A practical comparison of tensor train models: the effect of homogeneity

Jonathan Palucci [1] **Jacob Louis Hoover** [1]

Abstract

Tensor network methods have been used extensively in computational physics, and recently have begun to be applied in machine learning. We focus on a particular family of these tensor factorizations known as tensor trains, or matrix product states (MPS), as models for discrete multivariate probability distributions. There is a general correspondence between tensor networks and graphical models, and in particular, when restricted to non-negative valued parameters, an MPS is equivalent to Hidden Markov Model (HMM). Glasser et al. (2019) discuss this correspondence, and prove theoretical results about these non-negative models, as well as similar real– and complex–valued tensor trains. They supplement their theoretical results with evidence from numerical experiments. In this project, we re-implement models from their paper, and also implement time-homogeneous versions of their models. We replicate some of their results for non-homogeneous models, adding a comparison with homogeneous models on the same data. We find evidence that homogeneity decreases ability of the models to fit non-sequential data, but preliminarily observe that on sequential data (for which the assumption of homogeneity is justified), homogeneous models achieve an equally good fit with far fewer parameters. Surprisingly, we also found that the non time homogeneous positive MPS performs identically to a time homogeneous HMM.

1. Introduction

Tensor networks are a state of the art method for modelling complex quantum systems. They have been explored extensively in computational physics and have recently garnered attention as machine learning models, as a method to efficiently represent high-dimensional systems. They differ from neural networks in that they do not rely on nonlinear activation functions. Instead, they use the multiplicative interactions of tensors to capture complex patterns in the data (Miller et al., 2020). They have recently found applications in machine learning, including supervised learning (Stoudenmire & Schwab, 2016; Bradley et al., 2020), unsupervised learning (Han et al., 2018; Cheng et al., 2019), exploring the expressive power of neural networks (Khrulkov et al., 2019b), parameter reduction in neural networks (Novikov et al., 2015; Khrulkov et al., 2019a), and language modelling (Bradley & Vlassopoulos, 2020). Here we explore their potential as probabilistic models of multivariate distributions (Stokes & Terilla, 2019; Miller et al., 2020), partially replicating results from Glasser et al. (2019), with additional observations.

Modelling a distribution with a tensor network involves representing a high-dimensional tensor as a factorization into smaller tensors. There is a natural correspondence between tensor networks as models of distributions, and probabilistic graphical models. To achieve efficient representations of joint probability distributions, graphical models leverage a graph structure to express conditional dependence between the random variables. Given a discrete probability distribution over n random variables, each taking values in $\{1, \ldots, d\}$, the joint distribution may be represented as an n-dimensional table (which has d^n entries), where each entry corresponds to the joint probability of a particular outcome of the joint random variables $X_1, \ldots, X_n$. This array can be viewed as a tensor $T \in \mathbb{R}^{d^n}$ with non-negative real entries with values between 0 and 1, and which sum to 1, so $p(x_1, \ldots, x_n) = T_{x_1,\ldots,x_n}$. A tensor network can be defined as a factorization of this large tensor. This leads to a clear parallel between graphical models and tensor networks: in the former case, we are concerned with the factorization of a probability table and in the latter case, the factorization of large tensors into a network of connected smaller tensors. In this sense, discrete graphical models can be put into a natural correspondence with non-negative tensor networks (Robeva & Seigal, 2017).

In this paper, we will be concerned only with a specific kind of tensor network called a *tensor train*, or *matrix product state* (MPS). These models have a structure similar to a Markov model (see Figure 2b). An MPS is simply a network of order-3 tensors which are connected to each other along an unobserved *bond* dimension (which can be thought of as the number of hidden states in an HMM).

[1]Group 22, equal contribution.

IFT 6269 project report - Université de Montréal

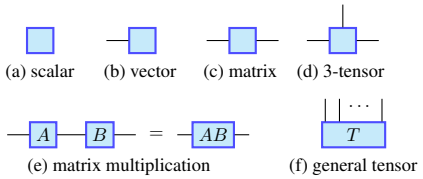


Figure 1: Examples of tensors.

2. Background

Similar to probabilistic graphical models, tensor networks also make use of a simple and intuitive graphical language. When a tensor network is used to model a joint probability distribution, there is a natural relationship with (undirected) graphical models. Even though the graph structure has different formal interpretation in each case, in both the graph diagram can be used to economically describe a joint distribution through select local relationships between variables.

Examples of simple tensors diagrams are given in Figure 1. In the graphical language of a tensor network, a box corresponds to a tensor, and a wire to an index. The number of wires connected to a box determines the order of the tensor. When two boxes are connected, this represents tensor contraction, corresponding to Einstein summation across the corresponding indices (a generalization of the trace operation or matrix multiplication). A tensor network is itself a tensor, with order corresponding to the number of wires with a free end. A network where all wires are connected on both ends (i.e. no free indices) is a scalar.

2.1. Tensor Networks as Probabilistic Models

A tensor network is simply any factorization of a tensor into a collection of other tensors. It will be useful when the factorization describes a low-rank (approximation) of a high dimensional object. In order for such a network to be a model of a probability distribution, we must simply describe how to evaluate the network at a particular outcome, and we must ensure that the values assigned to outcomes are non-negative values which sum to 1.

A joint probability distribution for discrete random variables $X_1, \ldots, X_N$ over $\{1, \ldots, d\}$ is naturally modelled by a tensor T with d^N entries, $T_{X_1,\ldots,X_N} = P(X_1, \ldots, X_N)$ (graphically, a tensor box with n wires). Contracting this network with inputs corresponding to the observation (that is, plugging in an encoding of observed values at each position), we obtain some scalar value, which, if we may ensure it is non-negative, may be normalized to obtain a probability model.

2.2. MPS models, positive, real and complex

There is a theoretical equivalence (in terms of expressive power) between non-homogeneous HMMs and positive MPSs, when viewed as probabilistic models (Critch, 2014). Glasser et al. (2019) also prove that there are non-homogeneous Born machines that cannot be modelled by non-homogeneous HMMs (they also hypothesize that the reverse direction is true). The comparison between HMMs positive MPSs and Born machines can also be examined in the homogeneous setting, however this a much more recent concern. Srinivasan et al. (2020) prove that there exist homogeneous Born machines that have no equivalent homogeneous HMM, and vice-versa. However, all these results are more theoretical in nature and do not focus on the practical applications of these different models.

3. The models

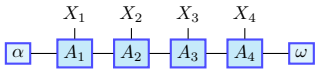
The tensor network models we are interested in are very simple: an MPS/tensor train consists of a collection of rank 3 tensors connected in a linear fashion, as depicted in Figure 2b. In our application, we call each of these tensors in the network *cores*, and they are each of the same dimensions, which will be $d \times D \times D$, where *physical dimension* d is the range of the input (the number of categories in the distribution that we are modelling, the vertical wire), and D is the *bond dimension* (the horizontal wires). We define these networks with *boundary vectors*, in order to have all of the cores be the same order.[1] The parameters of an MPS model consist in the values stored in the cores and boundary vectors.

Evaluating the tensor network T at a particular observation $(x_1, \ldots, x_n)$ to get a scalar involves getting a matrix from each core by selecting the element of the core at the observed value in the physical dimension, and then contracting the network to get a scalar (see Figure 2). This is equivalent to plugging in one-hot encodings of the observed values. Interpreting the resulting scalar as a probability may be done in two ways, as described below. To calculate the probability for a particular configuration $(x_1, \ldots, x_N)$ we contract this sequence with the network, as shown in Figure 3. This will return a scalar value since the corresponding network will have no free indices.

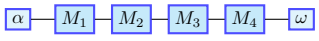
3.1. Positive MPS model

The most direct way to use an MPS as a probabilistic model is to restrict all the values of the tensor cores to be in $\mathbb{R}^{\geq 0}$ (we will refer to this as a *positive MPS* - even if they would be more accurately characterized as non-negative MPS). When evaluating, the resulting scalar may be interpreted

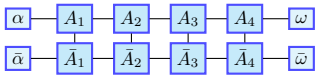
[1] We could equivalently remove the boundary vectors for the non-homogeneous models, and have the first and last cores be of order-2, but this would not be possible for the homogeneous models since the same order-3 tensor is used at each time step.



(a) A tensor train T, consisting of cores $A_1, \ldots, A_4$, and boundary states α, ω.



(b) $T_{x_1,\ldots,x_4}$, evaluated as $\alpha^\top M_1 \ldots M_4 \omega$ where $M_i = A_i(x_i)$.



(c) T contracted with its complex conjugate, to compute the normalizer.

Figure 2: An example of a MPS/tensor train A for a sequence of length 4. The contraction of the network in (a) with one-hot encoded inputs at each of the outgoing wires (evaluating the network at an observation) is interpreted as the unnormalized probability. The network is contracted with itself as in (b) to compute normalization constant Z_A.

directly as an unnormalized probability distribution:

$$\hat{p}_{\text{posMPS}}(X_1, \ldots, X_n) = T_{X_1,\ldots,X_n} \tag{1}$$

where T is an MPS. That is, we interpret as a probability mass function the function which takes in a particular configuration of the n random variables and contracts the network at this configuration to generate a scalar value. For a positive MPS, this scalar is guaranteed to be positive so it can be used as the unnormalized probability.

In order for the above scalar to be a valid probability, it must be properly normalized. Normalization is straightforward and efficient for MPS (normalization can in general be computed exactly for any tensor network whose graph is a tree; Bridgeman & Chubb, 2017). For a given positive MPS the normalization constant is calculated by simply contracting the network, summing over all possible values of the observed input.

3.2. Born Machine models

Another way to interpret an MPS as a probabilistic model is inspired by the Born Rule in quantum mechanics (Born, 1926), wherein the complex-valued wave function gives an unnormalized probability density as its magnitude squared. We will refer to this approach as the *Born Machine* interpretation for MPSs, which allows us to formally interpret such models as wavefunctions over n quantum spins (Miller et al., 2020).[2] This interpretation allows us to relax the non-negative requirement above.

In this paper, we use Born Machines with parameters in $\mathbb{R}$ (*real Born*), and in $\mathbb{C}$ (*complex Born*). To generate the unnormalized probability for a Born Machine, we use the same contraction algorithm as the positive MPS. The only difference is that the resulting scalar may be negative or complex. Taking the modulus squared of the output scalar ensures that the output is positive and may be interpreted as an unnormalized probability:

$$\hat{p}_{\text{Born}}(X_1, \ldots, X_n) = |T_{X_1,\ldots,X_n}|^2 . \tag{2}$$

For normalization in the Born machine interpretation, instead of simply summing over all possible values of the observed input, the network is contracted with its complex conjugate, along the physical dimension. This will generate a ladder type network (as shown in Figure 2c). Normalization is computed by contracting this entire network. Tensor contraction is associative, so the order of contraction makes no difference for the outputted scalar, however, contracting the network in specific orders can be much more efficient (Bridgeman & Chubb, 2017). This computation can be done efficiently by contracting the network as shown in Figure 3. The resulting normalization constant corresponds to the L_2 norm of the network.

3.3. Homogeneous and Non-homogeneous Models

An HMM can be time homogeneous or non-homogeneous, depending on whether or not one restricts the set of parameters which define the transition and emission probabilities to be identical at each step of the sequence. The same concept applies to an MPS. A time-homogeneous MPS has tensor cores which are all identical, while a non-homogeneous MPS allows different values in the tensor cores at each position in the sequence. A length n sequence has potentially n different tensor cores, making the number of parameters scale linearly with the sequence length (with possibly a hefty constant, if the dimensions are large). Conversely, a homogeneous MPS has a single tensor core which is repeated n times. Not only does this result in a much smaller model, but the recurrent nature of the model also provides the flexibility to model sequences of varying length.

In this work, we begin to investigate the differences between these two forms of MPS. Even though the idea of homogeneous and non-homogeneous MPSs is common, systematic comparison between the two has not been explored empirically. The use of homogeneous MPS models also allows for a more natural and appropriate comparison to HMMs, which are often used as homogeneous models.

4. Related Work

This paper is an extension to Glasser et al. (2019), which explores the expressive power of different tensor network factorizations for probabilistic modelling. We are interested in exploring similar questions but with a particular emphasis on practical comparisons. More specifically, perform a

[2] Informally, we can encode the probability distribution as a wavefunction $\Psi(\boldsymbol{x})$; then looking at a particular configuration of the distribution corresponds to measuring the state of the system and generates an unnormalized probability proportional to $|\Psi(\boldsymbol{x})|^2$.

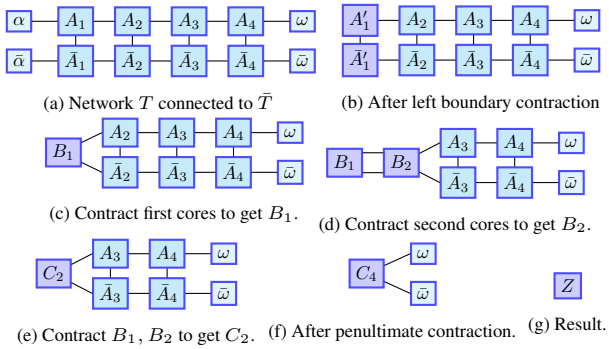


Figure 3: Demonstration of contraction to get normalization constant.

more thorough comparison of different MPS models with an HMM. In particular, Glasser et al. (2019) compare non-homogeneous MPS models to a homogeneous HMM, and do not explore the behaviour of homogeneous MPS models. To make, what we feel is a more appropriate comparison, we re-implemented the positive MPS and Born Machine models that they use, and also implement homogeneous variants of these models.

Another lingering question surrounding tensor network models is what is the best way to train them. If these models are to be used for standard machine learning tasks, this is an important problem. The code provided by Glasser et al. (2019) used custom implementations of the weight gradients and a gradient descent algorithm, which did not allow for easy experimentation with different model architectures or optimization methods. For this reason, our re-implementation is written from scratch in PyTorch, allowing possible modifications or extensions to the tensor network models, and allowing the use of various optimization algorithms available in the torch.optim package, with gradients computed by autograd.[3]

5. Training MPS models as ML estimators

To test how well these networks are able to fit to data, we train the different tensor network models as Maximum Likelihood estimators, to fit data using gradient descent. We compare these results with an HMM model trained using an Expectation Maximization algorithm (Baum-Welch).

Given dataset $D = \{\boldsymbol{x}^{(i)} : i \in \{1, \ldots, N\}\}$, where observations $\boldsymbol{x}^{(i)} = (x_1^{(i)}, \ldots, x_n^{(i)})$ generated according to some joint distribution over n discrete multivariate random variables, an MPS can be trained to approximate this distribution through standard maximum likelihood estimation, by minimizing the negative log-likelihood loss:

$$\ell(T; D) = -\sum_{i=1}^{N} \log \frac{\hat{p}_T(x_{1:n}^{(i)})}{Z} \tag{3}$$

The derivative of the log-likelihood which is used in the gradient updates can be given by:

$$\partial_w \ell = -\sum_{i=1}^{N} \frac{\partial_w \hat{p}_T(x_{1:n}^{(i)})}{\hat{p}_T(x_{1:n}^{(i)})} - \frac{\partial_w Z}{Z}. \tag{4}$$

where w is the weights of the model (the entries of the tensor cores and boundary vectors).

5.1. Contraction algorithm with numerical stability

Contracting these networks is particularly prone to cause overflow as the computations can quickly become very large (due to the fact that we are taking large sums of products when performing contractions). In order to avoid overflow issues, we implemented our contraction algorithm with a numerical stability trick. We cannot directly work with log probabilities with a tensor network, but we may still increase stability of the calculation during contraction by storing the logarithm of the accumulated magnitude.

Algorithm 1 get unnormalized probability, with stability

initialize direction unit vector $\tilde{v}_0 \leftarrow \alpha / \|\alpha\|$
initialize accumulated log norm scalar $c_0 \leftarrow \log \|\alpha\|$
$n \leftarrow$ length of sequence
for i in $\{1, \ldots, n-1\}$ **do**
  $v_i \leftarrow \exp[c_i]\tilde{v}_i$
  $\tilde{v}_{i+1}^{(\text{temp})} \leftarrow v_i$ contract with $A_{i+1} x_{i+1}$
  $c_{i+1} \leftarrow c_i + \log \left\| \tilde{v}_{i+1}^{(\text{temp})} \right\|$
  $\tilde{v}_{i+1} \leftarrow \tilde{v}_{i+1}^{(\text{temp})} \Big/ \log \left\| \tilde{v}_{i+1}^{(\text{temp})} \right\|$
end for
return $\hat{p}(x_{1:n}) = \exp[c_n] + \log(\tilde{v}_n^\top \omega)$

We implemented a stabilized MPS contraction algorithm, given in Algorithm 1. This algorithm works by contracting the network left to right, in the same manner described in Section 3 and shown in figures 2 and 3. Starting with the left boundary vector, at each step there is a vector which must be stored. Instead of storing this vector directly, we record its log magnitude before normalizing to get a unit vector. These log magnitudes may be summed, before being re-exponentiated at the end of the computation, before the final inner product with the right boundary vector.

5.2. Datasets

We used the same six datasets used in Glasser et al. (2019, see there for references). These are: 5 datasets from the

[3] In order to use autodifferentiation with complex numbers for these models, we needed to use a 'nightly' preview release of PyTorch (v1.8), since operations such as einsum and batched matrix multiplication are not currently supported for complex types in the stable release.

UCI Machine Learning Repository (Lymphography, SPECT Heart, Congressional Voting Records, Primary Tumor and Solar Flare) and the biofam dataset of family life states from the Swiss Household Panel biographical survey. Each dataset consists of sequences of categorical variables (coded as integers), where all sequences from a given dataset are the same length.

Note that of these datasets, biofam is unique in that each datapoint is naturally sequential (the features representing family life status from age 15 to 30 for an individual). The features in the other datasets are not naturally sequential.

5.3. Training regimen

Each model was trained for 2000 epochs using gradient descent, using the Adam algorithm (Kingma & Ba, 2017, as implemented in torch.optim), with an initial learning rate of 1×10^{-3} to 1×10^{-5} to, and a batch size of 50 for biofam, 20 for flare and votes, and 10 for the other datasets.

5.4. HMM model for comparison

For comparison with the tensor network models, we followed Glasser et al. (2019) in using the Hidden Markov Model implementation provided in the package pomegranate (Schreiber, 2018), and training by EM for 1000 iterations using the Baum-Welch algorithm.

6. Results and discussion

Results of training each of the six MPS models and the HMM model for comparison on each of the six datasets, for various values of bond/hidden dimension are given in Figure 4.

Qualitatively, for non-homogeneous models, we observe the same pattern that Glasser et al. (2019) report. That is, compared to Positive MPS, Born models achieve better fit, with the complex Born often outperforming the real Born. And, increasing the bond dimension (D) of the models leads to better fit across all datasets, as expected.

Additionally, we found that we actually achieved better results than Glasser et al. (2019), specifically with the Born machine models. This is presumably due to the numerical stability trick which was implemented as we noticed that this increased performance when used. Another possibility is that using boundary vectors which served as trainable parameters led to improved performance (they implemented their models without boundary vectors, an option not available to us, without compromising direct comparability with the homogeneous models).

Also as reported in the original paper, performance of non-homogeneous MPS is effectively identical to homogeneous HMM. This is in fact quite surprising. Given the theoretical correspondence between positive MPS and HMM, one would expect that the homogeneous HMM and homogeneous positive MPS should pattern together, however this was not the case, based on our results (which agree with the results reported in Glasser et al. (2019), though they do not discuss this puzzling fact). This alignment in the behaviour of the two models, one homogeneous and the other non-homogeneous is something that merits further exploration. Even though there may be a theoretical equivalence between the positive MPS and HMM, this equivalence does not hold in practice.

As a general observation, it seems the homogeneous models perform worse than the non-homogeneous models. This is to be expected given that the non-homogeneous models have many more parameters to fit to the data. There was one exception to this: the biofam dataset. Interestingly, and perhaps crucially, this was the only dataset which involved a natural sequential order (they represent the family life states from age 15 to 30). In this setting, the homogeneous and non-homogeneous models had effectively identical performance, even though the homogeneous models have a fraction of the parameters. This is a novel result which we would like to explore more.

6.1. A note on training tensor networks

Given that tensor networks have only recently begun to be used in machine learning, there are no standard practices for training them. In our experimentation, we found that stochastic gradient descent with a fixed learning rate caused a lot of instability. In order to avoid converging at a local minimum, a large learning rate was necessary, however this caused issues with exploding gradients, especially with the complex Born models. It seems that these models need a larger learning rate at the start of training in order to avoid local minima, but a smaller learning rate later in training in order to converge. Experimenting with optimizers in the PyTorch optim package, we found that the Adam algorithm led to the best results, while maintaining stability throughout training.This could be due to the fact that Adam implements a decaying learning rate throughout training, which seems to be important for these networks.[4]

A good choice of initialization may also be important. We began by initializing the tensor cores and boundary vectors uniformly as ones and scaled them by the inverse square root of the number of entries in the tensor ($1/\sqrt{D}$ for each boundary vector and $1/\sqrt{dD^2}$ for the tensor cores), to allow the initial tensor to be naturally interpreted as a probability model. This initialization led to stable training. We then achieved better results using the same initialization but injecting a small amount of gaussian noise ($\sigma^2 = 0.5$, before

[4]Other optimizers that implement decaying learning rates such as Adadelta and Adagrad also led to comparable results.

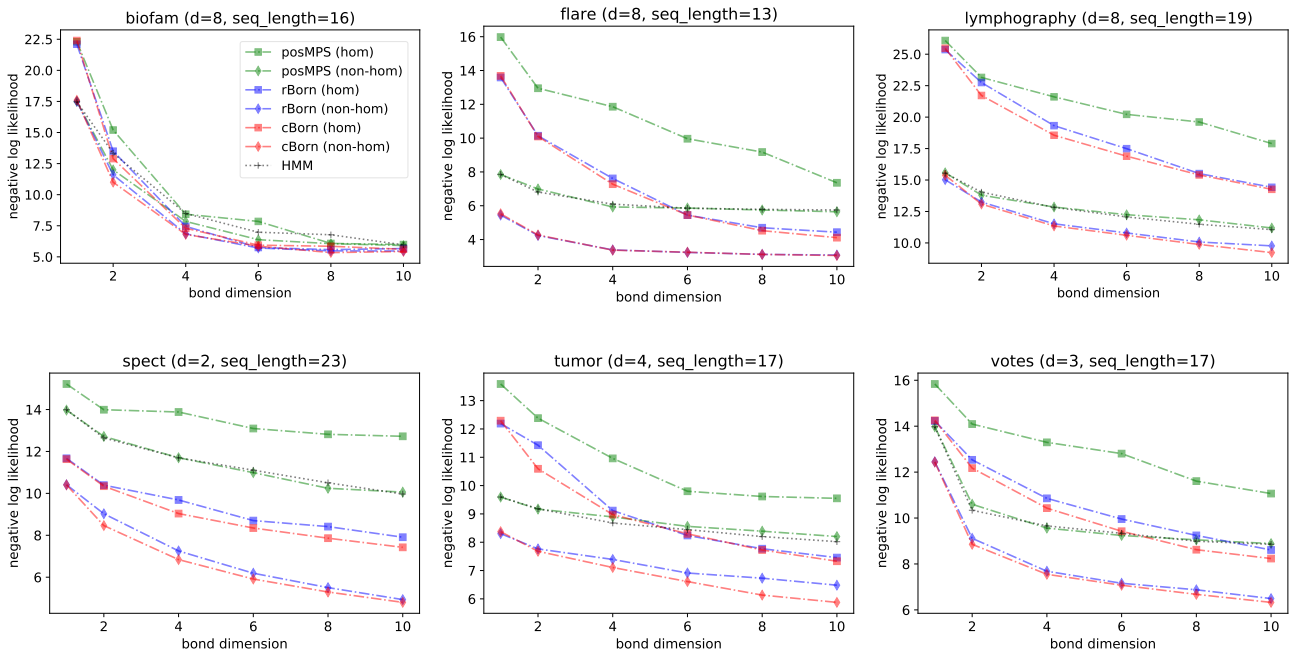


Figure 4: Negative log likelihood for each of the models, on each of the six datasets, showing the effect of varying bond dimension. Shared legend shown in first plot. Each tensor network model was run for 2000 epochs. An HMM model with corresponding hidden dimension is included for comparison (trained with EM for 1000 steps). Note that for the same bond dimension, Born models outperform positive MPS, on all datasets, and additionally, non-homogeneous models outperform corresponding homogeneous models, on all datasets, with the exception of biofam. In all cases, the HMM matches the non-homogeneous positive MPS.

scaling).[5]

7. Conclusion

Based on our results, it is not clear that non-homogeneous models should always be preferred over homogeneous ones in terms of ability to fit to data. It seems that this is highly dependent on the task at hand. In this selection of datasets, it does seem to be the case that the Born machines outperform positive MPS, but it isn't clear whether this would always be the case. In addition, the HMM model performs comparably to the other models given that it has fewer parameters (especially at higher bond dimensions). From a machine learning perspective, it would be interesting to fully understand the theoretical predictions about these models' expressive powers, and how this would relate to the structure that underlies the data.

For all datasets except lymphography, we see that the homogeneous Born models outperform the non-homogeneous positive MPS model at higher bond dimensions. It is possible that with bond dimensions higher than 10, we would see the same pattern for lymphography, though we did not explore high bond dimensions extensively due to computational cost. This suggests that certain datasets may favour the use of Born models. Understanding when this is the case is an interesting empirical question that can help to further understand when these models are appropriate.

One important aspect of these models' behaviour that we would like to test in the future is generalization performance. Tensor networks are claimed to generalize very well (Miller et al., 2020; Bradley et al., 2020), and in their preliminary experiments, Glasser et al. (2019) found that while the positive MPS and HMM achieved nearly identical fit on all datasets, the positive MPS generalized much better. This is an important issue for standard machine learning tasks and is a promising next step in comparing these models.

[5] We also experimented with a similar initialization but instead of all ones, we used a random gaussian initialization centered at 0 with a variance of 1. We then scaled this by $1/\sqrt{D}$ for the boundary vectors and $1/\sqrt{dD^2}$ for the tensor cores. However, this initialization led to instability during training, for reasons that are still unclear to us.

References

Born, M. Quantenmechanik der stoßvorgänge. *Zeitschrift für Physik*, 38(11-12):803–827, 1926.

Bradley, T.-D. and Vlassopoulos, Y. Language modeling with reduced densities, 2020.

Bradley, T.-D., Stoudenmire, E. M., and Terilla, J. Modeling sequences with quantum states: A look under the hood. *Machine Learning: Science and Technology*, 2020.

Bridgeman, J. C. and Chubb, C. T. Hand-waving and interpretive dance: an introductory course on tensor networks. *Journal of Physics A: Mathematical and Theoretical*, 50(22):223001, May 2017. ISSN 1751-8121. doi: 10.1088/1751-8121/aa6dc3. URL http://dx.doi.org/10.1088/1751-8121/aa6dc3.

Cheng, S., Wang, L., Xiang, T., and Zhang, P. Tree tensor networks for generative modeling. *Physical Review B*, 99 (15):155131, 2019.

Critch, A. Algebraic geometry of matrix product states. *Symmetry, Integrability and Geometry: Methods and Applications*, Sep 2014. ISSN 1815-0659. doi: 10.3842/sigma.2014.095. URL http://dx.doi.org/10.3842/SIGMA.2014.095.

Glasser, I., Sweke, R., Pancotti, N., Eisert, J., and Cirac, I. Expressive power of tensor-network factorizations for probabilistic modeling. In *Advances in Neural Information Processing Systems*, pp. 1496–1508, 2019.

Han, Z.-Y., Wang, J., Fan, H., Wang, L., and Zhang, P. Unsupervised generative modeling using matrix product states. *Physical Review X*, 8(3), Jul 2018. ISSN 2160-3308. doi: 10.1103/physrevx.8.031012. URL http://dx.doi.org/10.1103/PhysRevX.8.031012.

Khrulkov, V., Hrinchuk, O., Mirvakhabova, L., and Oseledets, I. Tensorized embedding layers for efficient model compression. *arXiv preprint arXiv:1901.10787*, 2019a.

Khrulkov, V., Hrinchuk, O., and Oseledets, I. V. Generalized tensor models for recurrent neural networks. *CoRR*, abs/1901.10801, 2019b. URL http://arxiv.org/abs/1901.10801.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2017.

Miller, J., Rabusseau, G., and Terilla, J. Tensor networks for probabilistic sequence modeling, 2020.

Novikov, A., Podoprikhin, D., Osokin, A., and Vetrov, D. P. Tensorizing neural networks. *CoRR*, abs/1509.06569, 2015. URL http://arxiv.org/abs/1509.06569.

Robeva, E. and Seigal, A. Duality of graphical models and tensor networks, 2017.

Schreiber, J. pomegranate: Fast and flexible probabilistic modeling in python. *Journal of Machine Learning Research*, 18(164):1–6, 2018. URL http://jmlr.org/papers/v18/17-636.html.

Srinivasan, S., Adhikary, S., Miller, J., Rabusseau, G., and Boots, B. Quantum tensor networks, stochastic processes, and weighted automata, 2020.

Stokes, J. and Terilla, J. Probabilistic modeling with matrix product states. *Entropy*, 21(12):1236, 2019.

Stoudenmire, E. and Schwab, D. J. Supervised learning with tensor networks. In *Advances in Neural Information Processing Systems*, pp. 4799–4807, 2016.