IOP Publishing Mach. Learn.: Sci. Technol. 3 (2022) 045027 https://doi.org/10.1088/2632-2153/aca271

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
6 September 2022

REVISED
31 October 2022

ACCEPTED FOR PUBLICATION
14 November 2022

PUBLISHED
20 December 2022

PAPER

Interaction decompositions for tensor network regression

Ian Convy[1,2,*] and **K Birgitta Whaley**[1,2]

[1] Department of Chemistry, University of California, Berkeley, CA 94720, United States of America
[2] Berkeley Quantum Information and Computation Center, University of California, Berkeley, CA 94720, United States of America
[*] Author to whom any correspondence should be addressed.

E-mail: ian_convy@berkeley.edu

Keywords: tensor regression, interaction decomposition, tensor network machine learning

Abstract

It is well known that tensor network regression models operate on an exponentially large feature space, but questions remain as to how effectively they are able to utilize this space. Using a polynomial featurization, we propose an interaction decomposition as a tool that can assess the relative importance of different regressors as a function of their polynomial degree. We apply this decomposition to tensor ring and tree tensor network models trained on the MNIST and Fashion MNIST datasets, and find that up to 75% of interaction degrees are contributing meaningfully to these models. We also introduce a new type of tensor network model that is explicitly trained on only a small subset of interaction degrees, and find that these models are able to match or even outperform the full models using only a fraction of the exponential feature space. This suggests that standard tensor network models utilize their polynomial regressors in an inefficient manner, with the lower degree terms being vastly under-utilized.

1. Introduction

Tensor network regression has emerged as a promising and active area of machine learning research, having achieved impressive results on common benchmark tasks such as the Movie 100K [1], MNIST [2–5], and Fashion MNIST [3–5] datasets. The effectiveness of these models can be attributed to the tensor-product transformation that is applied to the data features, which maps the original feature vector into an exponentially large vector space. By performing linear operations on this expanded feature space, tensor network models are able to generate regression outputs that are highly *non-linear* functions of the original features.

In most tensor network models, the tensor-product transformation is constructed from a set of vector-valued functions that each act on only a single data feature. The form of these functions is important to the operation of the model, as it determines how regression on the transformed space is related to regression on the original feature space. Conventional wisdom regarding the choice of these functions can be traced back to the parallel works of Stoudenmire and Schwab [2] and Novikov *et al* [1], who each proposed a different transformation scheme. The method from [2] 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 [1], 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.

The purpose of this work is to quantitatively assess how well tensor network models are able to utilize the exponential feature space induced by their tensor-product transformations. We shall focus specifically on models which are built upon the $[1,\ x]$ featurization from Novikov *et al* [1], since this allows us to easily interpret different regions of the transformed space in terms of *interactions* (products) 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*. Here the degree of an interaction is defined as the number of features that are multiplied together, such that,

e.g. interactions of degree three take the form $x_1x_2x_3$. 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 paper has the following structure. Section 2 provides an overview of tensor network regression, starting with a review of tensor operations and ending with a description of the tensor ring and tree tensor network architectures that we used for our tests. In section 3, we describe the motivation and mechanics of the interaction decomposition, and then 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, we also 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 section 4. 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.

2. Tensor network regression

2.1. Background

2.1.1. Tensor overview

Throughout this work, we consider machine learning models that are constructed using *tensors* [6, 7]. For our purposes, a tensor is simply a multidimensional array of numbers, 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 one of its elements. From this perspective, vectors and matrices can be viewed as first-order and second-order tensors respectively. We will denote tensors with order greater than one using uppercase letters $(A, B, C, \ldots)$, while vectors will be denoted using a lower case letter under an arrow $(\vec{a}, \vec{b}, \vec{c}, \ldots)$. 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 (all dimensions are indexed starting from zero). When referring to elements of a vector, the arrow symbol is dropped. To specify the ith member of a set of tensors, we use a superscript with parentheses, e.g. $A^{(i)}$.

There are a wide variety of operations that can be defined between tensors, but in this work we focus primarily on 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, such that each element of C is given by

$$C_{i_0 \ldots i_{m-1} j_0 \ldots j_{n-1}} = A_{i_0 \ldots i_{m-1}} B_{j_0 \ldots j_{n-1}} \tag{1}$$

and the order of C is the sum of the orders of A and 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

$$C_{jklm} = \sum_i A_{jik} B_{lmi}. \tag{2}$$

Note that C is fourth order rather than sixth order, since two of the dimensions of $A \otimes B$ were summed together.

2.1.2. Tensor networks

Although we have described the tensor product and the tensor contraction as operations which construct a new tensor C from existing tensors A and B, it is equally valid to take the reverse view and interpret equations (1) and (2) as representing an existing tensor C in terms of new components A and B. This approach is the foundation of the *tensor network* representation, in which a tensor of interest is decomposed into a set of contractions between component tensors [8, 9]. The number and order of these component tensors are ultimately arbitrary, but most networks are constructed such that the component orders are all significantly smaller than the order of the original tensor. Since the number of elements in a tensor scales exponentially with its order (for a fixed index size), these component tensors will generally contain far fewer

$$H_{ijkop} = \sum_{l,m,n,r=0}^{t-1} D_{lim} E_{jmrk} F_{lrn} G_{nop}$$

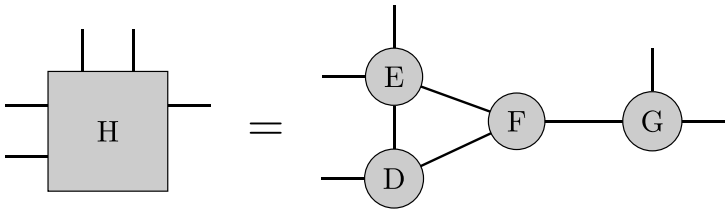


Figure 1. Tensor network representing fifth-order tensor H, depicted using explicit summations (top) and a tensor diagram (bottom). The network is generated by four different contractions, each of which is indicated in the diagram by a shared leg. While it is possible to discern the contraction pattern of the component tensors by studying the index notation, the diagram makes it obvious at a glance.

elements than the original tensor. This can allow for operations to be performed on the network that would have been computationally intractable on the original, higher-order tensor.

As a very simple example of a tensor network, consider the sixth-order tensor C that is represented by the contraction of two fourth-order tensors A and B, such that the elements of C are given by

$$C_{ijklmn} = \sum_{r=0}^{t-1} A_{ijkr} B_{lmnr}. \tag{3}$$

Assuming that each index in equation (3) 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 ($2t^4$). Using this representation, operations on C which are localized to indices $\{i, j, k\}$ or $\{l, m, n\}$ can instead be performed on A or B respectively, dramatically reducing their computational cost. Further inspection of equation (3) shows that the contraction of A and B can yield a combined tensor that has rank at most t across dimension r, which is far smaller than the largest possible rank of t^3. This implies that most sixth-order tensors can only be approximately represented by the contraction of A and B, with the quality of the approximation depending on both the underlying rank structure of C and the size constraints placed on A and B. Similar trade-offs occur across all tensor networks, with different contraction structures being better suited to represent different higher-order tensors [10, 11].

Due to its simplicity, the network formed from A and B in equation (3) can be clearly conveyed by simply listing out all of the indices explicitly. However, many tensor networks involve contractions between dozens or even hundreds of tensors, each with their own set of indices. For notational clarity, it is common to use *tensor diagrams* (also referred to as *Penrose notation* [12]) to represent the sets of contractions within more complicated tensor networks. In these 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. Figure 1 shows the relative simplicity of the diagram notation versus explicit summations when several tensors are involved in the network. We utilize tensor diagrams, along with explicit index expressions, throughout the remainder of this paper to help illustrate the relevant tensor operations.

2.1.3. Regression with tensors

Our work focuses on the use of tensor networks as a means of performing *regression* [13]. In a regression task, the goal is to learn (or estimate) the relationship between a set of m independent variables $\{x_i\}_{i=0}^{m-1}$ called *features* and a set of n dependent variables $\{y_i\}_{i=0}^{n-1}$ called *labels*. 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

$$\vec{y} \approx \vec{f}(\vec{x}; \mathcal{W}), \tag{4}$$

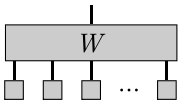
where $\mathcal{W}$ is a set of learned parameters which determines the behavior of the function. We do not expect the relationship in equation (4) to be exact for any intuitive function $\vec{f}$, except when the data is generated artificially. Indeed, an exact reconstruction will generally be undesirable for real-world data, since the labels

are likely to contain noise that should not be directly copied into the model. Once $\vec{f}$ is learned, it can be used to make predictions about the value of $\vec{y}$ for an unlabeled sample $\vec{x}$.

Tensor network regression can be understood as a specific form of *tensor regression* [14], in which a large tensor of regression coefficients is generated by a network of smaller component tensors. In tensor regression, the function $\vec{f}$ of equation (4) 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 a set of m vector-valued functions $\{\vec{h}^{(i)}\}_{i=0}^{m-1}$ that each take as input a single feature:

$$X(\vec{x}) = \bigotimes_{i=0}^{m-1} \vec{h}^{(i)}(x_i) \quad \rightarrow \quad \text{  }, \tag{5}$$

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 equation (5) 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:

$$f_k(\vec{x};\mathcal{W}) = \sum_{i_0 \ldots i_{m-1}} W_{k i_0 \ldots i_{m-1}} X_{i_0 \ldots i_{m-1}}(\vec{x}) \quad \rightarrow \quad \text{  }, \tag{6}$$

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 $W_{k i_0 \ldots i_m}$ 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 $\{x_i\}_{i=0}^{m-1}$ play in equation (6) depends on the form of X and therefore on the set of functions $\{\vec{h}^{(i)}\}_{i=0}^{m-1}$ that were chosen. For our work we follow Novikov *et al* and use functions of the form

$$\vec{h}^{(i)}(x_i) = \begin{bmatrix} 1 \\ x_i \end{bmatrix}, \tag{7}$$

which have been used in other implementations of tensor network regression [4, 15, 16]. When equation (7) is used to construct X, the regression function $\vec{f}(\vec{x};\mathcal{W})$ from equation (6) becomes

$$f_k(\vec{x};\mathcal{W}) = \sum_{i_0 \ldots i_{m-1}=0}^{1} W_{k i_0 \ldots i_{m-1}} x_0^{i_0} x_1^{i_1} \cdots x_{m-1}^{i_{m-1}}, \tag{8}$$

where $0^0 = 1$ is assumed. Equation (8) shows that tensor regression, when using the definition of $\vec{h}^{(i)}(x_i)$ from equation (7), 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. Regression on the elements of X can therefore generate functions $\vec{f}$ which have non-zero mixed derivatives with respect to the original features. For example,

$$\frac{\partial^2}{\partial x_0 \partial x_1} f_k(\vec{x};\mathcal{W}) = \sum_{i_2 \ldots i_{m-1}=0}^{1} W_{k 1 1 i_2 \ldots i_{m-1}} x_2^{i_2} x_3^{i_3} \cdots x_{m-1}^{i_{m-1}}. \tag{9}$$

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.

2.2. Regression using tensor rings and tree tensor networks

Although straightforward mathematically, the approach to tensor regression outlined in section 2.1.3 is often impractical due to the size of the weight tensor W. Inspection of equation (8) reveals that there are 2^m parameters in W, which is exponential in the number of features. Given that many regression tasks involve data with hundreds or even thousands of features, this method of parameterization cannot be used. Tensor network regression offers an alternative approach, in which the weight tensor is decomposed into a set of low-order component tensors. The parameters $\mathcal{W}$ of the regression model are then taken to be the elements

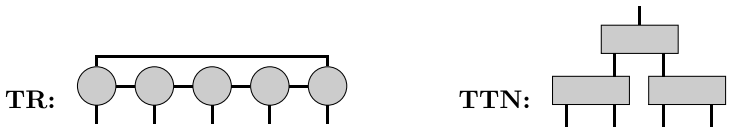


Figure 2. Tensor diagrams depicting a tensor ring (TR) and a tree tensor network (TTN), which can both be used for regression. The five component tensors of the TR are arranged in a line and connected together horizontally by five virtual indices, with a physical index dangling vertically from each component. By contrast, the three components of the TTN are arranged in a binary tree, with a pair of virtual indices connecting the two layers. The physical indices of the TTN are placed along the bottom layer, although when used for regression an additional physical index is added to the top tensor to represent the output, as shown here.

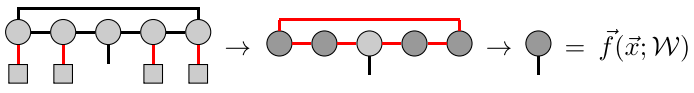
of these tensors, rather than the elements of W directly. This decomposition can be illustrated using figure 1, in which H serves as the weight tensor while the components D, E, F, and G contain the actual parameters of the model. Since the data tensor X is itself composed of vectors $\{\vec{h}^{(i)}(x_i)\}_{i=0}^{m-1}$, it is possible for the contraction in equation (6) to be carried out using only the *components* of W and X.

There exist a wide variety of tensor network architectures which can be used for regression. Our work here will focus on two of the more popular designs: tensor rings [17, 18] and tree tensor networks [19, 20]. The structures of these networks are illustrated using tensor diagrams in figure 2, and are described in the following subsections.

2.2.1. Tensor rings

The tensor ring (TR) is a popular type of tensor network, having been utilized for neural network compression [21, 22] and image reconstruction [23–25]. As depicted in figure 2, a TR is constructed from a 1D sequence of third-order tensors contracted along a set of *virtual indices* that link neighboring tensors together. The 'ring' part of a TR references the fact that the tensors at the beginning and end of the sequence are also contracted together, forming a closed loop. In addition to its two virtual indices, each component tensor also has a *physical index*, which becomes an index of the higher-order tensor after contraction of the virtual indices. When a TR is used for regression, the physical indices are contracted with the $\{\vec{h}^{(i)}(x_i)\}_{i=0}^{m-1}$ from the data tensor, except for one tensor whose physical index is left uncontracted to serve as the index of $\vec{f}$. The TR is closely related to the matrix product state (MPS), which is used heavily in quantum many-body physics [26] and also frequently utilized for tensor network regression [1, 2, 15]. The structure of an MPS network, also referred to as a *tensor train decomposition* [27], is identical to that of a TR, except that the tensors at the ends of the chain are second-order and thus not contracted together. In this work we use TRs rather than MPSs due to the greater symmetry of the former, which allows us to employ simpler contraction algorithms.

For tensor network regression to be practical, the sequence of contractions between components of X and components of the network must be carried out such that all of the intermediate tensors are low-order. For a TR, this can be easily achieved by performing the contractions in two stages, with the vectors $\{\vec{h}^{(i)}(x_i)\}_{i=0}^{m-1}$ first being contracted with their corresponding component tensors in the TR along the physical index, which produces a new set of second-order tensors. These matrices are then contracted together, along with the third-order tensor containing the regression output, using the virtual indices. For $m = 4$, the diagrams of these steps are given by



(10)

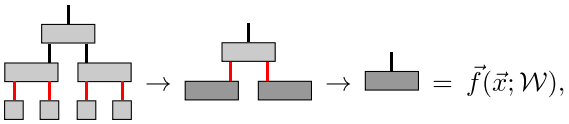
where the legs colored in red are those that are contracted from one step to the next. When using the functions from equation (7), the total number of parameters in a TR regression model is $2(m+1)r^2$, where r is the size of the virtual indices. Since the number of features m is generally fixed for a given regression task (after preprocessing), r serves as the primary tunable parameter in the model, with larger values of r placing fewer restrictions on the elements of W. If r is allowed to grow exponentially with m, then the TR can represent an arbitrary weight tensor W, although this generally defeats the purpose of using a tensor network. In practice, r is typically capped at between 10 and 100 for regression.

2.2.2. Tree tensor networks

A common alternative to the TR/MPS network is the tree tensor network (TTN), in which the component tensors are arranged in a (typically binary) tree pattern. TTNs have been used for quantum simulation [19, 28], efficient tensor representation [29] (where it is known as the *hierarchical Tucker decomposition*), and for regression [4, 30]. An example TTN is depicted in figure 2, showing that each component tensor has both a

horizontal and vertical position in the network. Similar to a TR, a TTN contains both virtual and physical indices, but only the lowest layer of component tensors are contracted directly with the data tensor X. While the structure of a TR can be applied easily to any value of m, a binary TTN works most efficiently when $m = 2^l$, where l is the number of layers in the tree. Although these networks can be modified to handle other values of m, our work here will only consider regression tasks where the number of features is a power of two.

When used for tensor regression, the components in the bottom layer of the TTN are first contracted with the $\{\vec{h}^{(i)}(x_i)\}_{i=0}^{m-1}$ of X, which generates a new layer of $\frac{m}{2}$ first-order tensors. This is shown in the second diagram of equation (11), where the new first-order tensors are depicted as dark blocks. These tensors are then contracted with the second layer of the tree, generating $\frac{m}{4}$ first-order tensors. This process repeats layer-by-layer until the regression output $\vec{f}(\vec{x})$ is generated by contracting the top tensor, which is shown in the third diagram of equation (11). The intermediate tensors created at each layer are always first-order, which ensures that the procedure will be computationally tractable. For $m = 4$, the tensor diagrams for the contractions are given by



$$= \vec{f}(\vec{x}; \mathcal{W}), \tag{11}$$

with the number of tensors being approximately halved after each layer is contracted. As in equation (10), the legs shown in red are contracted between each step. When using $\vec{h}^{(i)}(x_i)$ of the form in equation (7), the number of parameters in a TTN is $2mr + (\frac{m}{2} - 1)r^3$, which scales as $\mathcal{O}(r^3)$ in contrast with the $\mathcal{O}(r^2)$ scaling of a TR. As a result, the size r of the virtual index is typically chosen to be on the order of 10. As with a TR, a TTN can represent an arbitrary weight tensor W if r is allowed to scale exponentially with m.

3. Interaction decomposition

3.1. Motivation

Throughout our discussion of tensor network regression in section 2.2, 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 equation (8), 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.

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 the weight tensor W. In an interaction decomposition, the terms of the sum in equation (8) 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 labeled its interaction *degree*, such that x_1 has degree 1, x_1x_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

$$\vec{f}(\vec{x}; \mathcal{W}) = \sum_{j=0}^{m} \vec{d}^{\,(j)}(\vec{x}; \mathcal{W}), \tag{12}$$

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 section 3.2.

Using equation (12), 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 section 3.3. Furthermore, by choosing to keep only a specific subset $\mathcal{D}$ of the decomposition terms in equation (12), 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 section 3.4.

3.2. 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 equation (5). The nature of the $\vec{d}^{\,(j)}(\vec{x};\mathcal{W})$ terms from equation (12) 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 equation (7), we can expand X on a standard basis of $\mathbb{X}$ as

$$X = \bigotimes_{i=0}^{m-1} (\vec{e}_0^{\,(i)} + x_i \vec{e}_1^{\,(i)}) = \sum_{i_1,\ldots,i_m=0}^{1} x_0^{i_0} \cdots x_{m-1}^{i_{m-1}} \, \vec{e}_{i_0}^{\,(0)} \otimes \ldots \otimes \vec{e}_{i_{m-1}}^{\,(m-1)}, \tag{13}$$

where $\{\vec{e}_0^{\,(i)}, \vec{e}_1^{\,(i)}\}$ spans the two-dimensional space inhabited by $\vec{h}^{(i)}(x_i)$. In words, equation (13) shows that the tensor products $\vec{e}_{i_0}^{\,(0)} \otimes \ldots \otimes \vec{e}_{i_{m-1}}^{\,(m-1)}$ 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 equation (13), 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 $\vec{e}_{i_0}^{\,(0)} \otimes \ldots \otimes \vec{e}_{i_{m-1}}^{\,(m-1)}$ 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

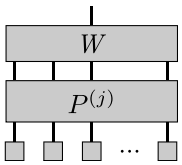
$$\dim(\mathbb{D}^{(j)}) = \binom{m}{j}, \tag{14}$$

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

$$\dim(\mathcal{D}) = \sum_{j \in \mathcal{D}} \binom{m}{j}, \tag{15}$$

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

$$f_k(\vec{x};\mathcal{W}) = \sum_{j=0}^{m} \sum_{i_0,\ldots,i_{m-1}=0}^{1} W_{k i_0 \ldots i_{m-1}} (P^{(j)} X)_{i_0 \ldots i_{m-1}} \rightarrow \sum_{j=0}^{m} \tag{16}$$



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 equation (16) 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 equation (16) 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.

3.3. Interaction decompositions of TR and TTN models

The interaction decomposition of a TR or TTN 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 equations (10) and (11), since the elements

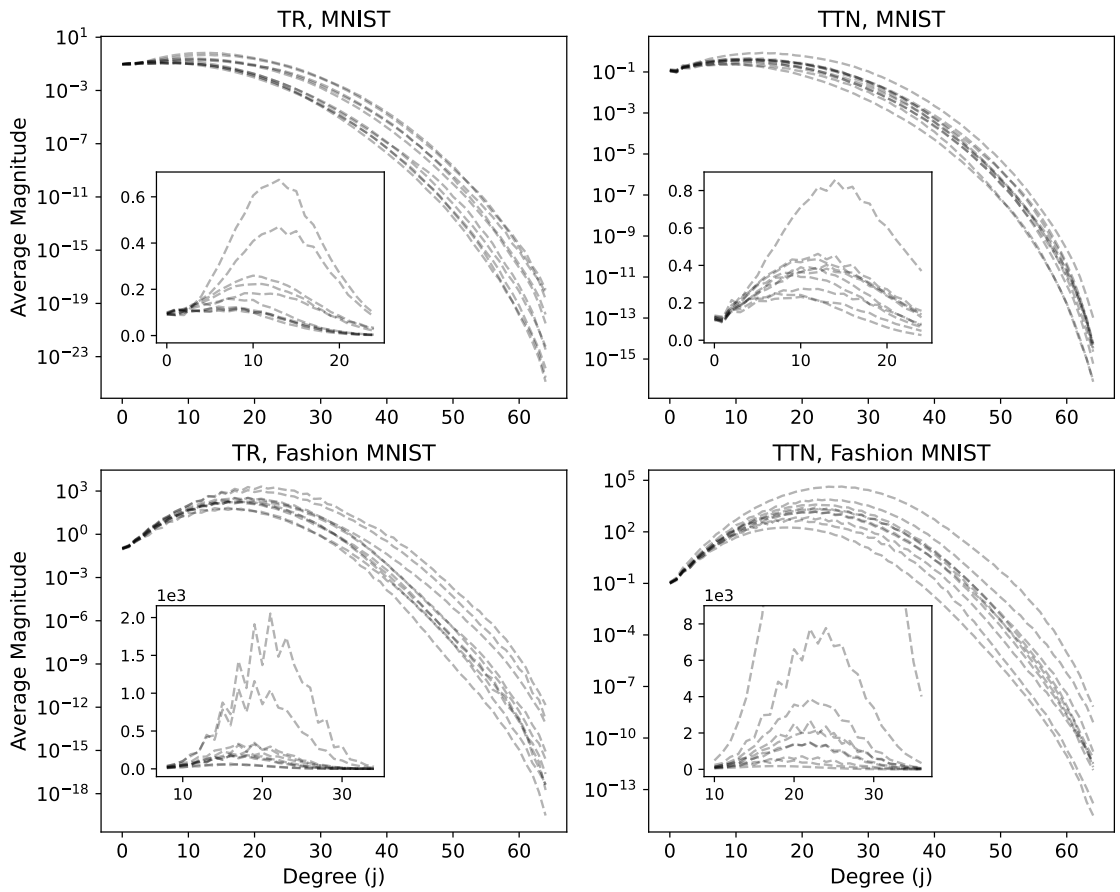


Figure 3. Plots of the L1 norm of $\vec{d}^{\,(j)}(\vec{x};\mathcal{W})$, averaged across the MNIST and Fashion MNIST test datasets, for ten TR models and ten TTN models trained using all interaction degrees. Each dashed line represents the average magnitudes from one of the models, which is plotted 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.

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 equation (14), 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 section 'Procedure for the interaction decomposition', 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 [31] and Fashion MNIST [32] datasets. 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 section 'Procedure for the interaction decomposition'), 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 optimized using gradient descent 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 L1 norm of $\vec{d}^{\,(j)}(\vec{x};\mathcal{W})$ for each image in the test dataset, and then averaged over the set. Figure 3 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

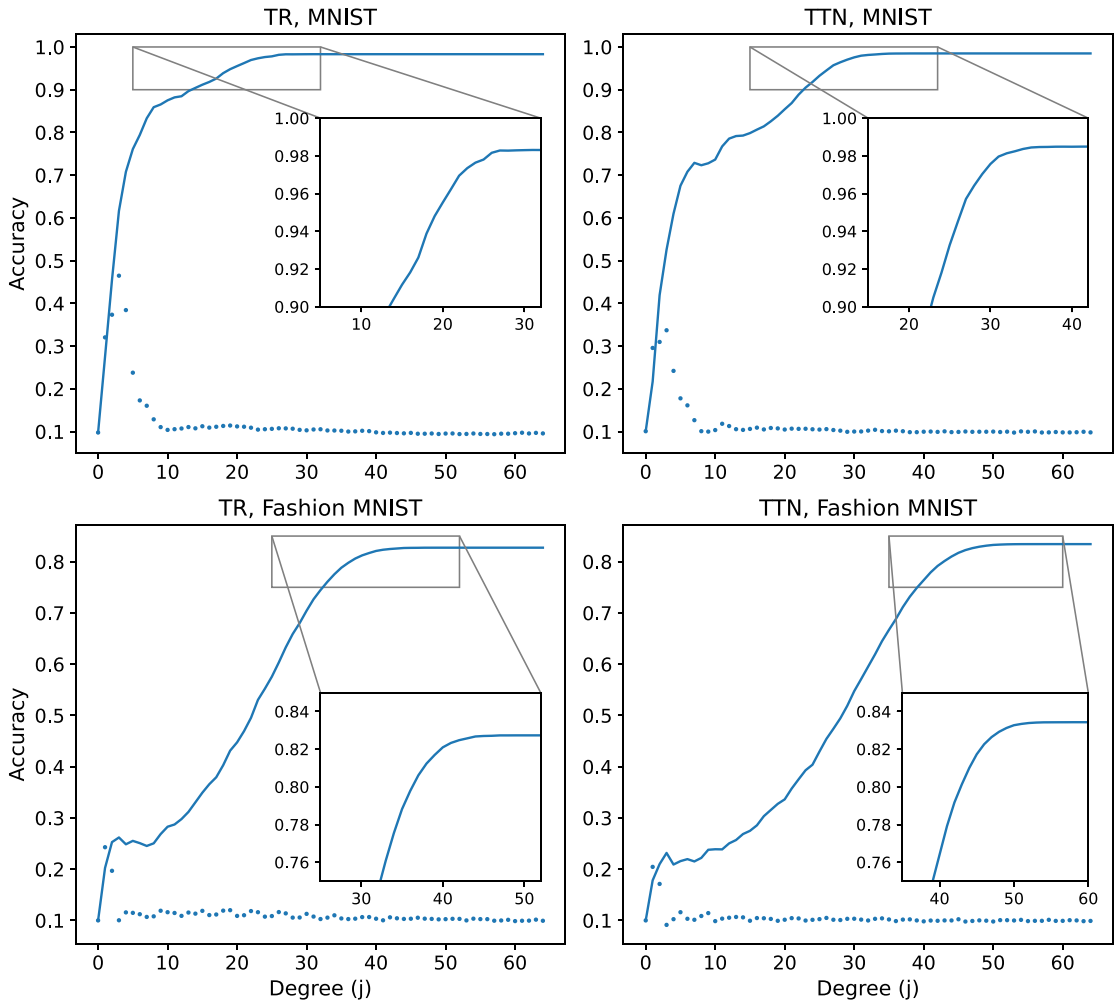


Figure 4. Plots of the average classification accuracy for the TR models and TTN models, trained using all interaction degrees, as a function of interaction degree j, using the MNIST and Fashion MNIST test datasets. The scatter plots show the accuracy of each $\vec{d}^{\,(j)}(\vec{x};\mathcal{W})$ term 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.

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 3 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 3 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 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 equation (15).

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 [2, 4, 15]. On Fashion MNIST the accuracies are

signficantly lower, at 82.73% for the TR and 83.43% for the TTN[3]. 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 3, 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 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 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 3 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.

3.4. Interaction decompositions as regression models

In section 3.3, 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

$$\vec{f}(\vec{x}; \mathcal{W}) = \sum_{j \in \mathcal{D}} \vec{d}^{\,(j)}(\vec{x}; \mathcal{W}). \tag{17}$$

Comparing this expression to the full interaction decomposition given in equation (12), it is clear that if $\mathcal{D}$ is the set of all interaction degrees (i.e. if $\mathcal{D} = \{0, 1, \ldots, m\}$), 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 $\mathcal{D}$. However, unlike in the case of sparse neural networks [34, 35], 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 section 'Procedure for the interaction decomposition', it is possible to efficiently train $\mathcal{D}$-degree models on the same regression tasks used in section 3.3, and thus compare their classification accuracies with those shown in figure 4. 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:

$$\text{Cumulative-}j: \ \vec{f}(\vec{x}; \mathcal{W}) = \sum_{j'=0}^{j} \vec{d}^{\,(j')}(\vec{x}; \mathcal{W}), \qquad \text{Degree-}j: \ \vec{f}(\vec{x}; \mathcal{W}) = \vec{d}^{\,(j)}(\vec{x}; \mathcal{W}). \tag{18}$$

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. For our numerical tests, we trained the

[3] Tensor networks can achieve significantly higher accuracies on 28×28 Fashion MNIST [3–5], but the decrease in performance at 8×8 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 [33] can achieve accuracies of 99.09% on 8×8 MNIST and 86.5% on 8×8 Fashion MNIST (see appendix section 'Regression model comparisons').

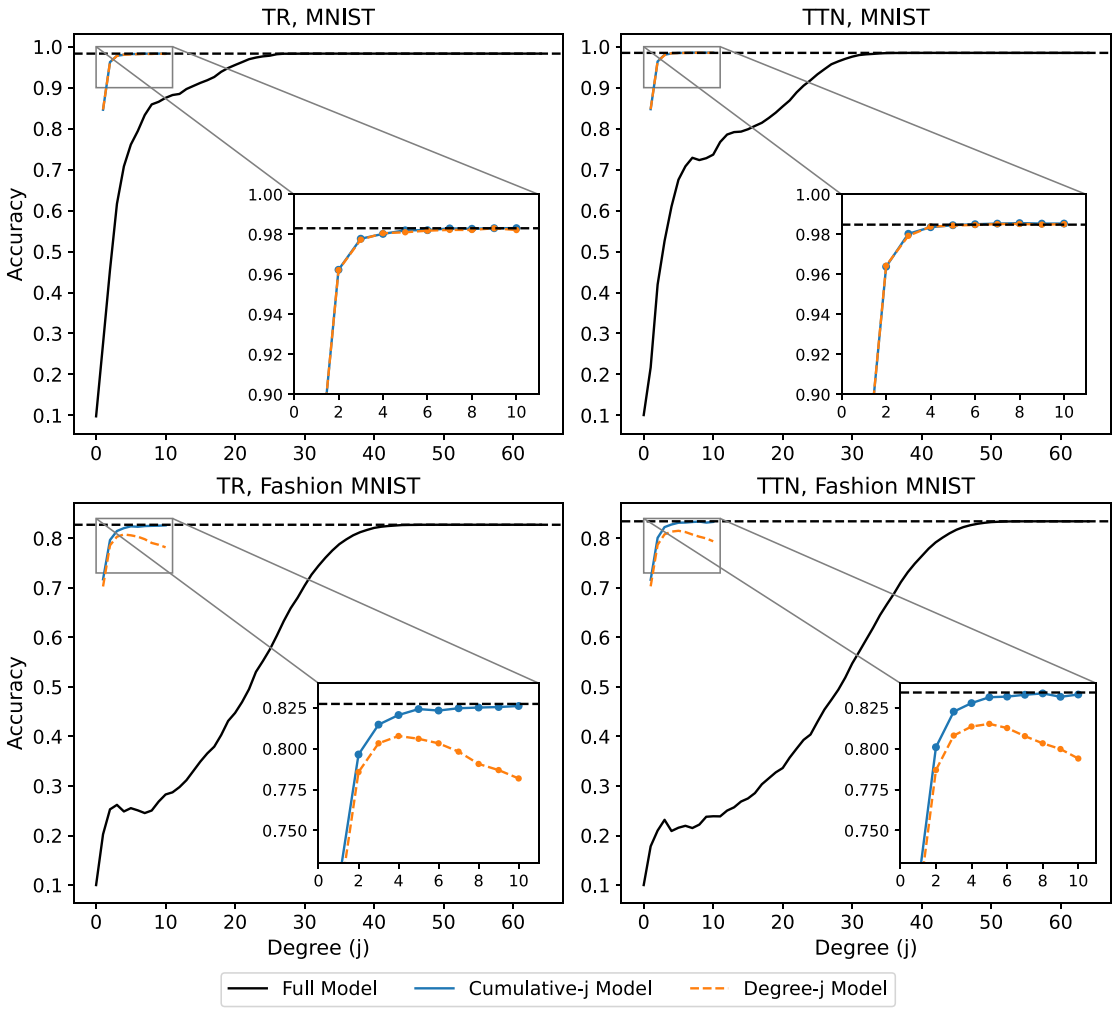


Figure 5. Plots of average accuracy on the MNIST and Fashion MNIST test datasets for $\mathcal{D}$-degree models parameterized by TRs and TTNs, with ten models being averaged for each degree. The solid black line indicates the cumulative accuracy of the standard tensor network models analyzed in section 3.3, as plotted in figure 4, 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 on 8×8 images from the MNIST and Fashion MNIST datasets prepared in the same manner described in section 3.3. 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 section 'Regression model comparisons').

Figure 5 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, 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 5 are computed in precisely the same manner as the line and scatter plots from figure 4, except that the models which generated figure 4 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 section 'Tabulation of $\mathcal{D}$-degree Model performance'.

The first trend to observe from figure 5 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, 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. 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 5 mark the accuracy of the full tensor network models when every degree contribution is included. Using these values as a benchmark, we can see that several of the cumulative-j TTN models ($6 \leqslant j \leqslant 10$) and degree-$j$ TTN models ($7 \leqslant j \leqslant 10$) 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 section 'Regression model comparisons', 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. Discussion

The exponential feature space induced by the transformation in equation (5) 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 section 3.3 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 section 3.2, 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 equation (14) across all significant degrees. It is important to note, however, that figure 4 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 section 3.3 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 section 3.4 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 equation (15). The results shown in figure 5 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 limitations 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:

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

Data availability statement

All data that support the findings of this study are included within the article (and any supplementary files).

Acknowledgment

Funding for this work was provided by the UC Noyce Initiative.

Appendix

Procedure for the interaction decomposition

In this section, 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

$$(A \,\tilde{\otimes}\, B)_{ji_0\ldots i_{m-1}k_0\ldots k_{n-1}} = \sum_{j_a+j_b=j} A_{j_a i_0\ldots i_{m-1}} B_{j_b k_0\ldots k_{n-1}}, \tag{19}$$

where the resulting tensor is of order $m+n+1$ and $0 \leqslant j \leqslant \max(j_a)+\max(j_b)$. Note that this operation attaches special significance to the first dimension, which we will hereafter refer to as the *degree index*. As shown in equation (19), 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 equation (2), such that the contraction of fourth-order tensors A and B is given by

$$C_{jklqr} = \sum_{j_a+j_b=j} \sum_{i} A_{j_a kil} B_{j_b qri}. \tag{20}$$

The utility of these degree-preserving operations becomes apparent if we alter the featurization in equation (7) to be

$$H^{(i)}(x_i) = \begin{bmatrix} 1 & 0 \\ 0 & x_i, \end{bmatrix}, \tag{21}$$

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 $H^{(k)}$:

$$H^{(i)} \tilde{\otimes} H^{(k)} = \begin{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, & \begin{bmatrix} 0 & x_i \\ x_k & 0 \end{bmatrix}, & \begin{bmatrix} 0 & 0 \\ 0 & x_i x_k \end{bmatrix} \end{bmatrix}, \tag{22}$$

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, equation (22) 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:

(a) 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.
(b) Construct (implicitly) a modified data tensor $\tilde{X}$ using the mappings from equation (21), such that $\tilde{X}(\vec{x}) = H^{(0)}(x_0) \tilde{\otimes} H^{(1)}(x_1) \tilde{\otimes} \ldots \tilde{\otimes} H^{(m-1)}(x_{m-1})$.
(c) 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.
(d) 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

$$F(\vec{x}; \mathcal{W}) = \left[\vec{d}^{\,(0)}(\vec{x}; \mathcal{W}), \quad \vec{d}^{\,(1)}(\vec{x}; \mathcal{W}), \quad \ldots, \quad \vec{d}^{\,(m)}(\vec{x}; \mathcal{W}) \right], \tag{23}$$

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 equation (19). 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 equation (19), 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 $\bar{j}_a \leqslant \bar{j}_b$. 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

$$s = (\bar{j}_a + 1)(\bar{j}_b + 1), \tag{24}$$

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.

Tabulation of $\mathcal{D}$-degree model performance

The following two tables show the results of the numerical tests described in section 3.4 and plotted in figure 5, along with the relevant cumulative accuracy values for the full models from figure 4. Table 1 gives the accuracies for MNIST, while table 2 provides them for Fashion MNIST. 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

Table 1. Table of average accuracy vs degree for the six different model types on MNIST, for figure 5.

	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 2. Table of average accuracy vs degree for the six different model types on Fashion MNIST, for figure 5.

	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 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.

	Parameters	Seconds per epoch	MNIST accuracy	Fashion MNIST accuracy
Linear	65	2	84.7(1)	71.35(8)
Bilinear	2081	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	1196 530	23	99.27(3)	86.64(9)

value of j, while for the full models they denote the cumulative accuracy of the output up to the jth interaction degree.

Regression model comparisons

In table 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 $1 \leqslant j \leqslant 4$. 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 [33] architecture, which contains the most parameters and achieves the best performance on both datasets.

ORCID iDs

Ian Convy https://orcid.org/0000-0003-1818-2677
K Birgitta Whaley https://orcid.org/0000-0002-7164-4757

References

[1] Novikov A, Trofimov M and Oseledets I 2016 Exponential machines (arXiv:1605.03795 [cs, stat])
[2] Stoudenmire E and Schwab D J 2016 Supervised learning with tensor networks *Advances in Neural Information Processing Systems* vol 29 (Red Hook, NY: Curran Associates, Inc.)
[3] Glasser I, Pancotti N and Cirac J I 2018 Supervised learning with generalized tensor networks (arXiv:1806.05964 [cond-mat, physics:quant-ph, stat])
[4] Stoudenmire E M 2018 Learning relevant features of data with multi-scale tensor networks *Quantum Sci. Technol.* **3** 034003
[5] Chen Y, Pan Y and Dong D 2021 Residual tensor train: a flexible and efficient approach for learning multiple multilinear correlations (arXiv:2108.08659 [cs])
[6] Kolda T and Bader B 2009 Tensor decompositions and applications *SIAM Rev.* **51** 455–500
[7] Hackbusch W 2012 *Tensor Spaces and Numerical Tensor Calculus (Springer Series in Computational Mathematics)* (Heidelberg: Springer)
[8] Biamonte J and Bergholm V 2017 Tensor networks in a nutshell (arXiv:1708.00006 [cond-mat, physics:gr-qc, physics:hep-th, physics:math-ph, physics:quant-ph])
[9] Bridgeman J C and Chubb C T 2017 Hand-waving and interpretive dance: an introductory course on tensor networks *J. Phys. A: Math. Theor.* **50** 223001
[10] Grasedyck L, Kressner D and Tobler C 2013 A literature survey of low-rank tensor approximation techniques *GAMM-Mitteilungen* **36** 53–78
[11] Evenbly G and Vidal G 2011 Tensor network states and geometry *J. Stat. Phys.* **145** 891–918
[12] Penrose R 1971 Applications of negative dimensional tensors *Combinatorial Mathematics and its Applications* vol 1 (Cambridge: Academic) pp 221–44
[13] Hastie T, Tibshirani R and Friedman J 2009 *The Elements of Statistical Learning: Data Mining, Inference and Prediction (Springer Series in Statistics)* 2nd edn (New York: Springer)
[14] Liu Y, Liu J, Long Z and Zhu C 2022 *Tensor Computation for Data Analysis* (Cham: Springer)
[15] Efthymiou S, Hidary J and Leichenauer S 2019 Tensornetwork for machine learning (arXiv:1906.06329 [cond-mat, physics:physics, stat])
[16] Meng Y M, Zhang J, Zhang P, Gao C and Ran S J 2021 Residual matrix product state for machine learning (arXiv:2012.11841 [cond-mat, physics:quant-ph])
[17] Zhao Q, Zhou G, Xie S, Zhang L and Cichocki A 2016 Tensor ring decomposition (arXiv:1606.05535 [cs])
[18] Mickelin O and Karaman S 2020 On algorithms for and computing with the tensor ring decomposition *Numer. Linear Algebra Appl.* **27** e2289
[19] Shi Y, Duan L and Vidal G 2006 Classical simulation of quantum many-body systems with a tree tensor network *Phys. Rev.* A **74** 022320
[20] Oseledets I V and Tyrtyshnikov E E 2009 Breaking the curse of dimensionality, or how to use SVD in many dimensions *SIAM J. Sci. Comput.* **31** 3744–59
[21] Wang W, Sun Y, Eriksson B, Wang W and Aggarwal V 2018 Wide compression: tensor ring nets *Proc. IEEE Conf. on Computer Vision and Pattern Recognition* pp 9329–38
[22] Pan Y, Xu J, Wang M, Ye J, Wang F, Bai K and Xu Z 2019 Compressing recurrent neural networks with tensor ring for action recognition *Proc. AAAI Conf. on Artificial Intelligence* vol 33 pp 4683–90
[23] Yuan L, Li C, Mandic D, Cao J and Zhao Q 2019 Tensor ring decomposition with rank minimization on latent space: an efficient approach for tensor completion *Proc. AAAI Conf. on Artificial Intelligence* vol 33 pp 9151–8
[24] Zhao Q, Sugiyama M, Yuan L and Cichocki A 2019 Learning efficient tensor representations with ring-structured networks *ICASSP 2019—2019 IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)* pp 8608–12
[25] He W, Yokoya N, Yuan L and Zhao Q 2019 Remote sensing image reconstruction using tensor ring completion and total variation *IEEE Trans. Geosci. Remote Sens.* **57** 8998–9009
[26] Schollwöck U 2011 The density-matrix renormalization group in the age of matrix product states *Ann. Phys., NY* **326** 96–192
[27] Oseledets I V 2011 Tensor-train decomposition *SIAM J. Sci. Comput.* **33** 2295–317
[28] Murg V, Verstraete F, Legeza O and Noack R M 2010 Simulating strongly correlated quantum systems with tree tensor networks *Phys. Rev.* B **82** 205105
[29] Grasedyck L 2010 Hierarchical singular value decomposition of tensors *SIAM J. Matrix Anal. Appl.* **31** 2029–54
[30] Liu D, Ran S-J, Wittek P, Peng C, García R B, Su G and Lewenstein M 2019 Machine learning by unitary tensor network of hierarchical tree structure *New J. Phys.* **21** 073059
[31] LeCun Y, Cortes C and Burges C 1998 MNIST handwritten digit database (available at: http://yann.lecun.com/exdb/mnist/)
[32] Xiao H, Rasul K and Vollgraf R 2017 Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms (arXiv:1708.07747 [cs, stat])
[33] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V and Rabinovich A 2015 Going deeper with convolutions *2015 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (Boston, MA: IEEE) pp 1–9
[34] Srinivas S, Subramanya A and Babu R V 2017 Training sparse neural networks *2017 IEEE Conf. on Computer Vision and Pattern Recognition Workshops (CVPRW)* pp 455–62
[35] Liu B, Wang M, Foroosh H, Tappen M and Penksy M 2015 Sparse convolutional neural networks *2015 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* pp 806–14