MPO + MPS for classification.

(Dated: March 23, 2022)

I. IDEAS AND SOME DETAILS OF CALCULATIONS.

A. Introduction and motivation.

The idea of using MPS for discriminative modeling, in particular for classification, was introduced in Ref. [1] and Ref. [2]. The setup is pretty straightforward: one maps all the features into an exponentially high dimensional space and then applies MPS to the resulting product state using MPS as a trainable ansatz. This idea was extended in Ref. [3] to networks with block product structure (which are powerful at capturing local correlations, which show high training performance, but poor generalization). Another powerful approach developed in Ref. [4] is to use PEPS for classification. While it shows high performance (in particular when combined with a CNN), it is very expensive numerically and so does not seem feasible for applications.

My idea is based on extension of Ref. [1] by adding an MPO layer to MPS. The guess is that it may be possible to reduce the number of parameters of a tensor network (TN) — so reduce space and time complexity — without compromising its expressivity.

One vague motivation comes from the deep neural networks, which generalize better than shallow ones. So, maybe a deeper MPS (i.e. MPS+MPO) may show better generalization properties. A more concrete advantage is that such an architecture may be more memory efficient as, rather than having an MPS of bond dimension χ, one may hopefully get equally good performance with MPS bond dim $\sqrt{\chi}$ and MPO bond dim $\sqrt{\chi}$. Ideally, if one had an infinite amount or resources, a better way to go would probably be to use some kind of block product PEPS, which should be ideal at capturing both local and non-local 2D correlations, but it seems impractical right now and in any foreseeable future. So, my hope is that MPS+MPO may be more suitable for 2D data (images) than MPS, but much simpler than PEPS.

B. Some implementation details.

Firstly, each feature (pixel intensity) is mapped into a vector, e.g., $x \to \phi(x) = (1, x, x^2, ...)$ or $x \to \phi(x) = (\sin x, \cos x)$, or using any other feature map. This way one maps a vectorized image $\mathbf{x} = (x_1, x_2, ..., x_n)$ into a product state

$$\Phi(\mathbf{x}) = \phi^{s_1}(x_1)\phi^{s_2}(x_2)...\phi^{s_n}(x_n), \tag{1}$$

where s_i is a local feature index. This may be represented by tensor diagram shown in Fig. 1. Then one contracts this state with a feature tensor W and obtains a kind of

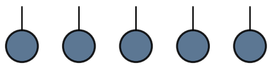


Figure 1. Feature vector mapped into a product state.

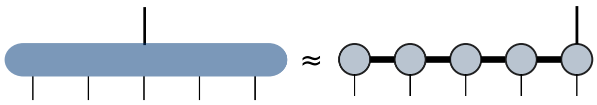


Figure 2. Approximation of a tensor by a matrix product state (MPS).

a decision function

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

which can then be used for, e.g., classification. Therefore, this approach is equivalent to linear regression in an exponentially large dimensional space. The problem is that, in general, weight tensor W consists of an exponentially large number of weights. One standard approximation of a weight tensor based on low rank factorization (compressed representation) is called tensor train or a matrix product state, which approximates a big tensor by a product of much smaller tensors:

$$W^{s_1 \dots s_n} = A^{s_1}_{i_1} A^{s_2}_{i_1 i_2} \dots A^{s_{n-1}}_{i_{n-2} i_{n-1}} A^{s_n}_{i_{n-1}}. \tag{3}$$

This decomposition is much easier to understand pictorically. In Fig. 2 I show MPS approximation as a tensor diagram. Now, my idea is to make a slightly different approximation and add an MPO (matrix product operator) layer with finite spacing — see Fig. 3. Of course, by no means this ansatz is more expressive than a simpler MPS. However, it may require a smaller number of parameters to reach the same expressivity. Also, due to finite spacing, it may be forced to learn more relevant features and so show better generalization. Finally, contraction of a TN with a product state can be represented as shown in Fig. 4, which is a graphical representation of Eq. 2.

Therefore, for actual calculations we need:

1. Map each image into a product state,
2. Construct MPS and MPO tensors.

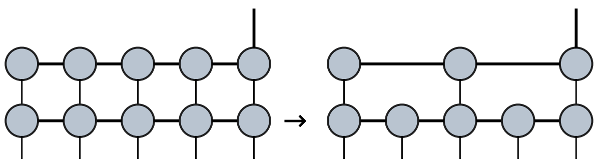


Figure 3. MPS+MPO ansatz. Left: zero spacing. Right: finite spacing.

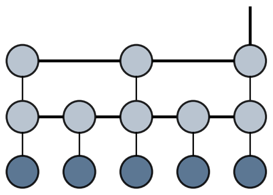


Figure 4. TN contracted with a product state.

C. Loss function

Next, to optimize the TN (weight tensor), we need to set up some loss function. A standard choice is to optimize so-called cross entropy using a softmax layer. A different approach was employed in Ref. [1] where mean square error optimization was used. In my calculations, I decided to optimize cross entropy without softmax normalization, but using simple "quantum mechanical" normalization (to mitigate overflow issues). By this I mean that if the output of the contraction of a TN with an image product state results in a vector $|\hat{y}\rangle$, then "probabilities" of each class are given by $p_i = (\langle y_i|\hat{y}\rangle)^2/\sqrt{\langle\hat{y}|\hat{y}\rangle}$, where $\langle y_i|$ is one for i-th class and zero for all other classes (a kind of a one-hot encoding). The cross entropy loss function is simply $\mathcal{L} = -\sum_{i\in correct} \ln p_i$. Therefore, by optimizing this loss function we increase "probabilities" of correct classes and reduce "probabilities" of incorrect classes (I keep writing "probability" in quotes because they do not necessarily represent real probabilities because of a calibration issue — see, e.g., Ref. [5]).

D. Optimization

There are different optimization strategies one may employ. For example, one may construct a full gradient and optimize all tensors at once. I found this strategy not particularly efficient (and slower than others). Another strategy (I have not tried) is a so-called Riemannian optimization mentioned in the context of MPS optimization, e.g., in Ref. [2]. A gold standard in quantum physics is DMRG, which is essentially an adaptive two-site optimization, where one contracts two neighboring tensors, optimizes the resulting tensor and then splits a new tensor into two ones using singular value decomposition (SVD) retaining some number of singular values (which defines MPS bond dimension). This is an interesting approach as it allows MPS ansatz capacity to grow and shrink adaptively depending on the data properties.

In my calculations I find that a simpler strategy generally works better, namely local one tensor at a time optimization. The idea is to simply fix all tensors except the first one and optimize the first one, then fix all tensors except the second one and optimize it an so on tensor by tensor.

To optimize the TN, we need to calculate derivatives of the loss function with respect to TN tensors W_i,

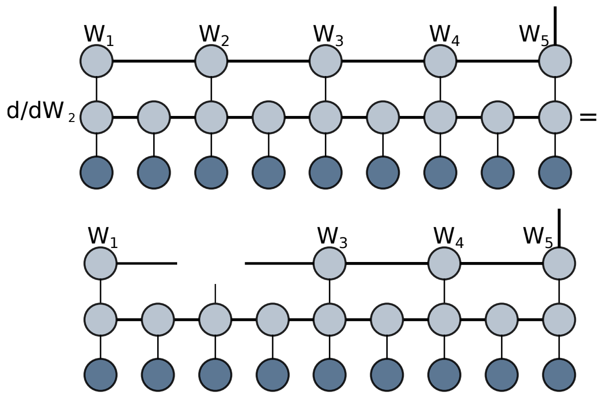


Figure 5. Derivative over an MPS tensor.

which are simply $\partial\mathcal{L}/\partial W_j = -\sum_i(\partial p_i/\partial W_j)/p_i$, where $\partial p_i/\partial W_j$ is essentially a function of $\partial|\hat{y}\rangle/\partial W_i$ and $|\hat{y}\rangle$, which means that we need to be able to efficiently calculate both $|\hat{y}\rangle$ and $\partial|\hat{y}\rangle/\partial W_i$.

The derivative $\partial|\hat{y}\rangle/\partial W_i$ over an MPS tensor may be represented by a tensor diagram — see Fig. 5 — which is easier to grasp than an analytical expression. The derivative over an MPO tensor can be obtained in a similar way by removing an MPO tensor. Also, just for a reference, if we did DMRG-like optimization, the derivative over a combined (say, MPS for concreteness) tensor would be obtained by removing two adjacent MPS tensors.

We can calculate a derivative every time for each tensor, but we would clearly do a lot of redundant work this way as we would need to contract the TN from the left and from the right over and over again. However, we can save the results of intermediate calculations and update them at each step, which would reduce time complexity from quadratic to linear in the input (number of pixels/features) size. All we need to do is to precalculate and update left and right TN contractions.

Last but not least, graphical representation is sufficient to set up most of tensor contraction using ITensor as all the contraction happen under the hood. For example, if one wants to contract two tensors, say A_{ijk} and B_{ikl}, in python one would need to write something like

$$\text{np.einsum}(\text{"}ijk, ikl \to jl\text{"}, A, B), \tag{4}$$

while using ITensor in julia one needs to define index objects, then tensors A and B using previously defined indices and then just write

$$A * B, \tag{5}$$

which, in my view, makes code a bit simpler and reduces chances of making tensor related bugs.

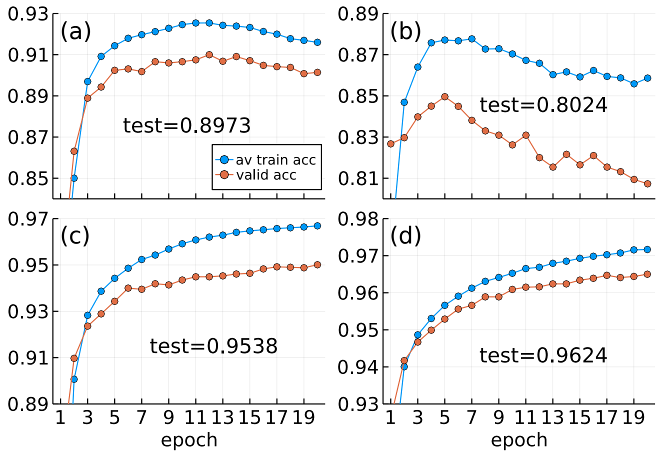


Figure 6. Validation and average training accuracy at each epoch. Optimizing one tensor at a time with one CGD step using $2^{12} = 4096$ images per batch. Also showing test loss at the last step. (a) $\chi_{\rm MPS} = 5, \chi_{\rm MPO} = 1$, frozen MPO, (b) $\chi_{\rm MPS} = 5, \chi_{\rm MPO} = 4$, frozen MPO, (c) $\chi_{\rm MPS} = 5, \chi_{\rm MPO} = 4$, unfrozen MPO, (a) $\chi_{\rm MPS} = 20, \chi_{\rm MPO} = 1$, frozen MPO.

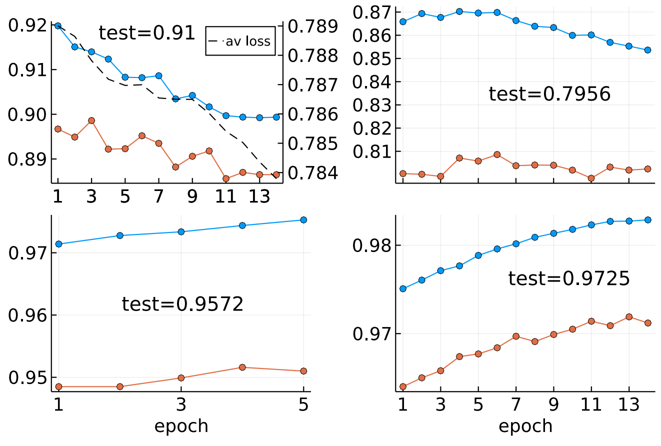


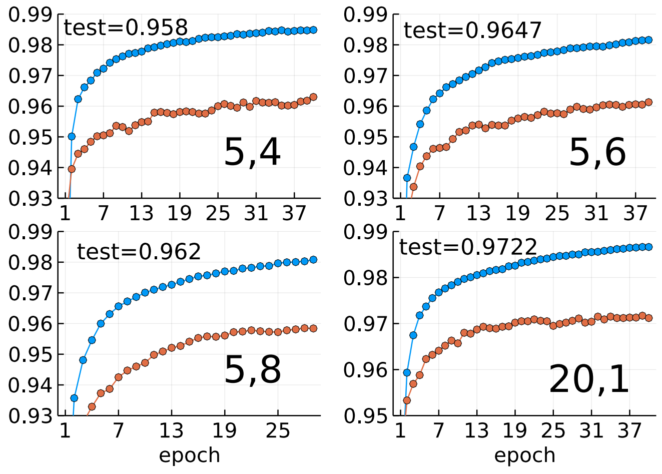
Figure 7. Further training 6 with 2 CGD steps per tensor update.

II. MPO EFFECT AT ZERO SPACING.

A. Frozen and unfrozen MPO.

Clearly, the results presented in Fig. 6 are not converged, so need to optimize further. Also, with frozen MPO training loss gets even worse, which does not make much sense and which also suggests that more "harsh" optimization, e.g., with more CGD steps, may be useful. So, let us optimize obtained MPSes and MPOs further with 2 CGD updates per tensor.

In Fig. 7 I show the results of further calculations. We can see that training a TN with an unfrozen MPO or with identity MPO and $\chi_{MPS} = 20$ leads to better performance, but not in other cases. In top left panel of Fig. 7 I also added average training loss, which does go down, while average training accuracy goes down too.



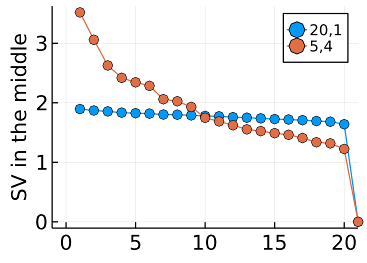


Figure 8. Accuracies with unfrozen MPO for $(\chi_{\rm MPS}, \chi_{\rm MPO})$. Bottom panel shows singular values in the middle of a TN for $(\chi_{\rm MPS}, \chi_{\rm MPO}) = (20, 1), (5, 4)$. Optimize them further!

This is a bit strange, but not impossible as loss is defined as $-\sum_{i\in\text{correct}} \log p_i$, while accuracy as the number of "correct" guesses with $p > 0.5$ divided by the total number of images.

So, from now I would compare TNs with unfrozen MPO. I would also increase the number of CGD steps from one/two to five or ten hoping to get faster training. It also may be that my training procedure is not the most efficient as I optimize MPS, then MPO and then repeat.

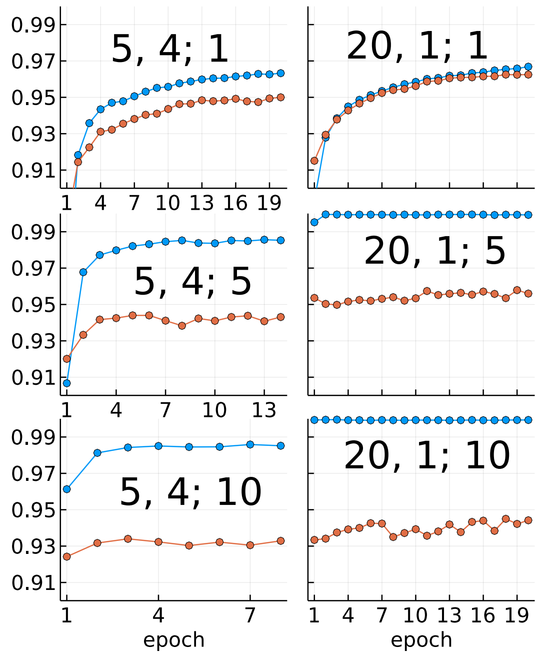


Figure 9. $(\chi_{\rm MPS}, \chi_{\rm MPO}; N_{\rm CGD})$. Blue dots: average training accuracy. Orange dots: validation accuracy. Number of training points per batch is 2^{12}.

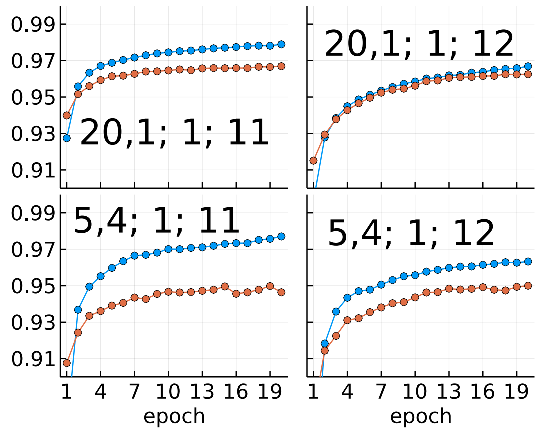


Figure 10. $(\chi_{\rm MPS}, \chi_{\rm MPO}; N_{\rm CGD}; \log_2 N_{\rm batch})$. Blue dots: average training accuracy. Orange dots: validation accuracy.

B. Effect of the number of CGD steps and number of images per batch.

An optimal number of CGD steps may depend on the number of images per batch. Thus, let us check how training is affected by changing the number of CGD steps and the number of images per batch. For concreteness, I would choose $N_{\rm CGD} = 1, 5, 10$ and $N_{\rm batch} = 2^{11}, 2^{12}$, so six tests in total. If Fig. 9 I show the results of corresponding calculations. Also, in Fig. 10 I show the results for different number of points per batch.

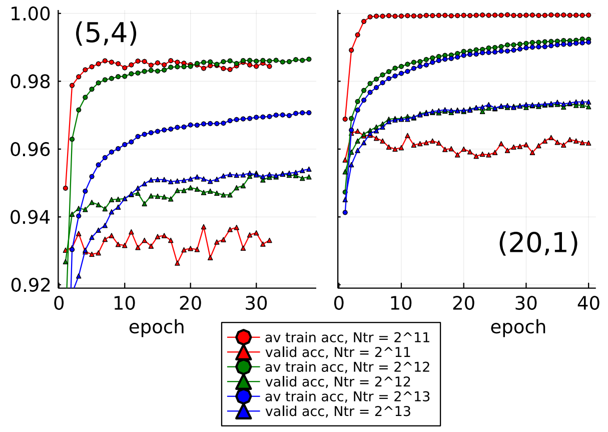


Figure 11. Accuracies increasing the number of images per batch keeping $N_{\rm CGD} = 2$.

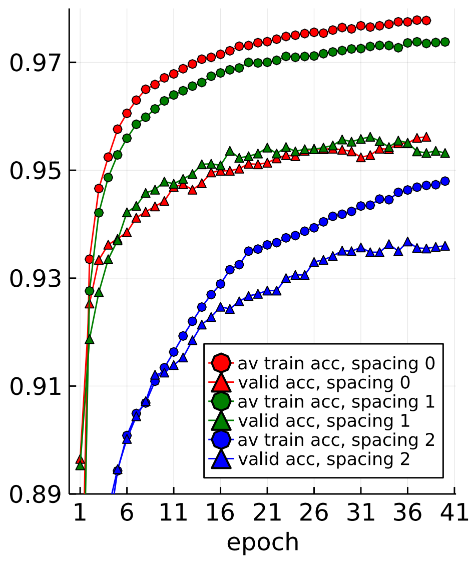


Figure 12. Spacing effect. Using 2 CGD steps, 2^{13} images per batch. $\chi_{\mathrm{MPS}} = 5$, $\chi_{\mathrm{MPO}} = 4$, zero and finite spacing.

III. EFFECT OF SPACING (UNFROZEN MPO).

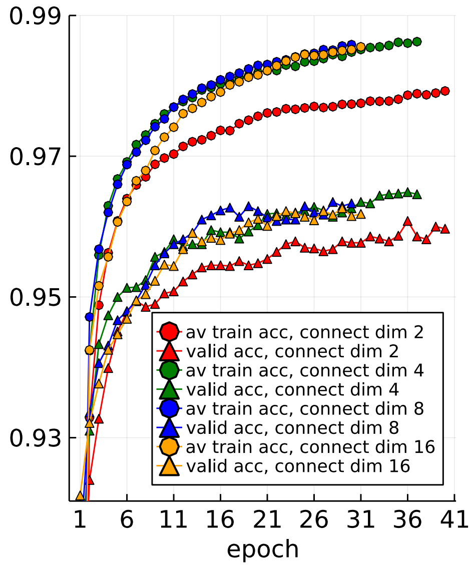


Figure 13. Connection dimension effect. Using 2 CGD steps, 2^{13} images per batch. $\chi_{\mathrm{MPS}} = 5$, $\chi_{\mathrm{MPO}} = 4$, zero spacing.

IV. EFFECT OF MPO OUTGOING DIMENSION AT ZERO AND FINITE SPACING.

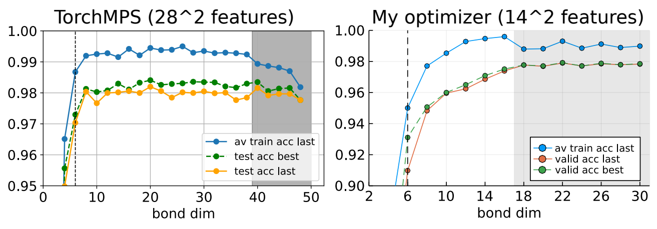


Figure 14. Accuracy vs MPS bond dimension (MPO bond dimension is one) using TorchMPS and my optimizer with Training one tensor at a time with 2 CGD iteration per tensor and 2^{13} images per batch (and extra optimization with 10000 images per batch).

V. EFFECT OF MPS BOND DIM AT FROZEN IDENTITY MPO.

VI. FULL ADAM OPTIMIZATION VS ONE TENSOR AT A TIME.

[1] E. M. Stoudenmire and D. J. Schwab, *Supervised learning with quantum-inspired tensor networks*, arXiv preprint arXiv:1605.05775 (2016).

[2] A. Novikov, M. Trofimov, and I. Oseledets, *Exponential machines*, arXiv preprint arXiv:1605.03795 (2016).

[3] J. Martyn, G. Vidal, C. Roberts, and S. Leichenauer, *Entanglement and tensor networks for supervised image classification*, arXiv preprint arXiv:2007.06082 (2020).

[4] S. Cheng, L. Wang, and P. Zhang, *Supervised learning with projected entangled pair states*, Phys. Rev. B **103**, 125117 (2021).

[5] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, *On calibration of modern neural networks*, in *International Conference on Machine Learning* (PMLR, 2017) pp. 1321–1330.