Matrix Product States

Glenn LeBlanc *

May 17, 2021

1 Introduction

1.1 Many-Body Wavefunctions as Tensors

Consider a spin-$\frac{1}{2}$ particle or two-level system. The systems's state is given by $|\psi\rangle \in \mathbb{C}^2$, and for some basis labeled by $\{|0\rangle, |1\rangle\}$ we can write

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$$

with

$$|\alpha|^2 + |\beta|^2 = 1.$$

This is the principle of superpostion– the system is in a superposition of the two basis states $|0\rangle$ and $|1\rangle$. Now if we add a second spin-$\frac{1}{2}$ particle or a second two-level system, the state of the many-body system is described by a vector $|\Psi\rangle$ in a tensor product Hilbert space $\mathcal{H} = \mathbb{C}^2 \otimes \mathbb{C}^2$ and may be in some superposition of the four basis states

$$|\Psi\rangle = \alpha |00\rangle + \beta |01\rangle + \gamma |10\rangle + \delta |11\rangle .$$

Generally, for N qubits (qudits) the system is fully parameterized by 2^N (d^N) complex numbers: $|\Psi\rangle \in \mathbb{C}^{2^N}$. Notice the exponential scaling here– storing the individual coefficients of $|\Psi\rangle$ for 30 sites requires about 8 GB of memory, and dealing with the raw coefficients of 40 sites requires over 8 TB. If we want to time evolve $|\Psi\rangle$ under some Hamiltonian $\hat{H}$

$$|\Psi(t)\rangle = e^{-it\hat{H}/\hbar} |\Psi\rangle$$

we need to exponentiate the matrix $-it\hat{H}/\hbar \in \mathbb{C}^{2^N \times 2^N}$ and multiply the vector $|\Psi\rangle$ by this result, which takes $\mathcal{O}\left(\left(2^N\right)^3\right)$ time. Thus it is extremely worthwhile to find more efficient ways of storing and manipulating the quantum state $|\Psi\rangle$ without completely sacrificing fidelity. As a first step, it is useful also to notice the isomorphism between the two spaces

$$\mathbb{C}^{2^N} \cong \mathbb{C}^{2\times\cdots\times 2}$$

meaning we can instead conceptualize $|\Psi\rangle$ as a *tensor*:

$$|\Psi\rangle \in \mathbb{C}^{2\times\cdots\times 2}$$

where in this paper a tensor Ψ is just a multi-dimensional array with some number of indices such that plugging in an assignment for each index spits out a complex number. More succinctly,

$$\Psi_{i_1 i_2 \cdots i_N} \in \mathbb{C}$$

A *contraction* between two tensors Ψ and Φ is a summation over a shared index:

$$T_{ijlm} = \sum_k \Psi_{ijk} \Phi_{lkm}$$

is an example of a contraction. Note that dot products, matrix multiplication, and trace are all

*This project was part of the Berkeley Physics Directed Reading Program, which allows undergraduates to explore novel material under the auspices of a graduate student mentor. Karthik Siva very graciously directed this project.

different vestiges of tensor contraction:

$$a \cdot b = \sum_k a_k b_k$$
$$(Ax)_i = \sum_k A_{ik} x_k$$
$$\mathrm{Tr}(A) = \sum_k A_{kk}$$

1.2 Tensor Networks

A *tensor network* is an undirected graph whose nodes represent tensors and whose edges correspond to tensor indices. An edge between two tensors corresponds to a contraction along the depicted axis of each tensor. Use of this graphical language for representing quantum systems is attractive since it unveils relevant entanglement properties and simplifies the expression of tensor network operations involving many indices [1]. The *rank* of a tensor is given by the number of indices that are associated with it— this is somewhat of an abuse of terminology, since rank has an entirely different meaning when associated with linear operators, and any tensor is isomorphic to a (multi)linear operator.

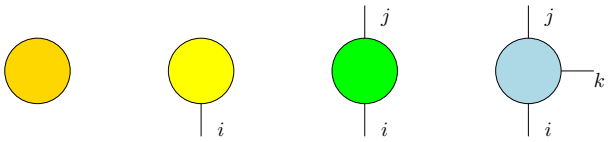

Figure 1: *A graphical depiction of a scalar c, vector v_i, matrix M_{ij}, and a tensor T_{ijk} of rank three.*

See figure 1 for a graphical depiction of tensors of rank zero to three. Each tensor is denoted as a node with free edges representing each index.

Figure 2 depicts the previous examples (dot product, matrix multiplication, trace) of tensor contraction using this graphical language.

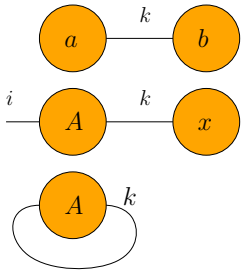

Figure 2: *A graphical depiction of three examples of tensor contraction.*

An example of an insight this graphical language provides is in proving trace cyclicality. The standard proof is as follows:

$$\begin{aligned}\mathrm{Tr}(ABC) &= \sum_{ijk} A_{ij} B_{jk} C_{ki} \\ &= \sum_{ijk} C_{ki} A_{ij} B_{jk} \\ &= \mathrm{Tr}(CAB)\end{aligned}$$

The tensor network proof is depicted in figure 3:

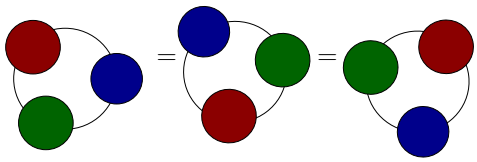

Figure 3: *Proof of trace cyclicality.*

The graphical depiction of trace provides simpler insight into the cyclic structure of the underlying tensor contractions required to calculate $\mathrm{Tr}(ABC)$, and thus allows for a totally visual and immediately obvious proof of the invariance of trace under cyclic permutations of A, B, and C.

2 Matrix Product States

A *matrix product state* (MPS) is a particular class of tensor network consisting of a chain of tensors each having one dangling edge and a bond between their nearest neighbors. See figure 4 for an example of an MPS with three sites with periodic and open boundary conditions. From here we only consider MPS with open boundary conditions.

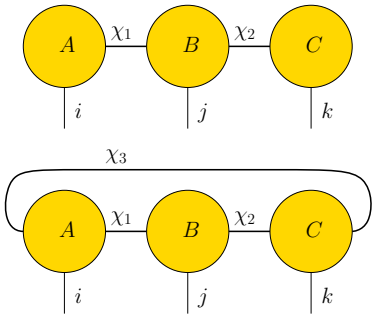

Figure 4: *Two three-site MPS, the first lacking periodic boundary conditions and the second with open boundary conditions.*

The indices α, β, γ in figure 4 are called *bond indices* and are associated with a *bond dimension* χ. The free indices i, j, k are *site indices*. Algebraically, the MPS decomposition of a tensor Ψ is written as:

$$\Psi_{i_1 i_2 \cdots i_N} = \sum_{\chi_1 \chi_2 \cdots \chi_{N-1}} A_{i_1}^{[1]\chi_1} A_{i_2}^{[2]\chi_1 \chi_2} \cdots A_{i_N}^{[N]\chi_{N-1}}$$

where the first and last local tensors $A^{[1]}$ and $A^{[N]}$ are matrices and every other tensor in the chain is of rank three. If each bond index is constrained to be of the same dimension χ then this representation approximates the 2^n coefficients of Ψ using

$$2\chi + (N-2)2\chi^2 + 2\chi = 4\chi + (2N-4)\chi^2$$

parameters, which grows as $O(N)$ for fixed χ. For a given choice of site indices (e.g., $i_1 = 0$, $i_2 = 1$, $i_3 = 0$, $\cdots$), the coefficient $\Psi_{010\ldots}$ is given by a matrix product– hence the name matrix product state.

Consider a state $|\Psi\rangle \in \mathbb{C}^{2\times\cdots\times 2}$ associated with N qubits. The *Schmidt decomposition* of $|\Psi\rangle$ with respect to a partition of sites $A : B$ is written

$$|\Psi\rangle = \sum_{\alpha}^{\chi_A} \lambda_\alpha |\Phi_\alpha^{[A]}\rangle \otimes |\Phi_\alpha^{[B]}\rangle$$

where χ_A is the *Schmidt rank* of the partition $A : B$ and is a natural measure of the entanglement between the qubits in A and those in B [2]. If $\max_A \chi_A = \chi$ and

$$2^N > 4\chi + (2N-4)\chi^2,$$

then the MPS decomposition of $|\Psi\rangle$ with uniform bond dimension χ is exact and requires less memory than storing the individual 2^N coefficients of $|\Psi\rangle$.

2.1 Left normality, right normality

Left and right normality is loosely a a generalization of orthonormality. Algebraically, if a tensor $T_{lir} \in \mathbb{C}^{b\times d\times b}$ is right normal, then

$$\sum_{ir} T_{lir} T^*_{l'ir} = \delta_{ll'}$$

and likewise if T is left normal then

$$\sum_{il} T_{lir} T^*_{lir'} = \delta_{rr'}.$$

See figure 5 for a graphical depiction of this property.

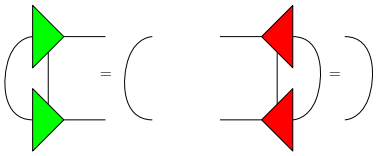

Figure 5: *Left normality (green) and right normality (red). Bottom tensors are complex conjugated.*

2.2 Generating an MPS from a tensor

We now give a presription for how to express an arbitrary wavefunction as an MPS of a given bond dimension, starting from an exact represntation as a vector. The first step of the iterative process for generating an MPS from a tensor is depicted in figure 6 for an initial tensor with three indices, each of dimension d. The input tensor ψ is reshaped into a matrix ψ_{MAT} by squashing indices the two leftmost indices into one index while keeping the rightmost index separated. The singular value decomposition (see appendix) is then performed on ψ_{MAT}. Σ is truncated and renormalized to Σ' such that the bond index between V and Σ is less than or equal to the bond dimension

χ. The truncated matrix Σ' is contracted to the left into U, resulting in the $d^2 \times \chi$ matrix ψ'_{MAT}. Finally, the index of dimension d^2 is unsquashed into two indices of dimension d each. The process is repeated for ψ', with the only change being that the middle site index is squashed with the bond index and the leftmost site index is kept separate. This process is repeated iteratively for every site index in the original tensor ψ until the tensor corresponding to the leftmost site is the only untouched tensor. See figure 7 for a depiction of this iterative process.

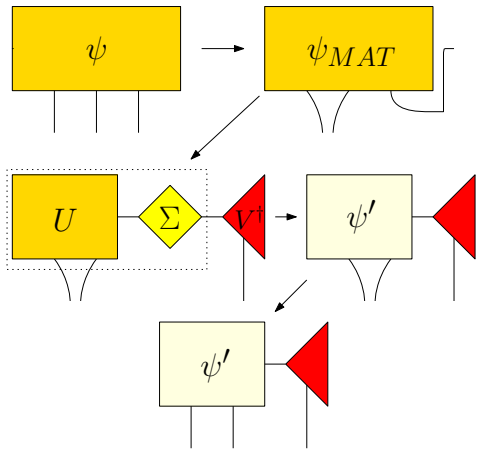

Figure 6: *Separating the first site index from ψ.*

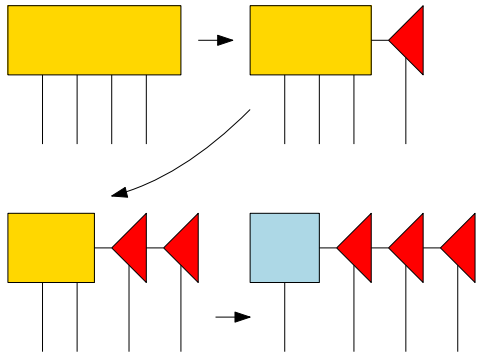

Figure 7: *How an MPS is generated using iterative SVD with an input tensor with four indices. The final orthogonality center is in blue.*

The leftmost site in figure 6 is an orthonormal matrix by definition of the SVD. Non-boundary sites in the final chain resulting from

If an MPS consists entirely of a nonnegative (possibly zero, as in figure 7) number of left normal tensors facing a single unconstrained tensor followed by a chain of right normal tensors we call the single unconstrained tensor the *orthogonality center*.

2.3 Moving the Orthogonality Center

The orthogonality center presents a useful way of evaluating the expectation of a local operator M on site i, depicted in figure 8.

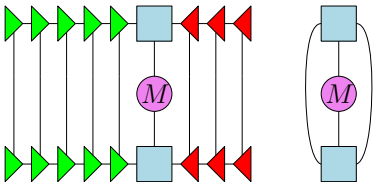

Figure 8: *A quick way to find $\langle M^{(i)} \rangle_\psi$ exploiting orthogonality. Left: the full contraction equivalent to $\langle \psi_{MPS} | M^{(i)} | \psi_{MPS} \rangle$. Left and right normal tensors can be removed, leaving the contraction on the right as the only necessary computation.*

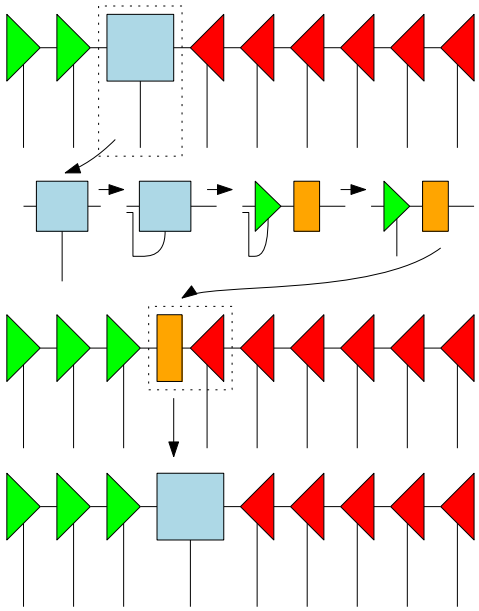

Figure 9: *Moving the orthogonality center one site to the right using the QR factorization. This can be repeated to move the orthogonality center to any site to the right of its current position. To move the orthogonality center to the left, a similar process is used substituting the LQ factorization for the QR factorization.*

However, this requires that the orthogonality center is already located on site i. Since this clearly will not always be the case, we need a way to move the orthogonality center from one site to another. The general idea is depicted in figure 9. Briefly, the orthogonality center's site index is squashed with the left (right) bond index. Next it is split into two matrices using the QR (LQ) decomposition, and the left (right) normal matrix is reshaped into a rank three tensor. The remaining matrix is contracted into the corresponding right (left) neighbor tensor.

2.4 Time Evolving Block Decimation

Time evolving block decimation (TEBD) is a method for efficiently simulating 1D quantum systems with low entanglement. By leveraging the fidelity of the system's MPS decomposition, a Suzuki-Trotter [3] decomposition of the propogator

$$\hat{U}(t) = \exp\{-i\hat{H}t/\hbar\}$$

can be applied step-by-step to the MPS with bond dimensions truncated after each step. For the 1-D transverse field Ising model

$$\hat{H} = \sum_{i=1}^{N-1} \sigma_z^{(i)}\sigma_z^{(i+1)} + J\sum_{i=1}^{N} \sigma_x^{(i)} \tag{1}$$

this amounts to applying p Trotterized operators (setting $\hbar = 1$):

$$\hat{U}(t) \approx \left[\left(\prod_{i=1}^{N-1} e^{-i\sigma_z^{(i)}\sigma_z^{(i+1)}\Delta t}\right)\left(\prod_{i=1}^{N} e^{-i\sigma_x^{(i)}\Delta t}\right)\right]^p$$

where the step size $\Delta t = \frac{t}{p}$. A depiction of one TEBD sweep with this Hamiltonian is shown in figure 10.

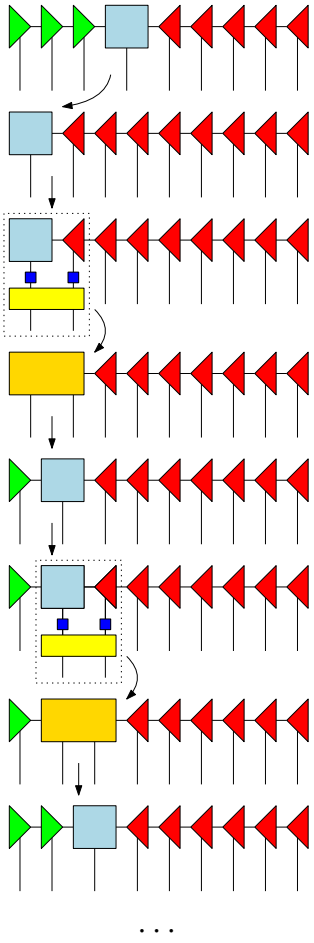

Figure 10: *One step of time evolving block decimation. The orthogonality center is moved to the leftmost site. Trotterized operators $R_x(\Delta t)$ in blue and $R_{zz}(\Delta t)$ in yellow are applied to the orthogonality center and its rightmost neighbor. A SVD is performed and the singular value matrix is truncated and contracted to the right to enforce the bond dimension χ. The sweep continues until reaching the last site, at which point the orthogonality center is again moved to the leftmost site and the above process is repeated $p-1$ additional times.*

This approximation is not sufficient when the Hamiltonian $\hat{H}$ generates sufficiently high entanglement, since enforcing the maximum bond dimension χ bounds the maximum amout of entanglement the MPS can ever reach. If, however, $\hat{H}$ consists entirely of single-site terms, then this method is exact since the systems entanglement cannot grow through time evolution under $\hat{H}$. See

the rightmost heatmap in figure 12 for an example of what time evolution under a nonentangling Hamiltonian might look like.

3 Package Overview

This section provides an overview of the real core of this project: a Julia package for creating and manipulating matrix product states, located at `https://github.com/gl3nnleblanc/pdrp2021`.

3.1 Toolchain

This package is written in Julia, a modern programming language incubated at MIT in 2009 and designed from the beginning with high performance in mind [4]. Julia is fast, easy to use, and open source. Git and TravisCI proved useful during development.

3.2 Algorithms

The following section contains pseucode outlining the main ideas of what was implemented in the package. Again, the full code written in Julia is available at the repository.

Algorithm 1: Helper function for splitting a tensor.

Data: input tensor T, d_{left}, d_{right}, χ
Result: length(Σ), T_{next}, V
$T_{MAT} \leftarrow$ reshape(T, d_{left}, d_{right});
T_{next}, Σ, $V^\dagger \leftarrow$ SVD(T_{MAT});
$V \leftarrow (V^\dagger)^\dagger$;
$\Sigma \leftarrow$ truncate_to_chi(Σ, χ);
$T_{next} \leftarrow$ truncate_to_chi(T_{next}, χ);
return length(Σ), T_{next}, V;

Algorithm 1 is a helper method to slice a tensor along some specified axis using the SVD, and runs in time $\mathcal{O}(d_{left}^2 d_{right} + d_{left} d_{right}^2)$ [5].

Algorithm 2: Generating an MPS from input tensor Ψ.

Data: input tensor Ψ, desired bond dimension χ
N $\leftarrow$ tensor rank of Ψ;
sites $\leftarrow$ [ ];
d_{next}, T_{next}, $T_{curr} \leftarrow$ split_tensor(Ψ, 2^{N-1}, 2, χ);
sites.append(T_{curr});
for $i \leftarrow$ *2 to* $N-1$ **do**
  d_{next}, T_{next}, $T_{curr} \leftarrow$ split_tensor(T_{next}, 2^{N-i}, 2^i, χ);
  if *length*(T_{curr}) $= 2\chi^2$ **then**
    $T_{curr} \leftarrow$ reshape(T_{curr}, χ, 2, χ);
  else
    $d_{prev} \leftarrow$ length(T_curr) $\div$ $(2 \times d_{next})$;
    $T_{curr} \leftarrow$ reshape(T_{curr}, d_{next}, 2, d_{prev});
  end
  sites.append(T_{curr});
end
_, T_{next}, $T_{curr} \leftarrow$ split_tensor(T_{next}, 2, d_{prev}, χ);
$T_{curr} \leftarrow$ reshape(T_curr, 2, 2, :);
sites.append(T_{curr});
sites.append(T_{next});
return MPS(sites) having bond dimension χ and orthogonality center at N;

Algorithm 2 returns an MPS wrapper consisting of an array of tensors arranged from left to right in descending order. Said another way, the Nth entry of this array is the leftmost site. For an input tensor T of rank N with each index having dimension 2, algorithm 2 requires N SVD computations on matrices of logarithmically decreasing size, resulting in a runtime of $\mathcal{O}(2^{2N})$, which is dominated by the runtime of the first SVD. The truncated SVD can be used to avoid exponential cost, but was not in this package, as there are currently no suitable TSVD implementations in Julia. (This would be a good next step.)

Algorithm 3: Moving the orthogonality center.

Data: MPS M, target_site
Result: MPS M'
$N \leftarrow M$.sites.length;
if *target_site* $= M.$*orthogonality_site* **then**
  return M;
else
  updated_sites $\leftarrow$ copy(M.sites);
  if *target_site* $> M.$*orthogonality_site* **then**
    for *every site between* $M.$*orthogonality_site and target_site* **do**
      $d_{left} \leftarrow$ site.left_dim;
      L, $Q \leftarrow$ LQ(reshape(site, d_{left}, :));
      contract L into left neighbor and replace entry in updated_sites;
    end
  else
    for *every site between* $M.$*orthogonality_site and target_site* **do**
      $d_{right} \leftarrow$ site.right_dim;
      Q, $R \leftarrow$ QR(reshape(site, :, d_{right}));
      contract R into right neighbor and replace entry in updated_sites;
    end
  end
end
return M'(updated_sites);

Algorithm 4: A single TEBD sweep.

Data: MPS M, H_1, H_2, t
Result: MPS M'
updated_sites = new array;
previous_site $\leftarrow M$.sites[N];
for $i = N$ *to* 2 **do**
  left_site $\leftarrow$ previous_site;
  right_site $\leftarrow M$.sites[i-1];
  $\psi_{ab\chi} \leftarrow$ contract(left_site, right_site);
  $\psi_{ab\chi} \leftarrow \sum_{i,j,k,l} \psi_{ij\chi}$ $[\exp(-itH_1/2)]_{ik}[\exp(-itH_1/2)]_{jl}$ $[\exp(-itH_2)]_{abjl}$;
  new_left_site, new_right_site $\leftarrow$ split_tensor(ψ, 2×left_edge_dim, 2×right_edge_dim);
  new_left_site $\mapsto$ updated_sites[i];
  previous_site $\leftarrow$ new_right_site;
end
previous_site $\mapsto$ updated_sites[1];
return M'(updated_sites);

Algorithm 4 performs a single sweep of TEBD, as depicted in figure 10. H_1 and H_2 represent the single and double body terms, respectively, of the Hamiltonian of interest. The runtime of a single sweep is $\mathcal{O}(N\chi^3)$ [2].

Algorithm 5: Full TEBD.

Data: MPS M, H_1, H_2, t, resolution
Result: MPS M'
$M' = M$;
for *1 to resolution* **do**
  $M' =$ single_tebd_step(M', H_1, H_2, t/resolution);
end
return M';

The runtime for 5 is clearly $\mathcal{O}(rN\chi^3)$ with resolution r.

Algorithm 3 moves the orthogonality center of an MPS. Since $\mathcal{O}(N)$ QR (LQ) decompositions and on matrices of size $2\chi \times \chi$ ($\chi \times 2\chi$) are required, along with $\mathcal{O}(N)$ contractions along indices of dimension χ, the total runtime is $\mathcal{O}(N\chi^3)$.

3.3 Examples

There are three example notebooks contained in the repository (as linked above). Some highlights:

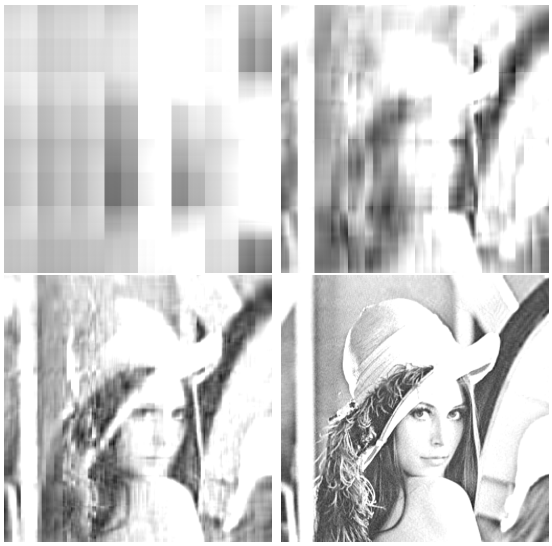

Figure 11: *Compressing an image with an MPS. Clockwise from top left:* $\chi = 2$, $\chi = 9$, $\chi = 20$, $\chi = 100$, *with compression ratios of 1927.5, 126.2, 31.3, and 2.24, respectively. Lossless storage as an MPS results in a compression ratio less than 1.*

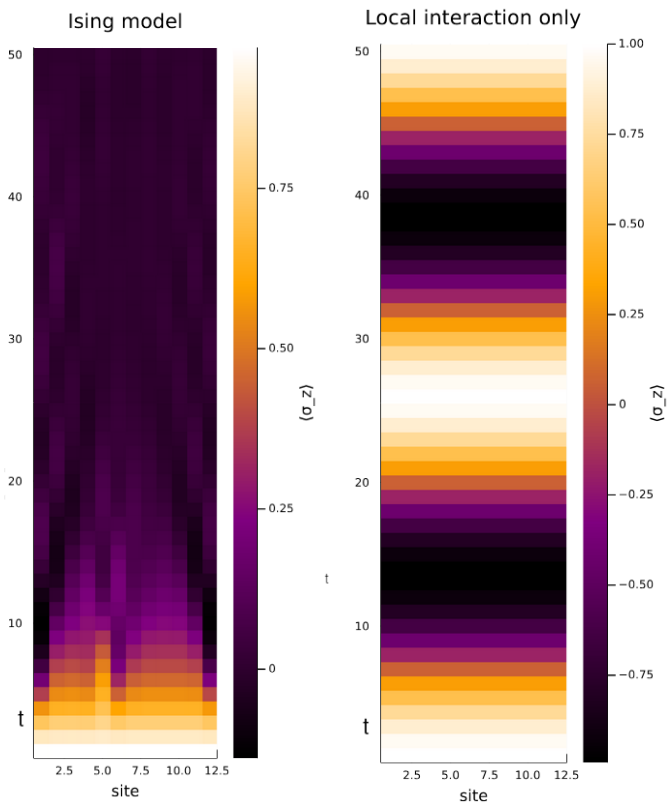

Figure 12: *Heatmaps of* $\langle \sigma_z \rangle$ *for each site in a chain of 12 qubits time evolving under a* $(J = 1)$ *1D Ising model with (left) and without (right) the local interaction term* $\sigma_z^{(i)}\sigma_z^{(i+1)}$.

Figure 11 displays four images compressed as MPS with successively higher bond dimensions. The initial 512×512 image is mapped to a tensor with 18 indices, each having dimension 2. The resulting tensor is fed into algorithm 2 over a variety of different bond dimensions. $\chi = 100$ results in very few visual artifacts, although artifacts become more visible when either the full RGB spectrum is used or the original image contains noisier features. Lossless storage as an MPS requires $\chi = 512$, which results in a compression ratio $\approx 0.3 < 1$, meaning redundant information is required to store the image as an MPS in this case, and hence there is no advantage to using an MPS when lossless compression is required. It would be interesting to investigate better methods of transforming the input image into a rank 18 tensor, as the visual artifacts seem to be closely tied to the way in which the image was reduced to a tensor.

Figure 12 displays heatmaps of $\langle \sigma_z \rangle$ for each site in a chain of 12 qubits time evolving under a $(J = 1)$ 1D transverse field Ising model (equation 1) with (left) and without (right) the local interaction term $\sigma_z^{(i)}\sigma_z^{(i+1)}$. Time increases from bottom to top. Using an MPS with $\chi = 32$ and running 1000 sweeps on an Intel i5-6600k @ 3.9 GHz took ≈ 3 minutes, whereas the same calculation with the full 4096×4096 Hamiltonian takes over 25 minutes. Gains in runtime efficiency increase exponentially with the number of sites for fixed χ.

Figure 13 plots the same values as the left heatmap in figure 12, but with the value of $\langle \sigma_z \rangle$ at each site superimposed on top of each other. At $J \approx 1$, $\langle \sigma_z \rangle \to 0$ with increasing t.

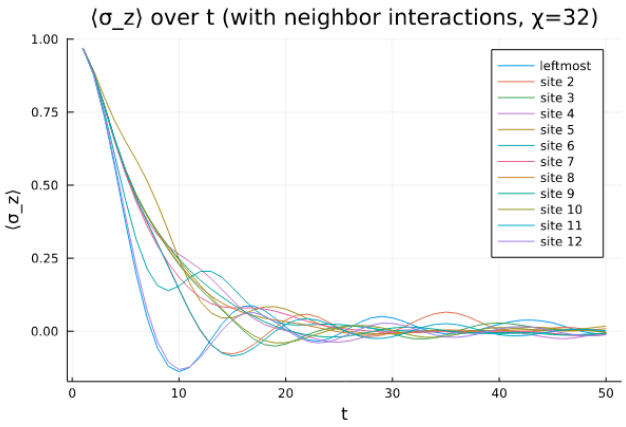

Figure 13: *Graph corresponding to the left heatmap in figure 12.*

3.4 Next Steps

This project felt extremely rewarding and I'm planning to continue sporadically adding things to it over the summer when I have the time. Two things that I'd like to implement are DMRG [6] and a neural network using an MPS whose sites correspond to lifted features [7, 8].

References

[1] R. Orus, A Practical Introduction to Tensor Networks: Matrix Product States and Projected Entangled Pair States, Annals Phys. 349 (2014) 117–158. arXiv:1306.2164, doi:10.1016/j.aop.2014.06.013.

[2] G. Vidal, Efficient classical simulation of slightly entangled quantum computations, Phys. Rev. Lett. 91 (2003) 147902. doi:10.1103/PhysRevLett.91.147902.
URL https://link.aps.org/doi/10.1103/PhysRevLett.91.147902

[3] N. Hatano, M. Suzuki, Finding exponential product formulas of higher orders, Lecture Notes in Physics (2005) 37–68doi:10.1007/11526216_2.
URL http://dx.doi.org/10.1007/11526216_2

[4] J. Bezanson, A. Edelman, S. Karpinski, V. B. Shah, Julia: A fresh approach to numerical computing, SIAM review 59 (1) (2017) 65–98.
URL https://doi.org/10.1137/141000671

[5] G. Golub, C. Van Loan, Matrix Computations, 4th Edition, The Johns Hopkins University Press, Baltimore, MD, 2013.

[6] N. Nakatani, Matrix product states and density matrix renormalization group algorithm, in: Reference Module in Chemistry, Molecular Sciences and Chemical Engineering, Elsevier, 2018. doi:https://doi.org/10.1016/B978-0-12-409547-2.11473-8.
URL https://www.sciencedirect.com/science/article/pii/B9780124095472114738

[7] S. Efthymiou, J. Hidary, S. Leichenauer, Tensornetwork for machine learning (2019). arXiv:1906.06329.

[8] W. Huggins, P. Patil, B. Mitchell, K. B. Whaley, E. M. Stoudenmire, Towards quantum machine learning with tensor networks, Quantum Science and Technology 4 (2) (2019) 024001. doi:10.1088/2058-9565/aaea94.
URL https://doi.org/10.1088/2058-9565/aaea94

A Singular Value Decomposition

The *singular value decomposition* decomposes a matrix $A \in \mathbb{F}^{n \times m}$ as

$$A = U \Sigma V^\dagger$$

where $V \in \mathbb{F}^{m \times m}$ is orthonormal, $\Sigma \in \mathbb{F}^{n \times m}$ is diagonal, and $U \in \mathbb{F}^{n \times n}$ is orthonormal. Equivalently,

$$A = \sum_{i=1}^{r} \sigma_i \left|u_i\right\rangle \left\langle v_i\right|$$

for orthonormal sets $\{|u_i\rangle\}_{i=1}^n$, $\{|v_i\rangle\}_{i=1}^m$ and non-negative values σ_i, with $\text{rank}(A) = r$. The time complexity to calculate the SVD of a A is $\mathcal{O}(n^2 m + m^2 n)$ [5].

B QR and LR Decompositions

The QR decomposition of a matrix $A \in \mathbb{F}^{n \times m}$ decomposes A as

$$A = QR$$

where $Q \in \mathbb{F}^{n \times n}$ is orthonormal and $R \in \mathbb{F}^{n \times m}$ is upper triangular. Similarly, the LQ decomposition expresses A as

$$A = LQ$$

where $Q \in \mathbb{F}^{m \times m}$ is orthonormal and $L \in \mathbb{F}^{n \times m}$ is lower triangular. Either decomposition can be calculated by the Gram-Schmidt process in $\mathcal{O}(m^2 n)$ time.