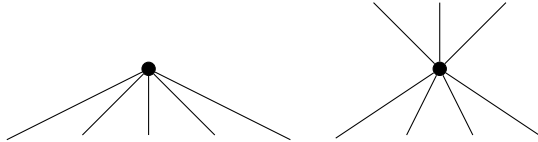


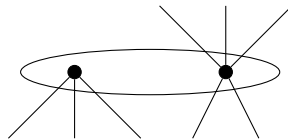
1 Tensors

Tensors are a representation of linear operators. Much like with bra-ket notation, we want a notation which will suggest correct operations. We can represent a tensor as a point with n legs radiating downward: it accepts n inputs (vectors) and gives a number back. Alternatively, we could have some legs radiating in (inputs) and some out (outputs).

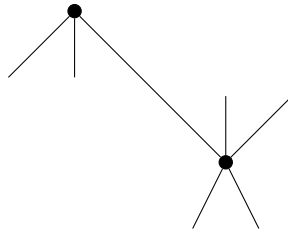


We can define two operations:

Tensor product Take two tensors, get together enough inputs to feed into both, and take the product of their results.




Contraction Take two tensors, attach a leg from each one to the other, and sum over all possible labels for that edge.



1.1 Examples

Matrix Multiplication Take two tensors, each with two legs (one in and one out), and contract them together. This yields another tensor with two legs: the matrix product.

Inner product Contract two one-input tensors together (flip one by complex conjugating) to get a number.

Trace Take a 1 to 1 tensor and contract the legs with each other. 

1.2 Tensor Network

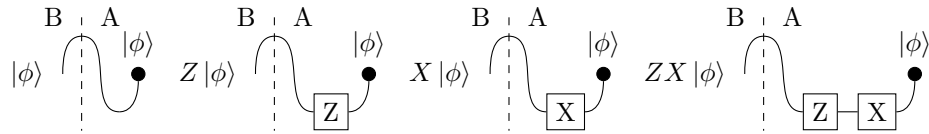
You can join tensors in these ways to create a tensor network. The network will have some “open” edges, and labeling them will give you a value: you take the product over all the original tensors, summing over all contracted “closed” edges. The order of contraction doesn’t matter, so you don’t need to know the history of how the network was assembled.

1.3 Teleportation

We can represent the bell state as a tensor of two inputs that spits out 1 if the labels on the inputs match. Furthermore, we can generate the other 4 members of the bell basis by jamming an X gate, a Z gate, or both onto one of the lines. For example:

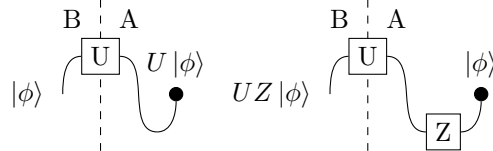
$$\begin{array}{c} \bullet \\ \text{---} \end{array} = \begin{array}{c} \text{---} \\ \text{---} \end{array} = |00\rangle + |11\rangle \quad \begin{array}{c} \text{---} \\ \text{---} \\ \boxed{Z} \end{array} = |00\rangle - |11\rangle$$

We can figure out “what would happen if alice measured her two qbits and got the bell basis” by attaching a bell state to the two qbits she’s measuring. If we do that, we get a straight line from Alice’s qbit to Bob’s, and sure enough, Bob now has Alice’s qbit. If she instead measured $|00\rangle - |11\rangle$, there would be a Z gate along the path. There are 4 possible outcomes, corresponding to the 4 possible measurements:



1.4 Computation by teleportation

Every computation is essentially a change of basis. Now when we teleport, why do we get our output in the same space? We get our output in the same basis because the bell state entangles $|0\rangle$ with $|0\rangle$ and $|1\rangle$ with $|1\rangle$. We could insert a unitary operator into our bell state:



If we measure $|00\rangle + |11\rangle$, Bob will get $U|\phi\rangle$. If we are less fortunate and measure $|00\rangle - |11\rangle$, we’ll get $UZ|\phi\rangle$, which is potentially inconvenient. To correct our result, we need to apply UZU^{-1} . However, if we can get around this snag, we can do our computation before we get our data.

1.5 Pauli Group

The Pauli Group is:

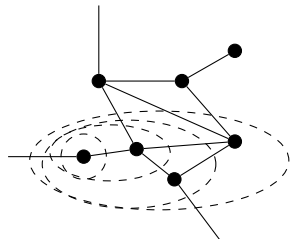
$$C_1 = \pm\{I, X, Z, Y\}$$

There's a bigger group, the Clifford Group C_2 , which commutes Paulis to Paulis. For example, $H \in C_2$, and $HXH^{-1} = Z$. There's a yet bigger set (not a group), C_3 , which commutes Paulis to Cliffords. C_3 contains $\pi/4 = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$. If your computation is in C_3 , your correction is in C_2 , and if your computation is in C_2 , your correction is in C_1 . You can repeatedly teleport, each time computing an easier correction until you hit C_1 .

1.6 Bubble Width

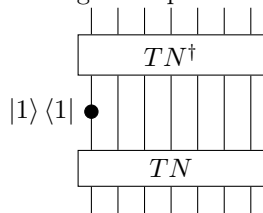
We can take a tensor network, and draw a "bubble" around one of the tensors. We're sad about every edge that goes through our bubble since we need to keep track of it. We want a sequence that minimizes the maximum number of edges through our bubble: that minimum is called the bubble width.

It takes d^w time to classically evaluate the tensor network.



1.7 Quantum circuits as Tensor Networks

You can write quantum circuits as tensor networks. If we want to figure out what the acceptance probability is of some tensor network, we can stick the tensor network on top of itself upside down. By projecting the output bits onto 1, we can get the probability of outputting 1.



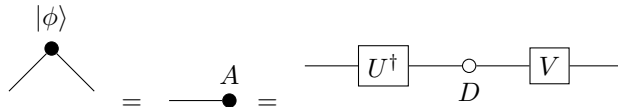
If we have a vector living in the tensor product of two spaces, we can schmidt decompose it:

$$|\phi\rangle = \sum_i \lambda_i |u_i\rangle \otimes |v_i\rangle$$

and write it as a matrix (rows being basis vectors of one space, columns of the other).

$$|\phi\rangle = U A V^\dagger = D$$

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

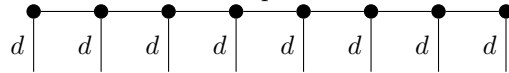


You can characterize the thickness of the bond by K (the rank of D , or alternately the number of nonzero λ_i).

1.8 MPS (Matrix Product State)

See <http://www.nature.com/nphys/journal/v11/n7/abs/nphys3345.html>

We can write some n -qubit state as a matrix product state, like so:



Say we have a gapped local hamiltonian in 1D, with ground state $|\Omega\rangle$. We would like to approximate the ground state by an MPS with low Schmidt rank. By default, we would expect the bond dimension to go as 2^{n-1} , which is going to be untractable.

$|\phi_{k_\epsilon}\rangle \approx_\epsilon |\Omega\rangle$, of Schmidt rank $\leq k_\epsilon$.

How does k_ϵ vary with ϵ ?

Well, worst case, the errors from each bond add up and we get $n\epsilon$. It would be nice to do better, and it's an open question if you can. Doing so would allow better algorithms.

$$\epsilon = \frac{1}{\text{poly}(n)} \quad k = 2^{\mathcal{O}(\log^{2/3} n)}$$

We want to take a product state not orthogonal to the ground state, then shrink it's orthogonal components. To do so, we'll use a (D, Δ) AGSP (Approximate Ground State Projector).

$$D = (\ln d)^s$$

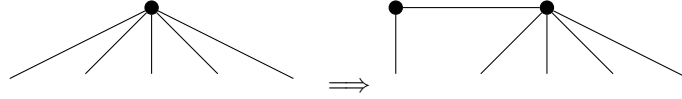
We want to shrink them by a factor of:

$$\Delta = e^{-s^{3/2}/u^{1/2}} = 1/\text{poly}(n)$$

$$D = (\ln d)^{u^{1/3} \ln^{2/3}(n)}$$

So if we just had one cut, we could truncate the Schmidt decomposition after just $\mathcal{O}(D)$ terms. That leaves us with an error of $n\epsilon$ in the ground state and k_ϵ across each cut.

The argument goes: you start with your state, which you don't know anything about. You repeatedly pull off one of your particles (schmidt decomposing, keeping only the top k components).

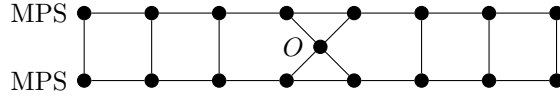


So let:

$$|\phi\rangle = \sum_i \lambda_i |v_i\rangle \otimes |w_i\rangle$$

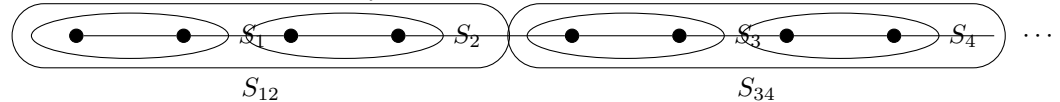
We can truncate by projecting to the left ($\text{span}\{|v_1\rangle \dots |v_n\rangle\}$) or to the right ($\text{span}\{|w_1\rangle \dots |w_n\rangle\}$). We're worried about our schmidt decompositions conflicting with each other, but if we do it to the left one way and to the right the other the errors just add.

We can efficiently compute a local observable O (the bubble width is small):



1.8.1 The Algorithm

As we go along, we're keeping track of some $S \subset \mathcal{H}_A$, where $|\Omega\rangle \in_\epsilon S \otimes \mathcal{H}_B$, but the dimension of S is polynomial in n . The algorithm tracks a basis for S , with each basis vector described by an MPS.



First, we combine neighboring particles into one subspace. We then recurse, combining blobs into bigger subspaces. By itself, this wouldn't help much: we'd still get an exponential space. We could randomly shrink our subspace, but that would increase the error massively.

Instead, we apply an AGSP. The whole point of an AGSP is to get a very favorable tradeoff between D and Δ . For example, we can pick $D^4 \Delta < 1/2$. This gives us a pump between dimension and error: first we trade off dimension for error 1-1, then we shrink the error. If we can:

- Construct the AGSP efficiently
- Bound the bond Schmidt rank of the AGSp of all cuts to $\text{poly}(n)$

Then we're pretty happy.