Anomaly Detection with Tensor Networks

Jinhui Wang*
Stanford University
Stanford, CA 94305, USA
wangjh97@stanford.edu

Chase Roberts
X – The Moonshot Factory
Mountain View, CA 94043, USA
chaseriley@google.com

Guifre Vidal
X – The Moonshot Factory
Mountain View, CA 94043, USA
guifre@google.com

Stefan Leichenauer
X – The Moonshot Factory
Mountain View, CA 94043, USA
sleichenauer@google.com

Abstract

Originating from condensed matter physics, tensor networks are compact representations of high-dimensional tensors. In this paper, the prowess of tensor networks is demonstrated on the particular task of one-class anomaly detection. We exploit the memory and computational efficiency of tensor networks to learn a linear transformation over a space with dimension exponential in the number of original features. The linearity of our model enables us to ensure a tight fit around training instances by penalizing the model's global tendency to predict normality via its Frobenius norm—a task that is infeasible for most deep learning models. Our method outperforms deep and classical algorithms on tabular datasets and produces competitive results on image datasets, despite not exploiting the locality of images.

1 Introduction

Anomaly detection (AD) entails determining whether a data point comes from the same distribution as a prior set of normal data. Anomaly detection systems are used to discover credit card fraud, detect cyber intrusion attacks and identify cancer cells. Since normal examples are readily available while anomalies tend to be rare in production environments, we consider the semi-supervised or one-class setting where only normal instances are present in the training set. It is important to remark that the outlier space is often much larger than the inlier space, though anomalous observations are uncommon. For instance, the space of normal dog images is sparse in the space of anomalous non-dog images. This discrepancy between data availability and space sizes makes anomaly detection hard, as one must manage a model's behavior over the entire input space while only having information of a minuscule subspace. Deep learning models generally struggle with this challenge due to their unpredictability and tendency to overfit. Linear models, however, do not face such a difficulty.

To gain control over our model's behavior on the entire input space, we employ a linear transformation as its main component and subsequently penalize its Frobenius norm. However, this transformation has to be performed over an exponentially large feature space for our model to be expressive—an impossible task with full matrices. Thus, we leverage tensor networks as sparse representations of such large matrices. All-in-all, our model is an end-to-end anomaly detector for general data that produces a normality score via its decision function $\mathcal{D}$. Our novel method is showcased on several tabular and image datasets. We attain significant improvements over prior methods on tabular datasets and competitive results on image datasets, despite not exploiting the locality of image pixels.

*Work done while a resident of X – The Moonshot Factory.

First Workshop on Quantum Tensor Networks in Machine Learning, 34th Conference on Neural Information Processing Systems (NeurIPS 2020).

2 Related Work

The early work of Stoudenmire and Schwab [39] demonstrated the potential of tensor networks in classification tasks, using the well-known density matrix renormalization group algorithm [43] to train a Matrix Product State (MPS) [36, 27] as a weight matrix in classifying *MNIST* digits [20]. Subsequent work has also applied tensor networks in further classification tasks [38, 40] and regression [30] while more recent focus has turned towards unsupervised settings [8, 16, 25].

The literature on anomaly detection (AD) is vast and we will focus on reviewing previous work in the **one-class context** for **arbitrary data** (e.g. not restricted to images). Kernel-based methods, such as the One-Class SVM (OC-SVM) [23], learn a tight fit of inliers in an implicit high-dimensional feature space while the non-distance-based Isolation Forest [21] directly distinguishes inliers and outliers based on partitions of feature values. Unfortunately, such classical AD algorithms presume the clustering of normal instances in some feature space and hence suffer from the curse of dimensionality, requiring substantial feature selection to operate on feature-rich, multivariate data [3].

As such, hybrid methods were developed to first learn latent representations using Auto-Encoders (AE) [45, 2, 37] and Deep Belief Networks (DBN) [12], that were later fed to a OC-SVM. End-to-end deep learning models, without explicit AD objectives, have also been devised. Auto-Encoder AD models [17, 34, 7] learn an encoding of inliers and subsequently use the reconstruction loss as a decision function. Other AE-variants, such as Deep Convolutional Auto-Encoders (DCAE) [24, 22], have also been studied by [37, 31]. Next, generative models learn a probability distribution for inliers and subsequently identify anomalous instances as those with low probabilities or those which are difficult to find in their latent spaces (in the case of latent variable models). Generative Adversarial Networks (GANs) [15] have been popular in the latter category, with the advent of AnoGAN [35], a more efficient variant [46] based on BiGANs [10], GANomaly [1] and ADGAN [9].

Deep learning models with objectives that resemble shallow kernel-based AD algorithms have also been explored. Such models train neural networks as explicit feature maps while concurrently finding the tightest decision boundary around the transformed training instances in the output space. Deep SVDD (DSVDD) [33] seeks a minimal-volume hypersphere encapsulating inliers, motivated by the Support Vector Data Description (SVDD) [41], while One-Class Neural Networks (OC-NN) [6] searches for a maximum-margin hyperplane separating normal instances from the origin, in a fashion similar to OC-SVM. Contemporary attention has been directed towards self-supervised models, mostly for images [14, 13, 18], with the exception of the more recent GOAD [4] for general data. These models transform an input point into several altered instances according to a fixed class of rules and train a classifier that predicts a score for each altered instance belonging to its corresponding class of transformation. Outliers are then reflected as points with extreme scores, aggregated over all classes. In particular, GOAD unifies DSVDD and the self-supervised GEOM model [14] by defining the anomaly score of each transformed instance as its distance from its class' hypersphere center.

3 Model Description

3.1 Overview

In this section, we introduce our model that we call Tensor Network Anomaly Detector (TNAD). TNAD falls into the category of end-to-end models with explicit AD objectives, but with a crucial caveat. Its notion of tightness does not rely on the volume of a decision boundary, which is an inadequate measure in the one-class setting. To illustrate this point, DSVDD and GOAD find hyperspheres around training instances but are not explicitly discouraged from mapping unseen or anomalous instances to the same side. Thus, shrinking the boundary volume does not ensure a tight fit around inliers. In the extreme case, DSVDD and GOAD may find tiny hyperspheres but still have a loose fit around inliers, as they may map all possible inputs to the same point—a problem acknowledged by the original authors of DSVDD as "hypersphere collapse" [41]. In the scenario where outliers are available, one can indeed judge the tightness of a fit by the separation of inliers and outliers with respect to a decision boundary but in the one-class setting where no such points of reference are available, the tightness of a model's fit on training instances must be gauged relative to its predictions on unseen instances. A naive way of achieving this would then be to observe the model's behavior on random samples from the input space but such a scheme may be prohibitively

expensive on a representative sample. To circumvent this difficulty, we design TNAD to incorporate a canonical measure of its overall tendency to predict normality, which can be computed efficiently.

A schematic of TNAD is depicted in Figure 1. A fixed feature map Φ is applied to map inputs onto the surface of a unit hypersphere in a vector space V with dimension exponential in the number of original features N. The training instances are sparse in this high-dimensional space V and thus enables the learnt component of our model to be expressive, despite being a simple linear transformation $P : V \to W$. Upon action by P, normal instances will be mapped close to the surface of a hypersphere in W of an arbitrarily chosen radius (set to be $\sqrt{e}$ in our experiments) while anomalous instances can be identified as those close to the origin. The decision function of the model with respect to an input $\boldsymbol{x}$, where a larger value indicates normality, is then

$$\mathcal{D}(\boldsymbol{x}) = \|P\Phi(\boldsymbol{x})\|_2^2 \tag{1}$$

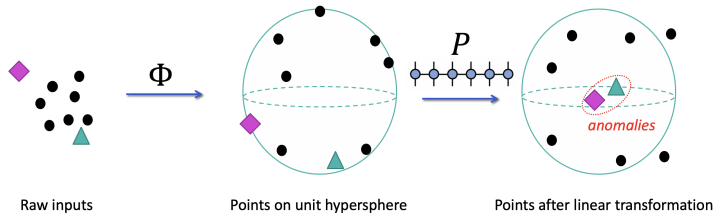

Figure 1: Schematic of Tensor Network Anomaly Detector (TNAD)

To accommodate the possible predominance of outliers, we allow $\dim W$ to have a smaller exponential scaling with N so that $\dim W \ll \dim V$ for P to have a large null-space. P can thus be understood informally as a "projection" that annihilates the subspace spanned by outliers. To parameterize P which has dimensions exponential in N, we leverage the Matrix Product Operator (MPO) tensor network, which is both memory and computationally-efficient. Finally, to obtain a tight fit around inliers, we penalize the Frobenius norm of P during training.

$$\|P\|_F^2 = tr\left(P^T P\right) = \sum_{i,j} |P_{ij}|^2 \tag{2}$$

where P_{ij} are the matrix elements of P with respect to some basis. Since $\|P\|_F^2$ is the sum of squared singular values of P, it captures the total extent to which the model is likely to deem an instance as normal. Concretely, if $\{v_i\}_{1\leq i\leq k} \subset V$ are singular vectors corresponding to the k non-zero singular values $\{s_i\}_{1\leq i\leq k}$ of P, $\mathcal{D}(\boldsymbol{x}) = \sum_{i=1}^k s_i^2 |\langle v_i, \Phi(\boldsymbol{x})\rangle|^2$ so penalizing $\|P\|_F^2 = \sum_{i=1}^k s_i^2$ has the effect of reducing $\mathcal{D}(\boldsymbol{x})$ for general inputs (while encouraging v_i's to be aligned with $\Phi(\boldsymbol{x})$'s seen during training, as the desired $\mathcal{D}(\boldsymbol{x})$ of a training inlier is $e > 0$). Ultimately, such a spectral property reflects the overall behavior of the model, rather than its restricted behavior on the training set.

3.2 Matrix Product Operator Model

In this section, the details of TNAD is expounded in tensor network notation—for which a brief introduction is included in the appendix while more comprehensive reviews can be found in [5, 26]. The input space $\mathcal{I}$ is assumed to be $[0, 1]^N$ for (flattened) grey-scale images and $\mathbb{R}^N$ for tabular data, where N is the number of features. Given a predetermined map $\phi : \mathbb{R} \to \mathbb{R}^p$ where $p \in \mathbb{N}$ is a parameter known as the **physical dimension**, an input $\boldsymbol{x} = (x_1, ..., x_N) \in \mathcal{I}$ is first passed through (see Figure 2) a feature map $\Phi : \mathcal{I} \to V = \otimes_{j=1}^N \mathbb{R}^p$ defined by

$$\Phi(\boldsymbol{x}) = \phi(x_1) \otimes \phi(x_2) \otimes ... \otimes \phi(x_N) \tag{3}$$

The map ϕ is chosen to satisfy $\|\phi(y)\|_2^2 = 1$ for all $y \in \mathbb{R}$ such that $\|\Phi(\boldsymbol{x})\|_2^2 = \prod_{i=1}^N \|\phi(x_i)\|_2^2 = 1$ for all $\boldsymbol{x} \in \mathcal{I}$, implying that Φ maps all points to the unit hypersphere in V. In our experiments, we used the following $2k$-dimensional embedding

$$\phi(x) = \frac{1}{\sqrt{k}} \left(\cos\left(\frac{\pi}{2}x\right), \sin\left(\frac{\pi}{2}x\right), ..., \cos\left(\frac{\pi}{2^k}x\right), \sin\left(\frac{\pi}{2^k}x\right)\right) \tag{4}$$

which possesses the following natural interpretation for grey-scale images when $p = 2k = 2$. Since $\phi(0), \phi(1)$ are the two standard basis vectors $\boldsymbol{e}_1, \boldsymbol{e}_2$ of $\mathbb{R}^2 = \mathbb{R}^p$, the set of binary-valued images $\mathcal{B} = \{\boldsymbol{x} \in \mathcal{I} : \ x_i \in \{0, 1\} \ \forall 1 \leq i \leq N\}$ is mapped to the standard basis of V. The squared F-norm of our subsequent linear transformation P then obeys $\|P\|_F^2 = \sum_{\boldsymbol{x} \in \mathcal{B}} \|P\Phi(\boldsymbol{x})\|_2^2$. Recalling $\|P\Phi(\boldsymbol{x})\|_2^2$ as the value of TNAD's decision function on an input $\boldsymbol{x}$, $\|P\|_F^2$ is thus conferred the meaning of the total degree of normality predicted by the model on $\mathcal{B}$—which is apt since images with the best contrast should be the most distinguishable. The $\|P\|_F^2$ penalty hence accounts for a representative, exponential subset of input images—a challenging feat for non-tensor network methods. Unfortunately, such an interpretation does not extend to tabular data but the ultimate role of Φ is nevertheless to segregate points close in the L^2-norm of the input space $\mathcal{I}$ by mapping inputs into the exponentially-large space V, buttressing the subsequent linear transformation P.

After the feature map, we learn a tensor $P_{i_1 \ldots i_q}^{j_1 \cdots j_N} : V \to W = \otimes_{j=1}^{q} \mathbb{R}^p$ where $q = \lfloor \frac{N-1}{S} \rfloor + 1$ for some parameter $S \in \mathbb{N}$ referred to as the **spacing**. Our parameterization of P in terms of rank-3 and 4 tensors is the variant of the Matrix Product Operator (MPO) tensor network in Figure 3.

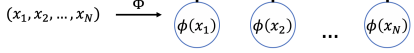

Figure 2: TNAD embedding layer.

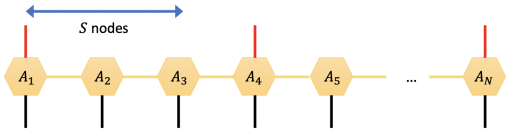

Figure 3: Matrix product operator parameterization for P.

The modified MPO only has an outgoing red leg every S nodes, beginning from the first, to allow $\dim W \ll \dim V$. The red legs again have dimension p while the gold legs have dimension b, which is another parameter known as the **bond dimension**. Intuitively, the gold legs are responsible for capturing correlations between features, for which a larger value of b is desirable. Next, tensor networks for TNAD's decision function and training penalty are depicted in Figures 4 and 5 respectively.

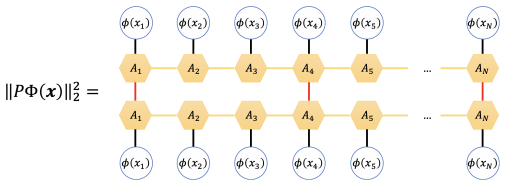

Figure 4: Squared norm of transformed vector.

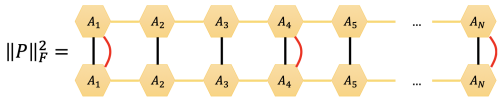

Figure 5: Squared F-norm of P.

Weaving the above together, our overall loss function over a batch of B instances $\boldsymbol{x}_i$ is given by

$$\mathcal{L}_{batch} = \frac{1}{B} \sum_{i=1}^{B} \left(\log \|P\Phi(\boldsymbol{x}_i)\|_2^2 - 1 \right)^2 + \alpha \text{ReLU}(\log \|P\|_F^2) \tag{5}$$

where α is a hyperparameter that controls the trade-off between TNAD's fit around training points and its overall tendency to predict normality. In words, P only sees normal instances during training which it tries to map to vectors on a hypersphere of radius $\sqrt{e}$, but it is simultaneously deterred from mapping other unseen instances to vectors of non-zero norm due to the $\|P\|_F^2$ penalty. The $\log$'s are taken to stabilize the optimization by batch gradient descent since the value of a large tensor network can fluctuate by a few orders of magnitude with each descent step even with a tiny learning rate. Finally, the ReLU function is applied to the F-norm penalty to preclude the trivial solution of $P = 0$.

3.3 Contraction Order and Complexity

The tensor networks for $\|P\Phi(\boldsymbol{x})\|_2^2$ and $\|P\|_F^2$ can be computed by a standard series of contractions. As depicted in Figure 6, the initial step in computing $\|P\Phi(\boldsymbol{x})\|_2^2$ is vertically contracting the black legs. In practice, only the bottom half of the network is contracted before it is duplicated and attached to itself. An alternative first step, that is more parallelizable, is also explained in the appendix.

At this juncture, observe that both $\|P\|_F^2$ and the resulting network for $\|P\Phi(\boldsymbol{x})\|_2^2$ are of the form in Figure 7, which can be computed efficiently by repeated zig-zag contractions. The overall time complexities of computing $\|P\Phi(\boldsymbol{x})\|_2^2$ and $\|P\|_F^2$ are $O\left(Nb^2(b+p)\left(\frac{p}{S}+1\right)\right)$ and $O\left(Nb^3p\left(\frac{p}{S}+1\right)\right)$, where only the former is needed during prediction. Meanwhile, the overall space complexity of TNAD is $O\left(Nb^2p\left(\frac{p}{S}+1\right)\right)$. Notably, all quantities in Eqn 5 can be computed exactly in time and space linear in N, forming the primary motivation behind our MPO parameterization. That said, we expect there to be better-performing tensor networks for P which are either costlier to compute exactly or require approximations to be tractable.

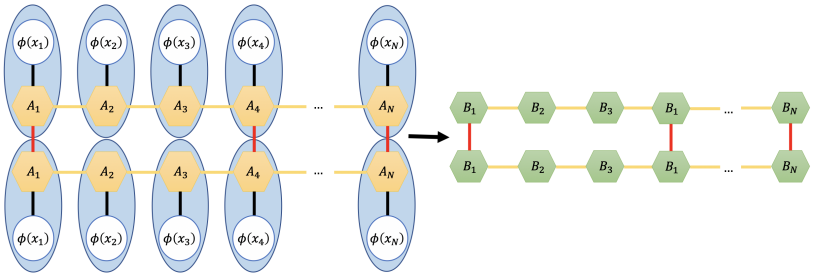

Figure 6: Initial steps in computing $\|P\Phi(\boldsymbol{x})\|_2^2$

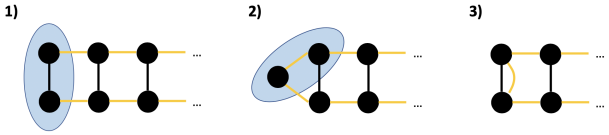

Figure 7: Zig-zag contraction that is repeated till completion.

4 Experiments

The effectiveness of TNAD as a general AD model is verified on both image and tabular datasets. The Area Under the Receiver Operating Characteristic curve (AUROC) is adopted as a threshold-agnostic metric for all experiments. TNAD was implemented with the TensorNetwork library [32] and optimized by batch gradient descent with the ADAM optimizer [19] in its default settings.

General Baselines: The general AD baselines evaluated are: One-Class SVM (OC-SVM) [23], Isolation Forest (IF) [21], and GOAD [4]. OC-SVM and IF are traditional anomaly detection algorithms known to perform well on general data while GOAD is a recent, state-of-the-art self-supervised algorithm with different transformation schemes for image and tabular data. OC-SVM and IF were taken off-shelf from the Scikit-Learn toolkit [28] while GOAD experiments were run with the official code of [4]. For all OC-SVM experiments, the RBF kernel was used and a grid sweep was conducted for the kernel coefficient $\gamma \in \{2^{-10}, ..., 2^{-1}\}$ and the margin parameter $\nu \in \{0.01, 0.1\}$ according to the test set performance in hindsight—providing OC-SVM a supervised advantage. For all IF experiments, the number of trees and the sub-sampling size were set to 100 and 256 respectively, as recommended by the original paper. GOAD parameters are reported in the specific subsections.

Remark about experimental protocol: Model selection has traditionally been a difficult problem in one-class AD since anomalies are needed to determine the AUROC but are presumed to be unavailable at the juncture of training. Even if anomalies were available, using them in a validation set also implicitly provides anomaly information, violating the one-class setting. As such, we adhered to the train-test protocol in the literature (e.g. [4, 14, 33]). We fix the bond dimension at $b = 5$, which is the largest value that allowed our image experiments to finish in reasonable time, and choose hyperparameters according to heuristics explained in Section 4.2. Subsequently, we conduct separate experiments which suggest that TNAD is insensitive to a wide regime of hyperparameter choices.

4.1 Image Experiments

Datasets: Our image experiments were conducted on the *MNIST* [20] and *Fashion-MNIST* [44] datasets, each comprising 60000 training and 10000 test examples of 28×28 grey-scale images belonging to ten classes. In each set-up, one particular class was deemed as the inliers and all original training instances corresponding to that class were retrieved to form the new training set, containing roughly 6000 examples. The trained models were then evaluated on the untouched test set.

Additional Image Baselines: To illustrate the strengths of our approach, we include further comparisons to Deep SVDD (DSVDD) [41] and ADGAN [9], which entail convolutional networks. DSVDD experiments were performed with the original code while ADGAN results are reported from [9, 14].

Preprocessing: For all models besides DSVDD, the pixel values of the grey-scale images were divided by 255 to obtain a float in the range $[0, 1]$. Due to the computational complexity of TNAD, a $(2, 2)$-max pool operation with stride $(2, 2)$ was also performed to reduce the size of the images to 14×14 only for our model. In the cases of TNAD, OC-SVM and IF, the images were flattened before they were passed to these models—which thus do not exploit the inherent locality of the images, as contrasted with the convolutional architectures employed by all other models. For GOAD, the images were zero-padded to size 32×32 to be compatible with the official implementation designed for *CIFAR-10*. Finally, for DSVDD, the images were preprocessed with global contrast normalization in the L^1-norm and subsequent min-max scaling to the interval $[0, 1]$, following the original paper.

Baseline Parameters: The convolutional architectures and hyper-parameters of all deep baselines (GOAD, DSVDD, ADGAN) follow their original work. GOAD was run with the margin parameter $s = 1$ and the geometric transformations of GEOM [14] involving flips, translations and rotations. DSVDD was run with $\nu = 0.1$ and a two-phased training scheme as described in the original paper.

TNAD Parameters: TNAD was run with physical dimension $p = 2$, spacing $S = 8$, sites $N = 14 \times 14 = 196$ and margin strength $\alpha = 0.4$. All tensors were initialized via a normal distribution with standard deviation 0.5. As an aside, TNAD is sensitive to initialization for large N since it successively multiplies many tensors, causing the final result to vanish or explode if each tensor is too small or big—we found a standard deviation of 0.5 to be suitable for $N = 196$ and the final performance of TNAD to not vary significantly once it was initialized in a reasonable regime. As a further precaution, TNAD was first trained for 20 "cold-start" epochs with learning rate 2×10^{-5} to circumvent unfortunate initializations and a subsequent 280 epochs with initial learning rate 2×10^{-3} decaying exponentially at rate 0.01, for a total of 300 epochs. A small batch size $B = 32$ was used due to memory constraints. Finally, since only the $\log$'s of tensor network quantities are desired, we employ the trick of rescaling tensors by their element of largest magnitude during contractions and subsequently adding back the $\log$ of the rescaling to stabilize computations.

Results and Discussion: Our results on image datasets are presented in Table 1. The mean AUROC across ten successful trials are reported for each model and each class chosen as the inliers. Occasionally, GOAD experienced "hypersphere collapse" while TNAD encountered numerical instabilities which led to *nan* outputs—these trials were removed. Ultimately, TNAD produces consistently strong results and notably emerges second out of all evaluated models on *MNIST*, surpassing all convolutional architectures besides GOAD despite not exploiting the innate structure of images. Furthermore, TNAD shows the lowest variation in performance other than the deterministic OC-SVM, possibly attributable to its linearity. OC-SVM exhibits a comparably strong performance though it was admittedly optimized in hindsight. Intriguingly, the common denominator of TNAD and OC-SVM operating in enormous spaces—exponentially large for TNAD and implicitly infinite-dimensional for OC-SVM (via the RBF kernel)—indicates that the adjacency of image pixels may not be too important in *MNIST* and *Fashion-MNIST*, as long as the transformed feature space is large enough.

Attaining the highest AUROC on most classes, GOAD undeniably triumphs all other evaluated models on images. However, GOAD's performance dip on *MNIST* digits $\{0, 1, 8\}$, which are largely unaffected by the horizontal flip and 180° rotation used in its transformations, suggests that its success relies on identifying transformations that leverage the underlying structure of images. Indeed, its authors [4] acknowledge that the random affine transformations used in GOAD's tabular experiments degraded its performance on images. As such, TNAD's performance as a general AD model is especially encouraging, considering its ignorance of the inputs being images. For image-specific applications, it is likely for tensor network parameterizations of P that capture two-dimensional correlations, such as PEPS [42], to fare better.

Table 1: Mean AUROC scores (in %) and standard errors on *MNIST* and *Fashion MNIST*

Dataset	c	SVM	IF	GOAD	DSVDD	ADGAN	TNAD
MNIST	0	**99.5**	96.4 ± 0.6	98.4 ± 0.4	98.2 ± 0.6	**99.5**	99.2 ± 0.0
	1	**99.9**	99.4 ± 0.1	96.5 ± 0.9	99.6 ± 0.1	**99.9**	99.8 ± 0.0
	2	92.6	75.1 ± 1.8	$\mathbf{99.6 \pm 0.0}$	90.3 ± 2.5	**93.6**	92.7 ± 0.3
	3	93.8	83.0 ± 1.2	$\mathbf{98.6 \pm 0.2}$	90.1 ± 2.3	92.1	$\mathbf{96.0 \pm 0.3}$
	4	**97.1**	87.0 ± 0.9	$\mathbf{99.2 \pm 0.2}$	94.5 ± 1.1	93.6	94.9 ± 0.3
	5	**95.5**	74.8 ± 0.8	$\mathbf{99.5 \pm 0.1}$	87.1 ± 1.4	94.4	95.1 ± 0.3
	6	98.8	86.9 ± 0.9	$\mathbf{99.9 \pm 0.0}$	98.8 ± 0.3	96.7	$\mathbf{98.9 \pm 0.0}$
	7	96.6	91.2 ± 0.7	$\mathbf{98.2 \pm 0.5}$	94.9 ± 0.6	96.8	$\mathbf{97.1 \pm 0.3}$
	8	90.8	73.7 ± 1.1	$\mathbf{96.9 \pm 0.4}$	93.3 ± 1.1	85.4	$\mathbf{94.9 \pm 0.3}$
	9	96.3	88.1 ± 0.6	$\mathbf{99.0 \pm 0.2}$	96.3 ± 0.9	95.7	$\mathbf{97.2 \pm 0.1}$
	avg	96.1	84.6	**98.6**	94.3	94.7	**96.6**
Fashion-MNIST	0	91.9	91.0 ± 0.2	$\mathbf{93.4 \pm 0.6}$	90.1 ± 0.8	89.9	$\mathbf{92.5 \pm 0.2}$
	1	**99.0**	97.6 ± 0.1	98.6 ± 0.2	$\mathbf{98.7 \pm 0.1}$	81.9	97.5 ± 0.1
	2	89.4	87.1 ± 0.4	$\mathbf{90.4 \pm 0.6}$	88.1 ± 0.7	87.6	$\mathbf{90.6 \pm 0.1}$
	3	**94.2**	93.2 ± 0.3	91.0 ± 1.5	$\mathbf{93.4 \pm 1.0}$	91.2	91.8 ± 0.2
	4	90.6	90.2 ± 0.5	$\mathbf{91.4 \pm 0.4}$	$\mathbf{91.8 \pm 0.5}$	86.5	90.5 ± 0.1
	5	91.8	$\mathbf{92.8 \pm 0.2}$	$\mathbf{94.7 \pm 0.7}$	89.1 ± 0.7	89.6	87.5 ± 0.3
	6	**83.5**	79.5 ± 0.6	$\mathbf{83.2 \pm 0.6}$	80.3 ± 0.8	74.3	82.7 ± 0.1
	7	**98.8**	98.3 ± 0.1	98.3 ± 0.5	98.4 ± 0.2	97.2	$\mathbf{98.9 \pm 0.0}$
	8	89.9	88.5 ± 0.9	$\mathbf{98.8 \pm 0.2}$	$\mathbf{92.9 \pm 1.3}$	89.0	92.0 ± 0.4
	9	98.2	97.6 ± 0.3	$\mathbf{99.3 \pm 0.2}$	$\mathbf{99.0 \pm 0.1}$	97.1	97.8 ± 0.2
	avg	**92.7**	91.6	**93.9**	92.2	88.4	92.2

In each row, the c-th class is taken as the normal instance while all other classes are anomalies. The top two results in each experiment are highlighted in bold. OC-SVM, which is abbreviated as SVM above, did not show variations in performance once it has converged so no standard errors are reported. ADGAN's results were borrowed from [9, 14] which did not include error bars.

4.2 Tabular Experiments

Datasets: We evaluate TNAD and other general baselines on 5 real-world ODDS [29] datasets derived from the UCI repository [11]: *Wine*, *Lympho*, *Thyroid*, *Satellite*, *Forest*. These were selected to exhibit a variety of dataset sizes, features and anomalous proportions—detailed information regarding them is presented in Table 2. Following the procedure of [4], all models were trained on half of the normal instances and evaluated on the other half plus the anomalies.

Preprocessing: The data was normalized so that training examples had zero mean and unit variance.

Baseline Parameters: GOAD employs random affine transformations with output dimension r for self-supervision on tabular data and trains a fully-connected classifier with hidden size h and leaky-ReLU activations. We adhere to the hyperparameter choices in the original paper, setting $r = 64$, $h = 32$ and 25 training epochs for the large-scale dataset *Forest* and $r = 32$, $h = 8$ and 1 training epoch for all other smaller-scale datasets. Finally, we also train DAGMM [47] using its original *Thyroid* architecture for epochs in $\{10000, 20000, 30000, 40000\}$ and report the best results.

TNAD Parameters: The dimensions of the input and output spaces V and W, which depend on the parameters N, p and S, are crucial to TNAD. As the number of features N varies across datasets, we choose p and S according to the following heuristics. We set $S = \lfloor \frac{N}{25} \rfloor + 1$ and subsequently choose p such that $10^4 \leq \dim W = p^{\lfloor \frac{N-1}{S} \rfloor + 1} \leq 10^{12}$, with a preference for smaller p on smaller datasets. The first rule imposes an appropriate nullspace of P while the second ensures that the dimension is large enough to exploit the prowess of tensor networks while concurrently small enough to ensure stable training (see Section 4.3). On the smallest datasets *Wine* and *Lympho*, we set $\alpha = 0.3$ while the other datasets used $\alpha = 0.1$. To motivate this choice, one expects a learnt $\|P\|_F^2$ to scale linearly with the training set size $|T|$ (consider the case of memorizing the training set). However, the first term in the loss function in Eqn 5 is independent of $|T|$ so for both terms in the loss function to be

comparable, α should be $\Theta\left(\frac{1}{\log|T|}\right)$. The two-phased training scheme is adopted as before for the small tensor networks in *Wine*, *Lympho*, *Thyroid* while lower learning rates of 5×10^{-6} damped and 5×10^{-4} undamped were used for larger models to stabilize training. A batch size of 32 was used for all datasets besides *Forest* which used 512. A summary of TNAD parameters is provided in Table 2.

Table 2: Information about ODDS datasets, sorted by size, and TNAD parameters used.

Dataset	# Train	# Test		TNAD Parameters					
		Normal	Anomalous	N	p	S	$\dim W$	α	lr
Wine	59	60 (85.7%)	10 (14.3%)	13	4	1	6.7e7	0.3	2e-3
Lympho	71	71 (92.2%)	6 (7.8%)	18	2	1	2.6e5	0.3	2e-3
Thyroid	1839	1840 (95.2%)	93 (4.8%)	6	6	1	4.7e4	0.1	2e-3
Satellite	2199	2200 (51.9%)	2036 (48.1%)	36	4	2	6.9e10	0.1	5e-4
Forest	141650	141651 (98.1%)	2747 (1.9%)	10	8	1	1.1e9	0.1	5e-4

Results and Discussion: Table 3 shows the mean AUROC from our experiments. Due to the large variance caused by its stochastic nature, GOAD was run for 500 trials on small-scale datasets and 100 trials on *Forest*. All other models were run for 10 trials. TNAD is the best performer across all datasets. Its drastic improvements over the respective best baseline and best deep baseline on the smallest datasets *Wine* and *Lympho* bear credence to the F-norm penalty's effectiveness in ensuring a tight fit around scarce inliers. GOAD's poorer performance on *Lympho*, *Satellite* and *Forest* supports the expectation that affine transformations may not suit general data. All-in-all, TNAD is arguably the best AD model out of those compared, given no prior domain knowledge of the underlying data.

Table 3: Mean AUROC scores (in %) and standard errors on ODDS datasets.

Dataset	OC-SVM	IF	GOAD	DAGMM	TNAD
Wine	**60.0**	46.0 ± 8.4	48.2 ± 24.7	51.7 ± 19.3	$\mathbf{97.3 \pm 4.5}$
Lympho	**92.5**	87.1 ± 2.3	68.9 ± 12.3	65.7 ± 16.6	$\mathbf{93.9 \pm 2.2}$
Thyroid	98.8	$\mathbf{99.0 \pm 0.1}$	95.8 ± 1.3	88.8 ± 6.8	$\mathbf{99.0 \pm 0.1}$
Satellite	**79.9**	77.2 ± 0.9	60.6 ± 5.3	72.1 ± 4.7	$\mathbf{81.3 \pm 0.5}$
Forest	**97.7**	71.7 ± 2.6	64.6 ± 4.7	60.9 ± 8.9	$\mathbf{98.8 \pm 0.6}$

The top two results in each experiment are highlighted in bold. OC-SVM did not show variations in performance once it has converged so no standard errors are reported.

4.3 Sensitivity to Hyperparameters

Table 4 reports the results of a grid sweep over hyperparameters α, b, S on *MNIST* digit "0" downsampled to 7×7. As expected, a steady albeit small improvement in TNAD's performance is observed as b is increased. Meanwhile, TNAD's consistent performance with S except for the extreme value $S = 48$ suggests that whether an image $\boldsymbol{x}$ is digit "0" only depends on the inner product of $\Phi(\boldsymbol{x})$ with a few (but more than one) singular vectors in the tensor product space V. In light of this, we also experimented with the limiting case of $S = 50$ and $\dim(W) = 1$ such that P effectively reduces to a Matrix Product State (MPS) [36, 27]. Such a MPS model failed to fit the training set—confirming that the extra degrees of freedom introduced by the MPO are necessary. Next, TNAD's stable performance with α implies that α only needs to be large enough for the F-norm penalty to set in. Finally, the effect of varying p was tested on another ODDS dataset *Breastw*. As shown in Figure 8, the average log decision score $\frac{1}{|T|}\sum_{\boldsymbol{x}\in T}\log \mathcal{D}(\boldsymbol{x})$ oscillates around one during training, due to the two conflicting objectives in the loss function. For small p, the oscillation amplitude decays with epochs so TNAD fits the training set. Unsurprisingly, this is not observed for large p, since successive multiplications of large tensors amplify individual changes caused by gradient descent. This is a call for improved optimization schemes (e.g. sweeping techniques) for future work. With the current gradient descent method, it is recommended to plot the average training log decision score with training epochs and select the largest value of p (guided by our heuristics) for which the oscillation eventually stabilizes.

Table 4: Hyperparameter sweep on *MNIST* with defaults $b = 5, S = 8, \alpha = 0.3$.

b	ROC	S	ROC	α	ROC
2	83.4	1	97.0	3e-3	83.8
3	96.9	2	97.1	1e-2	85.5
4	97.0	4	97.2	3e-2	92.1
5	97.2	8	97.2	1e-1	97.0
6	97.3	12	97.2	3e-1	97.2
7	97.4	24	97.2	1e0	97.2
8	97.6	48	94.1	3e0	97.3

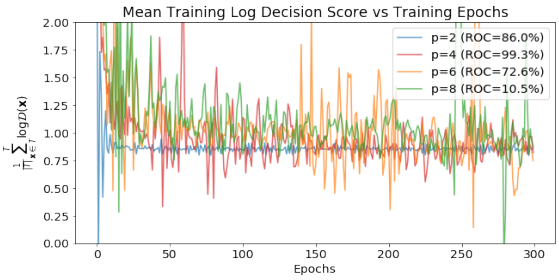

Figure 8: Train log decision score on *Breastw* for varying p (Test ROC) and defaults $b = 5, S = 1, \alpha = 0.3$.

4.4 Empirical time and memory costs

Finally, Table 5 presents the empirical training time, inference time per example, and (training) memory costs of the general AD models in our experiments. OC-SVM and IF were run on a Intel Xeon CPU @ 2.20GHz while GOAD and TNAD were run on a NVIDIA Tesla V100 GPU. As expected, the first general observation is that the time and memory costs of the classical models (OC-SVM and IF) are roughly two orders of magnitude smaller than the heftier GOAD and TNAD models. Next, GOAD's training time and memory scale unfavourably with the dataset size due to its expensive dataset transformations, as reflected in *MNIST* and *Forest*. Finally, although TNAD generally requires the longest training time, its inference time is relatively shorter since it only has to compute the decision function $\mathcal{D}$ during inference as opposed to both $\mathcal{D}$ and $\|P\|_F^2$ during training.

Remarkably, TNAD's inference time is 26 times faster than GOAD's on *MNIST*, despite GOAD being expedited by convolutions. Meanwhile, TNAD's inference time on tabular datasets is similar to GOAD's, with the former attaining an edge on *Thyroid*, *Satellite*, and *Forest*. This discrepancy in the relative inference times on image and tabular datasets primarily stems from GOAD's deeper and shallower architectures on the respective datasets. Ultimately, TNAD and GOAD consume comparable resources, albeit both significantly more than OC-SVM and IF.

Table 5: Training times (s), inference times (s / example) and memory costs (Mb) of AD models.

Dataset	OC-SVM			IF			GOAD			TNAD		
	Train	Infer	Mem	Train	Infer	Mem	Train	Infer	Mem	Train	Infer	Mem
MNIST	37	0.0045	16	5.8	1.6e-4	11	1400	0.034	4400	6200	0.0013	380*
Wine	0.0018	1.7e-5	0.15	0.87	0.0027	0.83	5.7	0.0042	100	170	0.0092	970
Lympho	0.0017	1.8e-5	0.15	0.83	0.0022	0.94	6.5	0.0041	160	220	0.0094	980
Thyroid	0.0085	2.6e-6	0.20	0.92	1.2e-4	1.6	150	0.0034	200	180	0.0014	970
Satellite	0.043	1.5e-5	0.51	0.98	7.4e-5	1.1	170	0.0035	230	770	4.9e-4	980
Forest	24	5.3e-5	0.78	4.7	2.1e-5	1.3	4600	3.9e-4	4700	740	3.4e-6	960

* TNAD's memory cost on *MNIST* is relatively smaller due to a large spacing $S = 8$.

5 Conclusion

In this paper, we have introduced TNAD as an adept anomaly detection model for general data. To the best of our knowledge, it is the first instance of a tensor network model that is competitive with state-of-the-art classical and deep methods, possibly pushing the rising field of tensor networks for machine learning into the practical regime. All-in-all, TNAD demonstrates how even elementary tensor network algorithms can become potent tools when thoughtfully applied to suitable problems.

Acknowledgements: The authors are grateful to X – The Moonshot Factory for supporting this research. X, formerly known as Google[x], is part of the Alphabet family of companies, which includes Google, Verily, Waymo, and others (www.x.company).

References

[1] Samet Akcay, Amir Atapour-Abarghouei, and Toby P. Breckon. Ganomaly: Semi-supervised anomaly detection via adversarial training. In *Computer Vision – ACCV 2018*, pages 622–637. Springer International Publishing, 2019.

[2] Jerone Andrews, Edward Morton, and Lewis Griffin. Detecting anomalous data using auto-encoders. *International Journal of Machine Learning and Computing*, 6:21, 2016.

[3] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[4] Liron Bergman and Yedid Hoshen. Classification-based anomaly detection for general data. In *International Conference on Learning Representations*, 2020.

[5] Jacob Biamonte and Ville Bergholm. Tensor networks in a nutshell, 2017.

[6] Raghavendra Chalapathy, Aditya Krishna Menon, and Sanjay Chawla. Anomaly detection using one-class neural networks, 2018.

[7] Jinghui Chen, Saket Sathe, Charu C. Aggarwal, and Deepak S. Turaga. Outlier detection with autoencoder ensembles. In *SDM*, 2017.

[8] Song Cheng, Lei Wang, Tao Xiang, and Pan Zhang. Tree tensor networks for generative modeling. *Phys. Rev. B*, 99:155131, Apr 2019.

[9] Lucas Deecke, Robert Vandermeulen, Lukas Ruff, Stephan Mandt, and Marius Kloft. Image anomaly detection with generative adversarial networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 3–17, 2018.

[10] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning, 2016.

[11] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[12] Sarah M. Erfani, Sutharshan Rajasegarar, Shanika Karunasekera, and Christopher Leckie. High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning. *Pattern Recognition*, 58:121 – 134, 2016.

[13] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations, 2018.

[14] Izhak Golan and Ran El-Yaniv. Deep anomaly detection using geometric transformations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 9781–9791, 2018.

[15] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2672–2680. MIT Press, 2014.

[16] Zhao-Yu Han, Jun Wang, Heng Fan, Lei Wang, and Pan Zhang. Unsupervised generative modeling using matrix product states. *Phys. Rev. X*, 8:031012, Jul 2018.

[17] Simon Hawkins, Hongxing He, Graham J. Williams, and Rohan A. Baxter. Outlier detection using replicator neural networks. In *Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery*, page 170–180. Springer-Verlag, 2002.

[18] Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and Dawn Song. Using self-supervised learning can improve model robustness and uncertainty. In *Advances in Neural Information Processing Systems 32*, pages 15663–15674. MIT Press, 2019.

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[20] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs*, 2, 2010.

[21] F. T. Liu, K. M. Ting, and Z. Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008.

[22] Alireza Makhzani and Brendan Frey. Winner-take-all autoencoders. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, page 2791–2799. MIT Press, 2015.

[23] Larry M. Manevitz and Malik Yousef. One-class svms for document classification. *J. Mach. Learn. Res.*, 2:139–154, 2002.

[24] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *Proceedings of the 21th International Conference on Artificial Neural Networks - Volume Part I*, page 52–59. Springer-Verlag, 2011.

[25] Jacob Miller, Guillaume Rabusseau, and John Terilla. Tensor networks for probabilistic sequence modeling, 2020.

[26] Román Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349:117–158, 2014.

[27] Stellan Östlund and Stefan Rommer. Thermodynamic limit of density matrix renormalization. *Phys. Rev. Lett.*, 75:3537–3540, 1995.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[29] Shebuti Rayana. ODDS library, 2016.

[30] Justin Reyes and Miles Stoudenmire. A multi-scale tensor network architecture for classification and regression, 2020.

[31] Charles Richter and Nicholas Roy. Safe visual navigation via deep learning and novelty detection. In *Robotics: Science and Systems*, 2017.

[32] Chase Roberts, Ashley Milsted, Martin Ganahl, Adam Zalcman, Bruce Fontaine, Yijian Zou, Jack Hidary, Guifre Vidal, and Stefan Leichenauer. Tensornetwork: A library for physics and machine learning, 2019.

[33] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4393–4402, 2018.

[34] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, page 4–11. Association for Computing Machinery, 2014.

[35] Thomas Schlegl, Philipp Seeböck, Sebastian M. Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *IPMI*, 2017.

[36] Ulrich Schollwöck. The density-matrix renormalization group in the age of matrix product states. *Annals of Physics*, 326(1):96–192, 2011.

[37] Philipp Seebock, Sebastian M. Waldstein, Sophie Klimscha, Hrvoje Bogunovic, Thomas Schlegl, Bianca S. Gerendas, Rene Donner, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised identification of disease marker candidates in retinal oct imaging data. *IEEE Transactions on Medical Imaging*, 38(4):1037–1047, 2019.

[38] Raghavendra Selvan and Erik B Dam. Tensor networks for medical image classification, 2020.

[39] E. Miles Stoudenmire and David J. Schwab. Supervised learning with quantum-inspired tensor networks. In *Advances in Neural Information Processing Systems 29*, NIPS'16, page 4799–4807, 2016.

[40] Edwin Stoudenmire. Learning relevant features of data with multi-scale tensor networks. *Quantum Science and Technology*, 3, 12 2017.

[41] David M. J. Tax and Robert P. W. Duin. Support vector data description. *Mach. Learn.*, 54(1): 45–66, 2004.

[42] F. Verstraete and J. I. Cirac. Renormalization algorithms for quantum-many body systems in two and higher dimensions, 2004.

[43] Steven R. White. Density matrix formulation for quantum renormalization groups. *Phys. Rev. Lett.*, 69:2863–2866, 1992.

[44] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[45] Dan Xu, Elisa Ricci, Yan Yan, Jingkuan Song, and Nicu Sebe. Learning deep representations of appearance and motion for anomalous event detection, 2015.

[46] Houssam Zenati, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek, and Vijay Ramaseshan Chandrasekhar. Efficient gan-based anomaly detection, 2018.

[47] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International Conference on Learning Representations*, 2018.

Appendix to Anomaly Detection with Tensor Networks

1 Introduction to Tensor Networks

A simplified overview of tensor networks is provided to make the paper accessible to a wider audience. A tensor A of rank k can be regarded as a generalized array whose $(i_1, ..., i_k)$-th element is $A_{i_1 \dots i_k} \in \mathbb{R}$. A common abuse of notation is to refer to the tensor itself as $A_{i_1 \dots i_k}$ since the indices $i_1, ..., i_k$ are "free" in the sense that they can be varied arbitrarily and the collection of all possible indices uniquely determines the tensor. In tensor network notation, a tensor of rank k is represented by a node with k legs, where each leg represents a certain index or axis of the tensor. As depicted in Figure 1a) and b), a rank-1 tensor or vector v_i has a single leg denoting index i while a rank-2 tensor or matrix M_i^j (upper index to be explained below) has two legs denoting indices i and j.

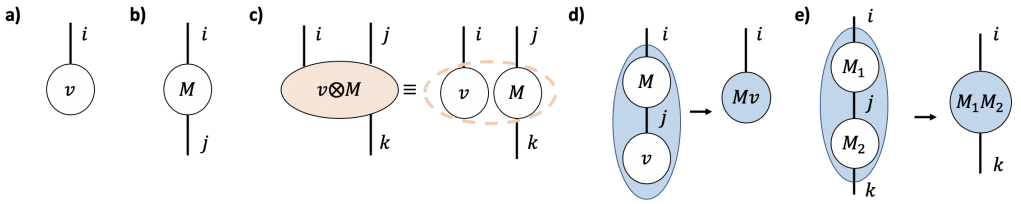

Figure 1: Common tensors and operations. a) vector v_i, b) matrix M_i^j, c) tensor product $(v \otimes M)_{ij}^k = v_i M_j^k$, d) matrix product with vector $(Mv)_i = M_i^j v_j$, e) matrix product with matrix $(M_1 M_2)_i^k = (M_1)_i^j (M_2)_j^k$.

Now, one can interpret the same tensor $A_{i_1 \dots i_k}$ in various ways. Letting the size of the axis along i_j be d_j, one can firstly view $A_{i_1 \dots i_k}$ as a vector in a $\prod_{j=1}^k d_j$-dimensional vector space whose standard basis vectors $\{e_{i_1,\dots,i_k}\}$ are indexed by $i_1, ..., i_k$ such that the coordinate of A with respect to $e_{i_1,\dots,i_k}$ is $A_{i_1 \dots i_k}$. This vector space is known as the tensor product space $\otimes_{j=1}^k \mathbb{R}^{d_j}$. Another perspective partitions the indices $i_1, ..., i_k$ at some i_p for $1 \leq p \leq k$ and re-expresses A as $A_{i_{p+1} \dots i_k}^{i_1 \dots i_p}$. In this case, we can see A as a linear transformation from a $\prod_{j=1}^p d_j$-dimensional vector space $\otimes_{j=1}^p \mathbb{R}^{d_j}$ with standard basis vectors $\left\{e_{i_1,\dots,i_p}\right\}$ to a $\prod_{j=p+1}^k d_j$-dimensional vector space $\otimes_{j=p+1}^k \mathbb{R}^{d_j}$ with standard basis vectors $\left\{e'_{i_{p+1},\dots,i_k}\right\}$. The coordinate of the transformed basis vector $A\left(e_{i_1,..,i_p}\right)$ with respect to $e'_{i_{p+1},\dots i_k}$ is $A_{i_{p+1} \dots i_k}^{i_1 \dots i_p}$. Viewing a $n \times m$ matrix M as a linear transformation from $\mathbb{R}^m$ to $\mathbb{R}^n$, we hence write it as M_i^j. Indeed, one can easily check that the i-th coordinate of the transformed j-th basis vector of the input space is M_i^j. Thus, one can generally deem a tensor as a linear transformation from the input space represented by its upper indices to the output space represented by its lower indices. It turns out that our model will employ a tensor with signature $P_{i_1 \dots i_q}^{j_1 \dots j_N}$ where all indices have dimension p. Then, P can be seen as a linear transformation from $V = \otimes_{j=1}^N \mathbb{R}^p$ to $W = \otimes_{i=1}^q \mathbb{R}^p$. Henceforth, we will adopt the notation for tensors with upper and lower indices for clarity, without affecting how tensors are reflected in tensor network diagrams.

Next, there are two important operations that enable the construction of new tensors. The tensor product $C = A \otimes B$ of tensors $A_{i_1 \dots i_p}^{j_1 \dots j_q}$ and $B_{k_1 \dots k_{p'}}^{l_1 \dots l_{q'}}$, with respective ranks $p + q$ and $p' + q'$, is

First Workshop on Quantum Tensor Networks in Machine Learning, 34th Conference on Neural Information Processing Systems (NeurIPS 2020).

a rank-$(p+p')+(q+q')$ tensor with elements defined by $C^{j_1\ldots j_q l_1\ldots l_{q'}}_{i_1\ldots i_p k_1\ldots k_{p'}} = A^{j_1\ldots j_q}_{i_1\ldots i_p} B^{l_1\ldots l_{q'}}_{k_1\ldots k_{p'}}$. In tensor network notation, the nodes A and B are placed side-by-side and C can be treated as their super-node with $(p+p')+(q+q')$ legs, as shown in Figure 1c). In our main text, we repeatedly apply the (associative) tensor product to build high-rank embeddings from a rank-1 embedding. Concretely, given a map $\phi : \mathbb{R} \to \mathbb{R}^p$ which only takes a single input, we construct a multivariable map $\Phi : \mathbb{R}^N \to \otimes_{i=1}^N \mathbb{R}^p$ defined by $\Phi(\boldsymbol{x}) = \phi(x_1) \otimes \ldots \otimes \phi(x_N)$, where $\boldsymbol{x} = (x_1, \ldots, x_N)$. Its elements are given by $\Phi(\boldsymbol{x})_{i_1\ldots i_N} = \phi(x_1)_{i_1} \times \ldots \times \phi(x_N)_{i_N}$. In tensor network notation,

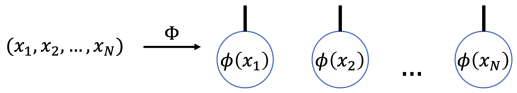

Figure 2: Embedding layer of our TNAD model.

The other operation, known as contraction, was inspired by Einstein's summation convention. Whenever a single index appears as both a lower and upper index, it is implicitly summed over. This is convenient in "multiplying tensors" as the product of a $n \times m$ matrix M^j_i with a m-dimensional matrix v_j can be written as $(Mv)_i = \sum_{j=1}^m M^j_i v_j = M^j_i v_j$, where the summation was explicit in the second expression. Similarly, the product of a $l \times m$ matrix $(M_1)^j_i$ with a $m \times n$ matrix $(M_2)^k_j$ becomes $(M_1 M_2)^k_i = \sum_{j=1}^m (M_1)^j_i (M_2)^k_j = (M_1)^j_i (M_2)^k_j$. Thus, operations such as matrix multiplication can be performed intuitively by just keeping track of indices appearing on both sides of the equation. Ultimately, the outcome of a contraction between two tensors is a single tensor with the combined indices of both tensors, minus the contracted indices.

In tensor network notation, the contraction of a repeated index is illustrated by connecting the two legs that the index appears in. As contracted indices (i.e. connected legs) do not appear in the result, only "dangling" or free legs become indices in the result of a diagram. Note that there is a distinction between a representing a contraction (via connecting legs in a diagram) and actually performing the contraction (computing the sums along connected legs and absorbing the two contracted nodes into a single node)—the latter will be depicted in our paper by encapsulating to be contracted nodes with a blue oval (see Figure 1d and e). This distinction is important because the time complexity of computing the result of a diagram depends on the order of performing contractions. As such, one usually identifies the entire diagram to be computed, before finding an efficient contraction sequence.

In our main paper, we adopt the following parameterization for a tensor $P^{j_1\ldots j_N}_{i_1\ldots i_q}$, where $q = \lfloor \frac{N-1}{S} \rfloor + 1$ for some $S \in \mathbb{N}$, based on a variant of the Matrix Product Operator (MPO) tensor network.

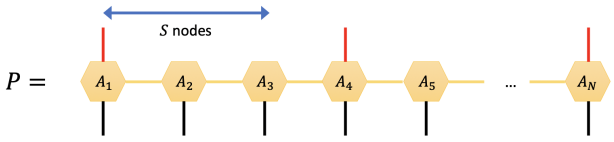

Figure 3: Matrix product operator parameterization for P.

P only has an outgoing red leg every S nodes while its connected gold legs represent contractions. The output vector obtained from performing the transformation P on an embedded vector $\Phi(\boldsymbol{x})$ (defined above) can then be illustrated by connecting the bottom legs of P to those of $\Phi(\boldsymbol{x})$, analogous to the matrix-vector product in Figure 1d. In explicit tensor indices, $(P\Phi(\boldsymbol{x}))_{i_1\ldots i_q} = P^{j_1\ldots j_N}_{i_1\ldots i_q} \Phi(\boldsymbol{x})_{j_1\ldots j_N}$.

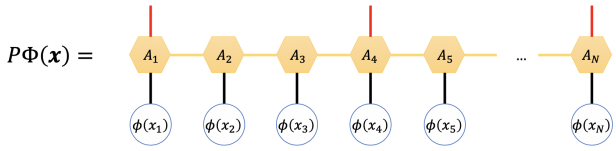

Figure 4: Transformed vector upon action by P

Finally, given a tensor $A^{j_1\dots j_q}_{i_1\dots i_p}$, we can construct a dual tensor A^* by exchanging the lower and upper indices of A, $(A^*)^{i_1\dots i_p}_{j_1\dots j_q} = A^{j_1\dots j_q}_{i_1\dots i_p}$. This is the natural generalization of the transpose operation for matrices which satisfies $\left(M^T\right)^j_i = M^i_j$. Since this is just a reshaping of the tensor, we often refer to the dual tensor of $A^{j_1\dots j_q}_{i_1\dots i_p}$ as $A^{i_1\dots i_p}_{j_1\dots j_q}$ for brevity. A quantity that will be particularly important in our main text is the contraction of a tensor $A^{j_1\dots j_q}_{i_1\dots i_p}$ with its dual $A^{i_1\dots i_p}_{j_1\dots j_q}$; $\|A\|_F^2 = A^{j_1\dots j_q}_{i_1\dots i_p} A^{i_1\dots i_p}_{j_1\dots j_q}$. Interpreting $A^{j_1\dots j_q}_{i_1\dots i_p}$ in terms of a linear transformation, $\|A\|_F^2$ is simply the sum of the squared matrix elements of the transformation and hence refers to the Frobenius norm squared. As a special case, the Frobenius norm of a vector is equivalent to its L^2 norm. If A is represented by a tensor network, the tensor network for $\|A\|_F^2$ can be obtained from duplicating the tensor network of A and connecting corresponding free legs between A and its duplicate (the dual tensor). This is exploited to draw the tensor networks of the squared L^2-norm $\|P\Phi(\boldsymbol{x})\|_2^2$ and the Frobenius norm $\|P\|_F^2$ as

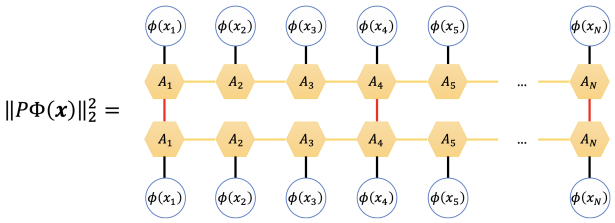

Figure 5: Squared L^2-norm of the transformed vector

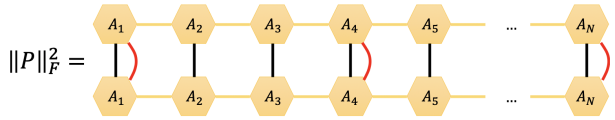

Figure 6: Frobenius norm of P

2 Alternative Contraction Scheme for TNAD's Decision Function

As depicted in Figure 7, alternative initial steps in computing $\|P\Phi(\boldsymbol{x})\|_2^2$ are vertical contractions of black legs followed by right-to-left contractions along horizontal segments between consecutive red legs. As before, only the bottom half of the network is contracted before it is duplicated and attached to itself in practice. Subsequently, the resultant network can again be computed via zig-zag contractions. Although this alternative scheme does not ultimately lead to a smaller overall time complexity, it greatly benefits from parallelism for large spacing S since each worker unit can be responsible for contracting different horizontal blocks between consecutive red legs.

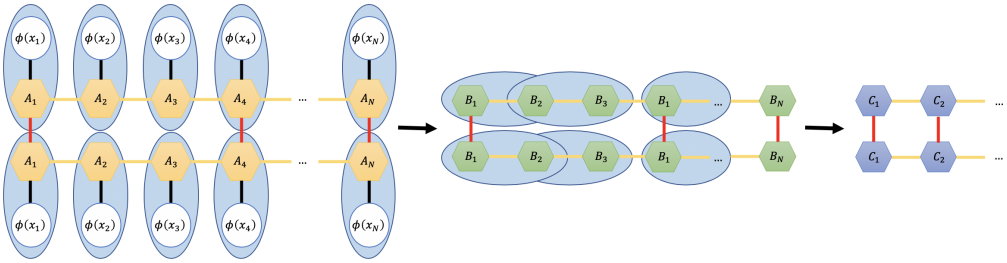

Figure 7: Alternative initial step in computing $\|P\Phi(\boldsymbol{x})\|_2^2$