Notes on tensor networks and machine learning

Anderson Misobuchi

January 29, 2022

1 Summary of MPS for supervised learning 1605.05775

This is a summary of the paper by Stoudenmire and Schwab [3], which uses a matrix product state (MPS) for classification of MNIST handwritten digits. Consider a data set $\{\mathbf{x}_n\}_{n=1}^{N_T}$, where each data example is a N-dimensional vector $\mathbf{x} = (x_1, \ldots, x_N)$ (i.e., N features). We consider a classification model in one-versus-all strategy: optimize set of functions indexed by label l

$$f^l(\mathbf{x}) = W^l \Phi(\mathbf{x}) \tag{1}$$

where

- W is the weight vector we want to learn.
- f is the decision function that classifies input $\mathbf{x}$ in class l if l is the index for which $f^l(\mathbf{x})$ is largest.
- Φ is the feature map embedding the data into a higher dimensional space.

One approach for this problem would be to use the kernel trick to deal with these exponentially large vectors. An alternative approach is to approximate the optimal weight vector W by a tensor network. Using tensor networks we can actually work in a very high dimensional space since there are algorithms to perform the contractions of the tensor very efficiently.

MPS decomposition: Approximate of rank-N tensor as a contracted chain of N lower order tensors.

$$W^l_{s_1 s_2 \ldots s_N} = \sum_{\{\alpha\}} A^{\alpha_1}_{s_1} A^{\alpha_1 \alpha_2}_{s_2} \ldots A^{l;\alpha_j \alpha_{j+1}}_{s_j} \ldots A^{\alpha_{N-1}}_{s_N} \tag{2}$$

Feature map: A local feature map $\phi^{s_j}(x_j)$ is applied to each input component x_j of Ndimensional vector $\mathbf{x}$. s_j runs from 1 to d. We will choose $d = 2$ in the following. A simple choice for local map is

$$\phi^{s_j}(x_j) = \begin{pmatrix} \cos \frac{\pi}{2} x_j \\ \sin \frac{\pi}{2} x_j \end{pmatrix} \tag{3}$$

and the feature map can be expressed as the product

$$\Phi^{s_1 s_2 \ldots s_N}(\mathbf{x}) = \phi^{s_1}(x_1) \otimes \phi^{s_2}(x_2) \otimes \ldots \otimes \phi^{s_N}(x_N) \tag{4}$$

Cost function: There might be many other choices of cost functions, but here we considerthe quadratic cost

$$C = \frac{1}{2} \sum_{n=1}^{N_T} \sum_{l} \left(f^l(x_n) - y^l_n \right)^2 \tag{5}$$

where y^l_n are the labels of the data set. If correct label for $\mathbf{x}_n$ is L_n, then $y^{L_n}_n = 1$ and $y^l_n = 0$ for any other l.

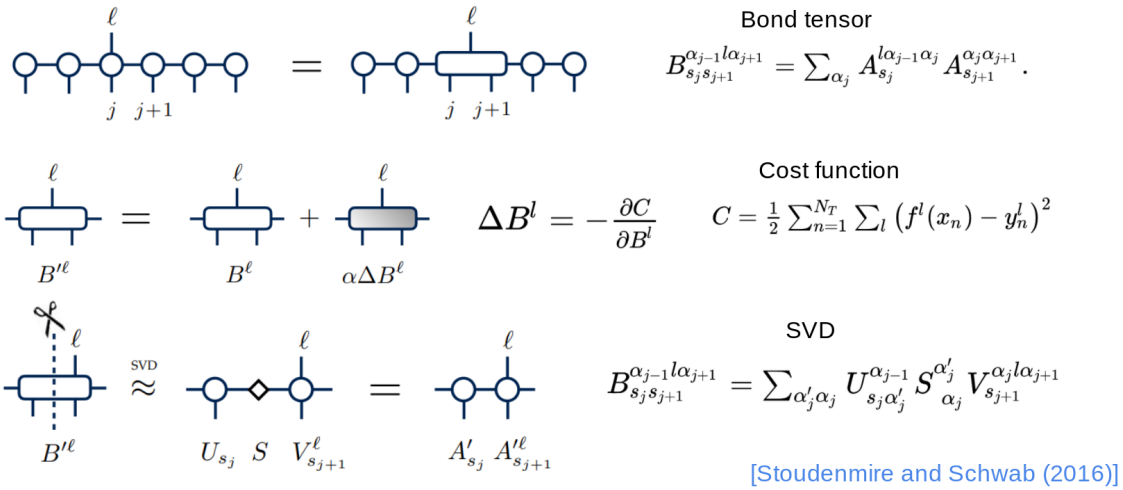

Figure 1: Diagrammatic notation for the supervised learning algorithm using matrix product states.

Bond tensor: The strategy of the algorithm is to vary only two neighboring MPS tensors at time. Consider tensors at sites j and $j+1$ and assume that label index l belongs to site j. Combine MPS tensors $A^l_{s_j}$ and $A_{s_{j+1}}$ by contracting the index α_j. This is going to give a rank 5-tensor we call bond tensor B

$$B^{\alpha_{j-1}l\alpha_{j+1}}_{s_j s_{j+1}} = \sum_{\alpha_j} A^{l\alpha_{j-1}\alpha_j}_{s_j} A^{\alpha_j\alpha_{j+1}}_{s_{j+1}}. \tag{6}$$

Now we want to update the bond tensor in order to decrease the cost function. Take the gradient of C with respect to B. In order to do that, it is convenient to define a projected input $\tilde{\Phi}$

Projected input: The projected input $\tilde{\Phi}$ is obtained by removing the nodes at sites j and $j+1$ from the MPS and contracting it with the feature tensor. Since the jth and $(j+1)$th index of the feature tensor are not contracted, the resulting tensor $\tilde{\Phi}$ is a rank-4 tensor.

Decision function: Decision function is computed from the projected input $\tilde{\Phi}_n$ and the current bond tensor B as

$$f^l(x_n) = \sum_{\alpha_{j-1}\alpha_{j+1}} \sum_{s_j s_{j+1}} B^{\alpha_{j-1}l\alpha_{j+1}}_{s_j s_{j+1}} \left(\tilde{\Phi}_n\right)^{s_j s_{s+1}}_{\alpha_{j-1}\alpha_{j+1}} \tag{7}$$

Gradient update: The gradient of the cost is given by (the full index structure is being omitted)

$$\Delta B^l = -\frac{\partial C}{\partial B^l} = \sum_{n=1}^{N_T} (y^l_n - f^l(x_n))\tilde{\Phi}_n \tag{8}$$

and then the updated B is obtained as $B \leftarrow \eta \Delta B$, where η is the analogous of the learning rate.

SVD: After we obtain the improved B, we want to decompose it using SVD to restore the original MPS form. At the same time, we can truncate the SVD which gives the best low rank approximation. The SVD is computed by treating B as a matrix with collective row index (α_{j-1}, s_j) and collective column index $(l, \alpha_j, s_j + 1)$. The SVD of B gives

$$B^{\alpha_{j-1} l \alpha_{j+1}}_{s_j s_{j+1}} = \sum_{\alpha'_j \alpha_j} U^{\alpha_{j-1}}_{s_j \alpha'_j} S^{\alpha'_j}_{\ \alpha_j} V^{\alpha_j l \alpha_{j+1}}_{s_{j+1}} \tag{9}$$

Sweeping: After restoring the MPS form, the index l is now carried by the $(j+1)$th site, so we can now repeat the same process and keep going back and forth through the MPS until the weights are optimized.

2 Summary of MPS for unsupervised learning 1709.01662

Consider a data set of binary strings $\mathbf{v} = \{v_1, \ldots, v_N\}$, where $v_j \in \{0, 1\}$ and $j = 1, \ldots, N$. In quantum mechanics, probabilities are captured by the norm squared of the wave function $\Psi(\mathbf{v})$ as

$$P(\mathbf{v}) = \frac{|\Psi(\mathbf{v})|^2}{Z}, \quad Z = \sum_{n=1}^{N_T} |\Psi(\mathbf{v}_n)|^2, \tag{10}$$

where Z the normalization factor, often called the partition function, and N_T is the size of the data set. We choose to model the wave function as a matrix product state (MPS)

$$\Psi(v_1, \ldots, v_N) = \sum_{\{\alpha\}} A^{\alpha_1}_{v_1} A^{\alpha_1 \alpha_2}_{v_2} \ldots A^{\alpha_{j-1} \alpha_j}_{v_j} \ldots A^{\alpha_{N-1}}_{v_N} \tag{11}$$

The cost function we want to minimize is the negative log-likelihood (NLL)

$$\mathcal{L} = -\frac{1}{N_T} \sum_{n=1}^{N_T} \log P(\mathbf{v}_n) = \log Z - \frac{1}{N_T} \sum_{n=1}^{N_T} \log |\psi(\mathbf{v}_n)|^2 \tag{12}$$

which is equivalent of minimizing the Kullback-Leibler divergence. The training algorithm is a DMRG-like gradient descent. It allows for dynamical adjustment on the bond dimensions. Initially, take all bond dimensions $D_j = 2$ except on the boundaries. Initialize all sites except the Nth to be left canonical. Canonicalization plays an important role here because it simplifies the computation of the partition function. The update is similar to the algorithm of Stoudemire and Schwab. We merge sites j and $j+1$. This is going to produce a rank 4-tensor

$$B^{\alpha_{j-1} \alpha_{j+1}}_{v_j v_{j+1}} = \sum_{\alpha_j} A^{\alpha_{j-1} \alpha_j}_{v_j} A^{\alpha_j \alpha_{j+1}}_{v_{j+1}}. \tag{13}$$

Then, we want to update B in such a way it will reduce the cost, so we consider the derivative

$$\frac{\partial \mathcal{L}}{\partial B^{\alpha_{j-1} \alpha_{j+1}}_{v_j v_{j+1}}} = \frac{Z'}{Z} - \frac{2}{N_T} \sum_{n=1}^{N_T} \frac{\Psi'(\mathbf{v}_n)}{\Psi(\mathbf{v}_n)}, \tag{14}$$

where Z' and Ψ' are the derivatives with respect to B. These derivatives can be better understood diagrammatically (see Appendix B of [1]). For example, $\psi'(\mathbf{v})$ is what you obtain by removing sites j and $j+1$ from the network (This is analogous to what was called $\tilde{\Phi}$ in Stoudemire and Swchab paper).

After updating B using a gradient descent (or a variant like stochastic gradient descent) step, we want to restore the MPS form and repeat the process in the next site. To do so, we split B using SVD

$$B^{\alpha_{j-1}\alpha_{j+1}}_{v_j v_{j+1}} = \sum_{\alpha'_j \alpha_j} U^{\alpha_{j-1}}_{v_j \alpha' j} S^{\alpha'_j}{}_{\alpha_j} V^{\alpha_j \alpha_{j+1}}_{v_{j+1}}. \tag{15}$$

The SVD is truncated in such a way that the bond dimensions are not greater than a chosen value $D_{\max}$. If we are 'sweeping' from right to left, we update $A_{v_j} = U_{v_j}$ and $A_{v_{j+1}} = SV_{v_{j+1}}$, because we want to preserve the canonical form of the network.

Generative sampling: The generative method is to directly sample using $P = |\psi|^2/Z$. Starting from the Nth bit, we sample this bit from the marginal probability

$$P(v_N) = \sum_{v_1,...,v_{N-1}} P(\mathbf{v}) \tag{16}$$

This can be done easily if all tensors except the last $A^{(N)}$ are left canonical so that

$$P(v_N) = \frac{|A^{(N)}|^2}{Z} = \frac{1}{Z} \sum_{\alpha_{N-1}, v_N} A^{v_N}_{\alpha_{N-1}} A^{\alpha_{N-1}}_{v_N}. \tag{17}$$

Given bit values $v_j, v_{j+1}, ..., v_N$, the $(j-1)$th bit is sampled according to the conditional probability

$$P(v_{j-1}|v_j, v_{j+1}, ..., v_N) = \frac{P(v_{j-1}, v_j, ..., v_N)}{P(v_j, v_{j+1}, ..., v_N)} \tag{18}$$

Roughly speaking, we can compute $P(v_j, v_{j+1}, ..., v_N)$ by removing the first $j-1$ sites of the MPS and contracting the remaining network with itself.

Canonical form: The MPS has a gauge freedom where we can insert the identity $I = MM^{-1}$ on each bound (and M can be different on each bond). We say that $A^{(j)}$ is left canonical if

$$\sum_{v_j \in \{0,1\}} \left(A^{(j)}_{v_j}\right)^\dagger A^{(j)}_{v_j} = I \tag{19}$$

and similarly definition for right canonical. A mixed canonical form, in which all tensors on the left of site j are left canonical and all tensor on the right of site j are right canonical, simplifies the computation of the partition function.

3 Summary of Learning relevant features 1801.00315

This is a summary of the paper [2]. Coarse graining is a key idea used in physics. It helps us to gain insight into a complicated system by marginalizing over smaller length scales while preserving properties at larger scales. Inspired by this idea, this paper aims to apply coarse graining to the data in an unsupervised fashion. This is similar to PCA when used as pre-processing step. Consider a data set $\{\mathbf{x}_j\}_{j=1}^{N_T}$ and the feature map

$$\Phi^{s_1 s_2 ... s_N} = \phi^{s_1}(x_1) \otimes \phi^{s_2}(x_2) \otimes ... \otimes \phi^{s_N}(x_N) \tag{20}$$

where $s_j = 1, 2, ..., d$. The idea of coarse graining applied to the data is to obtain a reduced representation $\tilde{\Phi}^{t_1 t_2}$ of the training inputs as

$$\tilde{\Phi}^{t_1 t_2} = \sum_{s_1 s_2 ... s_N} \mathcal{U}^{t_1 t_2}_{s_1 s_2 ... s_N} \Phi^{s_1 s_2 ... s_N}(\mathbf{x}_N) \tag{21}$$

in which $\mathcal{U}$ will be modeled here as a tree tensor network (TTN). Each layer of the tree network progressively applies a coarse graining to the training set feature vectors. It is useful to view the feature vectors as a matrix Φ_j^s, where

$$\Phi_j^s \equiv \Phi_{s_1 s_2 ... s_N}(\mathbf{x}_j) = \Phi_j^{s_1 s_2 ... s_N} \tag{22}$$

with s being a shorthand notation for $s_1 ... s_N$, so this is a rank $(N+1)$-tensor. Consider the SVD

$$\Phi_j^s = \sum_{nn'} U_n^s S_{n'}^n V_j^{\dagger n'} \tag{23}$$

and recall that the columns of U are orthonormal. More importantly, we can truncate the SVD to efficiently determine the transformation U in truncated form. Now define the feature space covariance matrix as

$$\rho_s^{s'} = \frac{1}{N_T} \sum_{j=1}^{N_T} \Phi_j^{s'} \Phi_s^{\dagger j} = \sum_n U_n^{s'} P_n U_s^{\dagger n} \tag{24}$$

The reason to define ρ is because this will allow us to define the coarse graining algorithm by determining an optimal tree tensor network.

The goal is to compute the eigenvectors U_n^s of the feature space covariance matrix ρ. Eigenvectors of ρ with small enough eigenvalues can be discarded. It is not doable to diagonalize ρ directly, so the strategy is to compute local isometries.

Isometry: A rank 3-tensor $U_t^{s_1 s_2}$ is an isometry if

$$\sum_{s_1 s_2} U_t^{s_1 s_2} U_{s_1 s_2}^{t'} = \delta_t^{t'} \tag{25}$$

Isometries can be interpreted as a unitary rotation followed by a projection.

Computation of first isometry U_1: Define U_1 such that when it acts on s_1, s_2 indices it maximizes the fidelity

$$F = \mathrm{Tr}(\rho) = \frac{1}{N_T} \sum_j \Phi_j^\dagger \Phi_j \tag{26}$$

After coarse graining the feature map using U_1, the fidelity is

$$F_1 = \frac{1}{N_t} \sum_j \Phi_j^\dagger U_1 U_1^\dagger \Phi_j \tag{27}$$

and we want to solve for the optimal U_1 that maximizes F_1. Define the reduced covariance matrix ρ_{12} by tracing over the $s_3, s_4, ..., s_N$ indices, so we can write

$$F_1 = \sum_{s_1 s_2 s_1' s_2' t} \left(U_1^\dagger\right)_{s_1' s_2'}^t \rho_{12}{}^{s_1' s_2'} U_1{}_t^{s_1 s_2} \tag{28}$$

Then, we construct U_1 using the eigenvectors corresponding to the D largest eigenvalues of ρ_{12}. Let us call r the rank of ρ_{12} and call $\{p_i\}_{i=1}^r$ the eigenvalues. We can determine D by using the following criteria

$$E = \frac{\sum_{i=D}^r p_i}{\mathrm{Tr}\ [\rho_{12}]} < \epsilon \tag{29}$$

where ϵ is a chosen threshold. The computation of the remaining isometries can be done in a simliar way (and can be done in parallel). We can compute ρ_{34} by tracing over all

indices except s_3 and s_4. From the diagonalization of ρ_{34} we obtain the second isometry U_2. This process can be done more efficiently by noticing that the reduced covariance matrices typically converge before summing over entire data set. After doing this for every pair of feature vectors we end up with a coarse grained feature tensor with the rank reduced by half. We can iterate the algorithm for $\log_2 N$ steps, where each step corresponds to the construction of one layer of the TTN. At the end, we have approximately diagonalized $\rho = UPU^\dagger$, where U is approximated by a tree tensor network. After learning the tree tensor network, one can use it for both supervised and unsupervised learning tasks.

References

[1] Zhao-Yu Han, Jun Wang, Heng Fan, Lei Wang, and Pan Zhang. Unsupervised generative modeling using matrix product states. *Physical Review X*, 8(3), Jul 2018.

[2] E Miles Stoudenmire. Learning relevant features of data with multi-scale tensor networks. *Quantum Science and Technology*, 3(3):034003, Apr 2018.

[3] E. Miles Stoudenmire and David J. Schwab. Supervised learning with quantum-inspired tensor networks, 2016.