Improvements to Gradient Descent Methods for Quantum Tensor Network Machine Learning

Fergus Barratt
TyTN
fergus.barratt.12@ucl.ac.uk

James Dborin
TyTN
james.dborin.18@ucl.ac.uk

Lewis Wright
TyTN
lewis.wright@kcl.ac.uk

Abstract

Tensor networks have demonstrated significant value for machine learning in a myriad of different applications. However, optimizing tensor networks using standard gradient descent has proven to be difficult in practice. Tensor networks suffer from initialization problems resulting in exploding or vanishing gradients and require extensive hyperparameter tuning. Efforts to overcome these problems usually depend on specific network architectures, or ad hoc prescriptions. In this paper we address the problems of initialization and hyperparameter tuning, making it possible to train tensor networks using established machine learning techniques. We introduce a 'copy node' method that successfully initializes arbitrary tensor networks, in addition to a gradient based regularization technique for bond dimensions. We present numerical results that show that the combination of techniques presented here produces quantum-inspired tensor network models with far fewer parameters, while improving generalization performance.

1 Introduction

Quantum Tensor Network Machine Learning (QTNML) is a new and rapidly developing area of research. Insights from many-body quantum physics have been combined with machine learning principles, producing novel machine learning models with heretofore unseen properties. This family of models has been applied to a variety of different machine learning problems, such as image classification [17, 1, 13], language modelling [12], and anomaly detection [20].

Despite the success of QTNML, the best method for optimizing tensor networks remains unclear. The Density Matrix Renormalization Group (DMRG) family of optimization algorithms is used throughout the literature. These techniques involve freezing a majority of the tensors whilst updating a small subset at a time. In some formulations pairs of tensors are contracted together. The new combined tensor is optimized and split using a Singular Value Decomposition (SVD). This is the procedure used in [17]. The most obvious alternative is performing gradient descent on all tensors at the same time, which we will call *Direct Gradient Descent* (DGD). DGD is used in [20], where it is noted that the tensor networks are initialized multiple times in order for training to occur.

There are two main advantages of DMRG optimization compared to DGD. Firstly, this method seems more effective at optimizing large tensor networks with many nodes. DGD suffers from exploding or vanishing gradients, which is further explored in the theory section below. There are theoretical reasons to believe that DMRG updates do not suffer from this problem. For example, when the

tensors fulfil certain isometric conditions, they will not suffer from exploding or vanishing gradients [18].

Secondly, DMRG methods naturally allow for automatic bond dimension selection within the tensor network. Only a single hyperparameter, the singular value cut off, needs to be tuned. When using DGD the bond dimension of the tensor network must be set in advance. This creates an extremely large hyperparameter search space, over all different bond dimension combinations. But in practice it is common to set all bond dimensions to be the same and simply search through a single set of bond dimension values.

There are problems with DMRG optimization algorithms however. They are hard to implement on popular machine learning frameworks, like tensorflow[11] and pytorch[15]. Although possible, it is certainly not as straightforward as implementing DGD and could present a stumbling block for widespread adoption of tensor network methods. Furthermore it is not clear how to combine tensors trained using DMRG, and tensors trained using DGD in the same training procedure. This is not an issue for models developed in [17], but in [1] it is clear there are benefits to be gained by combining QTNML methods with standard machine learning elements such as convolutional layers. Finally, there is no generalisation of DMRG algorithms for more complex tensor network structures, such as MERA[19, 16] or Tree Tensor Networks[2], which suit certain data structures better.

1.1 Our Contribution

In this paper we improve on the direct gradient descent scheme for tensor networks, and in doing so surpass the performance found in DMRG-inspired optimization algorithms. We present:

A An initialization scheme for quantum tensor network models which prevents exploding or vanishing gradients in the training process. This is supported with both theoretical and empirical justification. Models trained with this initialization show an increase in generalization performance over those trained using DMRG-inspired methods [17].

B A scheme to regularize the bond dimension during the gradient update process. The bond dimension is allowed to grow to increase model complexity if needed, whilst also penalizing growth in the size of the model. Models trained with this scheme show a significant decrease in parameter count for no significant loss of accuracy.

1.2 Tensor Networks

Many QTNML methods follow a similar pattern. A feature map, ϕ, is used to embed the input data vector, $\vec{x}$ into a much larger feature space,

$$\phi(\vec{x})_{\sigma_1 \sigma_2 \ldots \sigma_N} = \phi(x_0)_{\sigma_0} \otimes \phi(x_1)_{\sigma_1} \otimes \cdots \otimes \phi(x_N)_{\sigma_N}. \tag{1}$$

A linear transformation, $\mathbf{W}^{\delta}_{\sigma_1 \sigma_2 \ldots \sigma_N}$, can be used to create a decision function, f,

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

where δ is the dimension of the output space. The size of the transformation matrix, $\mathbf{W}$, is exponential in the size of the input vector, hence this transformation is not possible to perform in general. Instead the matrix $\mathbf{W}$ can be factorized into smaller tensors contracted together to approximate the larger tensor. This set of tensors and the contraction performed is known as a *tensor network*. An example representation of $\mathbf{W}$ factorizes the original tensor as a product of matrices, A_{i,σ_i}, aptly called *matrix product states*,

$$\mathbf{W} = \mathbf{A}^{\delta}_{0,\sigma_0} \mathbf{A}_{1,\sigma_1} \ldots \mathbf{A}_{N,\sigma_N}, \tag{3}$$

where the output leg is placed on the first matrix.

More generally, tensor networks consist of a number of *tensors*, connected by *tensor contractions*. For the purposes of this paper, tensors are multidimensional arrays of numbers. A tensor network, when contracted fully, produces a new tensor with some number of indices.

A generic tensor network consists of V tensors, $A_{\vec{v}_i}$, each indexed by a vector $\vec{v}_i$ of integers, taking on values in the space $\bigotimes_{j \in \mathbb{Z}^{r_i}} \mathbb{Z}^{D_i}$. The rank of tensor $A_{\vec{v}_i}$ can be written as $r_i = \dim(\vec{v}_i)$ An arbitrary tensor network decomposition of a tensor $W_{\vec{\xi}}$, has the following form,

$$W_{\vec{\xi}} = \sum_{\vec{\sigma}} \prod_i^V A_{\vec{v}_i}, \tag{4}$$

where $\vec{\sigma} = \bigcup_i \vec{v}_i \setminus \vec{\xi}$. Such a tensor network has E edges, with those edges connecting tensors k and l lying in the pairwise intersection $\vec{v}_k \cap \vec{v}_l$.

2 Initialization for Tensorization

One major application of tensor networks techniques is in *tensorization* of deep learning models [14]. Modern deep learning involves the use of deep learning networks with up to billions of parameters. Such liberal use of parameters makes it difficult to use such networks for inference in resource constrained environments: on mobile phones, microprocessors or practically sized cloud compute nodes. Many methods exist for reducing deep learning model size. Within the scope of *model compression* one can use *quantization*, *pruning* and (or) *clustering*, [9] . Another field of techniques uses *knowledge distillation* to convey knowledge from a large network to a small one, with an architecture that might be unlike the original [6].

Another complementary line of research, *tensorization* is a method by which expensive dense or convolutional layers in deep networks are replaced by tensor network factorizations of the form Eq. 4. These replacements can lead to large reduction in parameter count, which can lead to speedups for inference.

One important question is the best way to train these tensorized layers. One important tool in doing so is a means to initialize them effectively. Initialization of linear layers in neural networks is a well studied topic. Most commonly used, and available in most modern software packages [11, 15] are the *glorot*[3], *he*[5] and *lecun*[8] initialization methods. In this section we will show how such methods can also be used for tensorized neural network initialization, with appropriate modifications.

2.1 Probabilistic Initialization for Generic Tensor Network Models

In a deep learning model, the initial elements of the matrices (whatever structure they be a part of) are usually drawn from some probability distribution. If the model's matrices are represented by tensor networks, we can no longer set the values of these matrices directly, as they only appear as the result of the full contraction of the tensor network. We can only set the elements of the constituent tensors. However, we will show that we can set the elements of W probabilistically through cunning choices of the element distribution of these tensors.

2.1.1 Theory

Tensor networks for machine learning are most often used to represent matrices, and it is this application that we will consider going forwards. We will specialise further to the case where the *bond dimension* D_i of each index is the same $D_i = D \ \forall \ i$. The generalisation to arbitrary tensor decompositions and bond dimension distributions is natural.

In a tensorized deep learning layer, the weight matrix W is represented by a sum of products of elements of the tensors A making up the tensor network,

$$W_{ij} = \sum_{\vec{\sigma}} \prod_n A_{\vec{v}_n}. \tag{5}$$

A chosen distribution for each tensor element will lead to another distribution when all tensors are contracted, i.e. elements of W_{ij} For simplicity, we will consider the case where each variable is drawn independent and identically from P. We will show that this assumption suffices to derive a simple and robust initialization scheme.

Let the elements of the tensors be drawn from a distribution P, such that $\mathbb{E}[A_{\vec{v}_n}] = 0$, and $\mathrm{Var}(A_{\vec{v}_n}) = \sigma^2$. What can we say about the distribution of W_{ij}?

From Eq. 5 and the independence of the variables $A_{\vec{v}_n}$, it is clear that $\mathbb{E}(W_{ij}) = 0$.

Each term in the sum in Eq. 5 is the product of V variables drawn independently from P. The variance of each term $X_i = \prod_n^V A_{\vec{v}_n}$ in the sum is therefore,

$$\text{Var}\left(\prod_n A_{\vec{v}_n}\right) = \prod_n \sigma_n^2 = \sigma^{2V}, \tag{6}$$

where we have used the fact that the variables are i.i.d.

The elements W_{ij} are equal to the sum of D^E such random variables. The sum of n uncorrelated variables has a variance equal to the sum of the variances of the constituent variables. For each pair X_i, X_j, there is at least one n for which $\vec{v}_n^i$ and $\vec{v}_n^j$ differ. As such, the different elements of the sum are uncorrelated, and the variance of their sum is,

$$\text{Var}(W_{ij}) = D^E \sigma^{2V}. \tag{7}$$

By choosing the mean and variance of the constituent distributions, we can make sure that the mean and variance of the resulting W distribution takes on a reasonable value. Matching the mean and distribution of common choices of initializer, for example [5, 8, 3], should produce an effective initialization scheme for tensor network decomposed layers.

2.2 Algorithm

To initialize a tensorized layer such that the distribution of the elements W_{ij} is approximately normal, with mean 0 and variance σ_{target}^2, each $A_{\vec{v}_n}$ should be drawn i.i.d from P such that,

$$\mathbb{E}(A_{\vec{v}_n}) = 0, \tag{8}$$

and

$$\text{Var}(A_{\vec{v}_n}) = \left(\frac{\sigma_{\text{target}}^2}{D^E}\right)^{\frac{1}{V}}. \tag{9}$$

By initializing the tensors with a distribution with these properties, the mean and variance of the distribution of the elements of $\mathbf{W}$ is chosen appropriately. Such a method suffices for V and E sufficiently small. Tensorized layers naturally fall into this category, since the number of nodes scales as $O(\log(N_{\text{features}}))$. Other uses of tensor networks in machine learning do not have this property however. In the next section we will discuss an appropriate generalization for this case.

3 Initialization for Quantum-inspired Models

Tensor networks have also been used to construct models whose architecture differs radically from standard deep learning setups. Since such models take inspiration from tensor network methods as used in quantum physics, we will hereinafter term them *quantum-inspired* models. The simulation of quantum systems requires performing linear algebra in a space exponentially large in the number of subsystems – for example, for 100 electrons, matrix-vector operations in a $O(2^{100})$ dimensional vector space would be required for a full accounting of the physics.

Tensor network methods allow for the simulation of an important subset of the physics of such systems. These techniques have been used in a similar way in machine learning. Data is embedded into an exponentially large space, and the output of the model is a (massive dimensional) linear operation on the embedded data, represented as a tensor network.

Initialization of quantum-inspired models presents new challenges. The number of nodes in a tensor network used in this manner is on the order of the number of features. Even for the simplest datasets used in image classification, this can be on the order of 1000. With the conservative assumption that the number of edges (E) and nodes (V) grow linearly with the number of features and for bond dimension $D = 2$. A quantum-inspired tensor network model classifying MNIST requires summing $O(\max(D_i)^E) = 2^{784}$ numbers, each consisting of the product of $V = 784$ numbers. Naive initialization schemes for the elements of the tensors often lead to wild instability, since we must balance the product of an exponentially large and exponentially small number.
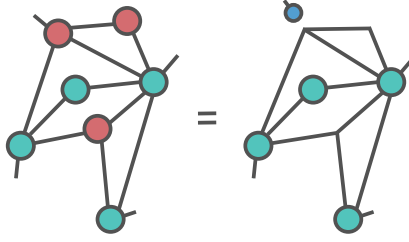
4

Figure 1: Copy node initialization. Replacing a fraction of the nodes of the tensor network with stacked copy nodes leads to an effective linear operator with known, manageable mean and variance. Red nodes are identified to be initialized as copy nodes. Red nodes with external legs are replaced by the tensor product of a copy node and a fixed vector.

As a result of these numerical problems, tensor network methods often use novel training methods, since SGD based training schemes for quantum-inspired techniques can be tricky to use, and practically can only be implemented when the number of features is sufficiently small.

In the following, we detail a practical scheme for initializing such tensor networks that can sidestep the problems associated with this instability.

3.1 Generic Tensor Network Models

The probabilistic method of Sec. 2.1 should still work in principle for quantum-inspired models. What will work mathematically however might not work using floating point numbers with finite precision. To initialize a network of a practicable size might require setting the variance below the floating point epsilon, or beyond the largest floating point number.

We would like a method with the same theoretical guarantees, but where initialization is easier. This can be achieved with the inclusion of *stacked copy nodes*.

A *copy node* is a multidimensional generalisation of the identity matrix. It is defined by the following equation,

$$C_{ijk...} = \begin{cases} 1 & i = j = k = ... \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

A copy node in a tensor network has the effect of pinning all of the legs of that tensor to the same value. One can naturally view a tensor network containing copy nodes as a *hyper*-tensor network, where single edges can be connected to multiple nodes.

A natural means of handling the explosion of nodes and edges in quantum-inspired tensor networks is to initialize some number of the nodes in the tensor network by copy nodes. Then use the above prescription to initialize all of the other nodes, making sure that only a practicable number of non-copy nodes remain. Replacing nodes in a tensor network with copy nodes results in a smaller hyper-tensor network. The above discussion of random initialization of tensor networks is insensitive to the node (edge) structure of the network. The smaller embedded hypernetwork can be initialized by initializing its nodes according to the prescription of the respective tensor network.

There is the question of definition for the input and output indices. To see the problem, consider the following simple example: If a node with both input and output indices is initialized with a copy node, the copy node will pin the input index on that node to the output node. The result is that the distribution of the output will not be as intended. Non-zero contributions to W will only arise when the vector input to a single node is the same as the output of the network.

The solution is to stack copy nodes across the input indices, and make sure nodes with output indices are densely initialized. We will show that this prescription leads to sensible initializations for the represented matrix.

3.2 Algorithm

We will initialize an n-node subnetwork of our tensor network as described above, and all other indices with copy nodes. To do so, choose $V - n$ nodes at random from the tensor network, skipping any with an output edge[1]. Each node is initialized with a copy node in the following scheme. Replace all completely internal nodes with pure copy tensors. For nodes with an input edge, initialize,

$$A_{\vec{v}_n}^{\xi_i} = C_{\vec{v}_n} \otimes \vec{v}_{\xi_i}, \tag{11}$$

where the internal indices are $\vec{v}_n$ as before, ξ_i are input indices and $\vec{v}$ is a constant arbitrary vector (see Fig. 1, where $\vec{v}$ is represented by the blue tensor).

The result is an embedded initialization, where some subgraph of the tensor network is initialized. Some input nodes are initialized to fixed local vectors (in quantum terms, they are left in a product state), while others form part of the initialized subgraph. This prescription is shown graphically in Figure 1. Red nodes are identified as being initialized with a copy node. Purely internal nodes are replaced with a copy node and nodes with an external leg are replaced with the tensor product of a copy node and a fixed vector.

The result is an initial value for the matrix W. Choose $v = (1, 0)$, and assume that the input tensors that are copy initialized are the first k in some order[2]. The resulting matrix will then have the following structure,

$$W_{tot} = \begin{pmatrix} W \\ 0 \end{pmatrix}, \tag{12}$$

where W is the matrix of the densely initialized subgraph. W is initialized using the algorithm of Sec. 2.2, where V and E are the number of nodes and (hyper-)edges in the subnetwork.

Such a scheme is especially suitable for the feature map found in [13]. Since the initial network can be chosen to act only on the linear variables, so we can guarantee that the properties at initialization match those of the equivalent linear model.

4 Automatic Bond Dimension Selection

In principle, for a tensor network with E edges there are E hyperparameters that should be set appropriately to define the model- a bond dimension D_i for each edge. In practise, given the difficulty of exploring such a large space of hyperparameters, all bond dimensions are set to a constant value and choices of the constant value are explored. This strategy is used in [1] and [17].

This has several drawbacks. A higher bond dimension between a pair of nodes leads to an ability to capture richer correlations between those tensors. A high bond dimension might be required at several edges in a network to do this, while in other parts a smaller bond dimension might suffice. If too small a global bond dimension is chosen, important correlations will be missed. If too high a global bond dimension, it might well be able to capture important correlations at those places where it is needed. But the model will contain parameters at further edges that increase the model complexity but offer no additional explanatory power.

These additional parameters can contribute to overfitting. Furthermore, for applications on edge devices where either tensorization or quantum-inspired models offer important opportunities for parameter efficient machine learning, such unnecessary parameters increase power and compute requirements for no additional benefit.

Gradient based training for neural networks offers a natural solution to these problems.

5 Rank Regularization

In this section we will detail a method for automatically choosing the bond dimensions of a tensor network model, which we term Rank Regularization (RR).

---

[1]By assumption we can do always do this, since the output dimension is much smaller than the input

[2]If they are not, the following matrix will have its rows permuted.

5.1 Defining 'soft' bond dimensions

In order to automatically determine an effective bond dimension for a network, we will relax the local bond dimensions of the network such that they can be trained as continuous parameters. These parameters can then be trained directly on the training data, or in the spirit of differentiable neural architecture search approaches [10], we can define custom update steps for these parameters that depend on a held-out validation set. In this paper we will explore the first approach.

We term these relaxed bond dimension parameters 'soft' bond dimensions, and they are defined as follows. At each edge on the tensor network model, define a 'rank regularizer', a diagonal matrix which is interposed between the tensors on that edge (see Fig 2). The original tensor network model can be recovered by setting each rank regularizer to the identity. A family of truncated models can be defined by setting each rank regularizer to a projector with $D_i < D$ ones on the diagonal and $D - D_i$ zeros. We can continuously explore this space of truncated models by using a shifted sigmoid function to define the diagonal of each rank regularizer, with a edge dependent 'soft' bond dimension defining the shift, see Fig. 2.

At training time, each soft bond dimension is variationally updated with the gradient of the loss function. Smaller soft bond dimension on some edges of the network will nullify the gradients of certain parameters. But if unmasking these parameters will lead to an increase in training performance, the bond dimension can increase. Importantly, gradient based training allows us to bias the training of the model towards models with smaller bond dimensions on different sites.

This can be done by adding a penalty to the loss function, for example an L1/L2 regularization on the vector of soft bond dimensions. The regularization strategy will define the final distribution of bond dimensions. We have found one effective strategy to be to compute a softened approximation to the number of parameters in the tensor network, by computing that metric as if each bond dimension was equal to its softened value.

Importantly, every step described so far can be performed differentiably with tools currently provided in standard software libraries.

Before using the network for inference, the model can be truncated at each edge., choosing a bond dimension that removes the elements of the tensors that are masked by the regularizers. Further, the rank regularizers can be simply absorbed into neighbouring tensors, such that the Floating Point Operations Per Second (FLOPS) cost of the forward operation of the model is no greater than the model without the regularizers (in fact, if any truncation is performed, the cost in both number of parameters and FLOPS is strictly lower).
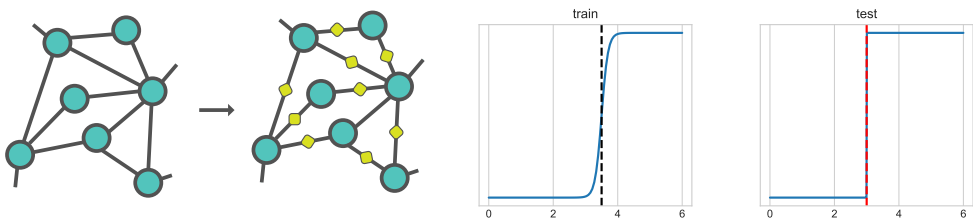

Figure 2: Rank Regularizers. a) A diagonal matrix, indicated by yellow diamonds, added to each internal edge in a tensor network, can be used to mask degrees of freedom in the tensor network, resulting in a reduction in the number of parameters. b) The form of the diagonal of the regularizers changes between training and inference, such that testing performance is that of the fully truncated network. The black dashed line indicates the soft bond dimension, whereas the red dashed line indicates the (rounded) final bond dimension choice.

6 Numerical Results

Ref. [17] is arguably the most well known application of tensor networks found in Machine Learning. Therefore, we consider the same experimental setup to benchmark our results.

6.1 Experimental Setup

We consider the image classification problem of the MNIST data set- 28×28 grey scaled images of digits from zero through to nine. Each image is fed through a $(2, 2)$ averaged pooling layer shrinking the image to 14×14 pixels, before being flattened using a zig-zag approach. Each pixel value is embedded int feature space using a localised feature map of the form form: for the jth pixel,

$$\phi(x_j)_{\sigma_j} = \left[\cos\left(\frac{\pi}{2}\hat{x}_j\right) + \sin\left(\frac{\pi}{2}\hat{x}_j\right)\right], \tag{13}$$

where $\hat{x}_j$ is the normalized pixel value. No bias in used in any of our results, and no further modifications were made to the dataset. The feature data is inputted into a Matrix Product State Tensor Network (MPS) which is then trained using stochastic gradient descent. The output leg of the MPS consists of un-normalized predictions with respect to each class, which are turned into class probabilities with a softmax function.

Tensor network operations are implemented by Quimb [4] and trained using TensorFlow Keras [11] for 100 epochs, with a batch size of 32 and a $80 : 20$ training validation split. We choose to train our tensor networks using Adam [7], with $\alpha = 0.001$, to optimize a categorical cross entropy cost function. Each tensor network is initialized and trained 10 times using the respective initialization scheme. We show the average result alongside error bars for 1 standard deviation, i.e. $\pm\sigma$.
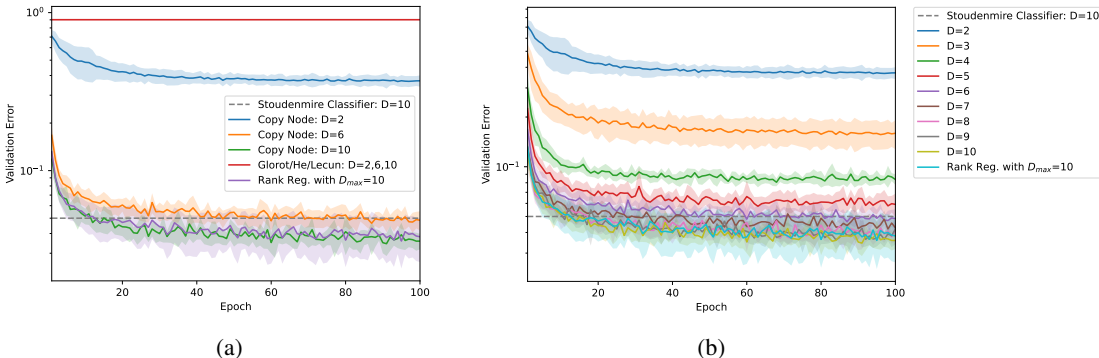
6.2 Initializers

Figure 3: Different MPS initializations results. a) Comparison of validation error during training between Glorot, He, Lecun and copy node initialization for $D = 2, 6, 10$. b) Validation error during training of copy node initialization for $D = 2, ..., 10$. Additionally, validation error for bond regularized MPS is also shown in both cases for $D_{max} = 10$. Our results outperform the different initialization schemes and [17] for $D \geq 6$.

Fig. 3a shows the average validation error during training of each MPS with different initializations and bond dimensions. Using either a Glorot, He or Lecun initialization of the MPS leads to a plateau in the loss and validation error for any bond dimension. On the other hand, using the copy node initialization scheme discussed in section 3 leads to a drop in validation error for all bond dimensions- as seen explicitly in Fig. 3b. We can see our initialization and training scheme start to outperform [17] in ~ 20 epochs for our equally sized $D = 10$ and rank regularized classifier. We observe no sign of over fitting due to the natural regulatory properties of Tensor Networks.

6.3 Rank Regularization

Fig. 4a shows the test accuracy and number of training parameters for each MPS of different bond dimensions, initialized using the copy node scheme. We see the expressive power of our classifier, outperforming [17] with a bond dimension of $D = 6$ and $\sim 64\%$ less parameters. Additionally, we show that our bond regularization scheme, discussed in section 5, for $D_{max} = 10$ performs equally as well as a $D = 10$ MPS with $\sim 22\%$ less parameters. Further increases
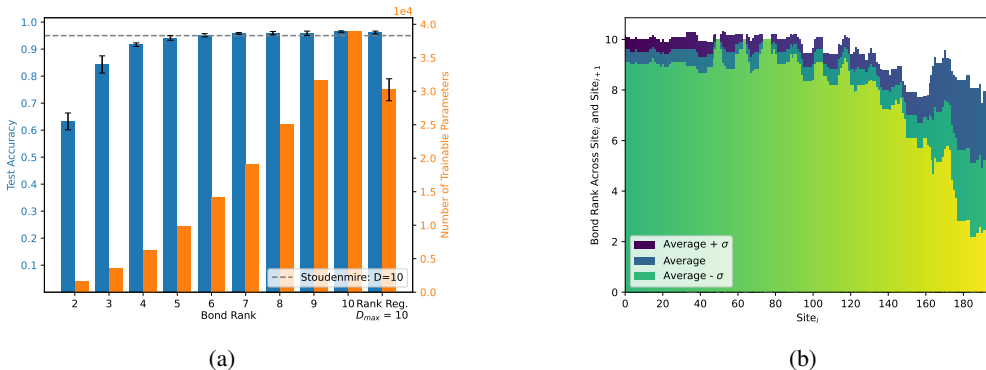
(a)	(b)

Figure 4: MPS model performance and bond regularization spectrum. a) Comparison of test accuracy and number of parameters for $D = 2, ..., 10$ MPS in addition to bond regularized MPS with $D_{max} = 10$. Our rank regularization scheme removes redundancies whilst performing equally as well as the $D = 10$ MPS. b) Spectrum of bond dimensions between neighbouring sites along the bond regularized MPS chain.

in bond dimension increase classifier performance further whilst still having less parameters than [17].

The dimension of each regularized bond, D_i, can be seen in Fig. 4b. The spectrum shows a drop in dimension towards the opposite end of the MPS chain with the output leg, since a lower dimension is needed to satisfy the amount of necessary information. We expect the peaks in the spectrum correspond to highly-valuable pixels shared by multiple digits. Therefore a higher resolution i.e. bond dimension is needed to differentiate between them.

7   Conclusions

In this paper we present two methods to improve the training of QTNML algorithms, which make it possible to train large tensor networks by performing gradient descent directly on all of the tensors. We have introduced the *copy node* initialization scheme, whereby certain tensors in the network are replaced by the tensor product of a copy node and a constant vector. The remaining nodes are then initialized such that when contracted they share the mean and variance with popular deep learning initialization distributions. We demonstrate that large MPS are able to be trained with this method without relying on DMRG-inspired updates, and without exploding or vanishing gradients. This method also extends to Tree Tensor Networks, MERA and PEPS. We also introduce a method to dynamically increase or decrease the bond dimension during training by introducing masked diagonal tensors between each node in the tensor network. Models trained with adaptive bond dimensions are able to achieve equivalent performance to higher bond dimension models with fewer parameters, focusing computational resources to the most important nodes in the classifying network. With these enhancements standard gradient descent is now a viable alternative to DMRG-inspired training procedures. These improvements should make it easier to develop tensor network machine learning algorithms for a wider variety of tensor network geometries.

We have shown that with two modifications to stand SGD, quantum tensor networks can be effectively and efficiently trained to solve machine learning problems. It would be interesting to explore the impact of these methods on a wider class of QTNML problems. It is well documented that the type of optimization algorithm that is used has a big impact on the quality of the trained model. We give evidence that ordinary stochastic gradient descent on all tensors simultaneously can produce higher accuracy models than DMRG equivalents by demonstrating this on the MNIST dataset, which is a good indication of success on more complicated datasets.

References

[1] Song Cheng, Lei Wang, and Pan Zhang. "Supervised learning with projected entangled pair states". In: *Phys. Rev. B* 103 (12 Mar. 2021), p. 125117. DOI: 10.1103/PhysRevB.103.125117. URL: https://link.aps.org/doi/10.1103/PhysRevB.103.125117.

[2] Song Cheng et al. "Tree tensor networks for generative modeling". In: *Phys. Rev. B* 99 (15 Apr. 2019), p. 155131. DOI: 10.1103/PhysRevB.99.155131. URL: https://link.aps.org/doi/10.1103/PhysRevB.99.155131.

[3] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

[4] Johnnie Gray. "quimb: A python package for quantum information and many-body calculations". In: *Journal of Open Source Software* 3 (Sept. 2018), p. 819. DOI: 10.21105/joss.00819.

[5] Kaiming He et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[6] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* (2015).

[7] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[8] Yann A. LeCun et al. "Efficient backprop". English (US). In: *Neural Networks*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Copyright: Copyright 2021 Elsevier B.V., All rights reserved. Springer Verlag, 2012, pp. 9–48. ISBN: 9783642352881. DOI: 10.1007/978-3-642-35289-8_3.

[9] Tailin Liang et al. "Pruning and quantization for deep neural network acceleration: A survey". In: *Neurocomputing* 461 (2021), pp. 370–403.

[10] Hanxiao Liu, Karen Simonyan, and Yiming Yang. "Darts: Differentiable architecture search". In: *arXiv preprint arXiv:1806.09055* (2018).

[11] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[12] Jacob Miller, Guillaume Rabusseau, and John Terilla. "Tensor Networks for Probabilistic Sequence Modeling". In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. Ed. by Arindam Banerjee and Kenji Fukumizu. Vol. 130. Proceedings of Machine Learning Research. PMLR, 13–15 Apr 2021, pp. 3079–3087. URL: https://proceedings.mlr.press/v130/miller21a.html.

[13] A Novikov, M Trofimov, and I Oseledets. "Exponential machines". In: *Bulletin of the Polish Academy of Sciences: Technical Sciences* (2018), pp. 789–797.

[14] Alexander Novikov et al. "Tensorizing Neural Networks". In: (Sept. 2015).

[15] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[16] Justin Reyes and Miles Stoudenmire. *A Multi-Scale Tensor Network Architecture for Classification and Regression*. 2020. arXiv: 2001.08286 [stat.ML].

[17] Edwin Stoudenmire and David J Schwab. "Supervised Learning with Tensor Networks". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper/2016/file/5314b9674c86e3f9d1ba25ef9bb32895-Paper.pdf.

[18] Zheng-Zhi Sun, Shi-Ju Ran, and Gang Su. "Tangent-space gradient optimization of tensor network for machine learning". In: *Physical Review E* 102.1 (2020), p. 012152.

[19] G. Vidal. "Entanglement Renormalization". In: *Phys. Rev. Lett.* 99 (22 Nov. 2007), p. 220405. DOI: 10.1103/PhysRevLett.99.220405. URL: https://link.aps.org/doi/10.1103/PhysRevLett.99.220405.

[20] Jinhui Wang et al. "Anomaly Detection with Tensor Networks". In: *arXiv e-prints*, arXiv:2006.02516 (June 2020), arXiv:2006.02516. arXiv: 2006.02516 [cs.LG].