what would result from Eq. (3.27).

One way to solve this problem is to reduce the magnitude of x_1. It is important to note that we do not wish to eliminate the presence of x_1 completely, as it contains information that the model may use to make its prediction, but we must minimize its effect on the contraction. To this end, we can shrink the elements of $A^{(1)}$ in the second column by a factor γ, thereby generating the modified matrix-value product

$$\begin{bmatrix} a_{11} & \gamma a_{12} \\ a_{21} & \gamma a_{22} \\ \vdots & \vdots \\ a_{r1} & \gamma a_{r2} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \end{bmatrix} = \begin{bmatrix} a_{11} + \gamma a_{12} x_1 \\ a_{21} + \gamma a_{22} x_1 \\ \vdots \\ a_{r1} + \gamma a_{r2} x_1 \end{bmatrix} = \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{r1} \end{bmatrix} + \gamma \begin{bmatrix} a_{12} x_1 \\ a_{22} x_1 \\ \vdots \\ a_{r2} x_1 \end{bmatrix}, \tag{3.31}$$

where we have split the final vector into the sum of a stochastic column and a non-stochastic portion which carries the data information. If $\gamma \ll 1$, then the first vector will dominate in the contraction and we will get a well-behaved contraction. The value of γ is a hyperparameter that may have to be tuned based on the number of features, though we found that values of 10^{-2} to 10^{-3} worked well for m on the order of 100 to 1000.

With the first component tensor contracted, the next step is to contract the resulting intermediate vector—the right side of Eq. (3.31)—and $\vec{h}^{(2)}(x_2)$ together with the second component tensor $A^{(2)}$ in a manner analagous to Eq. (3.28). Though this tensor is third-order rather than second-order, the initialization principles from $A^{(1)}$ still apply. To initialize $A^{(2)}$ as a stochastic tensor, we choose elements such that $\sum_{j_2} A_{j_1 i_2 j_2} = 1$ for all j_1, i_2 pairs. This is equivalent to creating a stochastic matrix of size $r \times 2r$ and then splitting the column index into an index of size 2 and and index of size r. To address the fact that the featurization vector $[1\ x_2]$ does not sum to 1, we can again shrink the elements of the component tensor to reduce the contribution from x_2. Since the element x_2 corresponds the index value of $i_2 = 2$, we can multiple all of the elements of $A^{(2)}$ with $i_2 = 2$ by γ to achieve the same effect as Eq. (3.31). To fully contract the network, we repeat this process for all tensors to the left of the output component, and then carry out the same procedure starting from $A^{(m+1)}$ and working right to left.

When working with models built using a TR rather than an MPS, the initialization method described above needs to be tweaked only slightly. In a TR there are no second-order component tensors, so there is no place to (conceptually) begin the contraction as a matrix-vector product akin to Eq. (3.30). Therefore, a starting point is chosen arbitrarily on the ring and then the tensors are initialized in the same manner as the third-order tensors in the MPS. Rather than mapping a stochastic vector and data vector to a new (approximately) stochastic vector, the tensors in a TR map a stochastic matrix and a data vector to a new (approximately) stochastic matrix.

3.3.3 TTN and STN initialization

The initialization of TTN and STN models is similar to the procedure that we described in the previous subsection for MPS/TR models, with the only significant difference being the order of the tensors and the connectivity of the network.

In a TTN model, for which all component tensors are third-order, the lowest layer of tensors are each contracted with two neighboring featurization vectors. These vectors have the same issue faced in Sec. 3.3.2, in that their elements sum to $1 + x_i$ rather than 1. This can once again be solved by shrinking the elements of the stochastic component tensors associated with x_i and x_{i+1}, which constitutes all but r of the elements in the tensor, using an appropriately chosen value of γ. The component tensors in the second layer and above do not contract directly with any featurization vectors, but instead take as input the intermediate vector generated by contraction of the previous layer. As a result, the component tensors in these layers can be initialized directly to a stochastic tensor without any further modification of the elements.

For an STN, the orders of the component tensors vary between second-order and sixth-order, with each tensor being contracted with a featurization vector. In order to properly generate the stochastic tensor elements for each component, we must trace through the contraction pattern of the spanning tree (see Eq. (3.11) for an example) to determine which index should sum to 1. The second-order components are straightforward, as they act on their featurization vector via the same matrix-vector product described in Eq. (3.28). Their tensor elements are therefore constrained to sum to 1 along the virtual index, and are shrunk in the same manner shown in Eq. (3.29). The initialization procedure then moves along the same path as the network contraction's scheme, focusing next on the component tensors which contract with the outputs of the second-order components, and so on. Each component is initialized to a stochastic tensor that is constrained to sum to 1 along the virtual index not associated with outputs from the previous contraction step, and then has the elements associated with x_i shrunken with an appropriate choice of γ.

Chapter 4

Interaction Decomposition of Tensor Network Models

This chapter is derived from previously published work by Convy and Whaley [28], which proposes a new algorithm for tensor network contraction that can separate out regressors of different degree in regression model.

4.1 Introduction

As discussed in Sec. 2.3.2, tensor network regression models generate predictions by contracting with the data tensor X, which is formed from the tensor product of featurization vectors shown in Eq. (2.20). These vectors each act on a single feature, and collectively generate a mapping from the original m-dimensional feature space into a 2^m tensor space. The purpose of the work in this chapter is to quantitatively assess how well tensor network models are able to utilize the exponential feature space induced by this tensor-product transformation. We shall focus specifically on models which are built upon the $[1,\ x]$ featurization from Novikov et al. [21] given in Eq. (2.22), since this will allow us to easily interpret different regions of the transformed space in terms of interactions between the original features. To this end, we introduce the *interaction decomposition* of a tensor network model, which casts the regression output as the sum of terms which each contain all feature products of a fixed degree. By applying this decomposition to tensor network models that were trained on a given machine learning task, we can determine the importance of each interaction degree to the final output of the model. Furthermore, by implementing new models that regress on only a subset of degrees, we can assess whether the tensor network models are under-utilizing those interactions.

The remainder of this chapter has the following structure. In Sec. 4.2, we describe the motivation and mechanics of the interaction decomposition, and in Sec. 4.3 we apply it to tensor network classifiers trained on the MNIST and Fashion MNIST datasets. From these tests, we find that some models utilize up to three-quarters of all interaction degrees generated by the tensor-product transformation, which collectively contain roughly 10^{19} different regressors. However, in Sec. 4.4 we determine that the tensor network classifiers are vastly under-utilizing the lower-degree interactions, since separate models trained using only interactions less than, e.g., sixth degree are able to achieve classifications accuracies very near those of the full regression models. We discuss the implications of these results and directions for future work in Sec. 4.5. The Appendix contains technical details about the procedure used to carry out the interaction decompositions, as a well as a tabulation of important numerical results.

4.2 The Interaction Decomposition

Throughout our discussion of tensor network regression in Sec. 2.3.3, the weight tensor W and data tensor X were treated principally as abstract objects, in that they were only operated on numerically via their component tensors. This was necessary on practical grounds, since the exponential scaling of both W and X makes it virtually impossible to perform operations on either tensor when the data has even a modest number of features m. That said, there is an obvious mathematical clarity that comes from working directly with W and X via the decomposition of Eq. (2.23), since the elements of X are simply products of the original features while the elements of W are the corresponding linear regression coefficients. If, for example, we wished to perform regression using only a specific portion of the feature products, then we could just set the elements of W for all other feature products to zero and learn the remaining parameters as usual. Such a straightforward modification is generally not possible when representing the weight tensor as a tensor network, since each element of W is a complicated function of all of the parameters in the model (see Sec. 2.3.3).

In this section we introduce the *interaction decomposition* of a tensor network, with the aim of recovering some of the fine-tuned control and interpretability that comes from an element-wise representation of W. In an interaction decomposition, the terms of the sum in Eq. (2.23) are grouped together by the number of features included in their product, for a total of $m+1$ groupings. The number of features in a given product is called its interaction degree, such that x_1 has degree 1, $x_1 x_2$ has degree 2, and so on, with the bias having degree 0. Under an interaction decomposition, the regression output $\vec{f}(\vec{x}; \mathcal{W})$ is written as

$$\vec{f}(\vec{x}; \mathcal{W}) = \sum_{j=0}^{m} \vec{d}^{\,(j)}(\vec{x}; \mathcal{W}), \tag{4.1}$$

where $\vec{d}^{\,(j)}(\vec{x}; \mathcal{W})$ is the contribution to the regression output from all terms of degree j. As with $\vec{f}(\vec{x}; \mathcal{W})$, these contributions are functions of both the original features $\vec{x}$ and the parameters $\mathcal{W}$ of the decomposed network. We discuss ways of interpreting this decomposition in terms of vector subspaces in Sec. 4.2.1.

Using Eq. (4.1), the relative importance of the jth interaction degree can be assessed by analyzing the average magnitude of $\vec{d}^{\,(j)}(\vec{x}; \mathcal{W})$, as well as its effect on the regression output. We carry out this analysis on TR and TTN models in Sec. 4.3. Furthermore, by choosing to keep only a specific subset $\mathcal{D}$ of the decomposition terms in Eq. (4.1), it is possible to construct a new type of regression model which we call a *$\mathcal{D}$-degree tensor network.* These networks utilize the same parameterization scheme for W as normal tensor network models of the same architecture, but are restricted to generating only the feature products of degrees contained in $\mathcal{D}$. By comparing the performance of a full tensor network with that of a $\mathcal{D}$-degree version of the network, we can quantify how effectively the standard network is able to utilize interaction degrees within $\mathcal{D}$. We introduce these models and perform numerical tests on them in Sec. 4.4.

4.2.1 Interaction Subspaces

In the context of vector spaces, the weight tensor W acts as a parameterized multilinear map between the data tensor space $\mathbb{X}$, which we take to be $\mathbb{R}^{2^m}$, and the label space $\mathbb{Y}$. Under this construction, the data tensor X is simply a vector within $\mathbb{X}$ whose elements are generated from the features in $\vec{x}$ via the tensor-product operations of Eq. (2.20). The nature of the $\vec{d}^{\,(j)}(\vec{x};\mathcal{W})$ terms from Eq. (4.1) can be understood by considering the action of W on a particular subspace decomposition of X. Using the definition of $\vec{h}^{(i)}(x_i)$ from Eq. (2.22), we can expand X on a standard basis of $\mathbb{X}$ as

$$X = \bigotimes_{i=1}^{m} (\vec{e}_0^{\,(i)} + x_i \vec{e}_1^{\,(i)}) = \sum_{i_1,\ldots,i_m=0}^{1} x_1^{i_1} \cdots x_m^{i_m}\ \vec{e}_{i_1}^{\,(1)} \otimes \ldots \otimes \vec{e}_{i_m}^{\,(m)}, \tag{4.2}$$

where $\{\vec{e}_0^{\,(i)}, \vec{e}_1^{\,(i)}\}$ spans the two-dimensional space inhabited by $\vec{h}^{(i)}(x_i)$. In words, Eq. (4.2) shows that the tensor products $\vec{e}_{i_1}^{\,(1)} \otimes \ldots \otimes \vec{e}_{i_m}^{\,(m)}$ form a basis for $\mathbb{X}$ upon which the coefficients of X take the form of feature products.

Looking at the structure of the tensor-product basis used in Eq. (4.2), it is possible to divide $\mathbb{X}$ into the direct sum of subspaces $\{\mathbb{D}^{(0)}, \ldots, \mathbb{D}^{(m)}\}$, where $\mathbb{D}^{(j)}$ is a subspace spanned by the basis tensors $\vec{e}_{i_1}^{\,(1)} \otimes \ldots \otimes \vec{e}_{i_m}^{\,(m)}$ such that j of the component basis vectors are of the form $e_1^{(i)}$ and $m - j$ are of the form $\vec{e}_0^{\,(i)}$. The coefficients of X on the bases in $\mathbb{D}^{(j)}$ consist of feature products of degree j, so we therefore refer to $\mathbb{D}^{(j)}$ as the *degree-j subspace* of $\mathbb{X}$. The dimension of $\mathbb{D}^{(j)}$ is given by

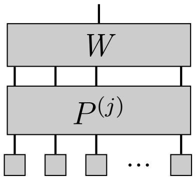
$$\dim(\mathbb{D}^{(j)}) = \binom{m}{j}, \tag{4.3}$$

which is the number of ways to draw j features from the total set of m features. The dimension of the combined feature space for a set $\mathcal{D}$ of degrees, denoted $\dim(\mathcal{D})$, is then

$$\dim(\mathcal{D}) = \sum_{j \in \mathcal{D}} \binom{m}{j}, \tag{4.4}$$

which is simply the sum of the subspace dimension for each degree in $\mathcal{D}$. If we denote the projection into the degree-j subspace as $P^{(j)}$, then the interaction decomposition becomes

$$f_k(\vec{x};\mathcal{W}) = \sum_{j=0}^{m} \sum_{i_1,\ldots,i_m=0}^{1} W_{ki_1\ldots i_m} (P^{(j)} X)_{i_1\ldots i_m} \to \sum_{j=0}^{m}$$



, (4.5)

where the regression output is a sum of contractions between W and the projection of X into each of the $m+1$ degree subspaces. Due to the linearity of tensor contractions, this equality can be easily verified by performing the sum over j first, since $\sum_j P^{(j)}$ gives the identity.

The form of Eq. (4.5) provides two interpretations of the degree contributions $\vec{d}^{\,(j)}(\vec{x};\mathcal{W})$. If we consider the projector $P^{(j)}$ acting on X, as originally envisioned, then $\vec{d}^{\,(j)}(\vec{x};\mathcal{W})$ is the contraction of W with the portion of X that inhabits $\mathbb{D}^{(j)}$. This could be viewed as a form of data tensor preprocessing, where elements of X corresponding to interaction degrees other than j are removed. Alternatively, the tensor diagrams in Eq. (4.5) show that it is equally valid to consider $P^{(j)}$ acting on the weight tensor W. Under this interpretation, the set $\{\vec{d}^{\,(j)}(\vec{x};\mathcal{W})\}_{j=0}^{m}$ consists of regressions on X performed by different models, each derived from a common tensor W by keeping only those elements corresponding to interactions of degree j. We will shift between these two interpretations freely throughout the remainder of this section, describing the interaction decomposition as a procedure which picks out different pieces of a tensor network model and which restricts the set of feature products that can be used for regression.

4.3 Interaction Decompositions of TR and TTN Models

The interaction decomposition of a TR (see Sec. 3.1.2) or TTN (see Sec. 3.1.3) regression model allows us to quantify the relative importance of the jth interaction degree to the overall value of the output. This information is not available when performing the standard contraction operations laid out in Eqs. (3.2, 3.8), since the elements of the intermediate tensors are sums of contributions from a wide range of interaction degrees, making it impossible to separate out the impact of any one degree. Given the unfavorable scaling of Eq. (4.3), a brute force evaluation of each feature product in $\vec{d}^{\,(j)}(\vec{x};\mathcal{W})$ is also impractical for even modest values of j. In Appendix 4.6.1, we describe an alternative procedure that efficiently contracts the TR and TTN component tensors in a manner that ultimately yields the same output as the standard contraction, but also separates out the various $\vec{d}^{\,(j)}(\vec{x};\mathcal{W})$ contributions.

In this section, we carry out these interaction decompositions on TR and TTN models that were trained to classify digits from the MNIST and Fashion MNIST datasets introduced in Sec. 2.3.1. These datasets have been widely used to evaluate tensor network models in the literature, and thus serve as reasonable benchmarks for our analysis. Given that the number of operations needed for a full interaction decomposition can scale quadratically with the number of features (see Appendix 4.6.1), we resized each image from 28×28 pixels to 8×8 pixels in order to reduce the computational burden of the tests. The grayscale pixels were also normalized to floating-point values on the range $[-0.5, 0.5]$ to improve the numerical stability of the networks. The bond dimension of the TR and TTN models was set to 20, providing them with sufficient representational power without excessive overfitting. The regression output $\vec{f}(\vec{x};\mathcal{W})$ was fit against one-hot encodings of the digit labels and

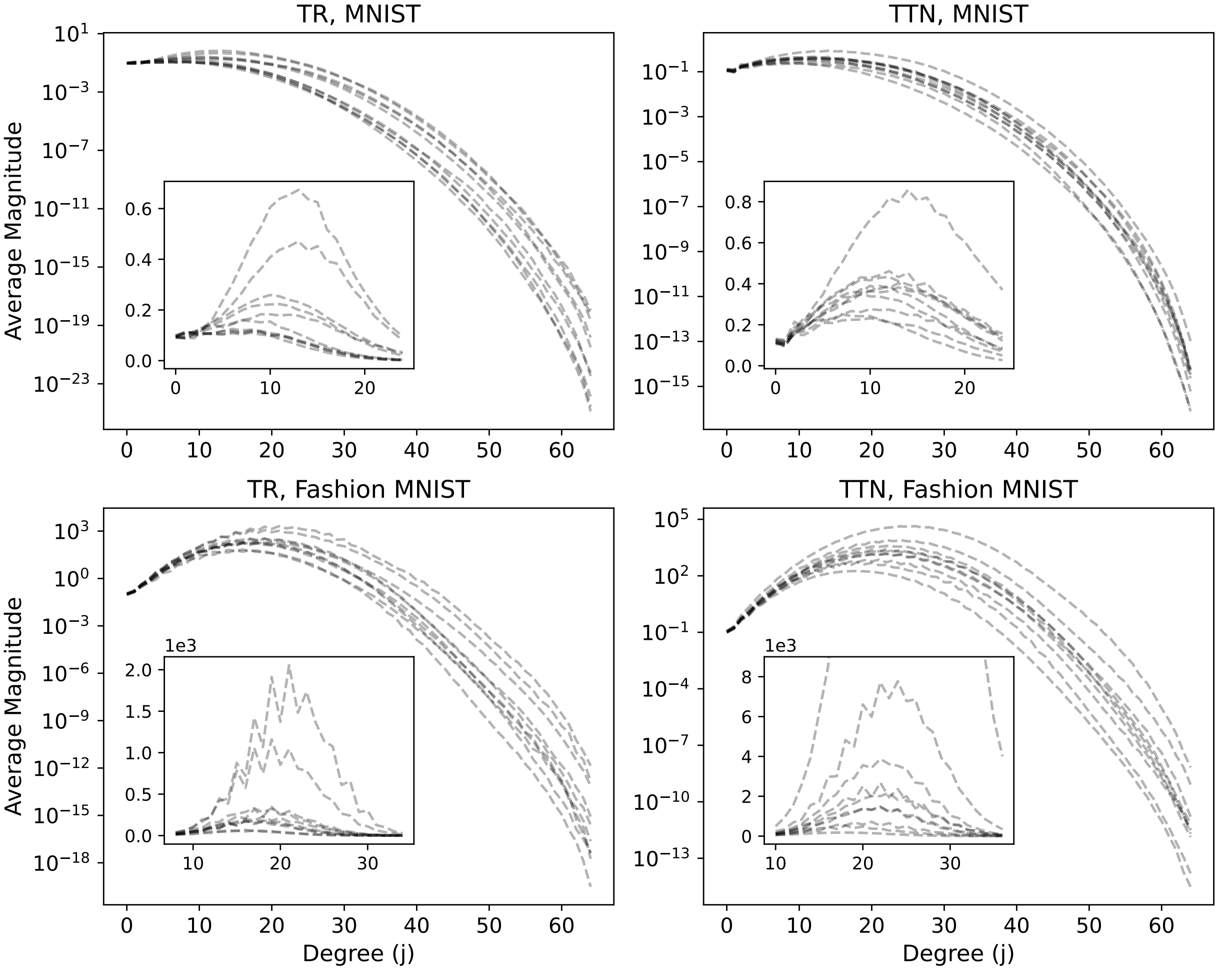


Figure 4.1: Plots of the L1 norm of $\vec{d}^{\,(j)}(\vec{x};\mathcal{W})$, averaged across the MNIST and Fashion MNIST test datasets, for ten TR models and ten TTN models trained using all interaction degrees. Each dashed line represents the average magnitudes from one of the models, which is plotted against the interaction degree j of the contribution. The trend for all models is broadly the same, with magnitudes starting near 10^{-1} for the bias term and then gradually rising to a peak before dropping off significantly by degree 45. Large variations in magnitude can be seen when comparing individual TR models and TTN models.

optimized using gradient descent (see Sec. 3.2.4) with a mean squared error loss function. During training the networks were contracted normally, with the interaction decomposition being performed at the end using the test dataset.

To begin our analysis, we focus first on the magnitudes of the different $\vec{d}^{\,(j)}(\vec{x};\mathcal{W})$ contributions. To produce a single magnitude for each degree, we computed the ℓ^1 norm of $\vec{d}^{\,(j)}(\vec{x};\mathcal{W})$ for each image in the test dataset, and then averaged over the set. Figure 4.1 shows the resulting magnitudes for ten TR models and ten TTN models, all trained using the same hyperparameters but with different initial values for the tensor elements. Across both datasets the TR and TTN plots show a similar pattern, with the degree magnitudes starting at approximately 10^{-1} for $j = 0$ and then growing steadily to some maximum value before declining again at larger j. The size and location of the peak varies significantly between the MNIST and Fashion MNIST models, with the MNIST models peaking from 0.1 to 1 at around degrees 10 to 15 while the Fashion MNIST models peak from 10^2 to 10^4 at around degrees 17 to 23. After the peak, the magnitudes begin to drop off precipitously, with interaction degrees greater then 45 typically having contributions orders of magnitude smaller than those from degrees before the peak. The inset plots of Figure 4.1 show that there is a significant amount of variation between individual models of a given network type and dataset, with some models having magnitudes 10 or even 100 times larger than others.

A significant limitation of the magnitude analysis from Figure 4.1 is that it can be difficult to assess the true importance of a degree contribution using only its average magnitude. Indeed, even if a set of degrees all have small individual magnitudes, their cumulative effect on the output may still be important. To better assess the “usefulness” of the degree contributions, we computed the accuracy of the TR and TTN classifiers as a function of interaction degree, both individually and cumulatively. Figure 4.2 shows these accuracies after averaging over the models of each network type. The cumulative accuracy (shown using a solid line) of degree j denotes the accuracy of the output generated by the sum of all degree contributions less than or equal to j, while the individual accuracy of degree j (shown as a point on the scatter plot) gives the performance of $\vec{d}^{\,(j)}(\vec{x};\mathcal{W})$ alone. The dimension of the expanded feature space corresponding to each data point is determined by Eq. (4.4).

From the plots of cumulative accuracy, we can see that the average performance of both the TR and TTN networks plateaus at slightly over 98% on MNIST (98.31% for the TRs and 98.47% for the TTNs when all degrees are included), which is consistent with prior work [22][56][57]. On Fashion MNIST the accuracies are signficantly lower, at 82.73% for the TR and 83.43% for the TTN.[1] These final accuracy values are of less significance to us than the interaction degree at which the curve flattens. On MNIST (Fashion MNIST) this occurs at approximately $j = 31$ (44) for the TRs, and at $j = 38$ (54) for the TTNs. Looking back at the magnitudes from Figure 4.1, this indicates that even contributions on the order of 10^{-3} can still improve the performance of the classifier. Interestingly, all four of the accuracy curves show a temporary flattening before degree 10, followed by a second upward rise. This effect is least visible on the TR MNIST curve and most visible on the two

[1] Tensor networks can achieve significantly higher accuracies on 28 x 28 Fashion MNIST [24][56][94], but the decrease in performance at 8 x 8 is much more severe than for standard MNIST, due to the greater image complexity. For comparison to more state-of-the-art methods, an Inception convolutional network [95] can achieve accuracies of 99.09% on 8×8 MNIST and 86.5% on 8×8 Fashion MNIST (see Appendix 4.6.3)

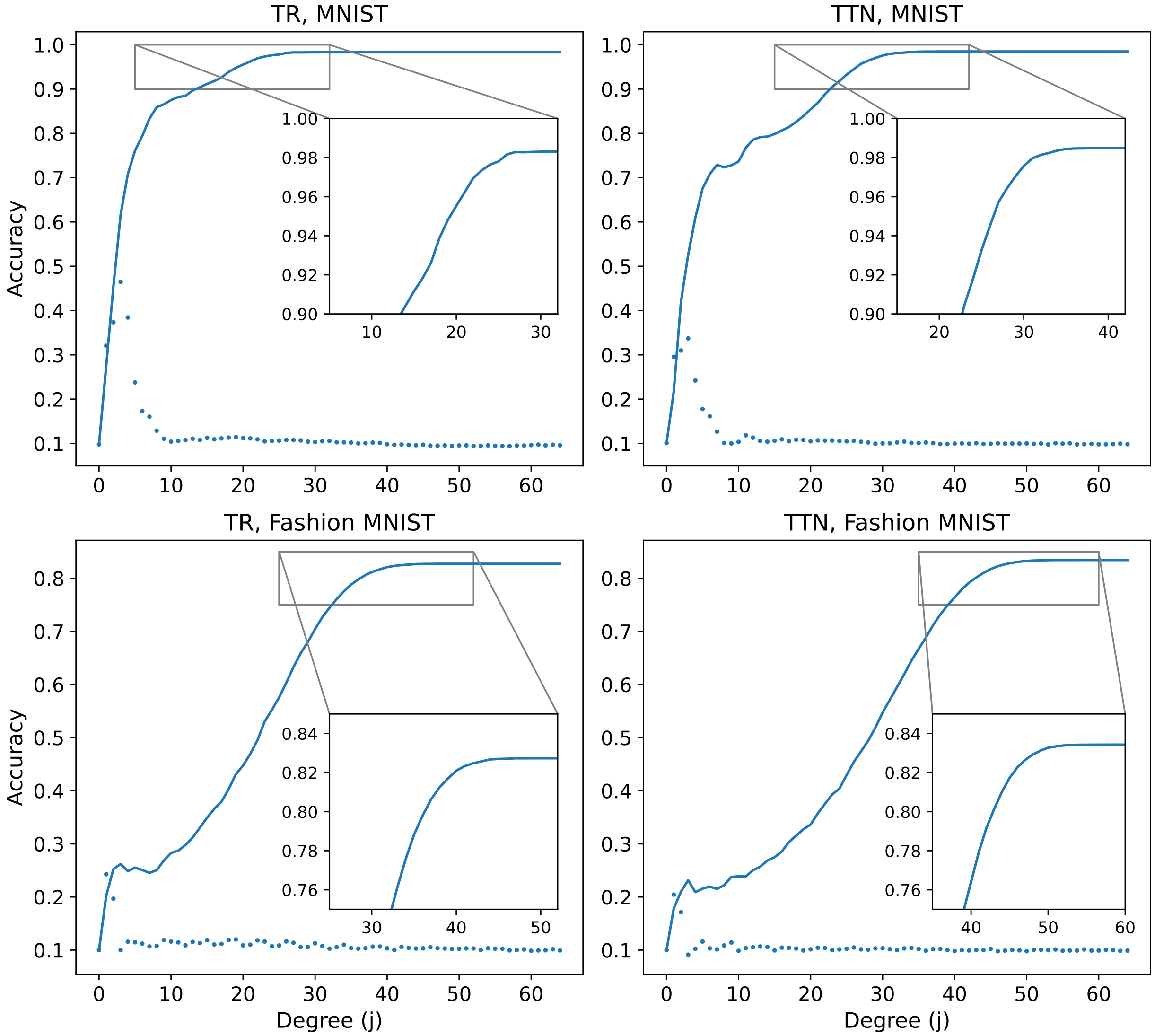


Figure 4.2: Plots of the average classification accuracy for the TR models and TTN models, trained using all interaction degrees, as a function of interaction degree j, using the MNIST and Fashion MNIST test datasets. The scatter plots show the accuracy of each $\vec{d}^{\,(j)}(\vec{x};\mathcal{W})$ term individually, while the solid line shows the accuracy of the sum of contributions from all degrees less than or equal to its position on the x-axis. On MNIST (Fashion MNIST), the cumulative accuracies of the TR and TTN models are equal to 98.31% (82.73%) and 98.49% (83.43%) respectively when all degrees are included, with the performance plateauing at degree 31 (44) for the TRs and 38 (54) for the TTNs. The accuracies of the individual contributions are all very low, with the vast majority of interaction degrees having almost no independent classification ability.

Fashion MNIST curves, with the latter pair of curves seeing most of their accuracy gains after degree 10.

Based on the accuracies of the individual contributions $\vec{d}^{\,(j)}(\vec{x};\mathcal{W})$, which are shown in Figure 4.2 using scatter plots, it is clear that only the first few interaction degrees are having their coefficients optimized such that they can classify images independently. The remaining contributions, which constitute the vast majority of regressors, have accuracies close to 10% and therefore do not separate the different digit classes to any appreciable extent when used in isolation. This suggests that the higher-degree $\vec{d}^{\,(j)}(\vec{x};\mathcal{W})$ have been trained essentially to correct or finesse the cumulative output from the lower degrees, since the cumulative accuracy continues to increase as their outputs are incorporated. This trend is particularly marked for the Fashion MNIST models, where only degrees 1 and 2 have accuracies above 12%. Indeed, plots C and D from Figure 4.1 show that many of the regressors in these models are being used to cancel out the large magnitudes of the intermediate degrees, since the final regression output needs to be roughly in the range [0, 1] to achieve a reasonable loss value.

4.4 Interaction Decompositions as Regression Models

In Sec. 4.3, we used the interaction decomposition as a tool to analyze tensor network models that had been trained using standard methods. As a result, the parameters of each model were optimized under the assumption that every interaction degree would contribute to the final output, without any truncation or isolation. This offers the greatest flexibility to the model in principle, but it can also obscure the potential success that a single degree or subset of degrees might have had if the parameters of the network had been optimized to improve their performance specifically.

In light of this fact, we propose a new type of tensor network model called the $\mathcal{D}$-degree tensor network. In these models, only interaction degrees in the set $\mathcal{D}$ are used to construct the regression output, such that

$$\vec{f}(\vec{x};\mathcal{W}) = \sum_{j\in\mathcal{D}} \vec{d}^{\,(j)}(\vec{x};\mathcal{W}). \tag{4.6}$$

Comparing this expression to the full interaction decomposition given in Eq. (4.1), it is clear that if $\mathcal{D}$ is the set of all interaction degrees (i.e. if $\mathcal{D} = \{0, 1, ..., m\}$), then the corresponding $\mathcal{D}$-degree network is equivalent to a standard tensor network with the same structure. However, we will focus our attention on models where $\mathcal{D}$ contains only a fraction of the $m+1$ possible interaction degrees. By restricting the regression in this manner, we are effectively inducing sparsity in the weight tensor W by zeroing the coefficients for all interaction degrees not included in $\mathcal{D}$. Unlike in the case of sparse neural networks [96][97], this sparsity does not necessarily lead to a reduction in the number of trainable parameters or to an improvement in the computational overhead. Instead, the sparsity leads to a simplification in the structure of the regression function, which can yield a model that is more easily interpretable while still achieving the same level of performance.

Using the decomposition procedure described in Appendix 4.6.1, it is possible to efficiently train $\mathcal{D}$-degree models on the same regression tasks used in Sec. 4.3, and thus compare their classification accuracies with those shown in Figure 4.2. For our tests, we selected $\mathcal{D}$-degree models that fell into two categories: the *cumulative-j* models, in which all degrees less than or equal to j are included in the output, and the *degree-j* models, in which the output is simply the contribution from the jth degree:

$$\text{Cumulative-}j:\ \vec{f}(\vec{x};\mathcal{W}) = \sum_{j'=0}^{j} \vec{d}^{\,(j')}(\vec{x};\mathcal{W}), \qquad \text{Degree-}j:\ \vec{f}(\vec{x};\mathcal{W}) = \vec{d}^{\,(j)}(\vec{x};\mathcal{W}). \tag{4.7}$$

These two groups describe only a small portion of the $2^{m+1}-1$ possible $\mathcal{D}$-degree models, but they will allow us to easily compare our results with the plots in Figure 4.2. For our numerical tests, we trained the models on 8×8 images from the MNIST and Fashion MNIST datasets prepared in the same manner described in Sec. 4.3. The $\mathcal{D}$-degree models take somewhat longer to train than standard tensor network models due to the added complexity of the interaction decomposition, but their times are still on par with those of neural network models (see Appendix 4.6.3).

Figure 4.3 shows average accuracies of the cumulative-j (blue plots) and degree-j (orange plots) models as a function of j, with each data point representing an average across ten models. These averages are plotted alongside the cumulative data from Figure 4.2, which shows the performance of the full tensor network models as a reference. We emphasize that the cumulative-j and degree-j curves in Figure 4.3 are computed in precisely the same manner as the line and scatter plots from Figure 4.2, except that the models which generated Figure 4.2 were trained using all of the interaction degrees rather than just the specific subset being plotted. The plots for the $\mathcal{D}$-degree models omit results for $j = 0$, since those models contain only the bias term and thus predict the same digit for every image. The data used to generate these plots is given in Appendix 4.6.2.

The first trend to observe from Figure 4.3 is that both the cumulative-j and degree-j models have accuracies that are significantly greater than the corresponding cumulative accuracy at degree j from the full tensor network models. This performance gap is notable, because it implies that the standard models are utilizing the feature-product regressors in a highly inefficient manner. For MNIST in particular, the degree-j classifiers with $j > 3$ were able to perform within 0.5% of the full model. As a comparison, the cumulative MNIST accuracy of the regular TR and TTN models using degrees $0-4$ is only 71% and 61% respectively. The disparity is even larger when looking at the single-degree accuracies from Figure 4.2, which show that most individual $\vec{d}^{\,(j)}(\vec{x};\mathcal{W})$ were unable to classify images at all when trained as part of a full tensor network model. When those degree contributions were optimized directly, however, they were able to perform classification with more than 98% accuracy. The degree-j models did not perform as well on Fashion MNIST, though they still achieved accuracies that were vastly higher than the corresponding cumulative accuracies from Figure 4.2. The cumulative-j models, on the other hand, were able to closely match the performance of the standard models, with the cumulative-10 TR and cumulative-8 TTN models coming within 0.1% of their full-degree counterparts.

The dashed horizontal lines in Figure 4.3 mark the accuracy of the full tensor network

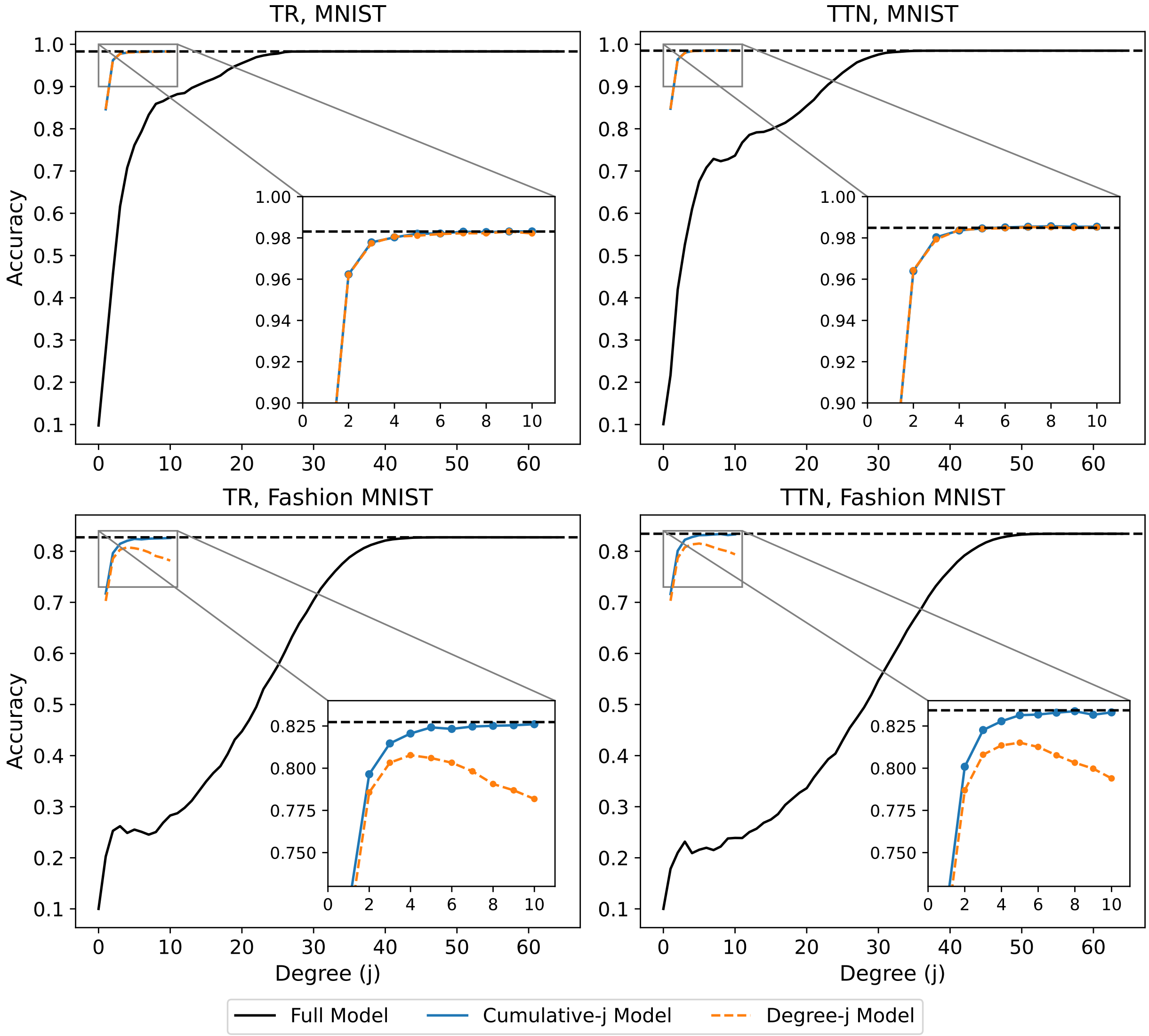


Figure 4.3: Plots of average accuracy on the MNIST and Fashion MNIST test datasets for $\mathcal{D}$-degree models parameterized by TRs and TTNs, with ten models being averaged for each degree. The solid black line indicates the cumulative accuracy of the standard tensor network models analyzed in Sec. 4.3, as plotted in Figure 4.2, with the dashed line showing the accuracy when all interaction degrees are included. The degree-j models generally demonstrate worse performance than the cumulative-j models, though the difference is very small on MNIST. The cumulative-j models were able to closely match the accuracies of the full models on both datasets, and even slightly exceed their performance on MNIST.

models when every degree contribution is included. Using these values as a benchmark, we can see that several of the cumulative-j TTN models ($6 \leq j \leq 10$) and degree-j TTN models ($7 \leq j \leq 10$) actually *outperformed* the corresponding full model on the MNIST dataset. This is a counter-intuitive result, as it suggests that regressing on all of the interaction degrees can actually yield slightly worse results than regressing on only a small subset of them. A cumulative-8 TTN model, for example, uses only one-billionth of the feature products contained within the data tensor X, yet achieves an average accuracy roughly 0.1% higher than a TTN model which has access to all of X.

Finally, we note that a comparison can be made between the performance of these $\mathcal{D}$-degree network models, which constrain the feature product coefficients to all be generated by the same low-rank tensor network, and a more general multilinear regression model in which every coefficient can be determined arbitrarily. In Appendix 4.6.3, we give results for this type of unconstrained regression on features products up to degree 4, which shows that the cumulative-j models achieve accuracies very near the arbitrary models of degree j, and can outperform them for larger values of j even when the tensor network models contain fewer trainable parameters. This demonstrates the utility of incorporating more interactions (up to a point), since constrained regression on higher-degree feature products is more effective than unconstrained regression on lower-degree feature products.

4.5 Discussion

The exponential feature space induced by the transformation in Eq. (2.20) lies at the heart of tensor network regression, and there is no doubt that these models utilize it to achieve a level of performance that far exceeds standard linear regression. That said, it is easy to feel incredulous toward the idea that tensor network models, or indeed any regression model, could truly make use of the 2^{64} different regressors that are generated from an 8×8 image. The goal of our work here has been to develop the interaction decomposition as a tool to test this claim, and then apply it to tensor network models under a pair of standard machine learning tasks. By evaluating the magnitudes and accuracies of the different interaction degrees, we can begin to draw conclusions about how effectively the exponential space is being utilized.

To this end, our results from Sec. 4.3 show that more than half of the interaction degrees contributed meaningfully to the output of the classifiers, with the Fashion MNIST TTN models in particular using up to degree 50. In the language of Sec. 4.2.1, this indicates that the tensor network models are utilizing a portion of the expanded feature space $\mathbb{X}$ that has a dimension on the order of 10^{19}, which can be computed by summing Eq. (4.3) across all significant degrees. It is important to note, however, that Figure 4.2 only shows the change in accuracy for the jth interaction degree when the entirety of subspace $\mathbb{D}^{(j)}$ is incorporated into the regression. It may very well be that the models in Sec. 4.3 are utilizing only a small portion of *this* space, and thus the number of relevant feature products could be far smaller than the upper-bound of 10^{19}. The interaction decomposition cannot separate out different parts of a given degree-j subspace, so future work might look into alternate algorithms that are able to divide up these spaces into meaningful components.

For a given interaction degree, one can ask not only *if* the set of feature products is being utilized by a tensor network model, but also *how well* the model is using them relative to some standard. In Sec. 4.4 we introduced the $\mathcal{D}$-degree tensor network to serve as this standard, since its parameters could be trained to maximize performance using only a specific subset of interaction degrees. In our tests, the networks were limited to interaction degrees of at most 10, which corresponds to an expanded features space of dimension at most 10^{11} as given by Eq. (4.4). The results shown in Figure 4.3 demonstrate that the full tensor network models are significantly under-utilizing the lower-degree interactions, since the $\mathcal{D}$-degree models are able to achieve accuracies up to 60 percentage points higher when constrained to those same degrees. This under-utilization was especially acute for Fashion MNIST, where the full models only reached cumulative accuracies of 25% for the first ten degrees, despite the fact that the cumulative-10 models had accuracies near 83%.

More significantly, some $\mathcal{D}$-degree models trained using only the first six interaction degrees were able to achieve accuracies on MNIST that were *greater* than those of models trained using all degrees. While this could simply be due to more overfitting in the full models, it might also point to inherent limitation in the tensor network representation of W. We know, for example, that the regression coefficients in a tensor network model are necessarily coupled together by the elements of the component tensors, which may force the model to use suboptimal coefficients for the lower interaction degrees in order to avoid harmful contributions from the higher degrees. Given that detailed, "under-the-hood" analyses of these models are possible using methods such as the interaction decomposition introduced here, the existence and nature of this compromise seems like a promising area for further study.

Taken together, the results discussed here support the following two conclusions:

1. Common tensor network models are capable of utilizing regressors from a large portion of the expanded feature space generated by the featurization from [21].

2. However, a comparable level of performance may also be achieved by regression on a minuscule fraction of that same space.

For those looking to use tensor network models for machine learning, there is cause here for both optimism and caution. While the first conclusion makes it clear that tensor network regression models can incorporate useful information from a wide range of interaction degrees, the second conclusion implies that it is difficult for these models to extract any *unique* information from the higher-degree regressors. In light of this, we believe that the $\mathcal{D}$-degree tensor network models, which have been absent in the literature up to this point, represent a promising approach for tensor network regression. Using these models, it is possible to exploit the representational efficiency of tensor networks while constraining the regression to a reasonable and interpretable set of feature products based on the inherent complexity of the dataset.

4.6 Appendix

4.6.1 Procedure for the Interaction Decomposition

In this subsection, we describe a procedure that can be used to carry out the interaction decomposition of any tensor network. At its core is a tensor operation that we call the *degree-preserving tensor product*, denoted $\tilde{\otimes}$, which is defined between $m+1$th order tensor A and $n+1$th order tensor B as

$$(A \tilde{\otimes} B)_{ji_0 \ldots i_{m-1} k_0 \ldots k_{n-1}} = \sum_{j_a + j_b = j} A_{j_a i_0 \ldots i_{m-1}} B_{j_b k_0 \ldots k_{n-1}}, \tag{4.8}$$

where the resulting tensor is of order $m+n+1$ and $0 \leq j \leq \max(j_a) + \max(j_b)$. Note that this operation attaches special significance to the first dimension, which we will hereafter refer to as the *degree index*. As shown in Eq. (4.8), the jth slice of $A \tilde{\otimes} B$ along the degree index is given by the sum of tensor products taken between slices of A and of B, such that the sum of the degree indices for those slices is equal to j. Like the normal tensor product, the degree-preserving tensor product is associative and commutative up to a permutation of the (non-degree) indices, and multilinear in its two arguments. Using this new variation of the tensor product, we can also define a *degree-preserving contraction* in the same manner as Eq. (2.4), such that the contraction of fourth-order tensors A and B is given by

$$C_{jklqr} = \sum_{j_a + j_b = j} \sum_i A_{j_a kil} B_{j_b qri}. \tag{4.9}$$

The utility of these degree-preserving operations becomes apparent if we alter the featurization in Eq. (2.22) to be

$$H^{(i)}(x_i) = \begin{bmatrix} 1 & 0 \\ 0 & x_{i,} \end{bmatrix}, \tag{4.10}$$

which simply embeds $\vec{h}^{(i)}(x_i)$ along the diagonal of a 2×2 matrix. Note that the degree index of this tensor matches up with the interaction degree of its non-zero elements, since the first row (index 0) is a constant while the second row (index 1) is x_i. This correspondence is maintained by the degree-preserving tensor product of $H^{(i)}$ and $H^{(k)}$:

$$H^{(i)} \tilde{\otimes} H^{(k)} = \left[\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & x_i \\ x_k & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & x_i x_k \end{bmatrix} \right], \tag{4.11}$$

where the non-zero elements all have an interaction degree equal to their position along the degree index. Since the zero elements do not contribute anything during a tensor contraction, Eq. (4.11) also indicates that any degree-preserving contraction between $H^{(i)}$ and $H^{(k)}$ would likewise maintain the correspondence between degree index and interaction degree.

Using the degree-preserving tensor product and contraction operations, along with the new featurization maps $H^{(i)}(x_i)$, the interaction decomposition of a tensor network regression model can be carried out using the following procedure:

1. Add a degree index of size one (i.e., an index that can only take a value of 0) to each component tensor of the network representing W. This increases the order of each tensor by one, but leaves the actual number of elements unchanged.

2. Construct (implicitly) a modified data tensor $\tilde{X}$ using the mappings from Eq. (4.10), such that $\tilde{X}(\vec{x}) = H^{(1)}(x_1) \tilde{\otimes} H^{(2)}(x_2) \tilde{\otimes} ... \tilde{\otimes} H^{(m)}(x_m)$.

3. Use degree-preserving contraction operations to contract $\tilde{X}$ with the tensor network, following whichever efficient contraction scheme is appropriate for the network architecture of the model.

4. If the decomposition is being used to contract a $\mathcal{D}$-degree network, then the degree index of all intermediate tensors can be be truncated to the largest degree in $\mathcal{D}$.

Since the contraction of the network is done using degree-preserving contractions, the contributions from each interaction degree are kept separate throughout the entire process. The final output of the interaction decomposition (without truncation) is a second-order tensor of the form

$$F(\vec{x}; \mathcal{W}) = \left[\vec{d}^{\,(0)}(\vec{x}; \mathcal{W}), \;\; \vec{d}^{\,(1)}(\vec{x}; \mathcal{W}), \;\; ..., \;\; \vec{d}^{\,(m)}(\vec{x}; \mathcal{W})\right], \tag{4.12}$$

where $\vec{d}^{\,(j)}(\vec{x}; \mathcal{W})$ is the degree-j contribution to the combined regression output $\vec{f}(\vec{x}; \mathcal{W})$. The computational cost of the procedure described above is best understood in terms of how much additional complexity it adds on top of a standard contraction of the network. This complexity comes from two sources: larger intermediate tensors due to the addition of the degree index, and an extra sum over the degree index that is present in the degree-preserving tensor product from Eq. (4.8). The first contribution is easy to characterize, since adding a degree index simply increases the size of the original tensors by a factor that is on the order of the maximum interaction degree j_{max} in the decomposition. The second contribution is more subtle, since the number of terms in the tensor-product sum depends on the relative sizes of the degree indices of the two inputs. Consider again the tensor product between A and B from Eq. (4.8), and let $\bar{j}_a$ and $\bar{j}_b$ be the largest value of the degree index for A and B respectively, with $\bar{j} = \bar{j}_a + \bar{j}_b$ and $\bar{j}_a \leq \bar{j}_b$. Then it it can be shown that the number of terms s needed to generate all $\bar{j} + 1$ slices of $A \tilde{\otimes} B$ is given by

$$s = (\bar{j}_a + 1)(\bar{j}_b + 1), \tag{4.13}$$

which scales as $\mathcal{O}(\bar{j}_a \bar{j}_b)$. This means that, for a fixed $\bar{j}$, the value of s can range from a minimum of $\bar{j} + 1$ if $\bar{j}_a = 0$ to a maximum of $\frac{1}{4}(\bar{j})^2 + \bar{j} + 1$ for the fully symmetric case when $\bar{j}_a = \bar{j}_b = \frac{1}{2}\bar{j}$. Given that the last contractions in the interaction decomposition will have $\bar{j}$ on the order of j_{max}, this means that the most complex degree-preserving tensor products can either scale as $\mathcal{O}(j_{max}^2)$ or $\mathcal{O}(j_{max})$, depending on the amount of symmetry between the two input tensors. The fact that more symmetric contraction schemes can lead to worse scaling (quadratic rather than linear in j_{max}) is an interesting property of this method, although the use of such schemes may still be desirable due to other computational advantages.

4.6.2 Tabulation of $\mathcal{D}$-degree Model Performance

Table 4.1 and Table 4.2 show the results of the numerical tests described in Sec. 4.4 and plotted in Figure 4.3 for MNIST and Fashion MNIST respectively, along with the relevant cumulative accuracy values for the full models from Figure 4.2. Each value represents the average percent test accuracy across ten different initializations of the given model type, with the standard error of the last digit shown in parentheses. For the cumulative-j and degree-j models the column label denotes the value of j, while for the full models they denote the cumulative accuracy of the output up to the jth interaction degree.

4.6.3 Regression Model Comparisons

In Table 4.3, we compare our TR and TTN models with several low-order multilinear models and a deep learning model, in terms of their number of trainable parameters, computation time per epoch, and average accuracies on the 8×8 image datasets. The linear, bilinear, trilinear, and tetralinear regression models perform unconstrained regression on feature products of degree less than or equal to 1, 2, 3, and 4 respectively, which are the same regressors used by the cumulative-j models for $1 \leq j \leq 4$. By “unconstrained”, we mean that the coefficients for each feature product can be set arbitrarily rather than being generated by a low-rank tensor network. To offer a comparison with state-of-the-art neural network algorithms, we also provide the corresponding numbers for a convolutional neural network (CNN) model based on the Inception [95] architecture, which contains the most parameters and achieves the best performance on both datasets.

	1	2	3	4	5	6	7	8	9	10	64
Full TR	27.5(5)	46(1)	62(1)	70.8(8)	76(1)	79(2)	83(3)	86(3)	87(3)	88(3)	98.31(2)
Cumulative TR	84.6(1)	96.23(3)	97.79(3)	98.03(3)	98.21(2)	98.21(3)	98.30(4)	98.28(3)	98.31(4)	98.31(2)	-
Degree TR	84.67(6)	96.22(2)	97.74(3)	98.06(3)	98.11(2)	98.19(3)	98.23(2)	98.22(3)	98.31(3)	98.22(3)	-
Full TTN	21.7(6)	42(1)	53(1)	61(1)	68(1)	71(2)	73(2)	72(2)	73(2)	74(3)	98.49(3)
Cumulative TTN	84.76(7)	96.39(2)	98.03(2)	98.36(2)	98.46(1)	98.51(2)	98.54(3)	98.57(1)	98.54(1)	98.54(1)	-
Degree TTN	84.76(8)	96.44(2)	97.93(2)	98.39(2)	98.44(3)	98.47(2)	98.52(3)	98.53(2)	98.49(2)	98.51(1)	-

Table 4.1: Table of average accuracy vs degree for the six different model types on MNIST, for Figure 4.3.

	1	2	3	4	5	6	7	8	9	10	64
Full TR	20.2(4)	25.3(5)	26(1)	25(1)	25.5(9)	25(1)	25(1)	25(1)	27(1)	28(1)	82.73(9)
Cumulative TR	71.73(6)	79.64(7)	81.47(5)	82.05(8)	82.42(7)	82.3(1)	82.47(6)	82.51(7)	82.54(9)	82.60(7)	-
Degree TR	70.27(8)	78.57(5)	80.33(8)	80.77(7)	80.60(6)	80.3(1)	79.82(7)	79.07(6)	78.7(1)	78.18(9)	-
Full TTN	17.8(3)	21.0(3)	23.2(8)	21(1)	22(1)	22(1)	22(1)	22(1)	24(2)	24(2)	83.43(6)
Cumulative TTN	71.63(7)	80.09(6)	82.26(7)	82.78(6)	83.14(6)	83.18(7)	83.29(7)	83.37(6)	83.17(9)	83.31(4)	-
Degree TTN	70.30(4)	78.69(6)	80.80(5)	81.35(6)	81.51(5)	81.26(8)	80.76(8)	80.33(9)	80.0(1)	79.4(1)	-

Table 4.2: Table of average accuracy vs degree for the six different model types on Fashion MNIST, for Figure 4.3.

	Parameters	Seconds per epoch	MNIST accuracy	Fashion MNIST accuracy
Linear	65	2	84.7(1)	71.35(8)
Bilinear	2,081	2	96.47(1)	80.20(6)
Trilinear	43,745	3	98.13(2)	82.32(7)
Tetralinear	679,121	11	98.46(1)	82.33(3)
Full TR	51,200	2	98.31(2)	82.73(9)
Full TTN	250,560	3	98.49(3)	83.43(6)
Cumulative-10 TR	51,200	29	98.31(2)	82.60(7)
Cumulative-8 TTN	250,560	18	98.57(1)	83.37(6)
Inception CNN	1,196,530	23	99.27(3)	86.64(9)

Table 4.3: Table of parameter number, seconds of computation per epoch (with batch size of 64), and average classification accuracies on MNIST and Fashion MNIST for various regression models. The averages were computed across ten different initializations, with the standard error of the last digit given in parentheses.

Chapter 5

Rank Analysis of Tensor Network Models

This chapter is derived from unpublished work which focuses on the degree-limited multilinear models considered in [28]. We analyze how the rank of the tensor network ansatz is impacted by regressors with different interaction degrees, and the implications this has for tensor network regression models.

5.1 Introduction

One of the biggest open questions in tensor network machine learning is how to choose the best network architecture for a given task. As discussed in Sec. 3.1, architectures differ not only in their contraction scheme and parameter count, but also in the arrangement of the virtual indices which connect component tensors together. The degree of connectivity between portions of the network can be quantified using the multiplex rank, which was introduced in Sec. 2.2.4 as the rank of the matricization of a given tensor. In the context of regression, we are interested in the multiplex rank of the weight tensor for different bipartions of the data features, as this will determine the bond dimension that is needed across that partition for an exact representation of the model by a tensor network.

In this chapter, we derive upper bounds for the multiplex rank of the weight tensor as a function of its maximum interaction degree and the size of the feature partitions. These upper bounds precisely quantify the minimum virtual bond dimension necessary to represent an arbitrary multilinear regression model up to the specified degree, and reveal how this bond dimension changes at different locations in the network. We provide a general description of our approach in Sec. 5.2, which makes use of braket notation from quantum physics. In Sec. 5.3, we explicitly work through some simple examples, including the linear and bilinear models, before developing a recursive algortihm that can solve for the multiplex rank for any feature number, partition size, and interaction degree.

With this general-purpose algorithm in hand, we then consider specific applications and interpretations of it in the context of tensor network regression. In Sec. 5.4.1, we quantify how the bond dimension of an MPS model must vary with both the interaction degree and location of the virtual index. Interestingly, we find that the patterns of growth are not simple, and that the bond dimension can go actually shrink as the partition size grows. In Sec. 5.4.2, we show how the size of the bond dimension is driven almost entirely by feature products which span the bipartition, and in Sec. 5.4.3 we show that the distance between features in an interaction can significantly impact its effect on the bond dimension. Sec. 5.4.4 demonstrates that using the interaction decomposition from Chapter 4 allows for a modest reduction in the bond dimension needed to represent a low-degree tensor regression model. In Sec. 5.5 we discuss our results and offer potential directions for future work. An Appendix is also provided which contains the more technical aspects of the work.

5.2 General Approach

5.2.1 Braket notation and excitations

For convenience and clarity, we introduce a new notation borrowed from quantum mechanics called *braket* notation, with which we will use to describe tensors and tensor operations. In this notation, any element of a vector space (e.g. a tensor from an mth-order tensor space) can be represented using a *ket*, which looks like $|W\rangle$ for the weight tensor W. The inner product between the weight tensor and the data tensor $|X\rangle$ from Eq. (2.20) is written as

$$y = \langle W|X\rangle \,, \tag{5.1}$$

where the notation $\langle A|B\rangle$ denotes the inner-product between tensors $|A\rangle$ and $|B\rangle$, with $\langle A|$ representing the dual tensor or *bra* of $|A\rangle$. Note that, like tensor diagrams, braket notation conveys the contraction of tensors without explicit reference to any indices, although it is generally taken to only represent full contractions between two tensors of the same shape.

Using braket notation, we can expand the weight and data tensors on a shared standard basis as

$$|X\rangle = \sum_{i_1,i_2,\ldots,i_m=0}^{1} x_1^{i_1} x_2^{i_2} \cdots x_m^{i_m} \, |i_1\rangle \, |i_2\rangle \cdots |i_m\rangle \tag{5.2}$$

$$|W\rangle = \sum_{i_1,i_2,\ldots,i_m=0}^{1} W_{i_1 i_2 \ldots i_m} \, |i_1\rangle \, |i_2\rangle \cdots |i_m\rangle \,, \tag{5.3}$$

where the expansion coefficients in each expression are precisely those found in Eq. (2.23) which describe the form of the regression function. To simplify our notation, we have employed a common braket shorthand and written the tensor product $|a\rangle \otimes |b\rangle$ as $|a\rangle \, |b\rangle$. The composite basis vectors $|i_1\rangle \, |i_2\rangle \cdots |i_m\rangle$ are elements of the same tensor product space $\mathbb{H} = \bigotimes_{i=1}^{m} \mathbb{V}^{(i)}$ inhabited by W and X, with $|i_j\rangle$ being a standard basis vector for the component space $\mathbb{V}^{(j)}$.

As shown in Eq. (5.2), each standard basis vector can be uniquely associated with a different product of features, with the jth feature being present in the product if $|i_j\rangle = |1\rangle$. Taking inspiration once again from quantum physics, we refer to $|i_j\rangle = |1\rangle$ as an *excitation* in $\mathbb{V}^{(j)}$, with the number of $|1\rangle$ states in a composite basis vector referred to as its *excitation number*. Different interaction degrees correspond to different excitation numbers across the composite basis of $\mathbb{H}$, such that all linear terms are associated with basis vectors that have one excitation, all bilinear terms are associated with basis vectors that have two excitations, and so on.

To better emphasize the different interaction degrees which contribute to y, we can adopt an alternative notation for the basis vectors of $\mathbb{H}$ which includes only the positions of the excitations. Using this notation, the basis state $|i_1\rangle |i_2\rangle \cdots |i_m\rangle$ with k excitations in subspaces $\{\mathbb{V}^{(j_1)}, \mathbb{V}^{(j_2)}, ..., \mathbb{V}^{(j_k)}\}$ is written as $|j_1, j_2, ...j_k\rangle$, where $j_1 < j_2 < ... < j_k$. We denote the state $|0\rangle |0\rangle \cdots |0\rangle$, which lacks any excitations, as $|0\rangle$. With this labeling, the data and weight tensors can be written out as

$$|X\rangle = |0\rangle + \sum_{j_1=1}^{m} x_{j_1} |j_1\rangle + \sum_{j_1=1}^{m-1} \sum_{j_2=j_1+1}^{m} x_{j_1} x_{j_2} |j_1, j_2\rangle + ... + x_1 x_2 \cdots x_m |1, 2, ..., m\rangle \tag{5.4}$$

$$|W\rangle = W_0 |0\rangle + \sum_{j_1=1}^{m} W_{j_1} |j_1\rangle + \sum_{j_1=1}^{m-1} \sum_{j_2=j_1+1}^{m} W_{j_1 j_2} |j_1, j_2\rangle + ... + W_{1,2,...,m} |1, 2, ..., m\rangle , \tag{5.5}$$

where $W_{j_1 \cdots j_k} \equiv \langle j_1, ..., j_k | W\rangle$ gives the regression coefficient for the feature product $x_{j_1} \cdots x_{j_k}$, and W_0 is the bias term. Note that Eqs. (5.4, 5.5) are equivalent to Eqs. (5.2, 5.3), except that the summations have been seperated out and relabeled to emphasize the different interaction degrees. The inner product expression in Eq. (2.23) can be similarly written out as

$$y = W_0 + \sum_{j_1}^{m} W_{j_1} x_{j_1} + \sum_{j_1=1}^{m-1} \sum_{j_2=j_1+1}^{m} W_{j_1 j_2} x_{j_1} x_{j_2} + ... + W_{1,2,...,m} x_1 x_2 \cdots x_m, \tag{5.6}$$

where the contributions from the different interactions degrees are separated out into distinct summations.

5.2.2 Partitioning the weight tensor

Our analysis of the weight tensor focuses on its multiplex rank, as that is the property which most directly determines the bond dimension and connectivity needed for a tensor network representation (see Sec. 2.2.4). When computing a multiplex rank, we partition the components of the tensor product space $\mathbb{H}$ into two disjoint sets, forming new spaces $\mathbb{A}$ and $\mathbb{B}$ such that $\mathbb{H} = \mathbb{A} \otimes \mathbb{B}$. A tensor $|T\rangle$ in $\mathbb{H}$ can then be decomposed into the sum of tensor products between tensors in $\mathbb{A}$ and $\mathbb{B}$,

$$|T\rangle = \sum_{i=1}^{r} |A^{(i)}\rangle |B^{(i)}\rangle , \qquad |A^{(i)}\rangle \in \mathbb{A}, |B^{(i)}\rangle \in \mathbb{B}, \tag{5.7}$$

where the elements of $|A^{(i)}\rangle$ and $|B^{(i)}\rangle$ are entirely unconstrained. The minimum value of r needed to represent $|T\rangle$ (using an appropriately chosen $\{|A^{(i)}\rangle\}_{i=1}^{r}$ and $\{|B^{(i)}\rangle\}_{i=1}^{r}$) in this form is the multiplex rank of the tensor with respect to partitions $\mathbb{A}$ and $\mathbb{B}$. The maximum value of this rank is given by $\min(|\mathbb{A}|, |\mathbb{B}|)$, where $|\mathbb{A}|$ and $|\mathbb{B}|$ are the dimensions of $\mathbb{A}$ and $\mathbb{B}$ respectively, while the minimum value for a non-zero tensor is 1.

An element-wise interpretation of Eq. (5.7) can be easily obtained by borrowing techniques from matrix analysis. We consider an expansion of $|T\rangle$ on a standard basis of $\mathbb{H}$ constructed from the tensor product of bases in $\mathbb{A}$ and $\mathbb{B}$:

$$|T\rangle = \sum_{i_A=1}^{|\mathbb{A}|} \sum_{i_B=1}^{|\mathbb{B}|} \tilde{T}_{i_A i_B} |i_A\rangle |i_B\rangle \tag{5.8}$$

where $\{|i_A\rangle\}_{i_A=1}^{|A|}$ and $\{|i_B\rangle\}_{i_B=1}^{|B|}$ are orthogonal tensors (which need not be rank one) that span $\mathbb{A}$ and $\mathbb{B}$ respectively, and $\tilde{T}$ is the matrix of expansion coefficients. We can then write $|T\rangle$ in terms of its "rows" by summing over the column index i_B of $\tilde{T}$:

$$|T\rangle = \sum_{i_A=1}^{|\mathbb{A}|} |i_A\rangle |T_{i_A}\rangle , \qquad |T_{i_A}\rangle \equiv \sum_{i_B=1}^{|\mathbb{B}|} \tilde{T}_{i_A i_B} |i_B\rangle , \tag{5.9}$$

where $|T_{i_A}\rangle$ is the i_Ath "row" of $|T\rangle$ and $\mathbb{T}_{\mathbb{A}} \equiv \text{span}(\{|T_{i_A}\rangle\}_{i_A=1}^{|\mathbb{A}|})$ is its corresponding "row space". The multiplex rank of $|T\rangle$ for this partition is thus given by $|\mathbb{T}_{\mathbb{A}}|$, which is simply the number of linearly independent rows in the matrix $T_{i_A i_B}$. Note that since $\mathbb{A}$ and $\mathbb{B}$ will usually be tensor product spaces themselves, $T_{i_A i_B}$ represents a matriciziation of the tensor T, with i_A and i_B serving as compound indices that jointly address every combination of positions (in an arbitrary ordering) along the axes of the multidimensional tensor element array.

To analyze the multiplex rank of a given weight tensor $|W\rangle$, we will partition the m features of its implied dataset into two disjoint sets $\mathcal{A}$ and $\mathcal{B}$, such that the set of all features $\mathcal{X}$ is given by $\mathcal{A} \cup \mathcal{B}$. Without loss of generality, we will take $\mathcal{A}$ to be the smaller of the two sets, so that $|\mathcal{A}| \leq |\mathcal{B}|$ and $|\mathcal{B}| = m - |\mathcal{A}|$. In order to more easily refer to features in different partitions, we introduce vectors $\vec{a}$ and $\vec{b}$ to hold the elements of $\mathcal{A}$ and $\mathcal{B}$ respectively in an arbitrary order. This allows us to hereafter consider the ith feature in $\mathcal{A}$ to be a_i and the jth feature in $\mathcal{B}$ to be b_j.

The partitioning of the features into $\mathcal{A}$ and $\mathcal{B}$ leads directly to a partitioning of the tensor product space $\mathbb{H}$ into vector spaces $\mathbb{A}$ and $\mathbb{B}$

$$\mathbb{H} = \mathbb{A} \otimes \mathbb{B}, \qquad \mathbb{A} = \bigotimes_{i \in \mathcal{I}_{\mathcal{A}}} \mathbb{V}^{(i)}, \quad \mathbb{B} = \bigotimes_{i \in \mathcal{I}_{\mathcal{B}}} \mathbb{V}^{(i)}, \tag{5.10}$$

where $\mathcal{I}_{\mathcal{A}} = \{i \mid x_i \in \mathcal{A}\}$ and $\mathcal{I}_{\mathcal{B}} = \{i \mid x_i \in \mathcal{B}\}$ are sets of indices corresponding to the features in $\mathcal{A}$ and $\mathcal{B}$ respectively. In words, Eq. (5.10) states that $\mathbb{H}$ can be decomposed into the tensor product of two vector spaces which are themselves formed from the component spaces $\mathbb{V}^{(i)}$ associated with features in $\mathcal{A}$ and $\mathcal{B}$ respectively. Using the excitation-based notation introduced in Sec. 2.3, we can write basis vectors of $\mathbb{H}$ as the tensor product of basis vectors in $\mathbb{A}$ and basis vectors in $\mathbb{B}$. As an example, if $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$, and we partition these four features into $\mathcal{A}$ and $\mathcal{B}$ such that $\vec{a} = [x_1, x_2]$ and $\vec{b} = [x_3, x_4]$, then the basis vector $|1\rangle\,|1\rangle\,|0\rangle\,|1\rangle$ associated with product $x_1x_2x_4$ is written as $|1,2\rangle\,|2\rangle$, since $a_1 = x_1$, $a_2 = x_2$ and $b_2 = x_4$. When indexing elements of $|W\rangle$, which can in general be associated with features in both $\mathcal{A}$ and $\mathcal{B}$, we use subscripts to denote indexing with respect to $\vec{a}$ and superscripts to denote indexing with respect to $\vec{b}$. Using this convention, the regression coefficient for $x_1x_2x_4$ in our example would be written as $W^2_{1,2}$.

Once the features have been partitioned into $\mathcal{A}$ and $\mathcal{B}$, we can expand $|W\rangle$ on the excitation bases of $\mathbb{A}$ and $\mathbb{B}$ and then sum together the components of $\mathbb{B}$, as was done in Eq. (5.9). This gives

$$|W\rangle = |0\rangle\,|W_0\rangle + \sum_{i_1=1}^{|A|} |i_1\rangle\,|W_{i_1}\rangle + \sum_{i_1=1}^{|A|-1} \sum_{i_2=i_1+1}^{|A|} |i_1, i_2\rangle\,|W_{i_1 i_2}\rangle + ... + |1, 2, ..., |A|\rangle\,|W_{1,2,...,|A|}\rangle\,, \tag{5.11}$$

where the vector $|W_{i_1...i_k}\rangle$ represents the vector in $\mathbb{B}$ associated with basis vector $|i_1, ..., i_k\rangle$ in $\mathbb{A}$. We refer to these as the "row vectors" of $|W\rangle$ with respect to the given partitioning, and they have the form

$$|W_{i_1...i_k}\rangle = W_{i_1...i_k}\,|0\rangle + \sum_{j_1=1}^{|B|} W^{j_1}_{i_1...i_k}\,|j_1\rangle + \sum_{j_1=1}^{|B|-1} \sum_{j_2=j_1+1}^{|B|} W^{j_1 j_2}_{i_1...i_k}\,|j_1, j_2\rangle + ... + W^{1,...,|\mathcal{B}|}_{1,...,|\mathcal{A}|}\,|1, ..., |\mathcal{B}|\rangle\,. \tag{5.12}$$

Focusing on the expansion terms in Eq. (5.12), we can see that basis vector $|j_1, ..., j_\ell\rangle$ in $\mathbb{B}$ is weighted by the regression coefficient corresponding to the feature product $a_{i_1}a_{i_2}\cdots a_{i_k}b_{j_1}b_{j_2}\cdots b_{j_\ell}$. Note that the features from $\mathcal{A}$ in the product are determined by the basis vector $|i_1, ..., i_k\rangle$ from $\mathbb{A}$ that $|W_{i_1...i_k}\rangle$ is associated with in Eq. (5.11), while the features from $\mathcal{B}$ are determined by $|j_1, ..., j_\ell\rangle$ from $\mathbb{B}$. As discussed earlier, the multiplex rank of the weight tensor across the partition is equal to the number of linearly independent row vectors, which is ultimately dictated by the values of the regression coefficients. Our analysis will therefore focus on how different forms of the regression function impact the rank structure of $|W\rangle$.

5.3 Rank Upper Bounds

5.3.1 Single-Feature Partition

To begin our analysis, we consider the simple case where a set of m features have been partitioned such that $\mathcal{A}$ contains only a single feature, which we chosen arbitrarily to be x_1. Therefore, we have $\vec{a} = [x_1]$ and $\vec{b} = [x_2, ..., x_m]$. Since $\mathbb{A} = \mathbb{V}^{(1)}$, the basis vectors of $\mathbb{A}$ are simply $\{|0\rangle\,, |1\rangle\}$, which denotes either the presence of absence of x_1 in the feature product. Expanding the weight tensor $|W\rangle$ as in Eq. (5.11) gives

$$|W\rangle = |0\rangle\,|W_0\rangle + |1\rangle\,|W_1\rangle\,, \tag{5.13}$$

where $|W_0\rangle$ contains the regression coefficients for products which do not include x_1 and $|W_1\rangle$ contains the coefficients for products which do include x_1. It is readily apparent that the rank of $|W\rangle$ across this single-feature partition is at most two, which is equal to the $\min(|\mathbb{A}|, |\mathbb{B}|)$ upper-bound mentioned in Sec. 5.2.2.

Let us assume that we are performing regression on all possible feature products. If the coefficients of the weight tensor are chosen at random, then its rank will be two with probability one. That said, it is worthwhile to consider the form of regression that would be necessary for $|W\rangle$ to have a rank of one across the partition, which is equivalent to having $|W_0\rangle \propto |W_1\rangle$. Full expressions for the two column vectors are given by

$$|W_0\rangle = W_0 |0\rangle + \sum_{i_1=1}^{|\mathcal{B}|} W^{i_1} |i_1\rangle + \sum_{i_1=1}^{|\mathcal{B}|-1} \sum_{i_2=i_1+1}^{|\mathcal{B}|} W^{i_1 i_2} |i_1, i_2\rangle + ... + W^{1,...,|\mathcal{B}|} |1, ..., |\mathcal{B}|\rangle \tag{5.14}$$

$$|W_1\rangle = W_1 |0\rangle + \sum_{i_1=1}^{|\mathcal{B}|} W_1^{i_1} |i_1\rangle + \sum_{i_1=1}^{|\mathcal{B}|-1} \sum_{i_2=i_1+1}^{|\mathcal{B}|} W_1^{i_1 i_2} |i_1, i_2\rangle + ... + W_1^{1,...,|\mathcal{B}|} |1, ..., |\mathcal{B}|\rangle , \tag{5.15}$$

from which it is clear that $|W_0\rangle \propto |W_1\rangle$ holds if and only if $W^{i_1...i_k} = cW_1^{i_1...i_k}$ for all excitation numbers k and indices $\{i_1, ..., i_k\}$. Note that the proportionality constant c must be the same for every coefficient.

5.3.2 Linear Regression

In linear regression, we consider a model of the form $\vec{f}(\vec{x}) = \vec{w}^{\ (i)} + \vec{w}^{\ (1)} x_1 + ... + \vec{w}^{\ (m)} x_m$ for some set of coefficient vectors $\{\vec{w}_i\}_{i=0}^m$. This means that the only non-zero elements of the weight tensor in a linear model are those associated with one or zero excitations. These elements can be grouped into those where the excitation is in $\mathbb{A}$ and those where it is in $\mathbb{B}$ (i.e. corresponding to a feature in $\mathcal{A}$ or a feature in $\mathcal{B}$). Under these conditions, Eqs. (5.11, 5.12) become

$$|W\rangle = |0\rangle |W_0\rangle + \sum_{i=1}^{|\mathcal{A}|} |i\rangle |W_i\rangle \tag{5.16}$$

$$|W_0\rangle = W_0 |0\rangle + \sum_{i=1}^{|\mathcal{B}|} W^i |i\rangle , \quad |W_i\rangle = W_i |0\rangle , \tag{5.17}$$

where the forms of the row vectors $|W_i\rangle$ are constrained by the fact that $\mathbb{B}$ cannot have any excitations if there is already an excitation in $\mathbb{A}$. Eq. (5.17) shows that every $|W_i\rangle$ is directly proportional to $|0\rangle$, so the second term in Eq. (5.16) becomes $(\sum_{i=1}^{|\mathcal{A}|} W_i |i\rangle) |0\rangle$ and thus the row space of $|W\rangle$ is spanned by $\{|W_0\rangle , |0\rangle\}$. This immediately implies that the multiplex rank of the linear regression weight tensor can be at most two for any partitions $\mathcal{A}$ and $\mathcal{B}$.

Since the row space is only two-dimensional in this simple case, we can compute the singular values of the matricized weight tensor W analytically as a function of the regression coefficients. This gives

$$\sigma_1^2 = \frac{||W||^2 - \sqrt{(||W||^2)^2 - 4\sum_{i=1}^{|\mathcal{A}|}(W_i)^2 \cdot \sum_{i=1}^{|\mathcal{B}|}(W^i)^2}}{2} \tag{5.18}$$

$$\sigma_2^2 = \frac{||W||^2 + \sqrt{(||W||^2)^2 - 4\sum_{i=1}^{|\mathcal{A}|}(W_i)^2 \cdot \sum_{i=1}^{|\mathcal{B}|}(W^i)^2}}{2}, \tag{5.19}$$

where σ_1 and σ_2 are the singular values of W with respect to an arbitrary bipartitioning. Note that σ_2 will always be non-zero so long as $||W||^2 > 0$ (i.e. at least one coefficient is non-zero), but σ_1 will vanish if the regression coefficients are all zero in just one of the partitions. The complexity of W is maximized, in a sense, when $\sigma_1 = \sigma_2$, which occurs when the bias b is zero and when the squared magnitudes of regression coefficients for features in $\mathcal{A}$ and $\mathcal{B}$ are equal.

5.3.3 Bilinear Regression

When we include bilinear terms in our regression model, the maximum number of excitations increases from one to two. As before, the non-zero elements of W can be grouped based on the number of excitations in $\mathbb{A}$ and $\mathbb{B}$, with a total of three different groups based on whether there is zero, one, or two excitations in $\mathbb{A}$. Using these groupings, $|W\rangle$ is given by

$$|W\rangle = |0\rangle |W_0\rangle + \sum_{i=1}^{|\mathcal{A}|} |i\rangle |W_i\rangle + \sum_{i=1}^{|\mathcal{A}|-1} \sum_{j=i+1}^{|\mathcal{A}|} |i,j\rangle |W_{ij}\rangle \,, \tag{5.20}$$

where terms of the form $|i,j\rangle |W_{ij}\rangle$ have two excitations in $\mathbb{A}$, terms of the form $|i\rangle |W_i\rangle$ have one excitation in $\mathbb{A}$, and the term $|0\rangle |W_0\rangle$ has no excitations in $\mathbb{A}$. The row vectors in these different groups are given by the following truncations of Eq. (5.12):

$$|W_0\rangle = W_0 |0\rangle + \sum_{i=1}^{|\mathcal{B}|} W^i |i\rangle + \sum_{i=1}^{|\mathcal{B}-1|} \sum_{j=i+1}^{|\mathcal{B}|} W^{ij} |i,j\rangle \tag{5.21}$$

$$|W_i\rangle = W_i |0\rangle + \sum_{i=1}^{|\mathcal{B}|} W_i^j |j\rangle \tag{5.22}$$

$$|W_{ij}\rangle = W_{ij} |0\rangle \,. \tag{5.23}$$

While Eqs. (5.21 - 5.23) can potentially describe $\frac{|\mathcal{A}|^2+|\mathcal{A}|+2}{2}$ unique vectors, it is important to note that this is not a tight upper bound on the dimension of the row space, since the $|W_{ij}\rangle$ are all proportional to one another regardless of the second-order coefficient values W_{ij}. Indeed, it is apparent by inspection that at most $|\mathcal{A}| + 2$ of the row vectors can be linearly independent, so we have that the multiplex rank is less than or equal to $|\mathcal{A}| + 2$. This bound is tight for all $|\mathcal{A}| > 1$, since with $|\mathcal{A}| = 1$ there cannot be two excitations in $\mathbb{A}$ and therefore the $|W_{ij}\rangle$ must all vanish. In contrast with the $r \leq 2$ bound found for linear regression in Sec. 5.3.2, the bound we have derived here for bilinear regression is not the same for all partitions, but rather scales linearly with the size of the smaller partition.

5.3.4 Multilinear Regression

Having worked through some simpler examples, we will now consider a generic upper bound on the multiplex rank of a weight tensor when used for multilinear regression of degree d. First, we note that the row vectors described in Eq. (5.12) inhabit subspaces which are the direct sum of *excitation subspaces* $\{\mathbb{E}^{(d_{\mathbb{B}})}\}_{d_{\mathbb{B}}=1}^{|\mathcal{B}|}$ of $\mathbb{B}$:

$$|W_{i_1 \dots i_{d_{\mathbb{A}}}}\rangle \in \bigoplus_{d_{\mathbb{B}}=0}^{\bar{d}_{\mathbb{B}}} \mathbb{E}^{(d_{\mathbb{B}})}, \quad \mathbb{E}^{(d_{\mathbb{B}})} = \text{span}(\{|i_1, ..., i_{d_{\mathbb{B}}}\rangle\}), \quad \bar{d}_{\mathbb{B}} = \min(d - d_{\mathbb{A}}, |\mathcal{B}|), \tag{5.24}$$

where $d_{\mathbb{A}}$ and $d_{\mathbb{B}}$ are the number of excitations in $\mathbb{A}$ and $\mathbb{B}$ respectively. In words, Eq. (5.24) states that the row vector $|W_{i_1 \dots i_{d_{\mathbb{A}}}}\rangle$ has support on basis vectors with up to $\bar{d}_{\mathbb{B}}$ excitations, where $\bar{d}_{\mathbb{B}}$ is the maximum possible number of excitations in $\mathbb{B}$ given that there are $d_{\mathbb{A}}$ excitations in $\mathbb{A}$. The dimension of $\mathbb{E}^{(d_{\mathbb{B}})}$ has the simple combinatorial form

$$\dim(\mathbb{E}^{(d_{\mathbb{B}})}) = \binom{|\mathcal{B}|}{d_{\mathbb{B}}}, \tag{5.25}$$

which is the number of ways to distribute $d_{\mathbb{B}}$ excitations among $|\mathcal{B}|$ features. Note that the excitation subspaces are very similar to the degree-j subspaces $\mathbb{D}^{(j)}$ from Sec. 4.2.1, except that those subspaces were spanned by basis vectors of the full tensor space $\mathbb{H}$ rather than the partitioned space $\mathbb{B}$.

The subspace decomposition given in Eq. (5.24) can be combined with a grid-based visualization of the row vectors from Eq. (5.12) to help guide our analysis. This grid has a height of $h = 1 + \min(d, |\mathcal{A}|)$ and a length of $\ell = 1 + \min(d, |\mathcal{B}|)$, where $\min(d, |\mathcal{A}|)$ and $\min(d, |\mathcal{B}|)$ are the maximum number of excitations that can be placed in $\mathbb{A}$ and $\mathbb{B}$ respectively. The position $(d_{\mathbb{A}}, d_{\mathbb{B}})$ on the grid, indexed from zero, is mapped to the generic expression given by Eq. (5.12) for the projection P of $|W_{i_1 \dots i_{d_{\mathbb{A}}}}\rangle$ onto $\mathbb{E}^{(d_{\mathbb{B}})}$:

$$P_{\mathbb{E}^{(d_{\mathbb{B}})}} |W_{i_1 \dots i_{d_{\mathbb{A}}}}\rangle = \sum_{j_1=1}^{|B|-d_{\mathbb{B}}+1} \sum_{j_2=j_1+1}^{|B|-d_{\mathbb{B}}+2} \cdots \sum_{j_{d_{\mathbb{B}}}=j_{d_{\mathbb{B}}-1}+1}^{|B|} W_{i_1 \dots i_{d_{\mathbb{A}}}}^{j_1 \dots j_{\mathbb{B}}} |j_1, ..., j_{d_{\mathbb{B}}}\rangle, \tag{5.26}$$

such that each row is uniquely identified by the number of excitations in $\mathbb{A}$ and each column is uniquely identified by the number of excitations in $\mathcal{B}$.

To give an example of how these grids are constructed, we can revisit bilinear regression from Sec. 5.3.3 and grid the row vectors in Eqs. (5.21 - 5.23) as

$$W_0 |0\rangle + \sum_{j_1} W^{j_1} |j_1\rangle + \sum_{j_1, j_2} W^{j_1 j_2} |j_1, j_2\rangle \tag{5.27}$$

$$W_{i_1} |0\rangle + \sum_{j_1} W_{i_1}^{j_1} |j_1\rangle + \quad 0 \tag{5.28}$$

$$W_{i_1 i_2} |0\rangle + \quad 0 \qquad\qquad + \quad 0, \tag{5.29}$$

where each row and column has a consistent number of excitation in $\mathbb{A}$ and $\mathbb{B}$ respectively. We have assumed here that $|\mathcal{A}| \geq 2$, so the dimensions $h = \ell = 3$ of the grid are determined by the maximum degree d (i.e. the maximum allowed number of excitations), which is two. The utility of this grid comes from its clean separation of the different excitation subspaces, with the ith column containing all projections into $\mathbb{E}^{(i)}$. This makes it easier to determine any linear dependence relationships that might exist between the row vectors, since linear dependence must hold within each subspace. Note that to reduce clutter we will generally omit the summation limits when writing out grid expressions.

It is worth considering how the grid pattern changes for different values of $|\mathcal{A}|$, $|\mathcal{B}|$, and d, since this will make it easier to understand the corresponding changes in the multiplex rank bound. When $d \leq |\mathcal{A}|$, the grid will be square and have a triangular structure like that seen in Eqs. (5.27 - 5.29), where all grid cells with $d_{\mathbb{B}} + d_{\mathbb{A}} > d$ (the sum of excitations in each partition greater than the total number of excitations) are zero. When $|\mathcal{A}| < d \leq |\mathcal{B}|$, we have that $h = |\mathcal{A}|$ and $\ell = d$, so the height of the grid will be smaller than its length. The grid will still have a triangle structure, with $d_{\mathbb{B}} + d_{\mathbb{A}} > d$ cells being zero, but the row vectors $|W_{i_1 \dots i_{\mathcal{A}}}\rangle$ at the bottom of the grid will have non-zero projections beyond the $\mathbb{E}^{(0)} = \text{span}(\{|0\rangle\})$ subspace. As an example, if we consider trilinear regression (i.e. $d = 3$) with $|\mathcal{A}| = 2$ and $|\mathcal{B}| \geq d$, then the grid of row vectors is

$$W_0 \,|0\rangle + \sum_{j_1} W^{j_1} \,|j_1\rangle + \sum_{j_1,j_2} W^{j_1 j_2} \,|j_1, j_2\rangle + \sum_{j_1,j_2,j_3} W^{j_1 j_2 j_3} \,|j_1, j_2, j_3\rangle \tag{5.30}$$

$$W_{i_1} \,|0\rangle + \sum_{j_1} W^{j_1}_{i_1} \,|j_1\rangle + \sum_{j_1,j_2} W^{j_1 j_2}_{i_1} \,|j_1, j_2\rangle + 0 \tag{5.31}$$

$$W_{i_1 i_2} \,|0\rangle + \sum_{j_1} W^{j_1}_{i_1 i_2} \,|j_1\rangle + 0 + 0, \tag{5.32}$$

where it is apparent from Eq. 5.32 that the row vectors with $d_{\mathbb{A}} = 2$ can have non-zero projections in $\mathbb{E}^{(1)}$. Similarly, if $d > |\mathcal{B}|$ then the grid dimensions will be $|\mathcal{A}| \times |\mathcal{B}|$, with the value of d determining how large the triangle of zeros will be in the bottom-right corner. For the most extreme case of $d = |\mathcal{A}| + |\mathcal{B}| = m$, there are no zeros in the grid and thus every row vector can have support in each excitation subspace.

With this groundwork laid, we now construct an algorithm which gives a tight upper bound for the multiplex rank of a given weight tensor. This bound will necessarily be a function of the partition sizes $|\mathcal{A}|$, $|\mathcal{B}|$ and the maximum interaction degree d. Our approach will be to sequentially populate the excitation subspaces, starting from the highest excitation, with linearly independent vectors drawn from the set of row vectors. There are two important quantities for us to consider here: the number $\alpha_{d_{\mathbb{B}}}$ of row vectors with support on the $d_{\mathbb{B}}$th excitation space, and the size $\beta_{d_{\mathbb{B}}} \equiv \dim(\mathbb{E}^{(d_{\mathbb{B}})})$ of the $d_{\mathbb{B}}$th excitation subspace. While $\beta_{d_{\mathbb{B}}}$ is just given by Eq. (5.25), the expression for $\alpha_{d_{\mathbb{B}}}$ is

$$\alpha_{d_{\mathbb{B}}} = \sum_{d_{\mathbb{A}}=0}^{\min(|\mathcal{A}|, d-d_{\mathbb{B}})} \binom{|\mathcal{A}|}{d_{\mathbb{A}}}. \tag{5.33}$$

The terms in the sum are the number of row vectors associated with $d_{\mathbb{A}}$ excitations in $\mathbb{A}$ (i.e. $|\{|W_{i_1 \dots i_{d_{\mathbb{A}}}}\rangle\}|$), which is simply the number of ways to distribute $d_{\mathbb{A}}$ excitations among the features in $\mathcal{A}$. The upper summation limit is the maximum number of excitations that can be placed in $\mathbb{A}$, given that $d_\mathbb{B}$ of the d total excitations have already been placed in $\mathbb{B}$. In the context of the grid visualization, this limit plus one indicates which row the triangle of zeros pattern starts for the $d_\mathbb{B}$th column.

With definitions for $\alpha_{d_\mathbb{A}}$ and $\beta_{d_\mathbb{A}}$ in hand, our algorithm for determining the multiplex rank r for the weight tensor is given by

$$r = \sum_{d_\mathbb{B}=0}^{\min(d,|\mathcal{B}|)} v_{d_\mathbb{B}}, \quad v_{d_\mathbb{B}} = \min\left[\alpha_{d_\mathbb{B}} - \sum_{d'_\mathbb{B}=d_\mathbb{B}+1}^{\min(d,|\mathcal{B}|)} v_{d'_\mathbb{B}}, \beta_{d_\mathbb{B}}\right], \tag{5.34}$$

where $v_{d_\mathbb{B}}$ is the number of linearly independent vectors placed in the $d_\mathbb{B}$th excitation subspace $\mathbb{E}^{(d_\mathbb{B})}$. The motivation for this recursive series is that we start by placing as many vectors as possible into the highest excitation subspace $\mathbb{E}^{(\bar{d}_\mathbb{B})}$, with the number of such vectors equal to either the size $\beta_{\bar{d}_\mathbb{B}}$ of the subspace or the number $a_{\bar{d}_\mathbb{B}}$ of row vectors that have support on that subspace (whichever is smallest). We then repeat this same process for the second-highest excitation subspace, except that the number of available row vectors is equal to $\alpha_{\bar{d}_\mathbb{B}-1}$ *minus* the number of row vectors that we already used to fill the highest excitation subspace. This subtraction is critical, because every vector with support in $\mathbb{E}^{(i)}$ also has support in $\mathbb{E}^{(j)}$ for $j < i$ (which is why the pattern of zeros in the grid is lower triangular), therefore $\alpha_{\bar{d}_\mathbb{B}-1}$ on its own counts vectors that we already used to span $\mathbb{E}^{(\bar{d}_\mathbb{B})}$. Repeating this analysis across the entire sequence of excitation subspaces yields the recursion relation in Eq. (5.34).

As a sanity check, we can confirm that Eq. (5.34) yields the ranks we found for the simpler cases considered in Secs. 5.3.1 - 5.3.3. For a single-feature partition where $|\mathcal{A}| = 1$, $|\mathcal{B}| > 1$, and $d = m$, Eq. (5.33) gives $\alpha_{d_\mathbb{B}} = 2$ for all $d_\mathbb{B}$. Then we have $\bar{d}_\mathbb{B} = |\mathcal{B}|$ and therefore $v_{|\mathcal{B}|} = \min(\alpha_{|\mathcal{B}|}, \binom{|\mathcal{B}|}{|\mathcal{B}|}) = 1$. By the recursion relation, $v_{|\mathcal{B}|-1} = \min(\alpha_{|\mathcal{B}|-1} - v_{|\mathcal{B}|}, \binom{|\mathcal{B}|}{|\mathcal{B}|-1}) = 1$, since $\alpha_{|\mathcal{B}|-1} = 2$. For $\mathbb{E}^{(|\mathcal{B}|-1)}$, we have that $\alpha_{|\mathcal{B}|-1} = 2$ and $v_{|\mathcal{B}|} + v_{|\mathcal{B}|-1} = 2$, so therefore $v_{|\mathcal{B}|-2} = \min(0, \binom{|\mathcal{B}|}{|\mathcal{B}|-1}) = 0$. Note that once the number of available vectors for a given excitation subspace is zero the recursion can be halted, since this implies that all row vectors have already been utilized. Putting everything together, the multiplex rank is given by Eq. (5.34) as $r = v_{|\mathcal{B}|} + v_{|\mathcal{B}|-1} = 2$, matching what we found in Sec. 5.3.1.

For linear regression, $d = 1$ and thus we need only consider v_0 and v_1 corresponding to $\mathbb{E}^{(0)}$ and $\mathbb{E}^{(1)}$ respectively. By Eq. (5.33) we have the $\alpha_1 = 1$ and $\alpha_0 = |\mathcal{A}| + 1$, while by Eq. (5.25) we have that $\beta_1 = |\mathcal{B}|$ and $\beta_0 = 1$. This means that $v_1 = \min(1, |\mathcal{B}|) = 1$ and $v_0 = \min(|\mathcal{A}| + 1 - 1, 1) = 1$, so $r = 2$ as found in Sec. 5.3.2. For bilinear regression, $\beta_2 = \frac{1}{2}|\mathcal{B}|(|\mathcal{B}|-1)$, $\alpha_2 = 1$, $\alpha_1 = |\mathcal{A}|+1$, and $\alpha_0 = \frac{1}{2}(|\mathcal{A}|^2+|\mathcal{A}|+2)$, so $v_2 = \min(1, \frac{1}{2}|\mathcal{B}|(|\mathcal{B}|-1)) = 1$, $v_1 = \min(|\mathcal{A}| + 1 - 1, |\mathcal{B}|) = |\mathcal{A}|$, and $v_0 = \min(\frac{1}{2}(|\mathcal{A}|^2 + |\mathcal{A}| + 2) - |\mathcal{A}| - 1, 1) = \min(\frac{1}{2}(|\mathcal{A}|^2 - |\mathcal{A}|), 1)$. The value of v_0 is 0 if $|\mathcal{A}| = 1$ and 1 otherwise, so $r = |\mathcal{A}| + 2$ if $|\mathcal{A}| > 1$ and $r = 2$ if $\mathcal{A} = 1$, precisely as we found in Sec. 5.3.3.

5.4 Applications to Tensor Network Regression

5.4.1 Bond dimension of low-degree regression models

As discussed in Sec. 2.2.4, the multiple rank of a tensor sets a lower bound on the bond dimension needed to exactly represent it using a tensor network. When considering the weight tensor represented in a tensor network regression model, the rank upper bounds derived in Sec. 5.3 translate into tight lower bounds on the bond dimension needed to carry out unrestricted multilinear regression up to degree d. By analyzing the multiplex rank bounds for each degree as a function of partition size $|\mathcal{A}|$ and feature number m, we can understand how the bond dimension grows and shrinks for different virtual bonds in the network.

As a specific example, we can consider the MPS regression model from Sec. 3.1.2, in which the featurization vectors are arranged in a line and joined together by a chain of third-order component tensors. The virtual bond between components $A^{(i)}$ and $A^{(i+1)}$ is constrained by the multiplex rank across partitions $\{x_1, .., x_i\}$ and $\{x_{i+1}, ..., x_m\}$, which can be computed using the recursive algorithm from Eq. (5.34) with $\mathcal{A}$ defined to be the smaller of the two partitions. In Figure 5.1, we show plots of the bond dimension needed for unconstrained multilinear regression using an MPS model as a function of virtual index position, with the ith position corresponding to the index between $A^{(i)}$ and $A^{(i+1)}$. We have taken $m = 784$, which is the number of pixels in the MNIST and Fashion MNIST images.

The plots in Figure 5.1 reveal some interesting behavior. The first row of plots, corresponding to linear and bilinear regression, conforms to the constant and linear scaling expressions from Secs. 5.3.2 and 5.3.3 respectively. In the bilinear curve we can see mirror symmetry at index position $\frac{m}{2} = 392$ which is shared across all plots, since this marks the point where an increase in the index position corresponds to a reduction in the size of $\mathcal{A}$. More interesting is the behavior observed for trilinear (degree 3) regression, in which the bond dimension increases rapidly at first but then suddenly plateaus at index position 40. The degree 5 plot shows an even more extreme discontinuity, as its rank actually starts to *decrease* at position 112, despite the fact that $|\mathcal{A}|$ continues to grow. In the bottom row of plots, corresponding to very high interaction degrees, we see complicated spiked curves with multiple minima and maxima, although the bond dimension still grows on average with the index position until the halfway point.

The precise relationship between multiplex rank, interaction degree, and partition size is complicated (see Appendix 5.6.1), but we can motivate some of the broader trends observed in Figure 5.1 by considering the impact of increasing $|\mathcal{A}|$. Since there are only a finite number of features, the growth of $|\mathcal{A}|$ must be offset by the shrinking of $|\mathcal{B}|$, and these two changes have opposing effects on the multiplex rank. An increase in $|\mathcal{A}|$ leads to a larger value of α_i in Eq. (5.33), which means that there are more row vectors available to populate the excitation subspaces and thereby increase the multiplex rank. On the other hand, a decrease in $|\mathcal{B}|$ leads to a decrease in the size of all excitation subspaces $\mathbb{E}^{(i)}$ (except $\mathbb{E}^{(0)}$), which will tend to reduce the multiplex rank. For any given increment $|\mathcal{A}| \to |\mathcal{A}| + 1$ the sizes of these two competing effects will vary, and therefore the multiplex rank may grow, shrink, or remain the same.

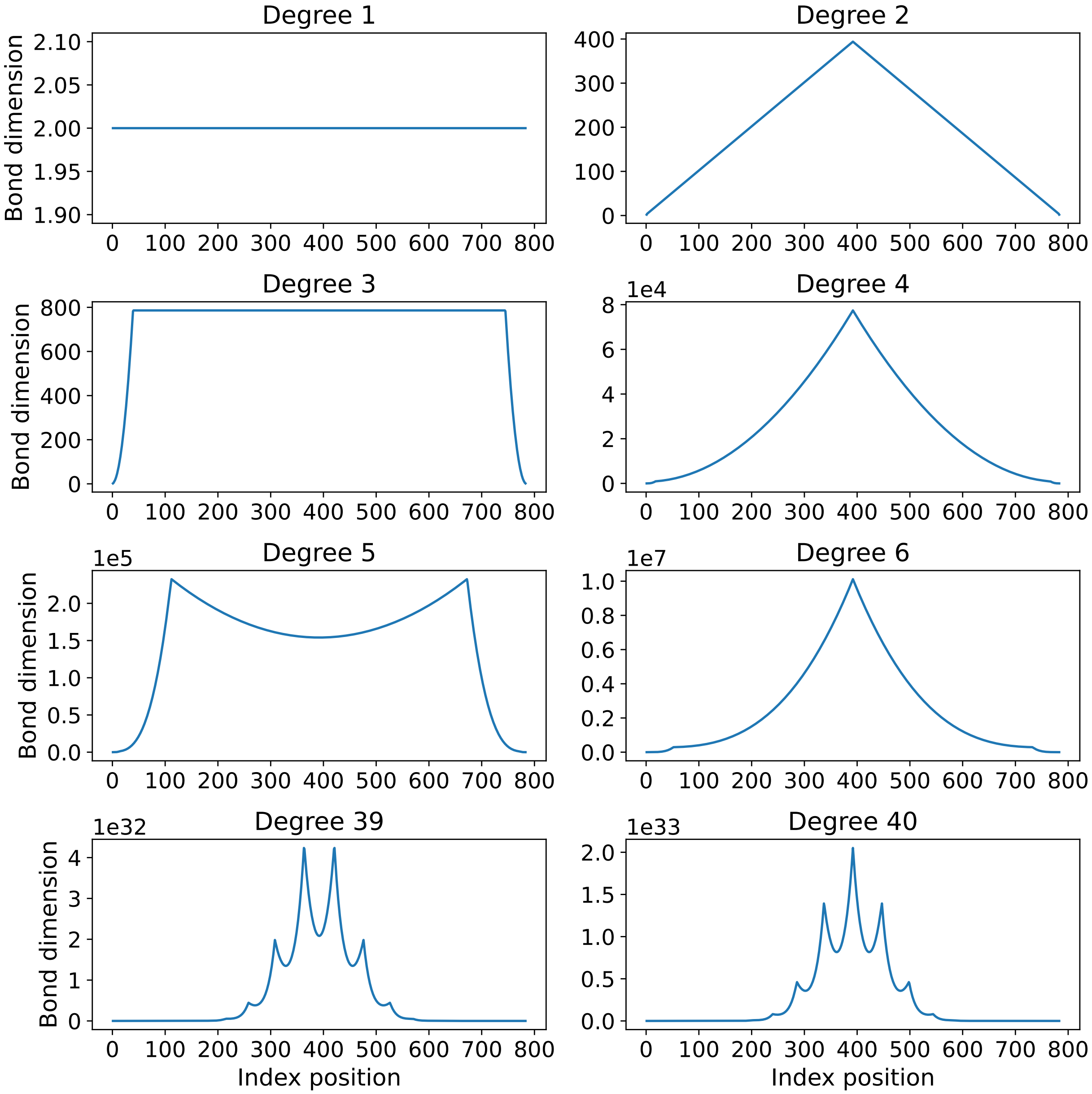


Figure 5.1: Plots of bond dimension versus virtual index position for an MPS model representing unconstrained multilinear regression up to the given degree. The bond dimension corresponds to the multiplex rank of the weight tensor with respect to partitions formed to the left and right of the virtual index. Since the multiplex rank is a function only of the two partition sizes rather than their specific contents, the plots are symmetric around the index position corresponding to $\frac{m}{2} = 392$, where $|\mathcal{A}| = |\mathcal{B}|$. At larger degrees, the bond dimension curves are complicated functions of $|\mathcal{A}|$, m, d, which we discuss in Appendix 5.6.1.

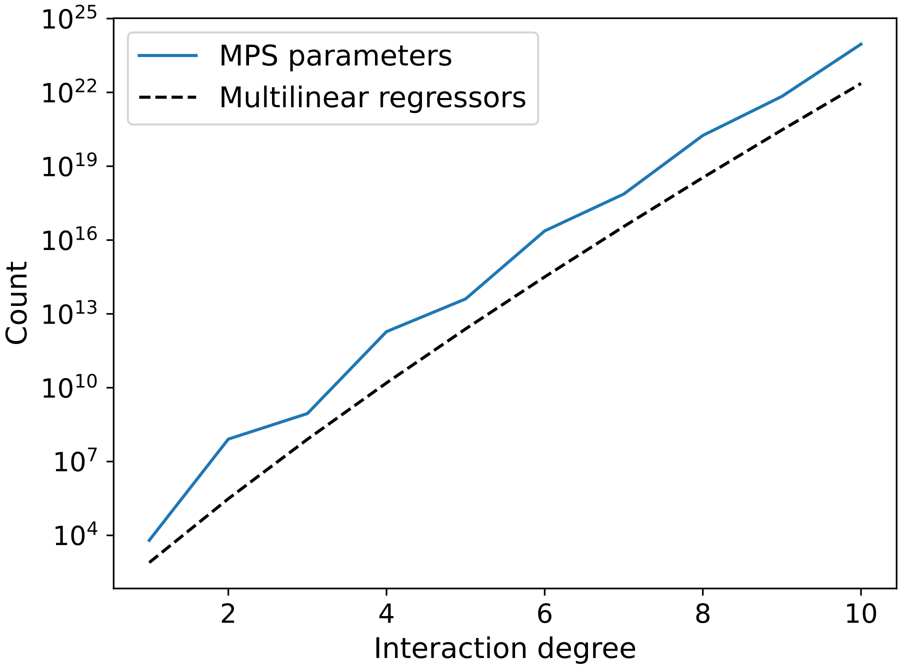


Figure 5.2: Plot of MPS parameters (blue) and regressor number (black) versus interaction degree for $m = 784$ features. The parameter curve gives the number of MPS component tensor elements needed to perform arbitrary multilinear regression up to the specified degree, while the regressor curve gives the number of non-zero elements in the weight tensor W.

Aside from the shapes of the curves in Figure 5.1, it is also worth considering the actual magnitudes of the bond dimension across different index positions and interaction degrees. These magnitudes dictate the number of parameters needed for an MPS model to perform unconstrained regression up to the specified degree, which can allow us to assess the efficiency of the tensor network representation as well as its limitations. Given that the multiplex rank provides a tight lower bound on the MPS bond dimension [49], the minimum number of model parameters $|\theta|$ is

$$|\theta| = 8 + \prod_{i=1}^{m-2} 2r_i r_{i+1}, \tag{5.35}$$

where r_i is the multiplex rank of W for partition $\mathcal{A} = \{x_1, ..., x_i\}$, and the eight additional parameters come from the 2×2 matrices at each end of the MPS[1]. In Figure 5.2, we give a plot of these parameter numbers as a function of maximum interaction degree (i.e. regression on all feature products less than or equal to the specified degree), alongside a curve showing the actual number of feature-product regressors. The most obvious thing to note is that the number of parameters needed for arbitrary multilinear regression is very large, with even bilinear regression already requiring over 10^7 parameters when $m = 784$ features. This is far in excess of the actual number of bilinear regressors, which is only about 300,000.

A similar inflation of parameter number is seen across all interaction degrees, and demonstrates that their is an inherent inefficiency in using a tensor network to exactly represent the weight tensor. This is unsurprising, however, as the purpose of networks such as MPS is to create low-rank *approximations* rather than identical recreations. Nonetheless, the bond dimension curves from Figure 5.1 paint a stark picture of just how significant the low-rank restriction is. Assuming that we set the bond dimension of our MPS regression model to 20, which is a reasonable value, the model is already placing massive constraints on the bilinear coefficients, let alone those of the higher-degree regressors. Indeed, a bond dimension of 20 is sufficient for unconstrained bilinear regression only when $m \leq 39$, at which point the curse of dimmensionality is minimal anyway for small interaction degrees. That being said, the utility of a tensor network regression model comes not from its ability to perform unconstrained regression at a low interaction degree, but rather in doing constrained regression across a wide range of degrees. From this perspective, the large magnitudes found in Figures 5.1 and 5.2 are neither surprising nor concerning, as we always intend to place severe restrictions on the multiplex rank of the weight tensor.

5.4.2 Significance of inter-partition regressors

In the previous subsection, we found that the multiplex rank of a high-degree weight tensor grows rapidly with the number of features in the smaller partition $\mathcal{A}$, but it turns out that not all of the regression coefficients contribute equally to this growth. Given the feature partitions $\mathcal{A} = \{x_1, ..., x_{|\mathcal{A}|}\}$ and $\mathcal{B} = \{x_{|\mathcal{A}+1|}, ..., x_m\}$, we can classify each feature-product regressor $x_{i_1} x_{i_2} \cdots x_{i_d}$ based which partitions the individual features are drawn from. We refer to a regressor as *intra-partition* if all of the features in the feature product are drawn from

[1] For simplicity we are neglecting the parameters in the output component, which is equivalent to assuming that the prediction vector $\vec{y}$ contains only a single element.

a single partition, while a regressor with at least one feature from each partition is classified as *inter-partition*. Using the notation introduced in Sec. 5.2.1, the regression coefficients corresponding to intra-partition regressors are written as either $W_{i_1 \ldots i_d}$ or $W^{j_1 \cdots j_d}$ for d-degree feature products, depending on whether the features all come from $\mathcal{A}$ or $\mathcal{B}$ respectively. On the other hand, coefficients for inter-partition regressors always have the form $W^{j_1 \cdots j_{d_\mathbb{B}}}_{i_1 \ldots i_{d_\mathbb{A}}}$, where $d_\mathbb{A}$ of the features are from $\mathcal{A}$ and $d_\mathbb{B}$ of them are from $\mathcal{B}$.

With this inter-partition versus intra-partition distinction in mind, we can look back at the row vector expressions in Eq. (5.12) and see that each row vector has only a single component that does not involve inter-partition coefficients. Unsurprisingly, this component corresponds to the basis vector $|0\rangle$, which spans the zero-excitation subspace $\mathbb{E}^{(0)}$. The one exception is $|W_0\rangle$, where every coefficient is intra-partition. If we require that all features be from either $\mathcal{A}$ or $\mathcal{B}$, then we are restricted to a single row vector that has support across all excitation subspaces (all columns of the matrix), and a large number of row vectors that that have support only on $\mathbb{E}^{(0)}$ (the first column of the matrix).

The practical significance of all this is that the multiplex rank of the weight tensor is determined almost entirely by the inter-partition coefficients. To give a concrete example, we can consider a weight tensor where all inter-partition coefficients are set to zero. This would result in a regression model for which the interaction degree is at most $|\mathcal{B}|$ among feature-products from $\mathcal{B}$ and at most $|\mathcal{A}|$ among feature-products from $\mathcal{A}$. The row vector grid for the weight tensor would be

$$\begin{array}{llllll}
W_0\,|0\rangle & + \displaystyle\sum_{j_1} W^{j_1}\,|j_1\rangle & + \displaystyle\sum_{j_1,j_2} W^{j_1 j_2}\,|j_1, j_2\rangle & + \ldots + & \displaystyle\sum_{j_1,\ldots,j_{|\mathcal{B}|}} W^{j_1 \cdots j_{|\mathcal{B}|}}\,|j_1, \ldots, j_{|\mathcal{B}|}\rangle & (5.36) \\
W_{i_1}\,|0\rangle & + \;\; 0 & + \;\; 0 & + \ldots + & 0 & (5.37) \\
W_{i_1 i_2}\,|0\rangle & + \;\; 0 & + \;\; 0 & + \ldots + & 0 & (5.38) \\
\vdots & \;\;\;\; \vdots & \;\;\;\; \vdots & \;\;\;\;\; \vdots & \vdots & (5.39) \\
W_{i_1 \ldots i_{|\mathcal{A}|}}\,|0\rangle & + \;\; 0 & + \;\; 0 & + \ldots + & 0, & (5.40)
\end{array}$$

which shows that there can be a maximum of two linearly independent row vectors for any values of $|\mathcal{A}|$ and $|\mathcal{B}|$. Note that in order for the multiplex rank to be one, all of the intra-partition regressors in either $|\mathcal{A}|$ or $|\mathcal{B}|$ would need to also be set to zero. This implies that, contrary to what might be expected, factorization of the weight tensor across the partition is not in general possible even if all inter-partition coefficients are removed.

5.4.3 Short-range and long-range interactions

Given a specific network architecture, it is natural to ask whether certain kinds of feature interactions are more costly to represent than others. It it is well-known in quantum physics, for example, that an MPS-based quantum state favors short-range correlations between sites on a 1-D lattice [60]. While it is possible to represent states with long-range correlations using an MPS, it generally requires a significantly larger bond dimension. In this subsection we carry out a similar analysis for short-range and long-range interactions in an MPS regression model, and quantify the impact of interaction distance on the bond dimension.

To begin, we induce a distance measure between the features by mapping them onto a 1-D grid. The initial ordering of the features is unimportant for our purposes, so we will simply take x_i to be the element on the ith grid point. With this spatial arrangement, the distance between x_i and x_j is given by $|i-j|$, which is one plus the number of features separating x_i and x_j on the grid. For m features, the maximum possible distance is $m-1$, while the shortest distance is 1. Since our analysis will be focused on bipartitions of the features that are contiguous on the grid, we define the *split point* to be the the value of i such that $x_1, ..., x_i$ are in the left partition and $x_{i+1}, ..., x_m$ are in the right partition.

Let us now consider a generic split point $s < \frac{m}{2} - 1$ which divides the features into a smaller partition $\mathcal{A} = \{x_1, ..., x_s\}$ and a larger partition $\mathcal{B} = \{x_{s+1}, ..., x_m\}$. When all inter-partition regressors are set to zero, the row vectors of the matricized weight tensor are identical to those in Eqs. (5.36 - 5.40) with $|\mathcal{A}| = s$ and $|\mathcal{B}| = m - s$. If we introduce an interaction between x_1 and x_m, which is to say that we assign a non-zero coefficient to regressor $x_1 x_m$, then row vector $|W_1\rangle$ becomes $W_1 |0\rangle + |W_1^m\rangle |m\rangle$. This increases the multiplex rank of the weight tensor by one, since the row vector now has support on $\mathbb{E}^{(1)}$ in addition to $\mathbb{E}^{(0)}$. If we introduce another interaction, this time between x_s and x_{s+1}, then $|W_s\rangle = W_s |0\rangle + W_s^{s+1} |s+1\rangle$ and we again increase the multiplex rank by one. It is important to note that the distances of the $x_1 x_m$ and $x_s x_{s+1}$ interactions are 1 and $m-1$ respectively, yet they both increase the multiplex rank by the same amount. This reflects the fact that excitation subspaces have no notion of distance, with a change of $W_i |0\rangle \rightarrow W_i |0\rangle + W_i^j |j\rangle$ in a row vector having the same implications for linear dependence regardless of the value of j. Indeed, for fixed partitions $\mathcal{A}$ and $\mathcal{B}$ the relevant quantity is the number of inter-partition interactions[2], rather than their distance.

The significance of the interaction distance become apparent when we consider the impact of a given interaction across *multiple* partition schemes. Continuing with the $x_1 x_m$ and $x_s x_{s+1}$ interactions, let us shift the bipartition split point from s to $s+1$, such that $\mathcal{A} = \{x_1, ..., x_{s+1}\}$ and $\mathcal{B} = \{x_{s+2}, ..., x_m\}$. With respect to these new partitions, the $x_s x_{s+1}$ interaction now lies entirely in $\mathcal{A}$, and therefore no longer contributes to the corresponding multiplex rank of the weight tensor. The $x_1 x_m$ interaction, by contrast, still spans the two partitions and will therefore continue to increase the multiplex rank. We can repeat this analysis for all possible split points $1 \leq s' \leq m-1$, and find that the $x_1 x_m$ interaction always spans the two partitions, while the $x_s x_{s+1}$ interaction only spans the partitions when $s' = s$.

For an MPS regression model, the significance of our analysis here is that a long-range interaction will increase the bond dimensions of more virtual indices than a short-range interaction will. The $x_1 x_m$ interaction, for example, could increase the bond dimension for every virtual index in the network, while the $x_s x_{s+1}$ interaction would only increase the bond dimension of the virtual index between x_s and x_{s+1}. In this sense, we can say that the MPS regression ansatz shows the same limitations as the MPS ansatz from quantum physics, in that long-range interactions require larger bond dimensions and are thus more costly to represent than short-range interactions.

To provide a concrete example, we will consider two different bilinear MPS regression

[2] We should note that the number of interactions is still not the only factor to consider, since the multiplex rank is reduced if the same feature participates in multiple interactions.

models: one has exclusively *nearest-neighbor* interactions, while the other has an equal number of interactions that are randomly distributed. In this context, a nearest-neighbor interactions encompasses any regressor of the form $x_i x_{i+1}$, or equivalently any interaction with a distance of 1. Given m features, there are $m-1$ such interactions, so both models will have a total of $m-1$ regressors of degree 2 (as well as m degree-1 regressors and a bias term). If we again consider an arbitrary split point $s < \frac{m}{2}-1$, then the only inter-partition regressor in the nearest-neighbor model will be $x_s x_{s+1}$. This means that the bond dimension for each virtual index will be at most three regardless of the value of m, which is much smaller in general than the maximum value of $\frac{m}{2}$ from Figure 5.1. By contrast, the bilinear model with $m-1$ randomly-distributed interactions will have an average bond dimension that scales linearly with m (maximum of about 300 for $m = 784$), and therefore be impractical for data with a large number of features.

5.4.4 Rank reduction through embedding

In Chapter 4, we utilized the interaction decomposition to selectively retain only a subset of regressors in a tensor network regression model, thereby generating a degree-limited model similar to those that we have been considering here. However, a key difference is that the interaction decomposition can generate low-degree regression functions without requiring that the high-degree elements of the weight tensor be set to zero. It is reasonable then to ask how the multiplex rank of a cumulative-d interaction decomposition model from Sec. 4.4 may differ from that of a standard tensor regression model, which we will refer to here as the "zeroed" model.

To see more clearly the difference between an interaction decomposed model and a zeroed model, we can look at the row vector grid corresponding to a cumulative-2 (i.e. bilinear) model:

$$W_0 \left|0\right\rangle + \sum_{j_1} W^{j_1} \left|j_1\right\rangle + \sum_{j_1,j_2} W^{j_1 j_2} \left|j_1, j_2\right\rangle \tag{5.41}$$

$$W_{i_1} \left|0\right\rangle + \sum_{j_1} W_{i_1}^{j_1} \left|j_1\right\rangle + \sum_{j_1,j_2} \phi_{i_1}^{j_1 j_2} \left|j_1, j_2\right\rangle \tag{5.42}$$

$$W_{i_1 i_2} \left|0\right\rangle + \sum_{j_1} \phi_{i_1 i_2}^{j_1} \left|j_1\right\rangle + \sum_{j_1,j_2} \phi_{i_1 i_2}^{j_1 j_2} \left|j_1, j_2\right\rangle , \tag{5.43}$$

where $\phi_{i_1 \ldots i_k}^{j_1 \ldots j_\ell}$ is a parameter whose value can be set arbitrarily, since it will not actually be included in the regression. Comparing Eqs. (5.41 - 5.43) to the bilinear grid from Eqs. (5.27 - 5.29), the most significant difference is that every row vector in the cumulative-2 weight tensor can span the entire set of excitation subspaces, since the values of the free parameters $\phi_{i_1 \ldots i_k}^{j_1 \ldots j_\ell}$ are not constrained to be zero. Given that the output of the model is determined exclusively by the $W_{i_1 \ldots i_k}^{j_1 \ldots j_\ell}$ coefficients , we can view Eqs. (5.41 - 5.43) as the *embedding* of a low-degree weight tensor within a tensor with maximal degree.

As shown in Appendix 5.6.2, the multiplex rank of an embedded weight tensor will in general be less than the corresponding weight tensor which has its higher-degree coefficients explicitly set to zero. As a simple example, we can easily show that almost all linear regression

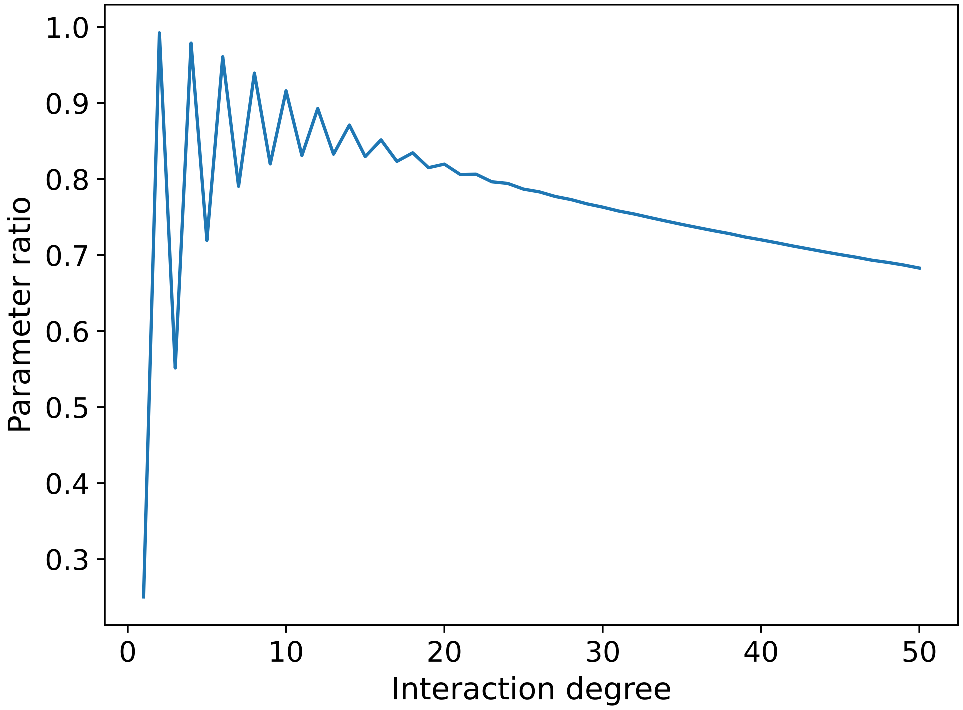


Figure 5.3: Plot of the parameter ratio for an MPS representing the embedded weight tensor over an MPS representing the zeroed weight tensor, with $m = 784$. In all cases the number of parameter is on the same order of magnitude, although the embedded network is always smaller. There is a clear oscillation in the ratio at lower degrees, which eventually smooths out into a roughly linear downward trend at higher degrees.

functions can be represented using rank-one weight tensors if the higher-degree contributions are ignored. Given row vectors $|W_0\rangle = W_0\,|0\rangle + \sum_j W^j\,|j\rangle$ and $|W_i\rangle = W_i\,|0\rangle + \sum_j \phi_i^j\,|j\rangle$, we will have $|W_i\rangle \propto |W_0\rangle$ for all i if

$$|W_i\rangle = \sum_{j=1}^{m} W_i\,|0\rangle + \frac{W^j W_i}{W_0}\,|j\rangle = \frac{W_i}{W_0}\,|W_0\rangle\,, \tag{5.44}$$

where we have set $\phi_i^j = \frac{W^j W_i}{W_0}$. In a full contraction of the weight tensor, the second-degree term present in Eq. (5.44) would interefere with the regression function, but using the interaction decomposition we can efficiently remove it. Note that ϕ_i^j will be undefined if $W_0 = 0$, so regression functions with no bias term are not factorizable. Interestingly, this means that shifting the data features can in principle lead to models of differing complexity, although in practice $W_0 = 0$ can be approximated to high precision.

Given a reduction in the multiplex rank, we can expect a reduction in the number of parameters needed to represent the weight tensor using a tensor network. In Figure 5.3, we plot the ratio of MPS parameter number for the embedded and zeroed weight tensors at different interaction degrees, with the raw parameter number for the zeroed model having been given in Figure 5.2. The plot clearly shows that the embedded and zeroed weight tensors have parameter numbers on the same order of magnitude, with the embedded MPS always requiring somewhat fewer elements. The most significant reductions are seen in the linear model and at higher interaction degrees, with odd degrees showing a greater effect than even degrees. The patterns seen in Figure 5.3 are driven by broader trends in the bond dimension, which can be seen in Figure 5.4 from Appendix 5.6.2.

5.5 Discussion

The objective of our work in this chapter was to find a precise quantitative relationship between the interaction degree of a regression model and the bond dimension needed to represent it via a tensor network. The key quantity underlying this relationship is the multiplex rank, and we therefore derived a general algorithm to compute its upper bound given a feature number, partition size, and interaction degree. With this algorithm, we then probed various properties of the bond dimension in the context of tensor network regression, with an eye toward understanding how the form of the weight tensor can place greater or lesser demands on the network.

The benefit of this kind of analysis is that it offers a clear mathematical framing for many questions that can be asked about tensor network models. We expect intuitively that a larger bond dimension should be associated with more sophisticated regression models, and in Sec. 5.4.1 we were able to demonstrate that this is true if a model's maximum interaction degree is taken as a measure of its sophistication. For MPS models, it is commonly assumed that the tensor network ansatz must favor short-range correlation between features, though precise descriptions of these "correlations" have been lacking. In Secs. 5.4.2 and 5.4.3, we showed that interactions between features far apart on the MPS chain have a larger impact on the bond dimension of the network than interactions with a shorter range. This reasoning could be generalized to other tensor network architectures, such as PEPS (see Sec. 3.1.4), to see whether they possess the same properties that have been observed in quantum physics.

Beyond answering existing question, the multiplex rank analysis in this chapter also provides a straightforward method of investigating new tensor regression algorithms. For the interaction decomposition models proposed in Sec. 4.4, we showed in Sec. 5.4.4 that the degree-limited regression function can be embedded into weight tensor of maximal degree, and that this tensor can have a multiplex rank smaller than a weight tensor whose higher-degree coefficients were explicitly set to zero. However, we also showed that this leads to only a modest reduction in the number of parameters in a model, and therefore does not itself represent an especially compelling reason to employ these kinds of models. Any algorithm which seeks to reduce the required bond dimension of a tensor network model can be similarly evaluated by probing the characteristics of its multiplex rank.

It is important to note that all of our work in this chapter was done without making reference to any specific properties of the underlying dataset. While the number of features was significant, the structure and identity of the those features was entirely irrelevant to our analysis. This was a deliberate choice, as the dataset determines the composition of the tensor network only indirectly via its impact on the weight tensor. In this light, our analysis is predicated on assuming a set of reasonable weight tensors from the outset, without worrying about why that type of weight tensor might be desirable for a given regression task. This is not uncommon in machine learning, as models are often employed without any clear indication that the properties of the data would demand that model type. Still, we would ideally want to tailor the tensor network based on the properties of the target dataset, and we explore a possible method for achieving this in Chapter 6.

5.6 Appendix

5.6.1 Multiplex rank as a function of partition size

In Sec. 5.4.1, we considered the minimum bond dimension needed for an MPS model to carry out unconstrained multilinear regression up to a given degree, and how it varied for different virtual indices in the chain. The plots in Figure 5.1 revealed a complex relationship between the bond dimension, which is equivalent to the multiplex rank, and the index position, which is a proxy for the partition size $|\mathcal{A}|$. In the present subsection we demonstrate how this relationship emerges out of Eq. (5.34), and assess its properties.

The curves in Figure 5.1 are largely in the regime where $d < |\mathcal{A}| < |\mathcal{B}|$, so this is where we will focus our analysis. Under such constraints, we have from Eqs. (5.33) and (5.34) that

$$\alpha_{d_{\mathbb{B}}} = \sum_{d_{\mathbb{A}}=0}^{d-d_{\mathbb{B}}} \binom{|\mathcal{A}|}{d_{\mathbb{A}}} = a_{d_{\mathbb{B}}+1} + \binom{|\mathcal{A}|}{d - d_{\mathbb{B}}} \tag{5.45}$$

$$v_{d_{\mathbb{B}}} = \min\left(a_{d_{\mathbb{B}}} - \sum_{d'_{\mathbb{B}}=d_{\mathbb{B}}+1}^{d} v_{d'_{\mathbb{B}}+1},\ \beta_{d_{\mathbb{B}}} \right) \tag{5.46}$$

$$r = \sum_{d_{\mathbb{B}}=0}^{d} v_{d_{\mathbb{B}}}, \tag{5.47}$$

where each of the summation upper limits now depend only on d. Note that with these new limits, $\alpha_{d_{\mathbb{B}}}$ can be written recursively in terms of $\alpha_{d_{\mathbb{B}}+1}$. If we now consider v_d, the number of linearly-independent vectors in the highest excitation subspace $\mathbb{E}^{(d)}$, it is clear that Eq. (5.46) becomes

$$v_d = \min(\alpha_d, \beta_d) = \min\left(\binom{|\mathcal{A}|}{0}, \binom{|\mathcal{B}|}{d} \right) = \alpha_d, \tag{5.48}$$

since $\binom{|\mathcal{A}|}{0} = 1$ will always be less than or equal to $\binom{|\mathcal{B}|}{d}$ for any value of d. If we now move from higher to lower excitations and evaluate v_{d-1}, we find

$$v_{d-1} = \min(\alpha_{d-1} - \alpha_d, \beta_{d-1}) = \min\left(\binom{|\mathcal{A}|}{1}, \binom{|\mathcal{B}|}{d-1} \right), \tag{5.49}$$

where $a_{d-1} - a_d$ is easily evaluated using the recursive relationship in Eq. (5.33).

The form of Eq. (5.49) places us at a cross-roads: if $d > 1$, then $\binom{|\mathcal{A}|}{1} < \binom{|\mathcal{B}|}{d-1}$ since $|\mathcal{A}| < |\mathcal{B}|$ and we have that $v_{d-1} = \binom{|\mathcal{A}|}{1}$. If $d \leq 1$, however, then $\binom{|\mathcal{A}|}{1} > \binom{|\mathcal{B}|}{d-1}$ and we have $v_{d-1} = \beta_{d-1}$. Even if $d > 1$, there will eventually be a value $0 \leq c < d$ such that $\binom{|\mathcal{A}|}{d-c} > \binom{|\mathcal{B}|}{c}$, and we refer to this value c as the *cross-over* point. This point is significant because it marks where $v_{d_{\mathbb{B}}}$ stops being determined by the value of $|\mathcal{A}|$, and starts being determined by the value of $|\mathcal{B}|$. For all $d_{\mathbb{B}} > c$ we have

$$\begin{aligned} v_{d_{\mathbb{B}}} &= \min\left(\alpha_{d_{\mathbb{B}}} - \sum_{d'_{\mathbb{B}}=d_{\mathbb{B}}+1}^{d} v_{d'_{\mathbb{B}}}, \beta_{d_{\mathbb{B}}}\right) = \min(\alpha_{d_{\mathbb{B}}} - \alpha_{d_{\mathbb{B}}+1}, \beta_{d_{\mathbb{B}}}) = \min\left(\binom{|\mathcal{A}|}{d-d_{\mathbb{B}}}, \binom{|\mathcal{B}|}{d_{\mathbb{B}}}\right) \\ &= \binom{|\mathcal{A}|}{d-d_{\mathbb{B}}}, \end{aligned} \tag{5.50}$$

which can be confirmed by simply following the recursion back to v_d. For $d_{\mathbb{B}} = c$, the minimum changes and we instead have

$$v_c = \min\left(\alpha_c - \sum_{d'_{\mathbb{B}}=c+1}^{d} v_{d'_{\mathbb{B}}}, \beta_c\right) = \min(\alpha_c - \alpha_{c+1}, \beta_c) = \min\left(\binom{|\mathcal{A}|}{d-c}, \binom{|\mathcal{B}|}{c}\right) = \binom{|\mathcal{B}|}{c}, \tag{5.51}$$

and thus $v_c = \beta_c$. The equality of $v_{d_{\mathbb{B}}}$ and $\beta_{d_{\mathbb{B}}}$ continues for all $d_{\mathbb{B}} < c$, since with more excitations shifting from $\mathcal{B}$ to $\mathcal{A}$ the number of row vectors only continues to increase relative to the size of the excitation subspace.

Putting the $d_{\mathbb{B}} > c$ and $d_{\mathbb{B}} \leq c$ regimes together, we have that the multiplex rank r is given by

$$r(|\mathcal{A}|, |\mathcal{B}|) = \sum_{d_{\mathbb{B}}=0}^{c} \binom{|\mathcal{B}|}{d_{\mathbb{B}}} + \sum_{d'_{\mathbb{B}}=c+1}^{d} \binom{|\mathcal{A}|}{d-d'_{\mathbb{B}}}, \quad c = \max d_{\mathbb{B}} \text{ s.t. } \binom{|\mathcal{A}|}{d-d_{\mathbb{B}}} > \binom{|\mathcal{B}|}{d_{\mathbb{B}}}, \tag{5.52}$$

which holds whenever the maximum interaction degree d is less than or equal to $|\mathcal{A}|$. In words, Eq. (5.52) states that contributions to the multiplex rank from higher excitation subspaces are limited by the small number of vectors that have support on them, thus giving a dependence on $|\mathcal{A}|$. However, at some cross-over point c the number of supported row vectors starts to exceed the size of the subspace, and thus the rank contributions begin to be limited by the dimension of the excitation subspaces as dictated by $|\mathcal{B}|$.

To understand the shapes of the bond dimension curves in Figure 5.1, we can use Eq. (5.52) to compute the finite difference $\Delta[r](|\mathcal{A}|, |\mathcal{B}|) = r(|\mathcal{A}|+1, |\mathcal{B}|-1) - r(|\mathcal{A}|, |\mathcal{B}|)$, which takes the form

$$\begin{aligned}\Delta[r](|\mathcal{A}|,|\mathcal{B}|) &= \sum_{d_{\mathbb{B}}=0}^{c'}\binom{|\mathcal{B}|-1}{d_{\mathbb{B}}} + \sum_{d'_{\mathbb{B}}=c'+1}^{d}\binom{|\mathcal{A}|+1}{d-d'_{\mathbb{B}}} - \sum_{d_{\mathbb{B}}=0}^{c}\binom{|\mathcal{B}|}{d_{\mathbb{B}}} - \sum_{d'_{\mathbb{B}}=c+1}^{d}\binom{|\mathcal{A}|}{d-d'_{\mathbb{B}}}\\
&= \sum_{d_{\mathbb{B}}=0}^{c}\binom{|\mathcal{B}|-1}{d_{\mathbb{B}}} - \binom{|\mathcal{B}|}{d_{\mathbb{B}}} + \sum_{d'_{\mathbb{B}}=c'+1}^{d}\binom{|\mathcal{A}|+1}{d-d'_{\mathbb{B}}} - \binom{|\mathcal{A}|}{d-d'_{\mathbb{B}}}\\
&\quad + \sum_{d''_{\mathbb{B}}=c+1}^{c'}\binom{|\mathcal{B}|-1}{d''_{\mathbb{B}}} - \binom{|\mathcal{A}|}{d-d''_{\mathbb{B}}}\\
&= -\sum_{d_{\mathbb{B}}=0}^{c-1}\binom{|\mathcal{B}|-1}{d_{\mathbb{B}}} + \sum_{d'_{\mathbb{B}}=c'+2}^{d}\binom{|\mathcal{A}|}{d-d'_{\mathbb{B}}} + \sum_{d''_{\mathbb{B}}=c+1}^{c'}\binom{|\mathcal{B}|-1}{d''_{\mathbb{B}}} - \binom{|\mathcal{A}|}{d-d''_{\mathbb{B}}},\end{aligned} \tag{5.53}$$

where c and c' are the cutoff points for $|\mathcal{A}|$, $|\mathcal{B}|$ and $|\mathcal{A}|+1$, $|\mathcal{B}|-1$ respectively (note that $c' \geq c$). While Eq. (5.53) does not immediately lend itself to interpretation, we can gain further insight by analyzing $\Delta[r]$ for fixed values of c. These expressions are easily derived from the last line of Eq. (5.53) by setting $c' = c$:

$$\Delta[r](|\mathcal{A}|,|\mathcal{B}|) = \begin{cases} \displaystyle\sum_{d_{\mathbb{A}}=0}^{d-2}\binom{|\mathcal{A}|}{d_{\mathbb{A}}} & c=0\\[2ex] \displaystyle\sum_{d_{\mathbb{A}}=0}^{d-3}\binom{|\mathcal{A}|}{d_{\mathbb{A}}} - 1 & c=1\\[2ex] \displaystyle\sum_{d_{\mathbb{A}}=0}^{d-4}\binom{|\mathcal{A}|}{d_{\mathbb{A}}} - \binom{|\mathcal{B}|-1}{1} - 1 & c=2\\ \vdots & \\ \displaystyle\sum_{d_{\mathbb{A}}=0}^{\lceil\frac{d}{2}\rceil-2}\binom{|\mathcal{A}|}{d_{\mathbb{A}}} - \sum_{d_{\mathbb{B}}=0}^{\lfloor\frac{d}{2}\rfloor-1}\binom{|\mathcal{B}|-1}{d_{\mathbb{B}}} & c=\left\lfloor\dfrac{d}{2}\right\rfloor, \end{cases} \tag{5.54}$$

where the lower limit $c = \left\lfloor\frac{d}{2}\right\rfloor$ marks the largest value of c for which $\binom{|\mathcal{A}|}{d-c}$ could ever be greater than $\binom{|\mathcal{B}|}{c}$. Note that we have introduced the index $d_{\mathbb{A}} = d - d_{\mathbb{B}}$ to simplify the expressions. The implication of Eq. (5.54) is that not only will the rate of change of the multiplex rank vary with $|\mathcal{A}|$ and $|\mathcal{B}|$, but the functional form of this dependence will also change when $|\mathcal{A}|$ passes through certain critical points that mark a change in the cutoff value c. As $|\mathcal{A}|$ increases the cutoff grows, and thus the finite difference moves through Eq. (5.54) from top to bottom.

To see how Eq. (5.54) operates in practice, we can use it to generate some of the curves observed in Figure 5.1. For linear regression ($d = 1$), the only valid cutoff point is $c = 0$, which by the given summation limit yields $\Delta[r] = 0$ as expected. For bilinear regression ($d = 2$), the cutoff point could be either $c = 0$ or $c = 1$ based solely off the range of c values given in Eq. (5.54), but we can easily verify that the $\binom{|\mathcal{A}|}{d-c} > \binom{|\mathcal{B}|}{c}$ inequality is not satisfied when $c = 1$. The finite difference is therefore given by $\Delta[r] = \binom{|\mathcal{A}|}{0} = 1$, which again agrees with the plot in Figure 5.1. A more interesting example is cubic regression, ($d = 3$), where $c = 0$ and $c = 1$ are both valid cutoffs. This results in the piecewise function

$$\Delta[r](|\mathcal{A}|, |\mathcal{B}|) = \begin{cases} |\mathcal{A}| + 1 & c = 0 \\ \\ 0 & c = 1 \end{cases} \tag{5.55}$$

which explains the parabolic rise and then sudden plateau in the cubic curve that was observed in Figure 5.1. Note that the values of $|\mathcal{A}|$ for which the cutoff changes can be solved for by expanding out the binomial coefficients on both sides of $\binom{|\mathcal{A}|}{d-c} > \binom{|\mathcal{B}|}{c}$. In the cubic case, it can be easily shown that $c = 1$ whenever

$$|\mathcal{A}| > \frac{\sqrt{1 + 8m} - 1}{2}, \tag{5.56}$$

whereas $c = 0$ for smaller values of $|\mathcal{A}|$.

Finally, we can use Eq. (5.53) to show how it is possible for the multiplex rank to actually *shrink* as $|\mathcal{A}|$ increases. Taking $d = 5$, we have cutoff values of $c \in \{0, 1, 2\}$, which leads to the piecewise function

$$\Delta[r](|\mathcal{A}|, |\mathcal{B}|) = \begin{cases} \binom{|\mathcal{A}|}{3} + \binom{|\mathcal{A}|}{2} + |\mathcal{A}| + 1 & c = 0 \\ \\ \binom{|\mathcal{A}|}{2} + |\mathcal{A}| & c = 1 \\ \\ |\mathcal{A}| - |\mathcal{B}| + 1 & c = 2. \end{cases} \tag{5.57}$$

Since $|\mathcal{A}| \leq |\mathcal{B}|$ by construction, the slope of the multiplex rank will be negative for every value of $|\mathcal{A}|$ once $c = 2$, which explains the sudden drop-off of the degree-5 curve in Figure 5.1. The more complicated curves corresponding to degrees 39 and 40 in that figure could also be laboriously explained using Eq. (5.54), but we will just note here that each of the observed cusps can be mapped back to a change in the cutoff value as $|\mathcal{A}|$ increases.

5.6.2 Rank of embedded weight tensors

In Sec. 5.4.4, we considered the multiplex rank of a low-degree weight tensor whose higher-degree coefficients are simply ignored rather than being set to zero. Given an appropriate choice of these freely-varying parameters, denoted $\phi_{i_1 \ldots i_k}^{j_1 \ldots j_\ell}$, we show here that it is possible to reduce the multiplex rank of this embedded weight tensor below that of the zeroed version.

Although the general problem of constrained rank minimization is NP-hard [98], it is relatively straightforward to derive a lower-bound for the multiplex rank of the cumulative-d weight tensors. To motivate our approach, it is helpful to first consider the simpler problem of embedding a $d \times d$ triangular matrix T into a generic $d \times d$ matrix M. Using T_{ij} to denote the fixed elements taken from T, and ϕ_{ij} to denote elements of M which are free to vary, we have

$$T = \begin{bmatrix} T_{1,1} & T_{1,2} & \dots & T_{1,d-1} & T_{1,d} \\ T_{2,1} & T_{2,2} & \dots & T_{2,d-1} & 0 \\ T_{3,1} & T_{3,2} & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ T_{d,0} & 0 & \dots & 0 & 0 \end{bmatrix}, \qquad M = \begin{bmatrix} T_{1,1} & T_{1,2} & \dots & T_{1,d-1} & T_{1,d} \\ T_{2,1} & T_{2,2} & \dots & T_{2,d-1} & \phi_{2,d} \\ T_{3,1} & T_{3,2} & \dots & \phi_{2,d-1} & \phi_{2,d} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ T_{d,1} & \phi_{d,2} & \dots & \phi_{d,d-1} & \phi_{d,d} \end{bmatrix}. \tag{5.58}$$

Note that T is a rotated version of the usual triangular matrix format, intended to better align with the form of the weight tensor matricizations.

The maximum rank of T is easily seen to be d, since it is always possible to choose elements such that every row is linearly independent of the others. The question then is whether the free parameters ϕ_{ij} in M can be chosen such that some of the row vectors become linearly dependent. The limiting factor in this dependence will necessarily be the fixed elements T_{ij}, and thus a lower-bound on the rank of M can be found by considering submatrices of M which contain only T_{ij} elements. The submatrix of this form with the highest possible rank is the contiguous square submatrix R whose upper-left element is $T_{1,1}$ and whose lower-left element is $T_{\lceil \frac{d}{2} \rceil, \lceil \frac{d}{2} \rceil}$, which has a maximum rank of $\lceil \frac{d}{2} \rceil$. Regardless of the values chosen for $\{\phi_{ij}\}$, the rank of M can never be made lower than the rank of R.

On the other hand, for non-singular R there will always exist an M that has a rank equal to this lower bound[3]. Consider the $d \times \lceil \frac{d}{2} \rceil$ submatrix generated by removing all columns of M not present in R. This submatrix has an additional $d - \lceil \frac{d}{2} \rceil$ rows vectors not present in R, but these can always be written as linear combinations of the rows of R. If we now add back in the removed columns and consider the entire matrix M, these same linear combinations can be used to the generate the complete row vectors by simply choosing values for the ϕ_{ij} that match the corresponding combinations of the T_{ij} in the added columns.

We can apply this same logic to the matricized weight tensor by considering the smallest submatrix whose row space spans all of the non-zero regression coefficients $W^{j_1 \dots j_\ell}_{i_1 \dots i_k}$. We therefore seek the smallest set of row vectors which each contain at least as many non-zero coefficients as any row vector outside the set, and which contains a number of row vectors greater than or equal to this coefficient count. The elements $W^{j_1 \dots j_\ell}_{i_1 \dots i_k}$ are non-zero wherever the total excitation number is less than or equal to the maximum interaction degree d of the model, which means that the width of row vector $|W_{i_1 \dots i_k}\rangle$ (i.e. the number of columns on which it has support) is given by the sum of the dimensions of the excitation subspaces less than or equal to $d - k$. The number of row vectors with k or fewer excitation is given by α_k from Eq. (5.33), where we assume $|\mathcal{A}| > d$. Putting these together, we want to find the

[3] The bound is not tight when R is singular, but we can always approximate the elements of a singular matrix to arbitrary precision using a full-rank matrix.

number of excitations n such that

$$\sum_{d_{\mathbb{A}}=0}^{n} \binom{|\mathcal{A}|}{d_{\mathbb{A}}} \geq \sum_{d_{\mathbb{B}}=0}^{d-n-1} \binom{|\mathcal{B}|}{d_{\mathbb{B}}}, \tag{5.59}$$

where the left side of the expression denotes the number of row vectors with at most n excitations and the right side denotes the maximum number of non-zero coefficients of row vectors with $n+1$ or more excitations.

The submatrix formed by all row vectors with up to n excitations has the shape $\sum_{d_{\mathbb{A}}=0}^{n} \binom{|\mathcal{A}|}{d_{\mathbb{A}}} \times \sum_{d_{\mathbb{B}}=0}^{d-n} \binom{|\mathcal{B}|}{d_{\mathbb{B}}}$, and thus its rank r will generically be given by

$$r = \min\left(\sum_{d_{\mathbb{A}}=0}^{n} \binom{|\mathcal{A}|}{d_{\mathbb{A}}}, \sum_{d_{\mathbb{B}}=0}^{d-n} \binom{|\mathcal{B}|}{d_{\mathbb{B}}} \right), \tag{5.60}$$

where we use the smallest value of n that satisfies Eq. (5.59). By the same reasoning used for T and M, this rank sets a lower bound that can in general be made tight by an appropriate choice of the free parameters $\phi_{i_1 \ldots i_k}^{j_1 \ldots j_\ell}$. In Figure 5.4, we show the result of applying Eq. (5.60) to the same MPS representation problem considered in Sec. 5.4.1 and plotted in Figure 5.1. For every interaction degree and index position, we find that the embedded weight tensor can be represented using a smaller bond dimension than the zeroed weight tensor, although the difference is not especially significant.

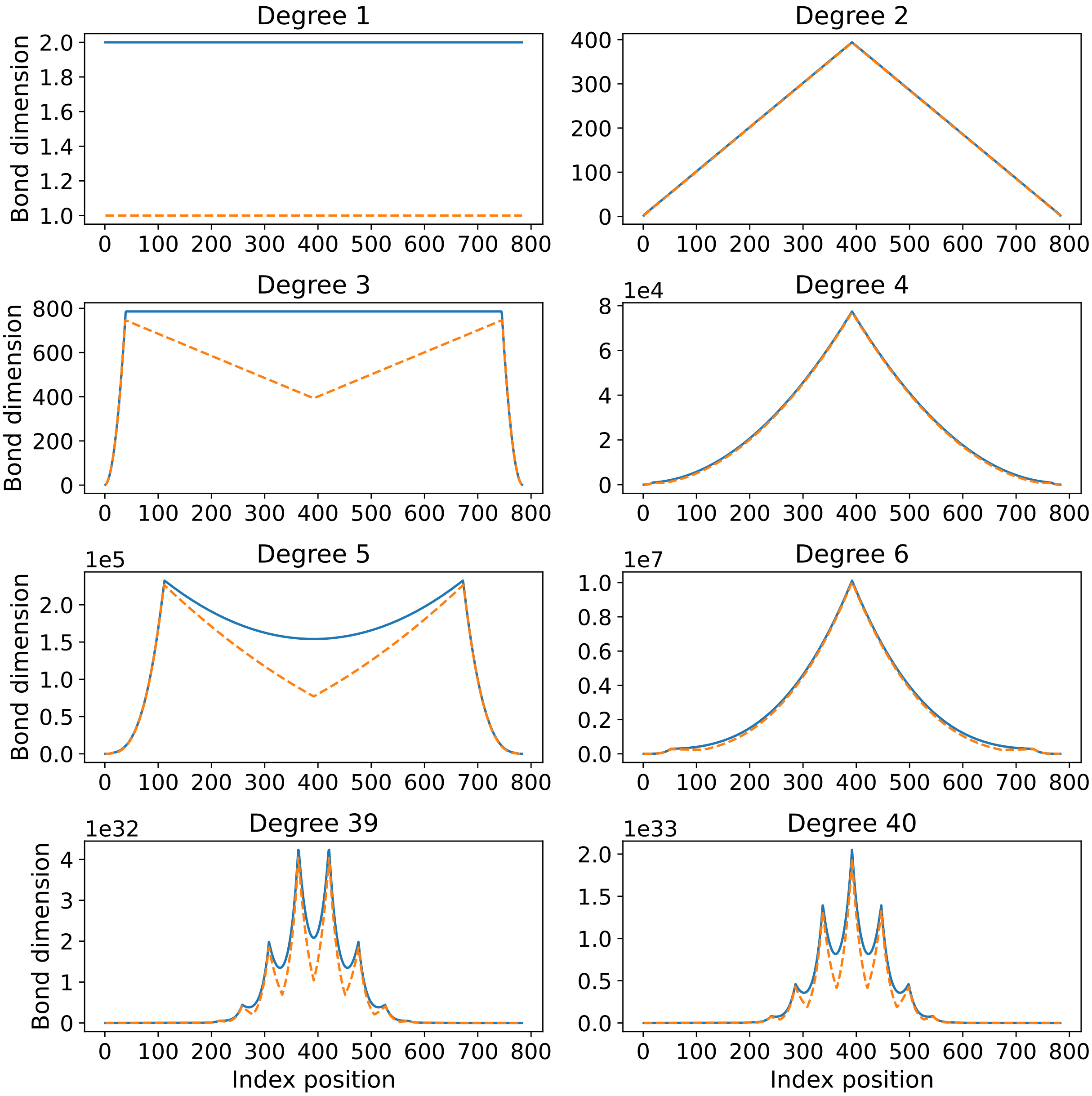


Figure 5.4: Plots of the MPS bond dimension for the embedded weight tensor (orange dashed line) and the zeroed weight tensor (blue solid line) as a function of virtual index position and interaction degree. The curves for the zeroed weight tensor are identical to those from Figure 5.1. The bond dimension of the embedded weight tensor representation is smaller for every index position and interaction degree, although the difference is never especially large in proportion.

Chapter 6

Mutual Information Scaling in Image Datasets

This chapter is derived from previously published work by Convy, Huggins, Liao, and Whaley [29], which carries out correlation scaling analysis on image datasets in a manner analogous to the entanglement scaling analysis from quantum physics.

6.1 Introduction

In Chapter 5, we explored how the bond dimension of a tensor network ansatz is impacted by the maximum interaction degree of the underlying regression model. A limitation of our approach there, however, was that we assumed a particular form for the weight tensor at the outset. This is fine for a theoretical analysis of tensor network models in general, but it fails to offer much guidance on designing a model for a *specific* machine learning task. Ideally, we would want tailor the properties of the weight tensor based on the available training set, along with prior knowledge we may have about the nature of the data, and then identify the optimal tensor network model for that particular weight tensor.

The challenge of designing a specialized tensor network ansatz is also faced in the field of quantum physics, where researchers seek to explore specific portions of Hilbert space using various network architectures. One of the most fruitful methods developed for this purpose has been the characterization of *entanglement scaling*, which describes how the bipartite entanglement in a quantum system grows with the partition size. As an example, it was discovered that the success of DMRG in 1-D systems was made possible by the short-range interactions present in many physical Hamiltonians, which lead to ground states that possess localized entanglement that obeys an "area law" or more properly a *boundary law* [71]. These discoveries have helped motivate the development of other network structures such as PEPS (see Sec. 3.1.4) and the multiscale entanglement renormalization ansatz (MERA) [99] to deal with multidimensional lattices and quantum critical points respectively.

The purpose of the work in this chapter is to take the entanglement scaling analysis that has been so illuminating in quantum many-body physics, and adapt it for use on the classical data commonly found in machine learning. Through this analysis, we seek to understand which tensor networks would be most appropriate for specific learning tasks. The chapter is organized into four sections, with Sec. 6.2 first reviewing how entanglement scaling relates to tensor network methods in quantum many-body physics. This analysis is then extended to classical data by using the mutual information (MI), which provides a generalized measure of correlation. We show that when using tensor networks for probabilistic classification of orthogonal inputs, the MI of the data provides a lower-bound on the entanglement and thus the connectivity of the tensors.

In Sec. 6.3, we introduce a numerical method for estimating the MI of a dataset given access to only a finite number of samples. We then test the accuracy of this method in Sec. 6.4 on a set of Gaussian distributions engineered to have different MI scaling patterns with respect to spatial partitioning of the variables. In Sec. 6.5 we estimate the MI scaling of MNIST and the Tiny Images, two well-known image datasets introduced in Sec. 2.3.1, and find evidence that the MI between a centered, square patch of pixels and the surrounding pixels scales with the boundary of the inner patch (a boundary law), rather than with the number of pixels (a volume law). This boundary-law scaling suggests that networks with an underlying 2-D grid structure such as PEPS or STNs (see Sec. 3.1.5) would be especially well-suited for machine learning on images.

6.2 Correlation Scaling

6.2.1 Entanglement Scaling in Quantum Systems

Entanglement is a defining property of quantum mechanics [100], and is the source of all correlations between components of a pure-state composite system [101]. Although there are multiple methods of quantifying entanglement, the *entropy of entanglement* is a widely used measure for entanglement between bipartitions of a composite system. For a pure state defined by the joint density matrix $\rho_{\mathcal{AB}}$ with reduced density matrices $\rho_{\mathcal{A}}$ and $\rho_{\mathcal{B}}$ corresponding to the bipartitions $\mathcal{A}$ and $\mathcal{B}$, the entanglement entropy is defined as the von Neumann entropy of $\rho_{\mathcal{A}}$ (or equivalently $\rho_{\mathcal{B}}$)

$$\mathrm{E}(\mathcal{A},\mathcal{B}) = -\mathrm{Tr}(\rho_{\mathcal{A}} \log \rho_{\mathcal{A}}). \tag{6.1}$$

A connection between the entanglement entropy of a quantum state and its structure can be made using the *Schmidt decomposition* [102], which is defined for state $|\psi\rangle$ on the combined Hilbert space $\mathbb{H}_{\mathcal{A}} \otimes \mathbb{H}_{\mathcal{B}}$ as

$$|\psi\rangle = \sum_{\alpha=1}^{r} \lambda_\alpha \, |s_\alpha^{\mathcal{A}}\rangle \, |s_\alpha^{\mathcal{B}}\rangle \,, \tag{6.2}$$

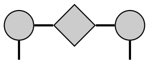
where r is the Schmidt rank, the λ_α are the Schmidt coefficients, and $|s_\alpha^{\mathcal{A}}\rangle \,, |s_\alpha^{\mathcal{B}}\rangle$ are the orthonormal Schmidt basis states in $\mathbb{H}_{\mathcal{A}}$ and $\mathbb{H}_{\mathcal{B}}$ respectively. Substituting Eq. (6.2) into Eq. (6.1) gives an expression for the entanglement in terms of the Schmidt coefficients

$$\mathrm{E}(\mathcal{A},\mathcal{B}) = -\sum_{\alpha=1}^{r} |\lambda_\alpha|^2 \log(|\lambda_\alpha|^2). \tag{6.3}$$

Formally, the Schmidt decomposition may be regarded as an SVD of the matrix C of coefficients that form $|\psi\rangle$:

$$\begin{aligned}|\psi\rangle &= \sum_{i,j} C_{ij}\,|i_{\mathcal{A}}\rangle\,|j_{\mathcal{B}}\rangle \\ &= \sum_{i,j,\alpha_1,\alpha_2} V_{i\alpha_1}\Lambda_{\alpha_1,\alpha_2}U^{\dagger}_{\alpha_2,j}\,|i_{\mathcal{A}}\rangle\,|j_{\mathcal{B}}\rangle \\ &= \sum_{i,j,\alpha}\lambda_\alpha V_{i\alpha}\,|i_{\mathcal{A}}\rangle\,U_{j,\alpha}\,|j_{\mathcal{B}}\rangle \\ &= \sum_{\alpha}\lambda_\alpha\,|s^{\mathcal{A}}_{\alpha}\rangle\,|s^{\mathcal{B}}_{\alpha}\rangle\,,\end{aligned} \tag{6.4}$$

where the rows of C correspond to the computational basis states $|i_{\mathcal{A}}\rangle$ in $\mathbb{H}_{\mathcal{A}}$ and the columns correspond to the computational basis states $|j_{\mathcal{B}}\rangle$ in $\mathbb{H}_{\mathcal{B}}$. The diagonal matrix Λ can be truncated so that it contains only the non-zero singular values of C, which are then equal to the Schmidt coefficients λ_α. Whenever there is more than one non-zero λ_α, the state possesses some degree of entanglement. Since the Schmidt decomposition is an SVD, the set of λ_α is guaranteed to be unique, and the Schmidt rank will be minimized with respect to all possible basis sets. Using the SVD matrices explicitly, we can write the Schmidt decomposition as a small tensor network

$$|\psi\rangle = \sum_{i,j,\alpha_1,\alpha_2} V_{i\alpha_1}\Lambda_{\alpha_1,\alpha_2}U^{\dagger}_{\alpha_2,j}\,|i_{\mathcal{A}}\rangle\,|j_{\mathcal{B}}\rangle \;\rightarrow$$

, (6.5)

where V, U are unitary matrices that map the basis states $|i_{\mathcal{A}}\rangle$, $|j_{\mathcal{B}}\rangle$ to the Schmidt bases of $\mathbb{H}_{\mathcal{A}}$ and $\mathbb{H}_{\mathcal{B}}$ respectively. It is important to note that this mathematical description of entanglement, which is based on the singular values, can be used to characterize a tensor regardless of whether it represents a truly quantum object.

The fact that Eq. (6.3) arises from a Schmidt decomposition is key to understanding the entanglement scaling properties of tensor networks. As discussed in Sec. 2.2.4, the collective size of the virtual indices which separate $i_{\mathcal{A}}$ and $j_{\mathcal{B}}$ in the network is lower-bounded by the multiplex rank of the tensor that it is representing. If we fix this collective index size to be some value q, then we can equivalently state that a tensor network will only be able to represent quantum states with Schmidt rank $r \leq q$. Through Eq. (6.3), this implies that the entanglement entropy represented by this tensor network bounded by

$$\mathrm{E}(\mathcal{A},\mathcal{B}) \leq \log(q), \tag{6.6}$$

where the inequality is saturated if $r = q$ and if the singular values are all $\frac{1}{q}$.

The expression in Eq. (6.6) can be refined by considering the geometry of the virtual indices in the network [103]. Assuming a maximum bond dimension given by t and a number of virtual indices n connecting the partitions, we will have $q = t^n$ and therefore Eq. (6.6) will become

$$\mathrm{E}(\mathcal{A},\mathcal{B}) \leq n\log(t). \tag{6.7}$$

For a fixed bond dimension t, differences in entanglement scaling between tensor networks will arise from differences in the value of n, which depends on the geometry of the network. For tensor networks which conform to the physical geometry of the composite system, such as MPS for 1-D systems and PEPS for 2-D systems, the number of indices connecting two partitions is determined by the size of the interface between the partitions. Given a simple partitioning of the system into a contiguous, hypercubic patch of length ℓ and the surrounding outer patch, the interface scales with the boundary of the inner patch. If the physical lattice dimension is d, then the entanglement follows a boundary-law scaling expression

$$\mathrm{E}(\mathcal{A},\mathcal{B}) \leq 2d\ell^{d-1}\log(t) = \mathcal{O}(\ell^{d-1}), \tag{6.8}$$

with ℓ being raised to a power one less than the dimension of the physical space.

The scaling behavior in Eq.(6.8) stands in sharp contrast to that of a random quantum state, whose entanglement will scale with the total size ℓ^d of the inner patch [104] rather than its boundary in what is referred to as a “volume law”. The success of methods like DMRG is only possible because the ground states of common Hamiltonians do not resemble states that have been randomly sampled from the Hilbert space, but instead tend to possess localized, boundary law entanglement that can be readily captured with the MPS ansatz. The existence of such scaling patterns has been proven for the ground states of 1-D gapped quantum systems [105], and for harmonic lattice systems of arbitrary dimension [106]. They have also been conjectured to exist in the ground states of most local, gapped quantum systems regardless of dimension [71]. Different tensor networks need to be employed when the ground state is suspected to violate the strict boundary law, with networks such as MERA being used to handle the $\log(\ell)$ corrections found in many critical-phase Hamiltonians [107]. In any case, the ultimate goal of these tensor network ansatzes is to match the known or predicted entanglement scaling of the quantum state with the entanglement scaling of the network.

6.2.2 Correlations in Classical Data

The preceding analysis used entanglement to quantify correlations in a system that was explicitly quantum mechanical. To carry out a similar analysis on classical data, we desire a more general quantity. A reasonable candidate is the *mutual information* (MI) [108], defined as

$$\mathrm{I}(a:b) = \mathrm{S}(a) + \mathrm{S}(b) - \mathrm{S}(a,b), \tag{6.9}$$

where $\mathrm{S}(a)$, $\mathrm{S}(b)$, and $\mathrm{S}(a,b)$ are the entropies of the probability distributions associated with marginal variables a, b and the joint outcome of a and b respectively. Qualitatively, the MI describes the amount of information we gain about one variable when we learn the state of the other, offering the most general measure of correlation. The MI can be calculated for either quantum or classical data, depending on whether the von Neumann or Shannon entropies are used. For a pure quantum state $\mathrm{S}(a,b) = 0$, and therefore the MI is equal to twice the entanglement.

An alternative but equivalent representation of the MI, which we make use of in Sec. 6.3, comes from the *Kullback-Liebler divergence* (KL-divergence), which is defined for two discrete probability distributions p and q on space $\mathcal{X}$ as

$$\mathrm{D}_{\mathrm{KL}}(p||q) = \sum_{x\in\mathcal{X}} p(x)\log\frac{p(x)}{q(x)}, \tag{6.10}$$

with an analogous definition for continuous variables that replaces the sum with an integral over probability densities. For a joint probability distribution p over variables a and b in spaces $\mathcal{A}$ and $\mathcal{B}$, the MI is equal to the KL-divergence between the joint distribution $p(a,b)$ and the uncorrelated product-of-marginals distribution $p(a)P(b)$, i.e.

$$\mathrm{I}(a:b) = \sum_{a\in\mathcal{A}}\sum_{b\in\mathcal{B}} p(a,b)\log\frac{p(a,b)}{p(a)p(b)}. \tag{6.11}$$

While not formally a metric, the KL-divergence can be viewed as measuring the distance between two distributions, so Eq. (6.11) represents the MI as the distance between $p(a,b)$ and the uncorrelated distribution $p(a)p(b)$.

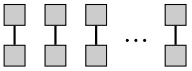
In the context of machine learning, the MI between features in a dataset can be measured by partitioning the features into two groups, assigning the collective state of each group to variables a and b respectively, and then measuring the amount of correlation that exists between the partitions. This parallels the bipartitioning of the quantum many-body system discussed in Sec. 6.2.1, and allows us to explore *MI scaling* in a similar manner to entanglement scaling.

6.2.3 Entanglement as a Bound on Mutual Information for Orthogonal Data

Given the connection between entanglement and tensor networks discussed in Sec. 6.2.1, and having introduced the MI as a classical measure of correlation in Sec. 6.2.2, we now show how the correlations in a classical dataset can guide the choice of network for machine learning. We focus on probabilistic classification, where the tensor network is used to approximate a probability distribution $p(x)$ of feature tensors generated from a classical data distribution $p(\vec{x})$ via Eq. (2.20). We show that for orthonormal inputs the entanglement of the tensor network between feature partitions $\mathcal{A}$ and $\mathcal{B}$ provides an upper bound on the MI of $p(x)$ between those same partitions. When designing a tensor network for a machine learning task, this relationship can be inverted so that the known MI of a given $p(x)$ sets a lower bound on the entanglement needed for the network to represent it. For non-orthogonal inputs these bounds do not hold rigorously, but may still serve as a useful heuristic for samples with negligible overlap.

To begin, let $p(\vec{x})$ be the probability distribution associated with feature vectors $\vec{x}$ of length d corresponding to some set $\mathcal{F}$ of d features. Using a tensor-product map of the form in Eq. (2.20), we can map the set of feature vectors $\{\vec{x}\}$ to a set $\mathcal{X}$ of orthogonal rank-one tensors $X \in \mathcal{X}$, generating a new distribution $p(X)$ from $p(\vec{x})$. The overlap of two tensors $X^{(j)}$ and $X^{(k)}$ is determined by the scalar products of the local feature maps

$$\langle X^{(j)}, X^{(k)}\rangle = \prod_{i=1}^{d} \vec{h}^{(i)}(x_i^{(j)})\cdot\vec{h}^{(i)}(x_i^{(k)}) \quad =$$



$, \qquad (6.12)$

where each feature map is a function of only a single feature. For this analysis we require that the vectors in the image of each local feature map must form an orthonormal set, so that a pair of feature vectors $\vec{x}^{\,(i)}$ and $\vec{x}^{\,(j)}$ will always be mapped to either the same tensor or to a pair of orthogonal tensors. For continuous features, such a mapping can be achieved by discretizing the real numbers into b bins, and then assigning values in each bin to a different b-dimensional basis vector. The $\vec{h}^{(i)}$ for this mapping will never be one-to-one, although as the dimensionality of their outputs grows the functions will come closer to being injective in practice.

Assuming that the images of the local feature maps are finite-dimensional, $\mathcal{X}$ will be finite and therefore $p(X)$ will be a discrete distribution that can be represented as a tensor W of the form

$$W = \sum_{X \in \mathcal{X}} \sqrt{p(X)} X, \tag{6.13}$$

where we have taken the square-root to ensure that W is normalized (i.e. $\langle W, W \rangle = 1$). With this representation, the probability of a given tensor X can be extracted by taking the square of its scalar product with W

$$p(X) = |\langle X, W \rangle|^2. \tag{6.14}$$

In the context of machine learning, W can be described as an idealized weight tensor which we seek to model using a tensor network. For a given network, we want to know which W, and therefore which $p(X)$, can be accurately represented.

To probe the correlations within $p(X)$, we partition the features into disjoint sets $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \cap \mathcal{B} = \emptyset$ and $\mathcal{A} \cup \mathcal{B} = \mathcal{F}$. Using this grouping, the underlying feature distribution $p(\vec{x})$ can be represented as the joint distribution $p(\vec{x}_{\mathcal{A}}, \vec{x}_{\mathcal{B}})$, where $\vec{x}_{\mathcal{A}}$ and $\vec{x}_{\mathcal{B}}$ are vectors containing values for the features in partitions $\mathcal{A}$ and $\mathcal{B}$ respectively. Similarly, $p(X)$ can be represented as the joint distribution $p(X_{\mathcal{A}}, X_{\mathcal{B}})$, where $\mathcal{X}_{\mathcal{A}} \ni X_{\mathcal{A}}$ and $\mathcal{X}_{\mathcal{B}} \ni X_{\mathcal{B}}$ are sets of orthogonal tensors created from the local maps of features in $\mathcal{A}$ and $\mathcal{B}$ respectively. For any tensor $X \in \mathcal{X}$, we have $X = X_{\mathcal{A}} \otimes X_{\mathcal{B}}$ for some $X_{\mathcal{A}}$ and $X_{\mathcal{B}}$. We can also define the marginal distributions $p(X_{\mathcal{A}})$ and $p(X_{\mathcal{B}})$ that describe the statistics within each partition separately. The MI $\mathrm{I}(X_{\mathcal{A}} : X_{\mathcal{B}})$ across the bipartition is given as in Eq. (6.9) using the entropies of these distributions.

To introduce the entanglement measure described in Sec. 6.2.1 as a bound on $\mathrm{I}(X_{\mathcal{A}} : X_{\mathcal{B}})$, we represent the normalized tensor W as the quantum state $|\psi_W\rangle$ and the tensors in $\mathcal{X}$ as orthonormal basis states $|X_{\mathcal{A}}, X_{\mathcal{B}}\rangle$, such that Eq. (6.13) becomes

$$|\psi_W\rangle = \sum_{\mathcal{X}_{\mathcal{A}}, \mathcal{X}_{\mathcal{B}}} \sqrt{p(X_{\mathcal{A}}, X_{\mathcal{B}})}\, |X_{\mathcal{A}}, X_{\mathcal{B}}\rangle\,, \tag{6.15}$$

where we have shifted to braket notation (see Sec. 5.2.1). This encoding of a probability distribution into a quantum state has been utilized previously in the study of quantum Bayesian algorithms [109]. The process of extracting $p(X_{\mathcal{A}}, X_{\mathcal{B}})$ described in Eq. (6.14) can be reimagined as projective measurements of $|\psi_W\rangle$ on an orthonormal basis, where the probabilities are used to reconstruct $p(X_{\mathcal{A}}, X_{\mathcal{B}})$. Since the MI between outcomes of local measurements on a quantum state is upper bounded by the entanglement of that state [110],

$|\psi_W\rangle$ must have a bipartite entanglement with respect to partitions $\mathcal{A}, \mathcal{B}$ that is at least as large as $\mathrm{I}(X_\mathcal{A} : X_\mathcal{B})$. The MI of $p(X)$ across a bipartition therefore provides a lower bound on the amount of entanglement needed in $|\psi_W\rangle$ with respect to that same partition

$$\mathrm{I}(X_\mathcal{A} : X_\mathcal{B}) \leq \mathrm{E}(\mathcal{A}, \mathcal{B}), \tag{6.16}$$

which through Eq. (6.7) sets a lower bound on the degree of connectivity n and/or bond dimension t needed in the tensor network representing $|\psi_W\rangle$.

In a typical machine learning setting, we will have access to samples from $p(\vec{x})$, which can then be encoded into tensors which form samples from $p(X)$. If we aim to estimate the MI numerically, as we will in Secs. 6.3 - 6.5, then it is generally easier to work with the original feature vectors sampled from $p(\vec{x})$ than with the feature tensors from $p(X)$. From the data processing inequality [111], $\mathrm{I}(X_\mathcal{A}, X_\mathcal{B})$ is upper-bounded by $\mathrm{I}(\vec{x}_\mathcal{A}, \vec{x}_\mathcal{B})$, so using the MI of the original features will yield a bound on the entanglement that may be larger than necessary to model $p(X)$, but will always be sufficient. Indeed, as the dimension of the feature maps increases, the gap between $\mathrm{I}(X_\mathcal{A}, X_\mathcal{B})$ and $\mathrm{I}(\vec{x}_\mathcal{A}, \vec{x}_\mathcal{B})$ will shrink—since the finer discretization preserves more information—and thus the estimates from both featurizations will converge.

The methodology described above may appear somewhat circuitous, in that we start from the tensorized entanglement formalism that is most natural for tensor networks, but then move back to a classical MI description of the original data features. At first glance it seems like a more direct approach would be to simply estimate the entanglement of $|\psi_W\rangle$ between partitions $\mathcal{A}$ and $\mathcal{B}$ directly, using some approximation $|\tilde{\psi_W}\rangle$ constructed from the available data

$$|\tilde{\psi_W}\rangle \propto \sum_{i=1}^{N} |X_\mathcal{A}^{(i)}, X_\mathcal{B}^{(i)}\rangle\,, \tag{6.17}$$

where $\{|X_\mathcal{A}^{(i)}, X_\mathcal{B}^{(i)}\rangle\}$ is a set of η samples from $p(X_\mathcal{A}, X_\mathcal{B})$. Such a construction was recently used for entanglement analysis by Martyn et al. [112] in the context of MPS image classification. Unfortunately, as evident in [112], the entanglement of $|\tilde{\psi_W}\rangle$ is artificially upper-bounded by $\log(\eta)$, irrespective of the actual properties of $p(X_\mathcal{A}, X_\mathcal{B})$. This saturation occurs because, for generic sample tensors $|X^{(j)}\rangle$ and $|X^{(k)}\rangle$ with d features, we have

$$\langle X^{(j)}|X^{(k)}\rangle = \prod_{i=1}^{d} \vec{h}^{(i)}(x_i^{(j)}) \cdot \vec{h}^{(i)}(x_i^{(k)}) \approx c^d \tag{6.18}$$

for some typical local overlap $c < 1$. As the number of features grows, the overlap between data tensors is exponentially suppressed. When calculating the entanglement, the near-orthogonality of tensors within $\mathcal{X}_\mathcal{A}$ and $\mathcal{X}_\mathcal{B}$ (when partitions $\mathcal{A}$ and $\mathcal{B}$ are both moderately sized) causes the partial trace to generate an almost maximally mixed state with a von Neumann entropy of approximately $\log(\eta)$. In contrast, by moving back to the original vector space of the data and using MI rather than entanglement, we can generally avoid the $\log(\eta)$ upper bound (in Sec. 6.6 we discuss specific circumstances where this limit can also appear in MI estimation).

6.3 Estimating Mutual Information

6.3.1 Setup and Prior Work

For our analysis in Sec. 6.2 to be of practical use, we need a method of estimating the MI of a probability distribution using only a finite number of samples. More formally, let $\{\vec{x}^{(i)}\}_{i=1}^{\eta}$ be a set of samples drawn from a distribution $p(\vec{x})$ whose functional form we do not, in general, have access to. For a bipartition $\mathcal{A}, \mathcal{B}$ of the dataset features, our goal is to estimate the MI of $p(\vec{x}_\mathcal{A}, \vec{x}_\mathcal{B})$ between the features in $\mathcal{A}$ and the features in $\mathcal{B}$ using these samples.

Several approaches to MI estimation [113] have been proposed and explored in the literature. For continuous variables, some methods discretize the variable space into bins, and then compute a discrete entropy value based on the fraction of samples in each bin [114][115]. Alternatively, kernel density estimators [116] can be used to directly approximate the continuous probability density function using a normalized sum of window functions centered on each sample, which is then used to calculate the MI [117]. A method developed by Kraskov et al. [118], which utilizes a k-nearest neighbor algorithm to calculate the MI, has become popular due to its improved error cancellation when calculating the MI from approximated entropies.

For this paper, we base our estimation method on more recent work by Koeman and Heskes [119] and Belghazi et al. [120]. In [119], the MI estimation problem is recast as a binary classification task between samples from $p(\vec{x}_\mathcal{A}, \vec{x}_\mathcal{B})$ and $p(\vec{x}_\mathcal{A})p(\vec{x}_\mathcal{B})$, which the authors modeled using a random forest algorithm. In [120], Belghazi et al. use a neural network to perform unconstrained optimization on the *Donsker-Varadhan representation* (DV-representation) of the KL-divergence between $p(\vec{x}_\mathcal{A}, \vec{x}_\mathcal{B})$ and $p(\vec{x}_\mathcal{A})P(\vec{x}_\mathcal{B})$, which provides a lower-bound on the MI. In our work, we found that a mixture of these two approaches was most effective. Specifically, we have used the binary classification framing proposed in [119], but approached the problem as a logistic regression task optimized using maximum log-likelihood on a neural network. To evaluate the MI, we used the DV-representation as in [120] to generate a lower-bound when possible. In practice this also gave us smoother MI curves and smaller errors. To our knowledge this overall approach has not be reported in the literature, though it appears similar in concept to a method proposed by Pool et al. [121] in the context of generative adversarial networks. In the next subsection we describe our algorithm in more detail.

6.3.2 Logistic Regression for MI Estimation

The logistic regression approach to MI estimation is built around the KL-divergence definition of the MI introduced in Eq. (6.11). In the context of our dataset, the variable spaces $\mathcal{A}$ and $\mathcal{B}$ describe the collective values of the features in partitions $\mathcal{A}$ and $\mathcal{B}$ respectively, with the sums taken over all allowed value combinations. For convenience, we simplify our notation such that $a \equiv \vec{x}_\mathcal{A}$ and $b \equiv \vec{x}_\mathcal{B}$ represent the feature values of each partition. To estimate the MI using the KL-divergence, we require an approximation for $f(a,b) = \log \frac{p(a,b)}{p(a)p(b)}$. This can be found via logistic regression by first recasting the joint and marginal probability distributions as conditional probabilities

$$p(a,b|\text{joint}) \equiv p(a,b), \qquad p(a,b|\text{marg}) \equiv p(a)P(b), \tag{6.19}$$

where $p(a,b|\text{joint})$ is the probability that the feature values a,b will be sampled from the joint distribution $p(a,b)$, and $p(a,b|\text{marg})$ is the probability that the values will be sampled from the product-of-marginals distribution $p(a)p(b)$. Using Bayes' theorem, the conditional probabilities can be reversed

$$p(a,b|\text{joint}) \propto \frac{p(\text{joint}|a,b)}{p(\text{joint})}, \qquad p(a,b|\text{marg}) \propto \frac{p(\text{marg}|a,b)}{p(\text{marg})}. \tag{6.20}$$

Substituting Eq. (6.19) and Eq. (6.20) back into $\log \frac{p(a,b)}{p(a)p(b)}$ gives

$$\log \frac{p(a,b)}{p(a)p(b)} = \log \frac{p(\text{joint}|a,b)}{p(\text{marg}|a,b)} + \log \frac{p(\text{marg})}{p(\text{joint})}, \tag{6.21}$$

where the first term is the log-odds of a binary classification problem where samples are taken from either $p(a,b)$ or $p(a)p(b)$ and the classifier must decide the most likely source for a given set of feature values a and b. The second term will equal zero if each source is equally likely to be sampled.

To get a numerical estimate of Eq. (6.21), we can train a parameterized function $f(a,b;\theta)$ to estimate the log-odds via standard logistic regression methods

$$f(a,b;\theta) \approx \log \frac{p(\text{joint}|a,b)}{p(\text{marg}|a,b)}, \tag{6.22}$$

using a training set that consists of an equal number of joint samples and marginal samples. In particular, we parameterized f using a dense feed-forward neural network to avoid introducing spatial bias, and optimized the network by minimizing the binary cross-entropy (i.e. maximizing the log-likelihood) across the samples.

Since the joint distribution is the actual source of our dataset, we already have η samples from it. To approximate a sample from the product-of-marginals distribution, we take a set of values for the features in $\mathcal{A}$ from a joint sample chosen at random, and then take values for the features in $\mathcal{B}$ from another randomly-chosen joint sample (the two sources could be the same sample, although this is unlikely for a large dataset). After selection, the features are combined together into a single mixed sample which, by construction, has no correlations across the partition. After training the network, the MI could be estimated by taking the average of f across the joint samples as a direct approximation[1] of the KL-divergence from Eq. (6.11)

$$\mathrm{I}(a:b) \approx \frac{1}{\kappa}\sum_{i=1}^{\kappa} f(a_i,b_i;\theta), \tag{6.23}$$

[1] For κ samples of $p(a,b)$, we have $\lim_{\kappa\to\infty} \frac{1}{\kappa}\sum_{i=1}^{\kappa} f(a_i,b_i;\theta) = \sum_{a\in\mathcal{A},b\in\mathcal{B}} p(a,b)f(a,b;\theta)$.

where a_i and b_i are the feature values of the ith joint sample taken from a validation set of size κ. However, a superior approach is to insert f into the DV-representation [122] of the MI

$$\mathrm{I}(a:b) \geq \frac{1}{\kappa}\sum_{i=1}^{\kappa} f(a_i, b_i; \theta) - \log \frac{1}{\kappa^2}\sum_{i=1}^{\kappa}\sum_{j=1}^{\kappa} e^{f(a_i, b_j; \theta)}, \tag{6.24}$$

which yields a lower-bound on the MI as $\kappa \to \infty$ and allows errors to cancel[2]. The inequality is saturated when $f(a, b; \theta) = \log \frac{p(a,b)}{p(a)p(b)}$, since as $\kappa \to \infty$ the second term vanishes and the first term gives the KL-divergence. Belghazi et al. carried out their MI estimation by maximizing Eq. (6.24) itself, but we have found in practice that the second term often overflows on datasets with large MI. Furthermore, the optimization algorithm would often attempt to maximize the second term, even though it must vanish in the optimal solution. We were able to mitigate these problems by instead training with the binary cross-entropy [123] as a loss function and only using Eq. (6.24) at the end to get the MI value of the optimized distribution. As a caveat, we found in practice that for certain distributions with larger MI values Eq. (6.23) generally yielded more stable and accurate estimates than Eq. (6.24), though the reason for this is not clear.

6.4 Numerical Tests with Gaussian Fields

6.4.1 Gaussian Markov Random Fields

To test the accuracy of the logistic regression algorithm, we need a distribution to sample from that has an analytic expression for the MI and that can model different MI scaling patterns. Both of these requirements are satisfied by Gaussian Markov random fields (GMRFs) [124], which are multivariate Gaussian distributions parameterized by the *precision matrix* $Q \equiv \Sigma^{-1}$, where Σ is the more familiar covariance matrix. With respect to Q, the Gaussian distribution with mean $\vec{\mu}$ is

$$p(\vec{x}) = \sqrt{\det(\frac{1}{2\pi}Q)}\,\exp[-\frac{1}{2}(\vec{x}-\vec{\mu})^T Q(\vec{x}-\vec{\mu})], \tag{6.25}$$

where $p(\vec{x})$ is the probability density of the variables $\vec{x}$. The element Q_{ij} of the precision matrix determines the *conditional correlation* between variables x_i and x_j, which describes the statistical dependence of the pair when all other variables are held fixed at known values. This is in contrast with the more familiar *marginal correlation*, governed by Σ, which describes the dependence between a pair of variables when the state of all other variables is unknown. If $Q_{ij} = 0$, the variables x_i and x_j are conditionally uncorrelated:

$$p(x_i, x_j | x_{k\notin\{i,j\}}) = p(x_i | x_{k\notin\{i,j\}})p(x_j | x_{k\notin\{i,j\}}) \iff Q_{ij} = 0. \tag{6.26}$$

By setting specific elements of the precision matrix to zero, the correlation structure and therefore the MI of the Gaussian can be tuned to a desired pattern. This flexibility allows

[2] For example, errors of the form $f(a, b; \theta) + \epsilon$ can be brought outside of the sums in Eq. (6.23) to give $\epsilon - \log e^{\epsilon}$, which cancels.

us to encode different MI scaling patterns into the distribution, which can then be extracted analytically using Eq. (6.9) and the expression for Gaussian entropy

$$\mathrm{S} = \frac{1}{2}\log[\det(2\pi e\Sigma)], \tag{6.27}$$

which combine together to give an expression for the Gaussian MI:

$$\mathrm{I}(a:b) = \mathrm{S}(a) + \mathrm{S}(b) - \mathrm{S}(a,b) = \frac{1}{2}\log\frac{\det(\Sigma_{\mathcal{A}})\det(\Sigma_{\mathcal{B}})}{\det(\Sigma)}, \tag{6.28}$$

where $\Sigma_{\mathcal{A}}$ and $\Sigma_{\mathcal{B}}$ are the covariance matrices corresponding to variables in partitions $\mathcal{A}$ and $\mathcal{B}$ respectively.

6.4.2 Test Setup

In the following subsections, we present test results of the logistic regression estimator on GMRFs representing three different correlation patterns: a boundary law with nearest-neighbor correlations, a volume law with weak correlations across all variables, and a distribution with sparse, randomized correlations. In the language of quantum many-body physics, the first two patterns reflect correlation structures that would be expected in ground states and random states respectively, while the GMRF with random sparse correlations shows the scaling for a heterogeneous distribution that lacks any spatial structure. These Gaussian distributions serve as both a means of testing the algorithm and as a clear illustration of the numerical MI plots that would be expected from different types of correlations within a dataset.

In the tests, each GMRF consisted of 784 variables, which mirrored the number of pixels in the 28 × 28 images taken from the MNIST and Tiny Images datasets that were analyzed in Sec. 6.5. To measure the scaling behavior of the MI in these GMRFs, we used a range of different bipartition sizes, with the partitions being selected such that they always formed a pair of contiguous patches when the variables were arranged in a 28 × 28 array. One member of each bipartition was formed from an inner square patch of variables centered on the array, whose side length we denote as ℓ. The other partition was an outer patch consisting of all other variables. The size of the inner partition ranged from a single variable ($\ell = 1$) to a 26 × 26 block ($\ell = 26$). For each bipartition, the MI was estimated using our logistic regression algorithm and the DV-representation, with the estimates plotted alongside the analytic MI curve of the GMRF to evaluate their quantitative and qualitative accuracy. Since our model used a stochastic gradient descent method for optimization, we averaged over multiple training runs to generate a representative curve. To explore the effect of sample size on the algorithm, we generated datasets from the GMRFs with 70,000, 700,000, and 7,000,000 joint training samples and created MI curves for each size using averages over 20, 10, and 5 trials respectively. Samples and covariance plots from the GMRF test distributions are given in Sec. 6.7.1.

6.4.3 Nearest-Neighbor Boundary-Law GMRF

As shown in Eq. (6.8), for the MI of a bipartition to obey a boundary law its magnitude must scale with the length of the boundary or interface between the partitions. Given a set of variables on a d-dimensional lattice, the simplest way to construct a boundary law is to have each variable be conditionally-correlated with only its $2d$ nearest neighbors. For variable x_{ij} on a two-dimensional grid at row i and column j, the conditional probability function would depend on the values of only four other variables

$$p(x_{ij}|\{x_{k\neq i,l\neq j}\}) = p(x_{ij}|x_{i+1,j}, x_{i-1,j}, x_{i,j+1}, x_{i,j-1}), \tag{6.29}$$

although the number of neighbors can be fewer if the variable is at an edge or corner since the grid is finite. After partitioning, the inner patch of variables will be conditionally correlated with only a single layer of variables surrounding its perimeter, so the MI between the inner and outer partitions will be proportional to ℓ. To encode the correlation structure of Eq. (6.29) into a precision matrix, all of the off-diagonal elements in each row of Q must be set to zero except those that correspond to the nearest neighbors, with the non-zero off-diagonal elements all assigned the same value q that determines the strength of the correlation. To guarantee that Q is positive definite, q should not exceed the magnitude of the diagonal elements divided by the number of nearest neighbors (see Sec. 6.4.4 for more details on these constraints).

The performance of the logistic regression algorithm on the nearest-neighbor GMRF is summarized in Figure 6.1, which plots the MI in *nats* against the side length ℓ of the square inner partition[3]. Since the x-axis is proportional to the perimeter of the inner patch rather than its area, we expect a boundary-law MI curve to be linear in ℓ. This is clearly evident in the analytic curve, which is linear up to a length of roughly 25 variables before leveling off. The linear pattern is broken near the boundaries because the marginal correlations between variables around the edges of the grid are smaller than those between variables closer to the center. Aside from the the 70,000 sample trial with weak correlations, the regression estimates were able to successfully reproduce the boundary-law scaling pattern, with the error shrinking as the number of samples increased. It is also interesting to note that the fractional errors of the different sample sizes are similar between the strong and weak correlations, suggesting that the source of the error is independent of the MI magnitude.

6.4.4 Uniform Volume Law GMRF

In contrast with the local correlations that give rise to a boundary law, we can imagine an alternative pattern in which each variable is equally correlated with every other variable. These correlations produce a volume law for the MI, since every variable in the inner partition must contribute equally to the correlations with the outer partition. To encode such a pattern into a GMRF, we set every off-diagonal element of the precision matrix Q to the same value q. To ensure that the precision matrix remains positive definite, the value q should be small

[3] When calculating quantities such as the entropy or MI using natural logarithms, the unit of information is a *nat* instead of a *bit*.

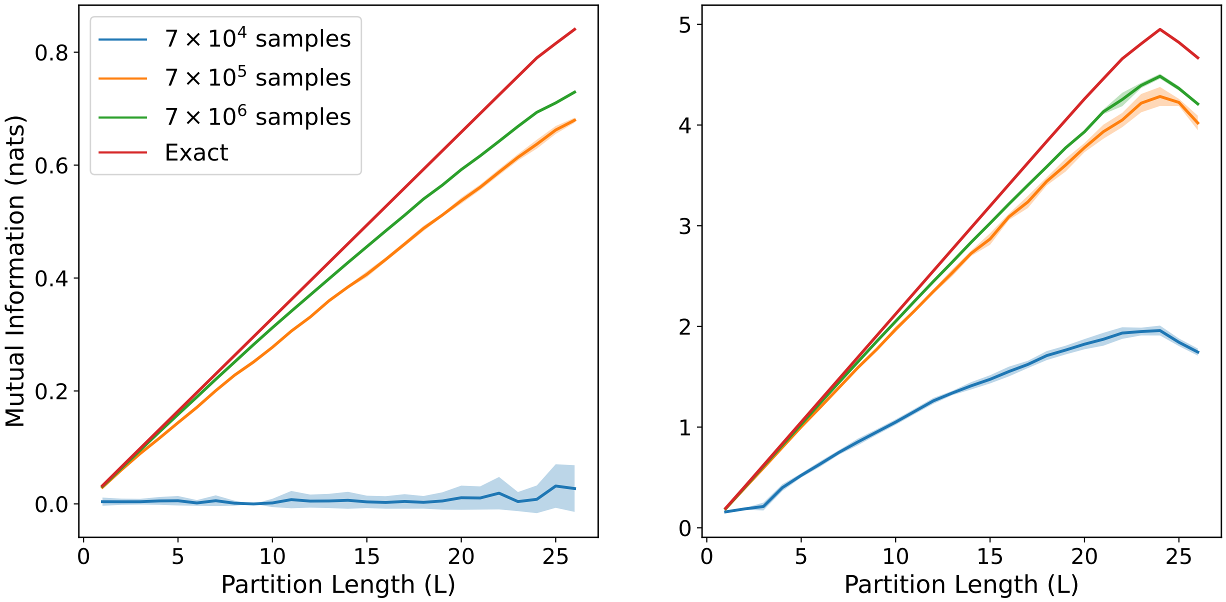


Figure 6.1: MI curves for a GMRF with nearest-neighbor correlations at different sample sizes, plotted relative to the side length ℓ of the inner partition. The plot on the left had a weaker correlation strength with $q = -0.12$, while the plot on the right had a stronger correlation strength with $q = -0.227$. The solid lines represent the averages over the trials, while the shaded regions show one standard deviation. The linear boundary-law scaling pattern of the GMRF is evident from the exact curve (red). With the exception of the weakly-correlated, 70,000 sample trial, this linear scaling is successfully reconstructed by the algorithm. The magnitude of the MI is underestimated in all trials, with the fractional error being similar for the strong and weak correlations.

enough to preserve *diagonal dominance*, a sufficient but not strictly necessary condition for a positive definite matrix in which the sum of the magnitudes of the off-diagonal elements of a row or column do not exceed the diagonal element

$$Q_{ii} > \sum_{i \neq j} |Q_{ij}| \text{ and } \; Q_{ii} > \sum_{j \neq i} |Q_{ij}|. \tag{6.30}$$

To create a uniform scaling pattern it suffices to set $Q_{ii} = 1$, which means we must have $q < \frac{1}{r-1}$ for an r-dimensional Gaussian. This provides an upper limit on the amount of correlation one Gaussian variable and can have with any other when the correlations are homogeneous, a limit that decreases as the number of variables grows larger.

The performance of our algorithm on a GMRF with these uniform correlations is summarized in Figure 6.2. We were able to accurately reproduce the shape and approximate magnitude of the analytic curves for both correlation strengths and for all sample sizes, although as expected the 70,000 sample trials had the largest error. Interestingly, the algorithm performed significantly better on the uniform GMRF than on the nearest-neighbor GMRF, even though the pairwise dependence between correlated variables in the former was much weaker than in the latter. This suggests that, for a given amount of MI, it is easier for the algorithm to find correlations that are spread out across many variables than to identify those that are concentrated in some sparse set.

It is worth noting that the shape of a volume-law curve should be quadratic on the axes used in Figure 6.2, yet from our plots it is clear that the quadratic form breaks down quickly for the weak correlations and never exists at all for the strong correlations. This distortion occurs because the MI is purely a function of the number of variables in each partition when the correlations are homogeneous, and due to the finite size of our grid any increase in the size of the inner patch necessarily comes at the cost of the outer patch. Correspondingly, any increase in the MI that comes from growing the inner patch is partially offset by the correlations that are lost when shrinking the outer patch. On the 28×28 grid used in Figure 6.2, the MI begins to decline at partition length $\ell = 20$, which marks the point where both partitions contain roughly the same number of variables (400 vs 384) and where the amount of correlation is therefore maximized.

6.4.5 Random Sparse GMRF

A third class of GMRF to explore is one where the correlations have no inherent spatial pattern yet are also non-uniform. Such a distribution could, for example, represent a dataset of features that are correlated but lack the in-built sense of position or ordering necessary to unambiguously map them onto a lattice (e.g., demographic data). If we nevertheless insist on embedding these features into a grid, we can expect that for most arrangements the MI will scale either as a volume law or in some irregular pattern, depending on whether the features all have similar correlation strengths.

For our tests, we engineered a spatially-disordered GMRF by taking the nearest-neighbor precision matrix used in Sec. 6.4.3 and randomly permuting the variables around the grid. Under this scheme, each row and column of the precision matrix Q has four non-zero off-diagonal elements in random positions. While the conditional correlations of this new distri-

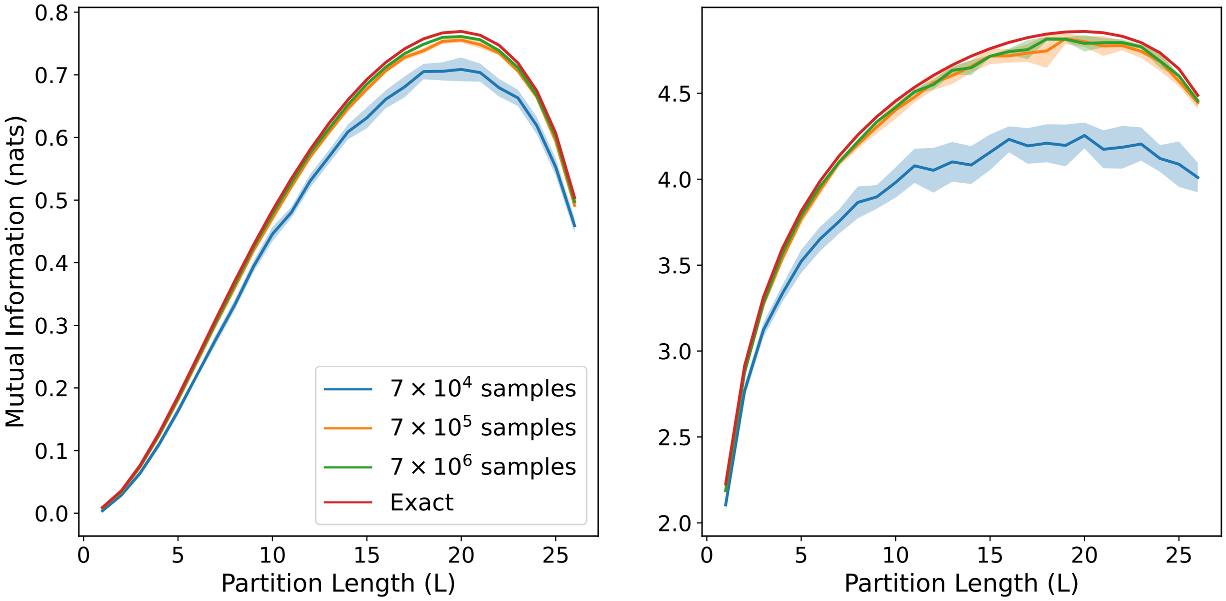


Figure 6.2: MI curves for a GMRF with uniform correlations at different sample sizes, plotted relative to the side length of the inner partition. The plot on the left had a weaker correlation strength with $q = -1.2 \times 10^{-3}$, while the plot on the right had a stronger correlation strength with $q = -1.27712 \times 10^{-3} \approx \frac{1}{783}$. The solid lines represent the averages over the trials, while the shaded regions show one standard deviation. The algorithm successfully reproduced the shape of the exact MI curve, with the larger sample sizes almost matching the analytic MI values. The finite size of the grid causes the curve to gradually bend over as the partition length increases.

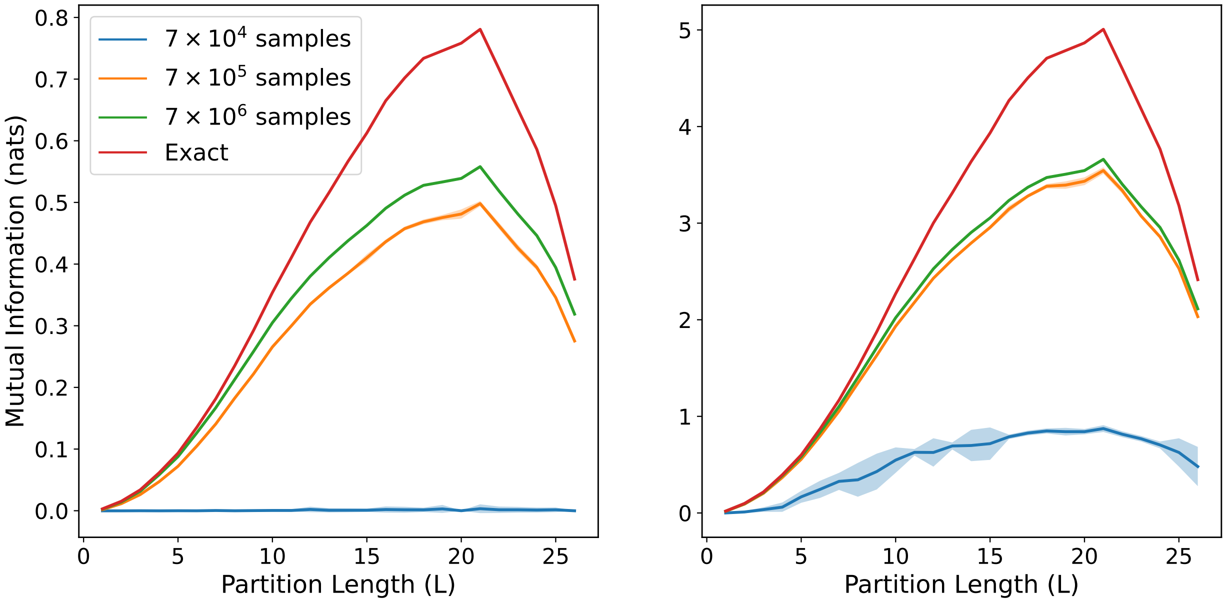


Figure 6.3: MI curves for a GMRF with spatially-randomized correlations at different sample sizes, plotted relative to the side length of the inner partition. The plot on the left had a weaker correlation strength with $q = -0.045$, while the plot on the right had a stronger correlation strength with $q = -0.11$. The solid lines represent the averages over the trials, while the shaded regions show one standard deviation. The quadratic, volume-law scaling is clear from the analytic MI curve at small ℓ, which for the strong correlations was reproduced across all sample sizes (though the 70,000 sample curve is greatly diminished). For the weak correlations the model was unable to find any correlations using the smallest sample size of 70,000, as was the case for the nearest-neighbor correlations (Figure 6.1, left panel).

bution are still sparse, they are no longer exclusively short-range but can instead span the entire grid. Since all of the non-zero off-diagonal elements of Q have the same magnitude q, the amount of correlation across any bipartition increases evenly with the number of correlated variable pairs shared between the partitions. Without any underlying spatial structure, the odds of a given pair being separated into two different partitions is roughly proportional to the volume of the smaller partition, assuming that the other partition is much larger. Under the inner-outer partitioning scheme used in our tests, we expect a volume law for small partition lengths, followed by the same bending-over observed in Sec. 6.4.4 for the uniform correlations.

The performance of our logistic regression algorithm on a GMRF with these spatially-randomized correlations is shown in Figure 6.3. As predicted, the analytic curves show similar scaling patterns to those of the uniform GMRF in Figure 6.2. The quality of the MI estimates, however, is more similar to the nearest-neighbor MI curves of Figure 6.1, where the model succeeded at replicating the analytic MI curve for all sample sizes when the correlation strength was large, but failed for the smallest sample size (70,000) when the correlations were weak. The estimation error is larger overall for the randomized variables than for the nearest-neighbor variables, and increasing the sample sizes appears to yield diminishing returns. This may partially stem from the reduction in pairwise correlation strength (quantified by q) that was required to keep the magnitude of the peak MI value consistent between the different GMRFs. However, a more likely explanation is that the nearest-neighbor correlations are able to reinforce one another due to their shared proximity, which results in the next-nearest-neighbor marginal correlations also being quite strong. This may make the correlations easier for a machine learning algorithm to detect, since they will impact a larger number of variables. In contrast, for the randomized GMRF the correlated variables are scattered far away from each other on average, which severely diminishes any reinforcement effect.

6.5 Application to Image Data

6.5.1 Setup

To explore the types of MI scaling patterns that might be seen in real data, we analyzed two sets of images: the 70,000 image MNIST handwritten-digits dataset [52], and 700,000 images taken from the Tiny Images dataset [54] converted to grayscale using a weighted luminance coding[4]. Sample images from these datasets and further details can be found in Sec. 2.3.1. These two datasets were chosen due to their differing levels of complexity: MNIST consists of simple, high-contrast shapes while the Tiny Images are low-resolution depictions of the real world with much more subtle color gradients. In our experiments, each image contained 784 pixels arranged in a 28×28 array, with the Tiny Images dataset being cropped from 32×32 by removing two pixels from each side. The pixel values, originally integers from 0 to 255, were rescaled to the range [0, 1].

To generate the MI estimates for these two datasets, we used the same partitioning method described in Sec. 6.4 for the GMRFs, with each image being split into a centered, square inner patch of increasing size and a surrounding outer patch. These partitions were then fed into the algorithm laid out in Sec. 6.3, with one key difference; the DV-representation of Eq. (6.24) proved to be unusable for both MNIST and the Tiny Images due to instability in the exponential term. While we were able to use the DV-representation to significantly reduce error on the GMRF tests, on the real datasets we had to instead make a direct estimate of the KL-divergence from Eq. (6.23). It is not clear why the DV-representation worked for the GMRFs but not for the image datasets, although this could be due to the larger MI and stronger correlations that are present in the real-world data.

6.5.2 Results

Figure 6.4 shows the MI of the MNIST and Tiny Images datasets as estimated by logistic regression, plotted relative to the side length ℓ of the inner pixel partition. The MI curves were generated from averages taken over twenty different trials, and plotted within a shaded

[4] For normalized RGB color values, each grayscale pixel was assigned the value 0.3R + 0.59G + 0.11B. See [125] for more information on grayscale conversions.

region containing one standard deviation. As with the GMRFs, this averaging helped smooth the curves and make their shapes easier to assess, especially for patch sizes with larger variance.

Looking first at the Tiny Images curve, we can see a moderately linear segment from 1 pixel length to roughly 18 pixels length, which then flattens out and begins to decrease at the 26 × 26 patch. Of the three scaling curves tested in Sec. 6.4, this overall shape is most consistent with the boundary-law scaling pattern of Sec. 6.4.3 (Figure 6.1, right panel). Unfortunately the variance of the algorithm increased significantly at larger MI values, making it more difficult to assess the pattern. For MNIST, the MI curve most closely resembles that of the strongly-correlated uniform GMRF (Figure 6.2), rising at a decreasing rate until it crests and gradually declines. However, this shape is not as distinct as that of a linear or quadratic curve, so it is difficult to use as evidence for a volume law.

Interestingly, the MNIST curve shows far less variance than the Tiny Images curve, despite the fact that it contains only a tenth of the images. For the GRMF tests done in Sec. 6.4, there was a clear reduction in the variance of each curve as the sample size increased, but this not observed in Figure 6.4. Indeed, the MNIST curve has a smaller variance at each patch size than the Tiny Images curve has at almost any patch size, even when the MI of the MNIST curve is larger. This suggests that there is some data-specific effect causing the discrepancy, perhaps attributable to the relative simplicity of the MNIST images relative to the more realistic Tiny Images.

Unlike in our GMRF tests, we do not have access to the underlying probability distributions that MNIST and the Tiny Images datasets were sampled from, so it is much more difficult to assess the accuracy of the curves in Figure 6.4. One approximate way of evaluating the estimates is to fit a GMRF to the empirical covariance matrix of the data, and then calculate the Gaussian MI analytically in the same manner as in Sec. 6.4. This new distribution is constrained to model only pairwise interactions between the variables, and all marginal and conditional distributions among the variables are forced to be Gaussian, so it is not representative of the true distribution. Nevertheless, due to its high entropy and simple correlation structure, a fitted GMRF is likely (but not guaranteed [126]) to provide a lower bound on the MI of the true distribution.

Figure 6.5 shows the Gaussian MI curves generated by fitting GMRFS to the covariance matrices of both the MNIST and Tiny Images datasets. It is important to note the scale of the y-axis: the MI values obtained from the fitted GMRFs are roughly five times larger than the predictions of the logistic regression algorithm that are shown in Figure 6.4, indicating a severe underestimation in the latter. The curve for the Tiny Images in Figure 6.5 is remarkably linear, only declining at the end because of the finite size of the image. This agrees with the shape of the logistic regression curve in Figure 6.4 and almost exactly resembles the boundary-law GMRF curve from Sec. 6.4.3 (Figure 6.1). The MNIST GMRF curve is also approximately linear up to an inner patch length of roughly $\ell = 15$ pixels, at which point the curve bends over and begins to decrease due to finite size effects; these are exacerbated by the fixed black border placed around each digit[5]. While the MNIST curves in Figures

[5] The MNIST digits themselves are only 20×20 pixels in size; since the digits are roughly centered in the 28×28 image, most of the outer pixels on the edges will be uniformly black and thus contribute nothing to

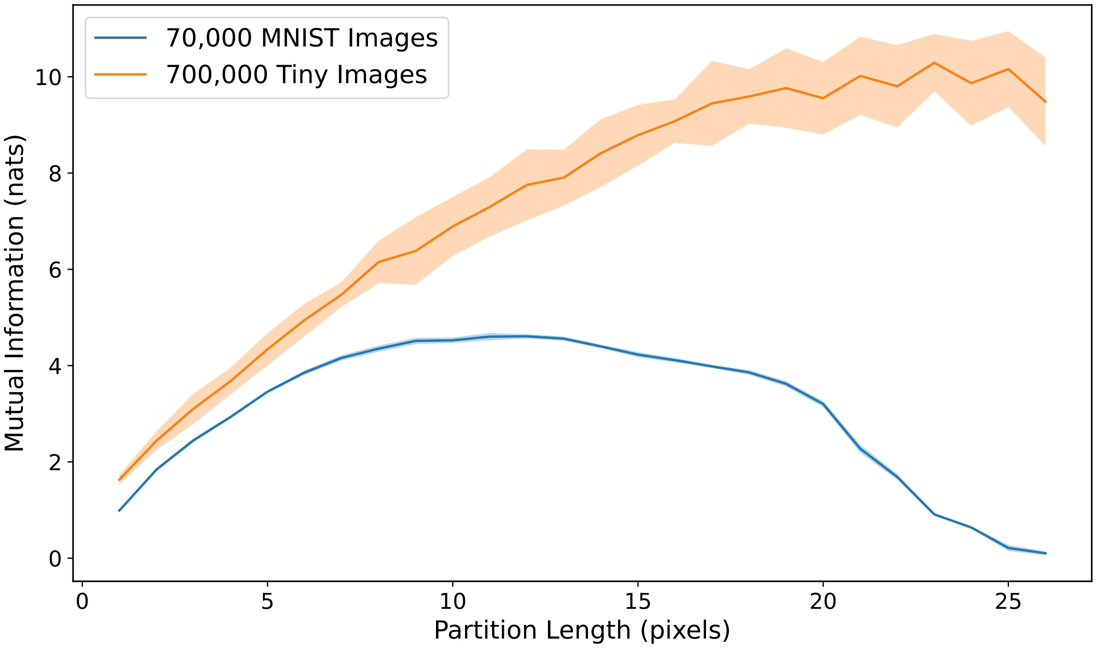


Figure 6.4: MI estimates for the MNIST and Tiny Images datasets using logistic regression, plotted relative to the side length ℓ of the inner partition. The solid lines are averages from twenty separate trials, while the shaded regions show one standard deviation. The MNIST curve most closely resembles the strongly-correlated uniform GMRF from Sec. 6.4.4, and exhibits minimal variance. The Tiny Images curve is most similar to the nearest-neighbor, boundary-law GMRF from Sec. 6.4.3 (Figure 6.1, right panel), but the shape is harder to pin down due to its high variance.

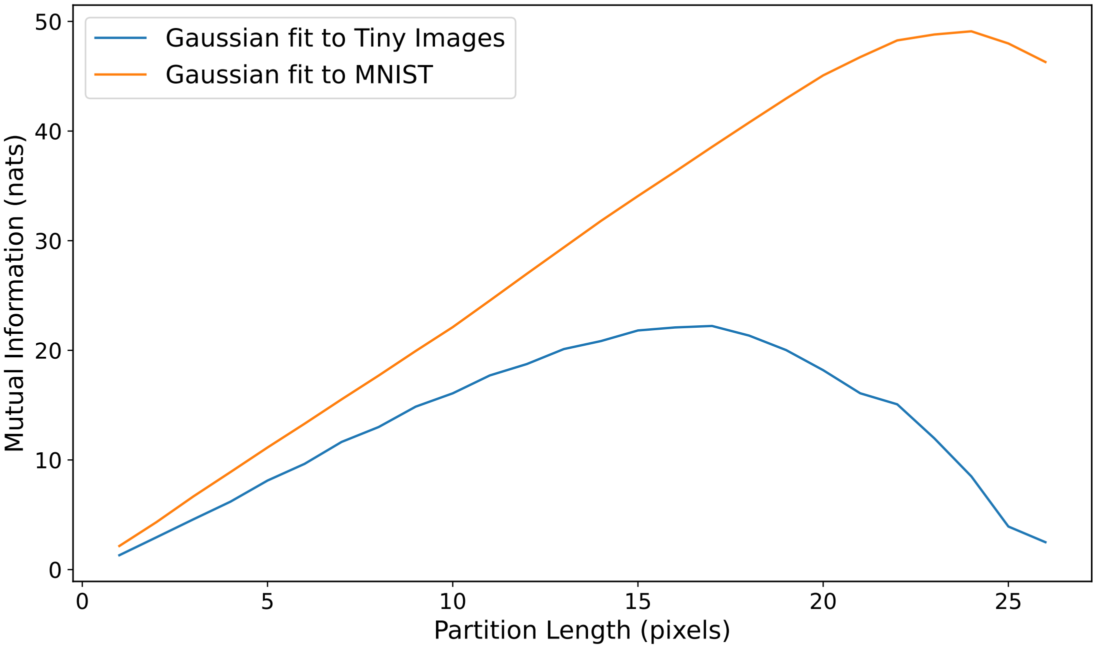


Figure 6.5: Analytic MI curves from the GMRFs fitted to MNIST and the Tiny Images, plotted relative to the side length ℓ of the inner partition. The Tiny Images curve shows a clear boundary law, while the MNIST curve also starts linear but gradually bends over. Since the GMRFs only model simple pairwise correlations, these MI values are very likely underestimates.

6.4 and 6.5 have somewhat similar shapes at larger patch sizes, the linearity of the Gaussian MNIST curve in Figure 6.5 at small ℓ is not present in the corresponding regression curve of Figure 6.4. Taken together, these results show that if the GMRF estimates are viewed as approximations of the simple, pairwise correlations in the images, then it is evident that the scaling behavior of those correlations obeys a clear boundary law in both datasets. Samples and covariance plots from the two fitted GMRFs are given in Figure 6.8.

Although the primary focus of this chapter is the use of logistic regression as a means of quantifying MI scaling, it is clear from Figure 6.5 that GMRF techniques offer a viable alternative. We provide here a brief discussion of the relative merits of each method. Compared to a stochastically-optimized neural network, a multivariate Gaussian is very simple to fit and provides a single, deterministic MI estimate via Eq. (6.28). The logistic regression algorithm, by contrast, shows significant variation across trials even when the dataset is fixed, a problem which becomes more severe at larger MI values (see, e.g., the Tiny Images plot in Figure 6.4). The simplicity of the GMRF comes at a cost, however, since Gaussians are inherently quadratic and thus incapable of modeling interactions between more than two variables. We would expect complex datasets to posses these higher-order dependencies, which favors the use of more expressive neural network models. At the same time, we can see from comparing Figure 6.4 with Figure 6.5 that the logistic regression method captures only a fraction of the total magnitude of the MI. Collectively, these observations suggest that the GMRF approach should be favored when the correlation patterns are simple or when only a rough lower-bound on the MI of a dataset is desired. By contrast, regression with a neural network is better suited to estimate the MI of data with more complex correlations.

6.6 Discussion

Recent work in quantum many-body physics has shown that the success of a tensor network ansatz is closely tied to the correlation structure of the underlying system. It stands to reason that similar logic should hold in machine learning. If true, this presents us with two main challenges. First, on a theoretical level, we must gain insight into the mathematical relationships that exist between dataset correlations and network architecture. At the same time, on a more practical level, we need to be able to quantify and characterize the kinds of correlation structures present in real-world data. Our work here addresses both of these problems, using the classical MI to establish an entanglement lower-bound for probabilistic classification tasks and finding clear evidence for boundary-law scaling in the Tiny Images dataset.

On the theoretical side, we established in Sec. 6.2.3 that the MI of the data features provides a lower bound on the entanglement needed for probabilistic classification of orthogonal samples by a tensor network. We showed that direct entanglement estimates, taken from the state representing the sample distribution, are artificially upper-bounded by the logarithm of the number of samples, regardless of the nature of the distribution. When the true entanglement is expected to exceed this bound, such as for data with a large number of features, a different measure of correlation such as the MI is therefore necessary. Given that

the MI.

the entanglement of a network with fixed bond dimension is $n \log t$ (Eq. (6.7)), an MI estimate can help determine both the connectivity of the network (n) and the size of the indices (t). While the lower bound should still hold approximately on samples with small overlaps, it will be useful to explore in future work whether and to what extent it is possible to generalize this bound to non-orthogonal featurizations. Additionally, there are many machine learning tasks where the ground truth cannot be expressed as a probability or modulus—e.g., regression over the real numbers $\mathbb{R}$—and which therefore fall outside of our analysis. It seems likely that the correlation structures in these tasks would still be important when choosing the right tensor network, but the mathematical relationship is not as clear as in the probabilistic cases studied here.

Assuming that the images analyzed in Sec. 6.5 can be mapped to tensors with minimal overlap and that therefore the bound in Sec. 6.2.3 applies, then our numerical results suggest that the MI of the Tiny Images obeys a boundary law. The evidence is less definitive for MNIST, although the analytic curve obtained by fitting a GRMF shows a clear boundary law for smaller patch sizes. This would indicate that the most appropriate tensor network to use for probabilistic classification of these datasets from a correlation standpoint is PEPS, whose connectivity follows a 2-D grid. However, given that exact contraction of a large PEPS network is impossible even with small bond dimension, it would be useful to look at alternative structures that still possess a 2-D geometry. Some possibilities include a TTN with four child nodes, or networks with a Cayley tree structure [127] possessing four nearest neighbors.

From a numerical perspective, our present work on MI estimation appears to be one of the few in the literature that seeks to quantify the spatial structure of the MI, or even just approximate the magnitude of the MI itself. Instead, most of the existing research focuses on MI as a minimization or maximization target, as seen in various independent component analysis algorithms [128] or in the training of generative models [129]. To our knowledge, the only other work that explores MI scaling is that of Cheng et al. [130], which characterized the MI of MNIST in the context of training sparse Boltzmann machines. The authors utilized side-to-side and checkerboard partitioning schemes, focusing their analysis on the degree to which the estimated MI value (using Kraskov's nearest-neighbor method) differed from the maximum MI value that could exist between the partitions. While their results showed that the estimate was significantly smaller than the maximum, it is unclear how much of this was actually an intrinsic property of the data or just a numerical limitation of the nearest-neighbor method used for estimation.

Indeed, recent work by McAllester and Stratos [131] has shown that lower-bound MI estimates based on sampling, such as our logistic regression algorithm using the DV representation, can never produce an estimate greater than $\mathcal{O}(\log \eta)$, where η is the number of samples. If we make the reasonable assumption that the Gaussian curves from Figure 6.5 underestimate the true MI, then we would need on the order of 10^{21} images to get a good estimate of the Tiny Images MI. This is of course impossible. For MNIST, the number of samples needed is on the order of 10^{8}, which is within the realm of possibility but would require a massive data collection and training scheme. On a practical level, this means that the DV representation cannot be used for MI estimation on datasets that have strong correlations, although it is unclear whether the $\log(\eta)$ bound tells us anything about direct

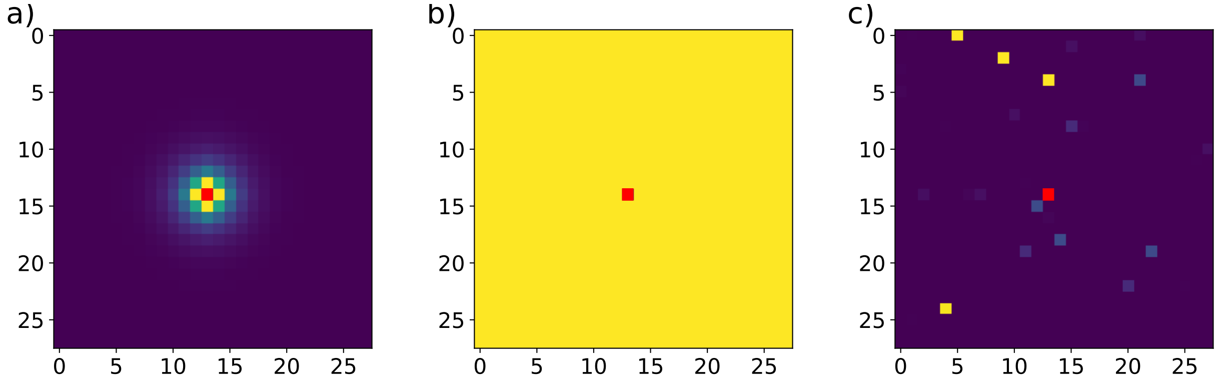


Figure 6.6: Covariance plots for the **a)** nearest-neighbor, **b)** uniform, and **c)** randomized GMRF distributions used in Sec. 6.4. The covariance values are taken with respect to the center variable highlighted in red, with brighter colors indicating stronger correlations and darker pixels indicating weaker correlations.

approximations of the KL divergence in the spirit of Eq. (6.23) (which was used to produce Figure 6.4). McAllester and Stratos recommend instead to minimize the cross-entropy as an upper bound on the entropy, then use Eq. (6.9) to get an estimate of the MI that is not a lower bound. This could be a useful direction for future work.

Tensor network machine learning is still in its infancy, and there is much work to be done in understanding the strengths and weaknesses of different network designs. It is likely that dataset correlations present in a given task will dictate the tensor structure that is best suited for the job, but determining which correlations are most important, and knowing how to assess that importance, is challenging. We have shown here that the scaling of the MI within a dataset can be systematically characterized in a manner that parallels the entanglement scaling analysis performed on quantum states, which may provide insight into these questions.

6.7 Appendix

6.7.1 GMRF Covariances and Sample Images

Figure 6.6 shows covariance plots of the three GMRFs tested in Sec. 6.4 with respect to a single variable highlighted in red. The magnitudes are expressed as colors to emphasize the importance of the correlation pattern rather than the specific covariance values. The variables that have the strongest covariance with the center variable are bright yellow, and correspond to the variables which have a non-zero conditional correlation with the center variable. In Figure 6.6a the four nearest-neighbor variables are clearly visible, while in Figure 6.6c those four variables are randomly distributed throughout the image. In Figure 6.6b the covariance matrix is uniformly yellow, as every variable is conditionally correlated with every other variable. Samples from these GRMFs are shown in Figure 6.7, where the subtly of the correlation effects is evident.

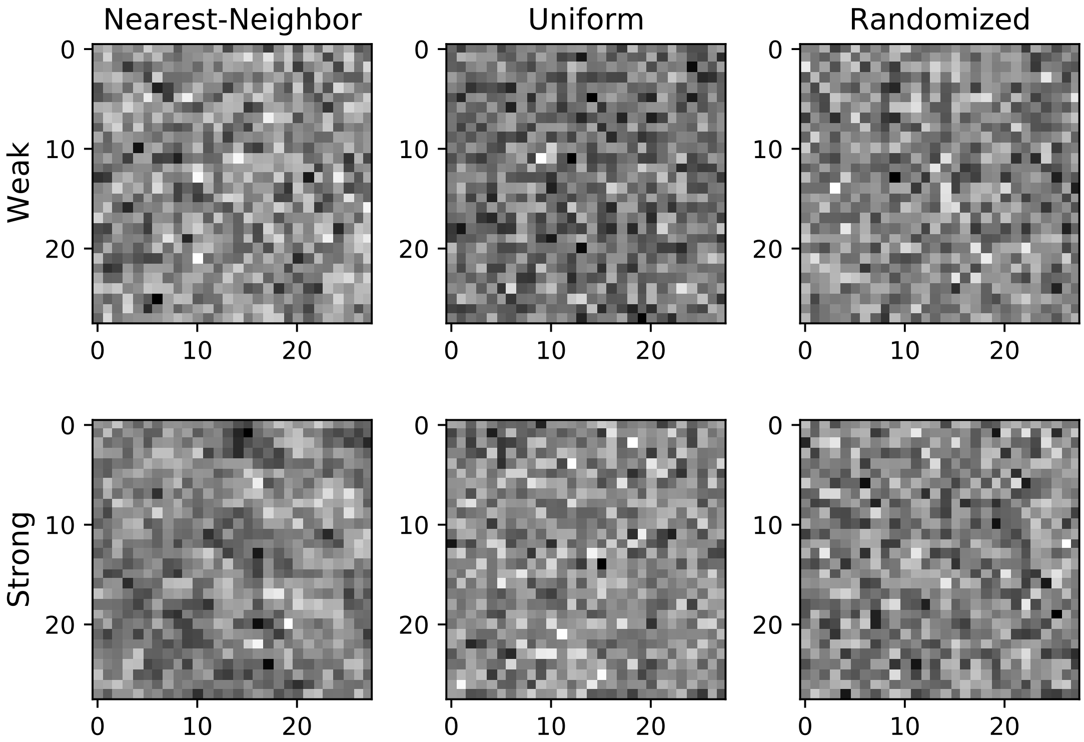


Figure 6.7: Sample "images" taken from the GRMF distributions of Sec. 6.4 at both strong and weak correlation strengths.

The covariance plots and sample images shown in Figure 6.8 are taken from the GRMFs fit to the Tiny Images and MNIST. The samples posses considerably more structure than those in Figure 6.7, which is consistent with the large MI values found in Figure 6.5. That said, the GRMFs are clearly not able to capture the full structure of the underlying dataset distributions, since the Tiny Images GMRF does not resemble any identifiable object and the MNIST GMRF sample does not resemble any digit. The covariance plots of Figure 6.8 both show strong nearest-neighbor correlations, which is consistent with the boundary-law scaling observed in Figure 6.5.

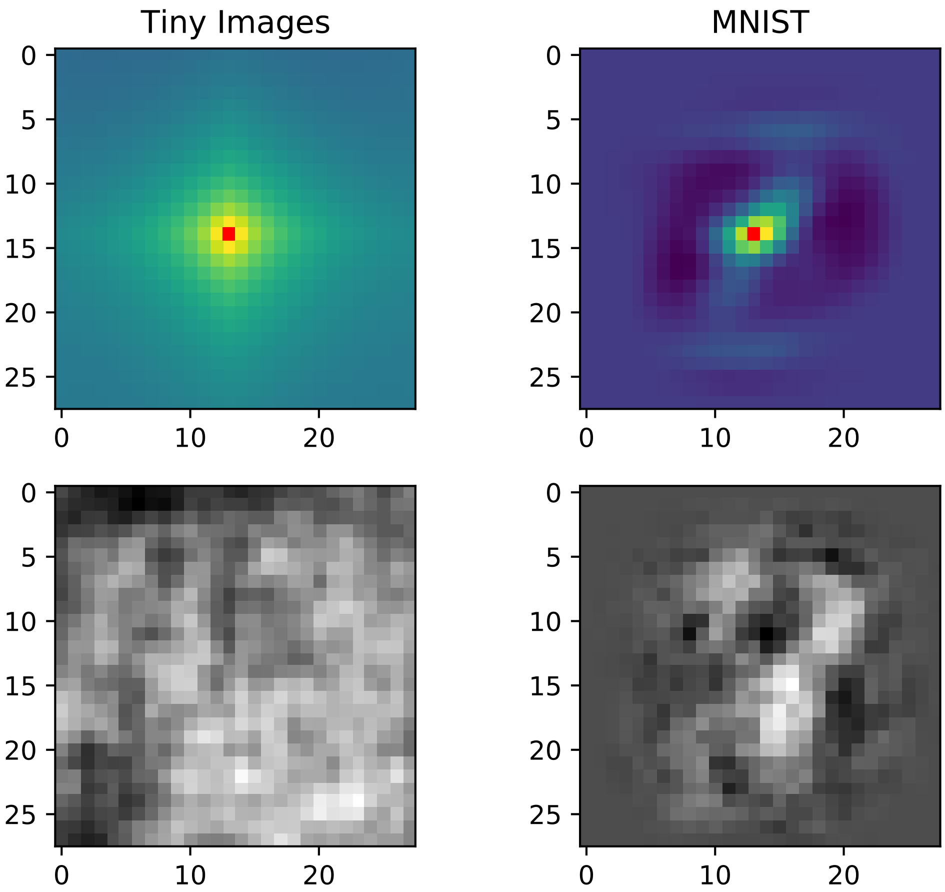


Figure 6.8: The covariances (top row) and sample images (bottom row) from GRMFs fit to the Tiny Images and MNIST datatsets. The covariance values are calculated with respect to the central pixel highlighted in red, with brighter colors indicating larger values. The Tiny Images covariance plot shows a strong nearest-neighbor pattern, while the MNIST plot has a more complicated and long-range structure. The sample images show some structure, but are not identifiable as a digit or object.

Chapter 7

Conclusions and Future Directions

The goal of this thesis was to probe the underlying properties of tensor network regression models, making use of tools from tensor analysis and quantum physics. Each work in Chapters 4-6 can be viewed as the first step in understanding a different facet of these machine learning algorithms, with much more yet to be understood. In this final chapter, we briefly summarize our findings and suggest potential directions for future work.

In Chapter 4, we asked whether the tensor-product feature space was truly "exponential" in practice, or if instead only small portions of it were being utilized by the tensor network models. To this end, we developed a novel form of tensor contraction called the interaction decomposition, and used it to split up the regression output from a classification model into separate contributions from each interaction degree. By analyzing the magnitudes and cumulative accuracies across the different degrees, we concluded that a large portion of the exponential space (roughly 75%) was indeed being utilized in a non-trivial manner by the model. However, this revelation was tempered by the fact that we were able to construct constrained models of degree ten or less that equaled and even outperformed the full models on MNIST and Fashion MNIST. This suggests that the tensor network models are not extracting anything useful from the higher interaction degrees that couldn't have been found in the lower degrees.

The burning question raised by these results is *why* the full tensor network models seem to do no better than their heavily-constrained counterparts. Is it simply because the higher-degree feature products contain little to no information useful for solving the classification task? We believe this to be unlikely, although it is difficult to provide any hard evidence to justify such skepticism. A more compelling explanation would be that the higher-degree and lower-degree regression coefficients are inherently coupled together via the elements of the component tensors, and therefore cannot be tuned independently of one another. It may be that the coefficient values necessary to fully utilize the low-degree regressors would lead to higher-degree coefficients that are deleterious to the output of the model, and thus a compromise has to be struck. An interesting next step would be to characterize the higher-degree coefficients that are generated from the constrained tensor network models, and see if the regression output is fatally corrupted by the inclusion of the higher-degree contributions.

Turning next to Chapter 5, we sought to understand the rank constraints placed upon a tensor network when it is used to represents arbitrary multilinear regression up to a given interaction degree. The specific quantity of interest was the multiplex rank, as this sets a lower bound on the combined bond dimension that the network must support between different feature partitions. After deriving a general algorithm for finding the multiplex rank as a function of feature number and partition size, we used it to compute the necessary bond dimensions for the virtual indices in an MPS model. Plots of these bond dimensions revealed some interesting patterns, while also driving home the massive number of network parameters that would be needed to represent the weight tensor for even modest interaction degrees. When looking for the primary force behind the large multiplex rank, we found that it was caused almost entirely by regressors which contained at least one feature from each partition. We also assessed how much the rank could be reduced if a low-degree weight tensors is embedded in a full-degree tensor, as is the case for the interaction decomposition models from Chapter 4, and found only a modest decrease.

For the work in this chapter, the most pertinent question is whether the rank upper-bounds or other properties can be translated into practical guidance for network design. Given a particular machine learning task, we would want to tailor the bond dimension and connectivity of the tensor network to match with the expected (or imposed) form of the weight tensor. To this end, our most significant finding is likely to be the large impact of inter-partition regressors, whose removal could massively shrink the size of the network. A useful direction for future work would then be to derive the necessary conditions for these regressors to vanish across some feature partition. With this knowledge, it would then be possible to optimize the position of the features in the network so as to minimize the bond dimension needed to represent the desired regression function.

Finally, we consider our work on correlation scaling from Chapter 6. Here, our focus shifted from analyzing properties of the weight tensor to analyzing properties of the underlying dataset. Inspired by the entanglement analysis used in quantum many-body physics, we looked for an analogous measure of correlation for classical features, and settled on the mutual information. To determine the size of the correlations, we developed a novel machine learning algorithm based on a dense feed-forward neural network that was capable of estimating mutual information using a finite number of samples. After testing the algorithm on Gaussian data with known correlations to verify its accuracy, we performed experiments on the MNIST and Tiny Images datasets. Our results indicated that both image sets seemed to obey an area law rather than a volume law in their correlation scaling, which mirrors the type of state that tensor networks are used to model in quantum physics.

As with our work in Chapter 5, the key question that we want answered is how to translate the correlation data into a set of guidelines for building tensor network models. We can be certain that some connection must exist here, given that the properties of the data ultimately determine the form of the weight tensor, which in turn constrains the form of the tensor network. A promising next step would be to derive analytic expressions for the elements of the weight tensor needed to minimize the mean-squared error, and then assess how the correlations are reflected in the resulting expressions. Alternatively, one could reverse our process here and instead generate data using a tensor network of some fixed design. The correlations in this synthetic dataset would then be estimated using our algorithm, with the resulting values being compared across different network designs and bond dimensions. While falling short of a mathematical proof, these comparisons could motivate useful heuristics for network design by matching the correlation scaling of the network with that of the dataset.

References

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. en. In: *Nature* 521.75537553 (May 2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. English. Ed. by Francis Bach. Cambridge, Massachusetts: The MIT Press, Nov. 2016. ISBN: 978-0-262-03561-3.

[3] Daniel Svozil, Vladimír Kvasnicka, and Jiri Pospichal. “Introduction to multi-layer feed-forward neural networks”. en. In: *Chemometrics and Intelligent Laboratory Systems* 39.1 (Nov. 1997), pp. 43–62. ISSN: 0169-7439. DOI: 10.1016/S0169-7439(97)00061-0.

[4] Yann LeCun and Yoshua Bengio. “Convolutional networks for images, speech, and time series”. In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[5] Zewen Li et al. “A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects”. In: *IEEE Transactions on Neural Networks and Learning Systems* 33.12 (Dec. 2022), pp. 6999–7019. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2021.3084827.

[6] Hojjat Salehinejad et al. “Recent Advances in Recurrent Neural Networks”. In: arXiv:1801.01078 (Feb. 2018). arXiv:1801.01078 [cs]. DOI: 10.48550/arXiv.1801.01078. URL: http://arxiv.org/abs/1801.01078.

[7] Kavi B. Obaid, Subhi Zeebaree, and Omar M. Ahmed. “Deep learning models based on image classification: a review”. In: *International Journal of Science and Business* 4.11 (2020), pp. 75–81.

[8] Shagun Sharma and Kalpna Guleria. “Deep Learning Models for Image Classification: Comparison and Applications”. In: *2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*. Apr. 2022, pp. 1733–1738. DOI: 10.1109/ICACITE53722.2022.9823516.

[9] Ali Bou Nassif et al. “Speech Recognition Using Deep Neural Networks: A Systematic Review”. In: *IEEE Access* 7 (2019), pp. 19143–19165. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2896880.

[10] Ceren Güzel Turhan and Hasan Sakir Bilge. “Recent Trends in Deep Generative Models: a Review”. In: *2018 3rd International Conference on Computer Science and Engineering (UBMK)*. Sept. 2018, pp. 574–579. DOI: 10.1109/UBMK.2018.8566353.

[11] Yu Zhang et al. “A Survey on Neural Network Interpretability”. In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 5.5 (Oct. 2021), pp. 726–742. ISSN: 2471-285X. DOI: 10.1109/TETCI.2021.3100641.

[12] Giuseppe Carleo et al. “Machine learning and the physical sciences”. In: *arXiv:1903.10563 [astro-ph, physics:cond-mat, physics:hep-th, physics:physics, physics:quant-ph]* (Mar. 2019). arXiv: 1903.10563. URL: http://arxiv.org/abs/1903.10563.

[13] Giacomo Torlai and Roger G. Melko. “Machine learning quantum states in the NISQ era”. In: *arXiv:1905.04312 [cond-mat, physics:quant-ph]* (May 2019). arXiv: 1905.04312. URL: http://arxiv.org/abs/1905.04312.

[14] Yoav Levine et al. “Bridging Many-Body Quantum Physics and Deep Learning via Tensor Networks”. en. In: *arXiv:1803.09780 [quant-ph]* (Mar. 2018). arXiv: 1803.09780. URL: http://arxiv.org/abs/1803.09780.

[15] C.R. Jesshope. “Computational physics and the need for parallelism”. en. In: *Computer Physics Communications* 41.2–3 (Aug. 1986), pp. 363–375. ISSN: 00104655. DOI: 10.1016/0010-4655(86)90075-5.

[16] Ari Harju et al. “Computational Physics on Graphics Processing Units”. en. In: *Applied Parallel and Scientific Computing*. Ed. by Pekka Manninen and Per Öster. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 3–26. ISBN: 978-3-642-36803-5. DOI: 10.1007/978-3-642-36803-5_1.

[17] Eric Cancès et al. “Computational quantum chemistry: A primer”. en. In: *Handbook of Numerical Analysis*. Vol. 10. Special Volume, Computational Chemistry. Elsevier, Jan. 2003, pp. 3–270. DOI: 10.1016/S1570-8659(03)10003-8. URL: https://www.sciencedirect.com/science/article/pii/S1570865903100038.

[18] Anders W. Sandvik. “Computational Studies of Quantum Spin Systems”. In: *AIP Conference Proceedings* 1297.1 (Nov. 2010), pp. 135–338. ISSN: 0094-243X. DOI: 10.1063/1.3518900.

[19] Kouichi Okunishi, Tomotoshi Nishino, and Hiroshi Ueda. “Developments in the Tensor Network — from Statistical Mechanics to Quantum Entanglement”. In: *Journal of the Physical Society of Japan* 91.6 (June 2022), p. 062001. ISSN: 0031-9015. DOI: 10.7566/JPSJ.91.062001.

[20] David W. Cohen. *An Introduction to Hilbert Space and Quantum Logic*. Problem Books in Mathematics. New York, NY: Springer, 1989. ISBN: 978-1-4613-8843-2. DOI: 10.1007/978-1-4613-8841-8. URL: http://link.springer.com/10.1007/978-1-4613-8841-8.

[21] Alexander Novikov, Mikhail Trofimov, and Ivan Oseledets. “Exponential Machines”. In: *arXiv:1605.03795 [cs, stat]* (May 2016). arXiv: 1605.03795. URL: http://arxiv.org/abs/1605.03795.

[22] Edwin Stoudenmire and David J Schwab. “Supervised Learning with Tensor Networks”. In: *Advances in Neural Information Processing Systems*. Vol. 29. Curran Associates, Inc., 2016.

[23] Zhongming Chen et al. “Parallelized Tensor Train Learning of Polynomial Classifiers”. In: *IEEE Transactions on Neural Networks and Learning Systems* 29.10 (Oct. 2018), pp. 4621–4632. ISSN: 2162-2388. DOI: `10.1109/TNNLS.2017.2771264`.

[24] Ivan Glasser, Nicola Pancotti, and J. Ignacio Cirac. “Supervised learning with generalized tensor networks”. In: *arXiv:1806.05964 [cond-mat, physics:quant-ph, stat]* (June 2018). arXiv: 1806.05964. URL: `http://arxiv.org/abs/1806.05964`.

[25] Edward Grant et al. “Hierarchical quantum classifiers”. en. In: *npj Quantum Information* 4.1 (Dec. 2018), pp. 1–8. ISSN: 2056-6387. DOI: `10.1038/s41534-018-0116-9`.

[26] Song Cheng, Lei Wang, and Pan Zhang. “Supervised Learning with Projected Entangled Pair States”. In: *arXiv:2009.09932 [cond-mat, physics:quant-ph, stat]* (Sept. 2020). arXiv: 2009.09932. URL: `http://arxiv.org/abs/2009.09932`.

[27] Jing Liu et al. “Tensor networks for unsupervised machine learning”. In: *Physical Review E* 107.1 (Jan. 2023), p. L012103. DOI: `10.1103/PhysRevE.107.L012103`.

[28] Ian Convy and K. Birgitta Whaley. “Interaction decompositions for tensor network regression”. en. In: *Machine Learning: Science and Technology* 3.4 (Dec. 2022), p. 045027. ISSN: 2632-2153. DOI: `10.1088/2632-2153/aca271`.

[29] Ian Convy et al. “Mutual information scaling for tensor network machine learning”. en. In: *Machine Learning: Science and Technology* 3.1 (Jan. 2022), p. 015017. ISSN: 2632-2153. DOI: `10.1088/2632-2153/ac44a9`.

[30] Ian Convy et al. “Machine learning for continuous quantum error correction on superconducting qubits”. en. In: *New Journal of Physics* 24.6 (June 2022), p. 063019. ISSN: 1367-2630. DOI: `10.1088/1367-2630/ac66f9`.

[31] Ian Convy and K. Birgitta Whaley. “A Logarithmic Bayesian Approach to Quantum Error Detection”. en-GB. In: *Quantum* 6 (Apr. 2022), p. 680. DOI: `10.22331/q-2022-04-04-680`.

[32] T. Kolda and B. Bader. “Tensor Decompositions and Applications”. In: *SIAM Review* 51.3 (Aug. 2009), pp. 455–500. ISSN: 0036-1445. DOI: `10.1137/07070111X`.

[33] W. Hackbusch. *Tensor Spaces and Numerical Tensor Calculus*. en. Springer series in computational mathematics. Heidelberg: Springer Verlag, 2012. ISBN: 978-3-642-28026-9.

[34] Thomas D. Howell. “Global properties of tensor rank”. en. In: *Linear Algebra and its Applications* 22 (Dec. 1978), pp. 9–23. ISSN: 00243795. DOI: `10.1016/0024-3795(78)90052-6`.

[35] Giorgio Ottaviani and Philipp Reichenbach. “Tensor Rank and Complexity”. en. In: arXiv:2004.01492 (July 2022). arXiv:2004.01492 [cs, math]. URL: `http://arxiv.org/abs/2004.01492`.

[36] Vin de Silva and Lek-Heng Lim. "Tensor Rank and the Ill-Posedness of the Best Low-Rank Approximation Problem". en. In: *SIAM Journal on Matrix Analysis and Applications* 30.3 (Jan. 2008), pp. 1084–1127. ISSN: 0895-4798, 1095-7162. DOI: 10.1137/06066518X.

[37] Richard Bellman. *Dynamic programming*. en. Princeton, NJ: Princeton Univ. Pr, 1984. ISBN: 978-0-691-07951-6.

[38] Amine Ammar and Francisco Chinesta. "Circumventing Curse of Dimensionality in the Solution of Highly Multidimensional Models Encountered in Quantum Mechanics Using Meshfree Finite Sums Decomposition". en. In: *Meshfree Methods for Partial Differential Equations IV*. Ed. by Michael Griebel and Marc Alexander Schweitzer. Lecture Notes in Computational Science and Engineering. Berlin, Heidelberg: Springer, 2008, pp. 1–17. ISBN: 978-3-540-79994-8. DOI: 10.1007/978-3-540-79994-8_1.

[39] Nico Vervliet et al. "Breaking the Curse of Dimensionality Using Decompositions of Incomplete Tensors: Tensor-based scientific computing in big data analysis". In: *IEEE Signal Processing Magazine* 31.5 (Sept. 2014), pp. 71–79. ISSN: 1558-0792. DOI: 10.1109/MSP.2014.2329429.

[40] M. V. Catalisano, A. V. Geramita, and A. Gimigliano. "Ranks of tensors, secant varieties of Segre varieties and fat points". en. In: *Linear Algebra and its Applications* 355.1 (Nov. 2002), pp. 263–285. ISSN: 0024-3795. DOI: 10.1016/S0024-3795(02)00352-X.

[41] Mari Carmen Bañuls. "Tensor Network Algorithms: A Route Map". en. In: *Annual Review of Condensed Matter Physics* 14.1 (Mar. 2023), annurev-conmatphys-040721–022705. ISSN: 1947-5454, 1947-5462. DOI: 10.1146/annurev-conmatphys-040721-022705.

[42] A. Cichocki et al. "Tensor Networks for Dimensionality Reduction and Large-Scale Optimizations. Part 2 Applications and Future Perspectives". en. In: *Foundations and Trends® in Machine Learning* 9.6 (2017). arXiv: 1708.09165, pp. 249–429. ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000067.

[43] Daniele Venturi. "The numerical approximation of nonlinear functionals and functional differential equations". en. In: *Physics Reports* 732 (Feb. 2018), pp. 1–102. ISSN: 03701573. DOI: 10.1016/j.physrep.2017.12.003.

[44] Steven P. Diaz and Adam Lutoborski. "Polynomial foldings and rank of tensors". In: *Journal of Commutative Algebra* 8.2 (Apr. 2016), pp. 173–206. ISSN: 1939-2346. DOI: 10.1216/JCA-2016-8-2-173.

[45] Peter D. Lax and Peter D. Lax. *Linear algebra and its applications*. en. 2nd ed. Pure and applied mathematics. Hoboken, N.J: Wiley-Interscience, 2007. ISBN: 978-0-471-75156-4.

[46] I. V. Oseledets and E. E. Tyrtyshnikov. “Breaking the Curse of Dimensionality, Or How to Use SVD in Many Dimensions”. en. In: *SIAM Journal on Scientific Computing* 31.5 (Jan. 2009), pp. 3744–3759. ISSN: 1064-8275, 1095-7197. DOI: 10.1137/090748330.

[47] W. Hackbusch and S. Kühn. “A New Scheme for the Tensor Representation”. en. In: *Journal of Fourier Analysis and Applications* 15.5 (Oct. 2009), pp. 706–722. ISSN: 1531-5851. DOI: 10.1007/s00041-009-9094-9.

[48] C. Hubig et al. “Strictly single-site DMRG algorithm with subspace expansion”. en. In: *Physical Review B* 91.15 (Apr. 2015), p. 155115. ISSN: 1098-0121, 1550-235X. DOI: 10.1103/PhysRevB.91.155115.

[49] I. V. Oseledets. “Tensor-Train Decomposition”. en. In: *SIAM Journal on Scientific Computing* 33.5 (Jan. 2011), pp. 2295–2317. ISSN: 1064-8275, 1095-7197. DOI: 10.1137/090752286.

[50] Lars Grasedyck. “Hierarchical singular value decomposition of tensors”. In: *SIAM Journal on Matrix Analysis and Applications* 31.4 (2010), pp. 2029–2054.

[51] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. en. 2nd ed. Springer Series in Statistics. New York: Springer-Verlag, 2009. ISBN: 978-0-387-84857-0.

[52] Yann LeCun, Corinna Cortes, and Chris Burges. *MNIST Handwritten Digit Database*. 1998. URL: http://yann.lecun.com/exdb/mnist/.

[53] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms”. In: arXiv:1708.07747 (Sept. 2017). arXiv:1708.07747 [cs, stat]. DOI: 10.48550/arXiv.1708.07747.

[54] A. Torralba, R. Fergus, and W.T. Freeman. “80 Million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.11 (Nov. 2008), pp. 1958–1970. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2008.128.

[55] Yipeng Liu et al. *Tensor Computation for Data Analysis*. en. Cham: Springer International Publishing, 2022. ISBN: 978-3-030-74385-7. DOI: 10.1007/978-3-030-74386-4.

[56] E. Miles Stoudenmire. “Learning relevant features of data with multi-scale tensor networks”. en. In: *Quantum Science and Technology* 3.3 (Apr. 2018), p. 034003. ISSN: 2058-9565. DOI: 10.1088/2058-9565/aaba1a.

[57] Stavros Efthymiou, Jack Hidary, and Stefan Leichenauer. “TensorNetwork for Machine Learning”. In: *arXiv:1906.06329 [cond-mat, physics:physics, stat]* (June 2019). arXiv: 1906.06329. URL: http://arxiv.org/abs/1906.06329.

[58] Ye-Ming Meng et al. “Residual Matrix Product State for Machine Learning”. In: *arXiv:2012.11841 [cond-mat, physics:quant-ph]* (Dec. 2021). arXiv: 2012.11841. URL: http://arxiv.org/abs/2012.11841.

[59] Tianxiang Gao and Vladimir Jojic. “Degrees of Freedom in Deep Neural Networks”. In: arXiv:1603.09260 (June 2016). arXiv:1603.09260 [cs, stat]. DOI: `10.48550/arXiv.1603.09260`. URL: `http://arxiv.org/abs/1603.09260`.

[60] J. Eisert. “Entanglement and tensor network states”. In: *arXiv:1308.3318 [cond-mat, physics:quant-ph]* (Sept. 2013). arXiv: 1308.3318. URL: `http://arxiv.org/abs/1308.3318`.

[61] Michele Dolfi et al. “Matrix product state applications for the ALPS project”. en. In: *Computer Physics Communications* 185.12 (Dec. 2014), pp. 3430–3440. ISSN: 0010-4655. DOI: `10.1016/j.cpc.2014.08.019`.

[62] A. Weichselbaum et al. “Variational matrix-product-state approach to quantum impurity models”. In: *Physical Review B* 80.16 (Oct. 2009), p. 165117. DOI: `10.1103/PhysRevB.80.165117`.

[63] Wen Wei Ho et al. “Periodic Orbits, Entanglement, and Quantum Many-Body Scars in Constrained Models: Matrix Product State Approach”. In: *Physical Review Letters* 122.4 (Jan. 2019), p. 040603. DOI: `10.1103/PhysRevLett.122.040603`.

[64] Ulrich Schollwöck. “The density-matrix renormalization group in the age of matrix product states”. In: *Annals of Physics*. January 2011 Special Issue 326.1 (Jan. 2011), pp. 96–192. ISSN: 0003-4916. DOI: `10.1016/j.aop.2010.09.012`.

[65] Dingheng Wang et al. “Compressing 3DCNNs based on tensor train decomposition”. en. In: *Neural Networks* 131 (Nov. 2020), pp. 215–230. ISSN: 0893-6080. DOI: `10.1016/j.neunet.2020.07.028`.

[66] Xiaokang Wang et al. “ADTT: A Highly Efficient Distributed Tensor-Train Decomposition Method for IIoT Big Data”. In: *IEEE Transactions on Industrial Informatics* 17.3 (Mar. 2021), pp. 1573–1582. ISSN: 1941-0050. DOI: `10.1109/TII.2020.2967768`.

[67] Longhao Yuan, Qibin Zhao, and Jianting Cao. “Completion of High Order Tensor Data with Missing Entries via Tensor-Train Decomposition”. en. In: *Neural Information Processing*. Ed. by Derong Liu et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 222–229. ISBN: 978-3-319-70087-8. DOI: `10.1007/978-3-319-70087-8_24`.

[68] Wenqi Wang, Vaneet Aggarwal, and Shuchin Aeron. “Tensor Completion by Alternating Minimization under the Tensor Train (TT) Model”. In: arXiv:1609.05587 (Sept. 2016). arXiv:1609.05587 [cs, math]. DOI: `10.48550/arXiv.1609.05587`. URL: `http://arxiv.org/abs/1609.05587`.

[69] Qibin Zhao et al. “Tensor Ring Decomposition”. en. In: *arXiv:1606.05535 [cs]* (June 2016). arXiv: 1606.05535. URL: `http://arxiv.org/abs/1606.05535`.

[70] Peter Pippan, Steven R. White, and Hans Gerd Evertz. “Efficient matrix-product state method for periodic boundary conditions”. en. In: *Physical Review B* 81.8 (Feb. 2010), p. 081103. ISSN: 1098-0121, 1550-235X. DOI: `10.1103/PhysRevB.81.081103`.

[71] J. Eisert, M. Cramer, and M. B. Plenio. "Area laws for the entanglement entropy - a review". In: *Reviews of Modern Physics* 82.1 (Feb. 2010). arXiv: 0808.3773, pp. 277–306. ISSN: 0034-6861, 1539-0756. DOI: 10.1103/RevModPhys.82.277.

[72] Vasily Pestun, John Terilla, and Yiannis Vlassopoulos. "Language as a matrix product state". In: *arXiv:1711.01416 [cond-mat, stat]* (Nov. 2017). arXiv: 1711.01416. URL: http://arxiv.org/abs/1711.01416.

[73] Yuhan Liu et al. "Entanglement-guided architectures of machine learning by quantum tensor network". In: *arXiv:1803.09111 [cond-mat, physics:quant-ph, stat]* (Mar. 2018). arXiv: 1803.09111. URL: http://arxiv.org/abs/1803.09111.

[74] Yaoyun Shi, Luming Duan, and Guifre Vidal. "Classical simulation of quantum many-body systems with a tree tensor network". In: *Physical Review A* 74.2 (Aug. 2006). arXiv: quant-ph/0511070, p. 022320. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.74.022320.

[75] V. Murg et al. "Simulating strongly correlated quantum systems with tree tensor networks". en. In: *Physical Review B* 82.20 (Nov. 2010), p. 205105. ISSN: 1098-0121, 1550-235X. DOI: 10.1103/PhysRevB.82.205105.

[76] Frank Verstraete and J. Ignacio Cirac. "Renormalization algorithms for quantum-many body systems in two and higher dimensions". In: *arXiv preprint cond-mat/0407066* (2004).

[77] F. Verstraete et al. "Criticality, the Area Law, and the Computational Power of Projected Entangled Pair States". en. In: *Physical Review Letters* 96.22 (June 2006), p. 220601. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.96.220601.

[78] Michael Lubasch, J. Ignacio Cirac, and Mari-Carmen Bañuls. "Algorithms for finite projected entangled pair states". en. In: *Physical Review B* 90.6 (Aug. 2014), p. 064425. ISSN: 1098-0121, 1550-235X. DOI: 10.1103/PhysRevB.90.064425.

[79] Kenta Ozeki and Tomoki Yamashita. "Spanning Trees: A Survey". en. In: *Graphs and Combinatorics* 27.1 (Jan. 2011), pp. 1–26. ISSN: 1435-5914. DOI: 10.1007/s00373-010-0973-2.

[80] Silvère Bonnabel. "Stochastic Gradient Descent on Riemannian Manifolds". In: *IEEE Transactions on Automatic Control* 58.9 (Sept. 2013), pp. 2217–2229. ISSN: 1558-2523. DOI: 10.1109/TAC.2013.2254619.

[81] Christian Lubich, Ivan V. Oseledets, and Bart Vandereycken. "Time Integration of Tensor Trains". In: *SIAM Journal on Numerical Analysis* 53.2 (2015), pp. 917–941. ISSN: 0036-1429.

[82] Sebastian Holtz, Thorsten Rohwedder, and Reinhold Schneider. "On manifolds of tensors of fixed TT-rank". en. In: *Numerische Mathematik* 120.4 (Apr. 2012), pp. 701–731. ISSN: 0945-3245. DOI: 10.1007/s00211-011-0419-7.

[83] Roman Orus. "A Practical Introduction to Tensor Networks: Matrix Product States and Projected Entangled Pair States". In: *Annals of Physics* 349 (Oct. 2014). arXiv: 1306.2164, pp. 117–158. ISSN: 00034916. DOI: 10.1016/j.aop.2014.06.013.

[84] Bart Vandereycken. “Low-Rank Matrix Completion by Riemannian Optimization”. In: *SIAM Journal on Optimization* 23.2 (Jan. 2013), pp. 1214–1236. ISSN: 1052-6234. DOI: 10.1137/110845768.

[85] Roeland Wiersema and Nathan Killoran. “Optimizing quantum circuits with Riemannian gradient flow”. In: arXiv:2202.06976 (May 2022). arXiv:2202.06976 [quant-ph]. DOI: 10.48550/arXiv.2202.06976. URL: http://arxiv.org/abs/2202.06976.

[86] Léon Bottou. “Large-Scale Machine Learning with Stochastic Gradient Descent”. en. In: *Proceedings of COMPSTAT’2010*. Ed. by Yves Lechevallier and Gilbert Saporta. Heidelberg: Physica-Verlag HD, 2010, pp. 177–186. ISBN: 978-3-7908-2604-3. DOI: 10.1007/978-3-7908-2604-3_16.

[87] Herbert Robbins and Sutton Monro. “A Stochastic Approximation Method”. In: *The Annals of Mathematical Statistics* 22.3 (Sept. 1951), pp. 400–407. ISSN: 0003-4851, 2168-8990. DOI: 10.1214/aoms/1177729586.

[88] Atilim Gunes Baydin et al. “Automatic differentiation in machine learning: a survey”. In: *Journal of Marchine Learning Research* 18 (2018), pp. 1–43.

[89] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. en. In: *arXiv:1412.6980 [cs]* (Jan. 2017). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

[90] Dumitru Erhan et al. “Why Does Unsupervised Pre-training Help Deep Learning?” en. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, Mar. 2010, pp. 201–208. URL: https://proceedings.mlr.press/v9/erhan10a.html.

[91] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. “A survey of transfer learning”. In: *Journal of Big Data* 3.1 (May 2016), p. 9. ISSN: 2196-1115. DOI: 10.1186/s40537-016-0043-6.

[92] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. en. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, Mar. 2010, pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

[93] J. R. Norris. *Markov Chains*. 1st ed. Cambridge University Press, Feb. 1997. ISBN: 978-0-521-48181-6. DOI: 10.1017/CBO9780511810633.

[94] Yiwei Chen, Yu Pan, and Daoyi Dong. “Residual Tensor Train: a Flexible and Efficient Approach for Learning Multiple Multilinear Correlations”. In: *arXiv:2108.08659 [cs]* (Aug. 2021). arXiv: 2108.08659. URL: http://arxiv.org/abs/2108.08659.

[95] Christian Szegedy et al. “Going deeper with convolutions”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA, USA: IEEE, June 2015, pp. 1–9. ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298594. URL: http://ieeexplore.ieee.org/document/7298594/.

[96] Suraj Srinivas, Akshayvarun Subramanya, and R. Venkatesh Babu. “Training Sparse Neural Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. July 2017, pp. 455–462. DOI: `10.1109/CVPRW.2017.61`.

[97] Baoyuan Liu et al. “Sparse Convolutional Neural Networks”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015, pp. 806–814. DOI: `10.1109/CVPR.2015.7298681`.

[98] Christopher J. Hillar and Lek-Heng Lim. “Most Tensor Problems Are NP-Hard”. In: *Journal of the ACM* 60.6 (Nov. 2013), 45:1–45:39. ISSN: 0004-5411. DOI: `10.1145/2512329`.

[99] G. Vidal. “A class of quantum many-body states that can be efficiently simulated”. In: *Physical Review Letters* 101.11 (Sept. 2008). arXiv: quant-ph/0610099, p. 110501. ISSN: 0031-9007, 1079-7114. DOI: `10.1103/PhysRevLett.101.110501`. URL: `http://arxiv.org/abs/quant-ph/0610099`.

[100] E. Schrödinger. “Discussion of Probability Relations between Separated Systems”. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 31.4 (Oct. 1935), pp. 555–563. ISSN: 0305-0041, 1469-8064. DOI: `10.1017/S0305004100013554`.

[101] Martin B. Plenio and S. Virmani. “An introduction to entanglement measures”. In: (Apr. 2005). URL: `https://arxiv.org/abs/quant-ph/0504163v3`.

[102] Artur Ekert and Peter L. Knight. “Entangled quantum systems and the Schmidt decomposition”. In: *American Journal of Physics* 63.5 (May 1995), pp. 415–423. ISSN: 0002-9505, 1943-2909. DOI: `10.1119/1.17904`.

[103] G. Evenbly and G. Vidal. “Tensor Network States and Geometry”. en. In: *Journal of Statistical Physics* 145.4 (Nov. 2011), pp. 891–918. ISSN: 0022-4715, 1572-9613. DOI: `10.1007/s10955-011-0237-4`.

[104] Don N. Page. “Average entropy of a subsystem”. In: *Physical Review Letters* 71.9 (Aug. 1993), pp. 1291–1294. ISSN: 0031-9007. DOI: `10.1103/PhysRevLett.71.1291`.

[105] M. B. Hastings. “An area law for one-dimensional quantum systems”. In: *Journal of Statistical Mechanics: Theory and Experiment* 2007.08 (Aug. 2007), P08024–P08024. ISSN: 1742-5468. DOI: `10.1088/1742-5468/2007/08/P08024`.

[106] M. Cramer et al. “Entanglement-area law for general bosonic harmonic lattice systems”. In: *Physical Review A* 73.1 (Jan. 2006), p. 012309. ISSN: 1050-2947, 1094-1622. DOI: `10.1103/PhysRevA.73.012309`.

[107] G. Vidal et al. “Entanglement in quantum critical phenomena”. In: *Physical Review Letters* 90.22 (June 2003). arXiv: quant-ph/0211074, p. 227902. ISSN: 0031-9007, 1079-7114. DOI: `10.1103/PhysRevLett.90.227902`.

[108] Nader Ebrahimi, Ehsan S. Soofi, and Refik Soyer. “Information Measures in Perspective: Information Measures in Perspective”. In: *International Statistical Review* 78.3 (Dec. 2010), pp. 383–412. ISSN: 03067734. DOI: `10.1111/j.1751-5823.2010.00105.x`.

[109] Guang Hao Low, Theodore J. Yoder, and Isaac L. Chuang. “Quantum inference on Bayesian networks”. In: *Physical Review A* 89.6 (June 2014), p. 062315. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.89.062315.

[110] Shengjun Wu, Uffe V. Poulsen, and Klaus Mølmer. “Correlations in local measurements on a quantum state, and complementarity as an explanation of nonclassicality”. In: *Physical Review A* 80.3 (Sept. 2009), p. 032319. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.80.032319.

[111] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. 6. print. Wiley series in telecommunications. Wiley, 1991. ISBN: 978-0-471-06259-2.

[112] John Martyn et al. “Entanglement and Tensor Networks for Supervised Image Classification”. In: *arXiv:2007.06082 [quant-ph, stat]* (July 2020). arXiv: 2007.06082. URL: http://arxiv.org/abs/2007.06082.

[113] Liam Paninski. “Estimation of Entropy and Mutual Information”. In: *Neural Computation* 15.6 (June 2003), pp. 1191–1253. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/089976603321780272.

[114] R. Moddemeijer. “On estimation of entropy and mutual information of continuous distributions”. In: *Signal Processing* 16.3 (Mar. 1989), pp. 233–248. ISSN: 0165-1684. DOI: 10.1016/0165-1684(89)90132-1.

[115] R. Steuer et al. “The mutual information: Detecting and evaluating dependencies between variables”. In: *Bioinformatics* 18.Suppl 2 (Oct. 2002), S231–S240. ISSN: 1367-4803, 1460-2059. DOI: 10.1093/bioinformatics/18.suppl_2.S231.

[116] V. A. Epanechnikov. “Non-Parametric Estimation of a Multivariate Probability Density”. In: *Theory of Probability & Its Applications* 14.1 (Jan. 1969), pp. 153–158. ISSN: 0040-585X. DOI: 10.1137/1114019.

[117] Young-Il Moon, Balaji Rajagopalan, and Upmanu Lall. “Estimation of mutual information using kernel density estimators”. In: *Physical Review E* 52.3 (Sept. 1995), pp. 2318–2321. ISSN: 1063-651X, 1095-3787. DOI: 10.1103/PhysRevE.52.2318.

[118] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. “Estimating mutual information”. In: *Physical Review E* 69.6 (June 2004), p. 066138. ISSN: 1539-3755, 1550-2376. DOI: 10.1103/PhysRevE.69.066138.

[119] Mike Koeman and Tom Heskes. “Mutual Information Estimation with Random Forests”. In: *Neural Information Processing*. Ed. by Chu Kiong Loo et al. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 524–531. ISBN: 978-3-319-12640-1. DOI: 10.1007/978-3-319-12640-1_63.

[120] Mohamed Ishmael Belghazi et al. “MINE: Mutual Information Neural Estimation”. In: *arXiv:1801.04062 [cs, stat]* (Jan. 2018). arXiv: 1801.04062. URL: http://arxiv.org/abs/1801.04062.

[121] Ben Poole et al. “On Variational Bounds of Mutual Information”. In: *International Conference on Machine Learning*. PMLR, May 2019, pp. 5171–5180. URL: http://proceedings.mlr.press/v97/poole19a.html.

[122] Avraham Ruderman et al. "Tighter variational representations of f-divergences via restriction to probability measures". In: *Proceedings of the 29th International Coference on International Conference on Machine Learning*. 2012, pp. 1155–1162.

[123] Daniel Ramos et al. "Deconstructing Cross-Entropy for Probabilistic Binary Classifiers". In: *Entropy* 20.33 (Mar. 2018), p. 208. DOI: 10.3390/e20030208.

[124] Havard Rue, Leonhard Held, and Leonhard Held. *Gaussian Markov Random Fields: Theory and Applications*. Chapman and Hall/CRC, Feb. 2005. ISBN: 978-0-429-20882-9. DOI: 10.1201/9780203492024. URL: https://www.taylorfrancis.com/books/9780429208829.

[125] Christopher Kanan and Garrison W. Cottrell. "Color-to-Grayscale: Does the Method Matter in Image Recognition?" In: *PLoS ONE* 7.1 (Jan. 2012). Ed. by Eshel Ben-Jacob, e29740. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0029740.

[126] Carlos A. L. Pires and Rui A. P. Perdigão. "Minimum Mutual Information and Non-Gaussianity Through the Maximum Entropy Method: Theory and Properties". In: *Entropy* 14.6 (June 2012), pp. 1103–1126. ISSN: 1099-4300. DOI: 10.3390/e14061103.

[127] Wei Li, Jan von Delft, and Tao Xiang. "Efficient simulation of infinite tree tensor network states on the Bethe lattice". In: *Physical Review B* 86.19 (Nov. 2012), p. 195137. ISSN: 1098-0121, 1550-235X. DOI: 10.1103/PhysRevB.86.195137.

[128] Wei Kong et al. "A review of independent component analysis application to microarray gene expression data". In: *BioTechniques* 45.5 (Nov. 2008), pp. 501–520. ISSN: 0736-6205, 1940-9818. DOI: 10.2144/000112950.

[129] Xi Chen et al. "InfoGAN: interpretable representation learning by information maximizing generative adversarial nets". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Curran Associates Inc., Dec. 2016, pp. 2180–2188. ISBN: 978-1-5108-3881-9.

[130] Song Cheng, Jing Chen, and Lei Wang. "Information Perspective to Probabilistic Modeling: Boltzmann Machines versus Born Machines". In: *Entropy* 20.8 (Aug. 2018), p. 583. ISSN: 1099-4300. DOI: 10.3390/e20080583.

[131] David McAllester and Karl Stratos. "Formal limitations on the measurement of mutual information". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 875–884.