Simulating Quantum Computers with Tensor Networks

Erik Recio

Supervised by: Antonio Acín
Co-supervised by: Korbinian Kottmann and Paolo Stornati

Institute of Photonic Sciences, 08860 Castelldefels, Barcelona

September 2022

Erik Recio: erik.recio@icfo.eu

We are setting up a pipeline to simulate variational quantum algorithms for relevant system sizes. Most simulators only allow for a few qubits. With tensor networks we can access much larger systems. One of the most promising algorithms is the variational quantum eigensolver, where a parametrized circuit is optimized to prepare an approximation of the ground state. We are investigating the reverse process of disentangling a ground state to a product state variationally, so we can study the necessary circuit depth needed in order to achieve the disentangling.

Keywords: Quantum Algorithms, Tensor Networks, Variational Quantum Eigensolver

Acknowledgements

Thanks to Antonio Acín and ICFO for providing the environment that has allowed me to investigate this work. I would also like to thank Korbinian Kottmann and Paolo Stornati for co-supervising this thesis and helping me all along the way.

Contents

1 Introduction

Most quantum algorithms are concerned with preparing a specific state starting from the 0-state. This process is unitary, and thus, invertible. Here we want to investigate this reverse process, the decoding, starting from a specific state $|\psi\rangle$ and trying to get to $|0\rangle^n$. Studying the nuances of decoding states can give us clues on how to better create them in a more efficient and precise manner. In this work we simulate variational quantum algorithms with tensor networks with the objective of decoding certain states and study the circuits that make it possible. We will also attempt to increase the number of sites of our quantum systems thanks to the use of tensor networks, which gives us a linear scaling on the number of coefficients that need to be set in order to define the state (instead of an exponential scaling with the exact wave function).

2 Tensors

A tensor is the generalization of scalars (that have no indices), vectors (that have exactly one index), and matrices (that have exactly two indices) to an arbitrary number of indices (figure 1) [Sto21]. A tensor with r indices is called a rank-r tensor. Each index has its dimension (d_i). In order to obtain the total number of components of a tensor, we multiply the dimensions of each index. The total number of components of a tensor would then be $\prod_{i=1}^{r} d_i$. If the dimension is the same for every index, then the total number of components of the tensor is d^r [BB17].

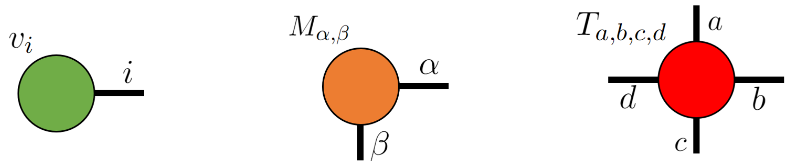

Figure 1: Diagrammatic representation of different multidimentional tensors.

3 Matrix Product States

A generic wave function of a quantum many-body system requires an exponentially large memory to be stored. For a spin chain of spin 1/2 and length n, we would need to store $O(2^n)$ complex numbers on the computer's memory. This exponential cost makes direct calculations of quantum mechanical systems unfeasible. Matrix Product States attempt to improve this scalability assuming a small error [Ba2].

We define the wave function of a many-body quantum state as:

$$|\psi\rangle = \sum_{s_1,\dots,s_n} c_{s_1,\dots,s_n} |s_1\rangle \otimes \dots \otimes |s_n\rangle$$

These coefficients $c_{s_1,\dots,s_n}$ can be interpreted as a tensor, and therefore can be redefined as the contraction of tensors A^{s_i} [Sch11]:

$$c_{s_1,\dots,s_n} = \sum_{\alpha,\beta,\dots,\gamma} A^{s_1}_{\alpha,\beta} A^{s_2}_{\beta,\delta} \dots A^{s_n}_{\gamma,\alpha}$$

The indices $\alpha, \beta, ..., \gamma$ can be contracted to recover the tensor $c_{s_1,...,s_n}$. This is an exact relation (figure 2). We have simplified the rank-n tensor c with n rank-3 tensor A. These tensors A can be interpreted now as lists of matrices (the 2 greek indices make up the matrix and the spin s_i makes the list), making it possible to apply Singular Value Decomposition and reduce the dimension of the tensors A as we will see later on this chapter.

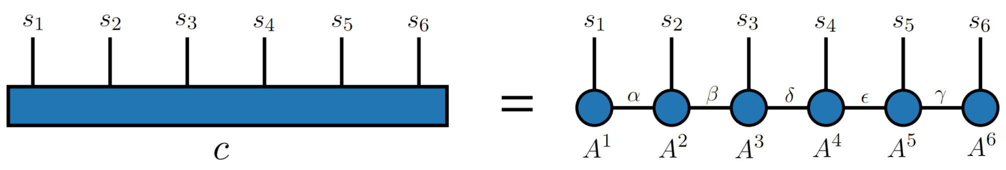

Figure 2: Schematic form of the MPS.

We can now compare the number of parameters in the quantum mechanical representation of the wave function and the matrix product state ansätz. We consider a many-body quantum system. We denote the local Hilbert space dimension (e.g., the spin dimension if we have a spin chain) d. In the wave function's quantum mechanical representation, the number of lattice points is n such that there is a total of $O(d^n)$ parameters. For the matrix product state we set the maximal dimension of the greek indices to be at most D (usually called bond dimension in the literature). Each tensor A has 3 indices: s_i of dimension d and 2 greek indices of dimension D. Therefore, since there are n of these tensors, the number of parameters in the matrix product state is $O(ndD^2)$, which increases linearly on the number of sites n instead of exponentially.

The exponential scaling in the matrix product state is hidden in the bond dimension D. We need to have an exponentially large D to have an exact equivalence. The MPS is just an approximation that can be useful in some cases. We can truncate this bond dimension by fixing a smaller maximal value of D. This truncation is at the core of the matrix product state and it can be done using the singular value decomposition (SVD) of the tensors A. The singular value decomposition decomposes a matrix as the product of three different matrices and highlights the singular values of the matrix. Given a generic rectangular matrix M, it is always possible to find the matrices U, Λ, V such that:

$$M = U\Lambda V^\dagger$$

U is a matrix containing the left eigenvectors of M and therefore has orthonormal columns ($U^\dagger U = \mathbb{1}$). Λ is a diagonal matrix with non-negative entries. The diagonal elements of Λ are the singular values of M. $V^\dagger$ is a matrix that contains the right eigenvectors of M and therefore has orthonormal rows ($V^\dagger V = \mathbb{1}$). As mentioned earlier, a tensor of a matrix product state $A^{s_i}_{\alpha,\beta}$ can be thought of as a list of matrices. We can perform the SVD for each of these rectangular matrices and keep only the largest singular values. Moreover, we can remove the right and left eigenvectors of the small singular values of the matrix and reshape them back to the previous form. The bond dimension of the new tensor A^{s_i} will be reduced, but the biggest contribution to the wave function given by the biggest singular values will be kept. In this manner we can achieve a good approximation of the wave function with a linear increase on the number of sites.

4 Variational Quantum Eigensolver

The VQE (or Variational Quantum Eigensolver) is a hybrid quantum-classical algorithm used to find an approximation of the minimum eigenvalue of a given Hamiltonian. As can be seen on figure 3, a quantum processor prepares a state according to a set of variational parameters (unitary $V(\vec{\theta})$). Then, using measurement outputs of the prepared state

$$|\psi(\vec{\theta})\rangle = V(\vec{\theta})\,|0\rangle$$

we calculate its expected energy with the given Hamiltonian:

$$\langle 0|V(\vec{\theta})^{\dagger} H V(\vec{\theta})|0\rangle = \langle \psi(\vec{\theta})|H|\psi(\vec{\theta})\rangle$$

The variational parameters are then optimized by a classical computer and fed back to the quantum machine in a closed loop. This optimization is stopped when either a convergence criterion or a maximum number of iterations defined by the user is reached.

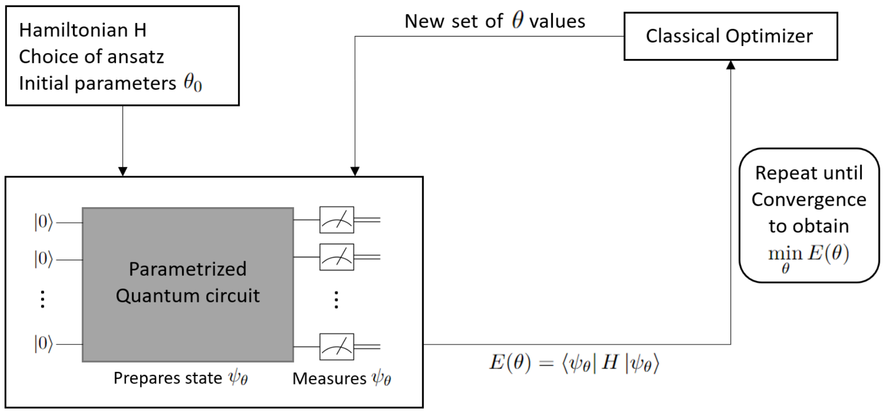
Hamiltonian H
Choice of ansatz
Initial parameters θ_0
New set of θ values
Classical Optimizer
Repeat until Convergence to obtain $\min_{\theta} E(\theta)$
$|0\rangle$
$|0\rangle$
$|0\rangle$
Parametrized Quantum circuit
Prepares state ψ_θ
Measures ψ_θ
$E(\theta) = \langle\psi_\theta|\,H\,|\psi_\theta\rangle$

Figure 3: Schematic form of the VQE algorithm.

5 Magnetization and Entanglement measures

Magnetization is the density of magnetic dipole moments (spins in our case) that are induced in a magnetic material when it is placed in a magnetic field. We will measure it with the following quantity:

$$M = \frac{1}{n}\sum_{i}^{n} \langle\psi|\,Z_i\,|\psi\rangle \tag{1}$$

Quantum entanglement is the physical phenomenon that occurs when a group of particles interact in a way such that the global state cannot be described by the state of its individual particles alone (it is more than the sum of its parts). We will get a measure of the entanglement of our systems by using the Von Newmann entropy (see equation 2). We will divide our spin chain in half and calculate the entropy of one of the subsystems (since it's symmetric, it will be the same quantity for both of them) [ECP10].

$$S = -\operatorname{Tr}(\rho \log \rho) \tag{2}$$

6 Explanation of the problem

In short, the problem is about putting the ground state of a given Hamiltonian into a quantum circuit that will decode it and end with the state $|0\rangle^{\otimes n}$. The quantum circuit will be composed of a number of equal layers with variational parameters. These parameters will change from run to run until we reach a good enough result on the decoding. Notice we are using the VQE algorithm, but reversed. We start from the ground state and we decode it to an all-$|0\rangle$ state.

The states that we will be decoding will be ground states of the transverse field Ising model Hamiltonian. Since we are looking at one-dimensional systems, this Hamiltonian can be written as the following:

$$H_{\text{Ising}} = -\sum_i J_i Z_i Z_{i+1} - \mu \sum_i h_i X_i \tag{3}$$

We will study the case where the interaction between sites is the same regardless of the site. Similarly with the magnetic field, its strength will be constant through the whole chain. Redefining the Hamiltonian H and the constant h in order to simplify the expression, we obtain the following:

$$H_{\text{Ising}} = -\sum_i Z_i Z_{i+1} - h \sum_i X_i \tag{4}$$

In order to solve this expression and find the ground state, we use the density-matrix renormalization group algorithm (DMRG). Once we have the ground state, we are ready to put it through our pipeline of layers, just like in VQE.

The circuit used is divided into k layers of the same type. Every layer will have the same gates, but their variational parameters will be different. Every layer is a unitary (U_j) that is applied to the state that we are decoding. The type of layer used in our simulations can be seen in Figure 4. A rotational gate R_y for every qubit and a CNOT gate for each neighbouring qubit.

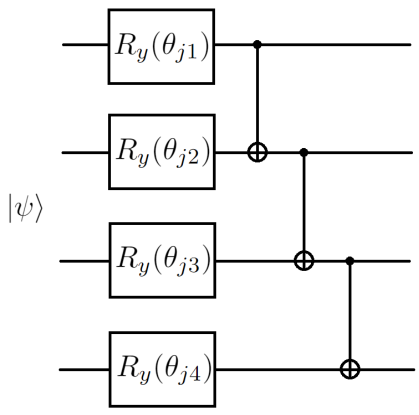

Figure 4: Schematic form of the j-th layer.

Stacking k of these layers together creates our variational circuit, as we can see in Figure 5. m is the number of qubits that will be decoded into the state $|0\rangle$ or, alternatively, the number of trash qubits.

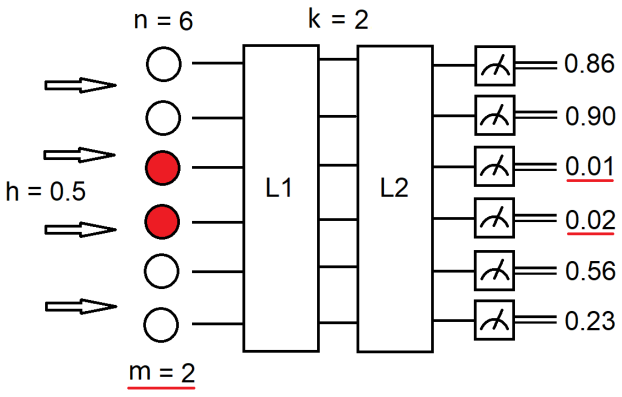

Figure 5: Schematic form of our quantum circuit.

The Hamiltonian that we are using in our reversed VQE will be the one in eq. 5. This is what VQE will try to find the minimum eigenvalue of.

$$H_{\mathrm{VQE}}(m) = \frac{1}{2m}\sum_{i}^{m}(\mathbb{1} - Z_i) \tag{5}$$

Therefore, the energy (or alternatively loss function L) will be the expression we find in equation 6. Since the result of this function is positive, and equal to 1 for an all-1 state and equal to 0 for an all-0 state, its minimization makes sure that all of the m qubits go to the state $|0\rangle$.

$$E(\vec{\theta}) = L = \langle\psi(\vec{\theta})|H_{\mathrm{VQE}}(m)|\psi(\vec{\theta})\rangle = \frac{1}{2m}\sum_{i}^{m}(1 - \langle\psi(\vec{\theta})|\, Z_i\, |\psi(\vec{\theta})\rangle) \tag{6}$$

After calculating L, the information is passed to a classical optimization algorithm (Nelder Mead in this case) and we obtain a new set of parameters, from wich we will start the process again. The first set of parameters $\vec{\theta_0}$ are random values.

7 Results

This work is the result of a classical simulation with code written in Julia (Appendix A). Julia is a programming language similar to Python but with a process of compilation before execution. This makes the language very suitable for research because it is highly optimized during the compilation and a speed advantage is gained in the long run.

Before starting with the VQE, we can analyse the ground states resulting from the DMRG algorithm (figure 6). For the magnetization (the order parameter of the phase transition), we see that it's not happening at the same value of h for every number of sites n. The phase transition starts happening at low h for small n and increases until it reaches $h = 1$ the more qubits we add (which is the known value for the thermodynamic limit on the Ising model). Regarding the entropy, we see that it increases at the change of phase. For every value of n, the entropy increases until the phase transition finishes.

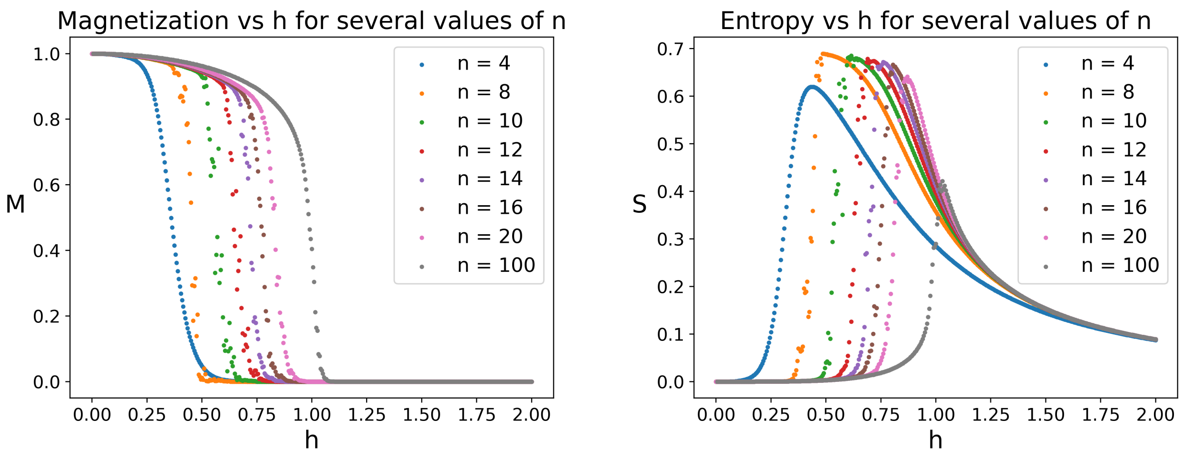

Figure 6: Magnetization and Entropy of the ground state for several cases of total sites. DMRG calculation has been done with 5 sweeps and 10 as the maximum bond dimension, for all values of n.

One more thing to notice is that for some values of n, the dots at the phase transition are not smooth. This is due to the DMRG algorithm not achieving the true ground state, just a close approximation. As stated before, when in transition, entropy grows a lot so more parameters are needed to define the wave function exactly. The MPS approximation of the algorithm turned out to be not too exact, although enough for our purpose.

We start the simulations with a small number of sites, increasing it step by step until the computing time is too large to continue. We test some values of m (trash qubits) for each of the following cases (n being the total number of sites, h the Ising transversal field and k the number of layers):

$$n = 4, 8, 15, 21$$

$$0 \leq h \leq 2$$

$$k = 1, 2, 3, 4$$

7.1 $n = 4$ sites

As mentioned before (figure 6), the change of phase occurs around $h = 0.3$ instead of $h = 1$ due to the boundary conditions and the low number of qubits simulated. Looking at the Von Neumann entropy, we see that its peak matches with the phase transition. The ground state is more entangled as the system becomes critical at the transition.

The first results are in the Figure 7. For $m = 1$ (1 trash qubit), we can see the minimum loss L obtained for each value of k layers. Some of the expected results are present:

- For low k, the minimum loss follows closely the entropy of the system. The circuit can not decode entirely the ground state, having a small but clear limit when minimizing the function. This limit follows the entropy, meaning that the more entangled a system is, the longer the circuit has to be in order to be able to disentangle the state and the more error we will have for short circuits.
- The more layers we add, the more accurate the system becomes. Every layer we add reduces the value of L, getting us closer to the all-$|0\rangle$ state.
- For 3 and 4 layers we can also see that there is no curve limiting the decoding. We have just a flat line (with local minima). Although the error for 2 layers is very small, from 3 layers on is when the value of the loss function is negligible and a very good decoding is achieved.

We can see some unexpected results as well:

- There is a small noisy bump in the loss function for $0.10 < h < 0.25$ and $k = 2, 3, 4$. We suspect this to be due to the structure of the layer used. Since we have a lot of CNOTs, our layers entangle the system very fast and the job of the rotational gates is to counter that, since for low values of h the ground state is almost $|0\rangle$ at every site.
- For 1 layer the loss function doesn't exactly follow the entropy and has a pointy maximum around $h = 0.7$ instead of 0.5 like the entropy in figure 6.
- There are many floating points that don't seem to belong to a specific curve. This is due to the local minima not allowing the optimization to find the absolute minimum.

Another way of looking at the plots is with the one on figure 8. We have the same 4 figures from before plotted together. Here we gain perspective on the value of the loss function. The plot for 2 layers becomes a small bump, meanwhile the local minima from 3 and 4 layers disappears. We will use both types of plots depending on what suits the most each case.

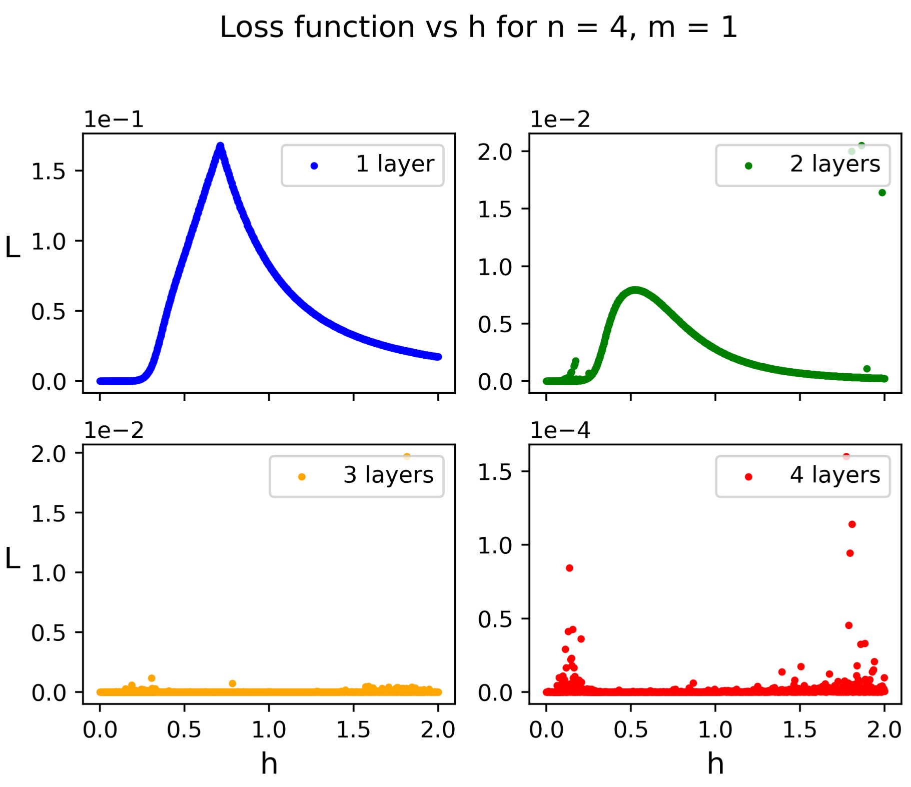

Figure 7: Minimum value achieved by the loss function for each value of h for $m = 1$ (trash qubits) and different values of k (number of layers).

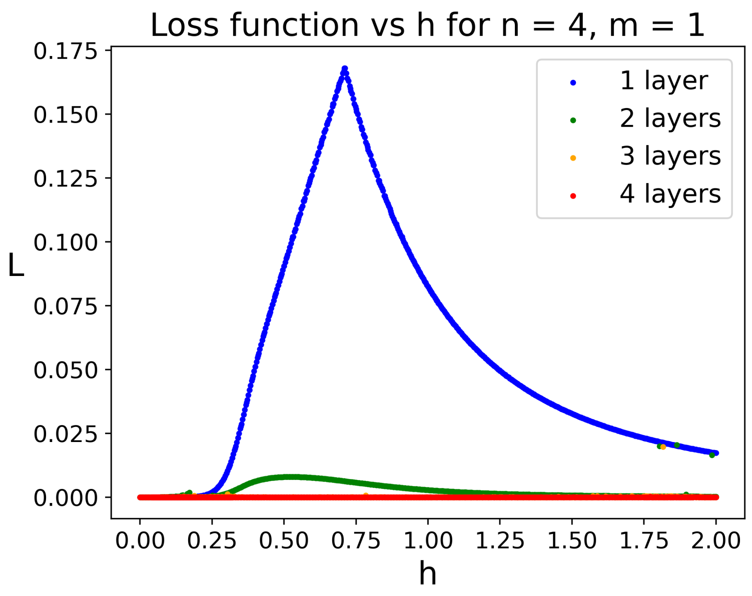

Figure 8: Minimum value achieved by the loss function for each value of h for $m = 1$ (trash qubits) and different values of k (number of layers).

Following the same style, we can see all the plots for $n = 4$ together on figure 9. We can see that all the figures are very similar. The peaks have all the same approximate values and the curves look more or less the same. For 1 and 4 trash qubits the curve is more pointy and for 1 layer we have the exception, as mentioned before, that its peak it's not exactly on the phase transition, although is very close.

Another thing to notice on figure 9 is that increasing the number of trash qubits doesn't affect too much on the quality of the decoding. The values on the loss function are almost the same. What changes is the appearance of local minima, which is non-existent for $k = 1$.

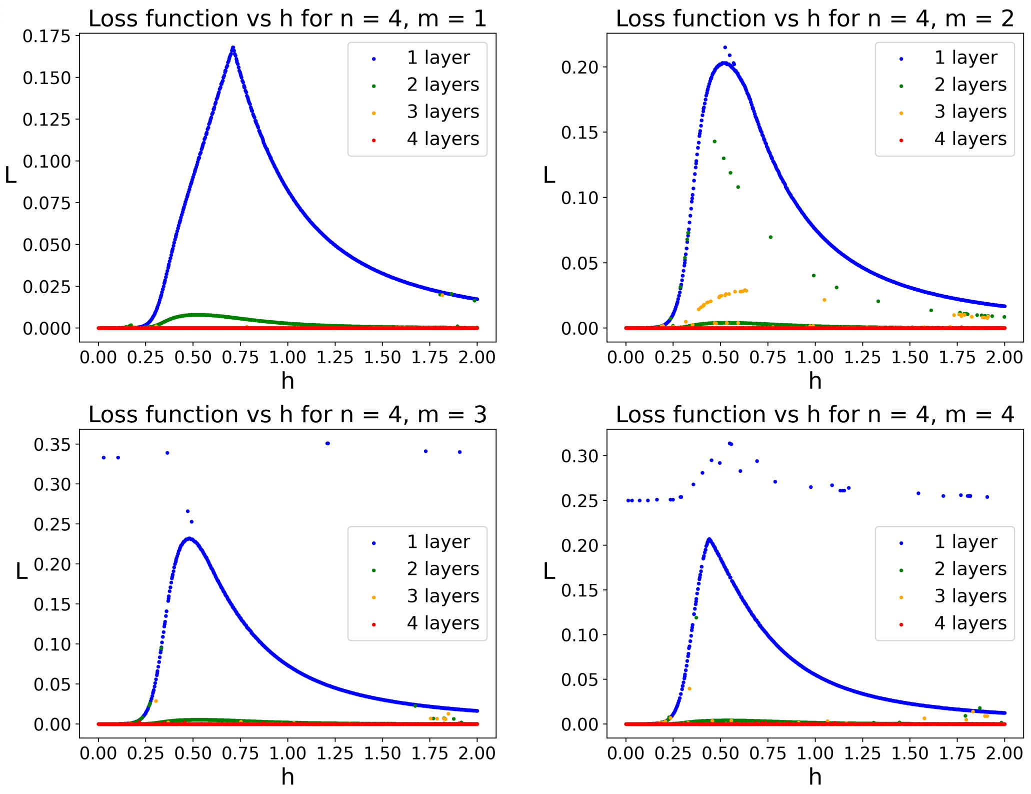

Figure 9: Minimum value achieved by the loss function for several m values (trash qubits). In each plot there are four curves, one for every number of layers considered.

7.2 $n = 8$ sites

For 8 qubits we can start to appreciate that the phase transition is moving to the right (figure 6) at around $h = 0.5$. The entropy continues to follow the phase transition as expected. On figure 10, we can see the same results as we saw for 4 qubits, but now the convergence into local minima is getting very relevant, bigger the more trash qubits we try to decode. The values for the global minimum of the loss function also remain the same for any value of m, as seen previously for 4 qubits.

A difference to notice with respect to 4 sites is that the values of L for 1 and 2 layers have increased. As expected, the more sites we try to decode, the harder it will be.

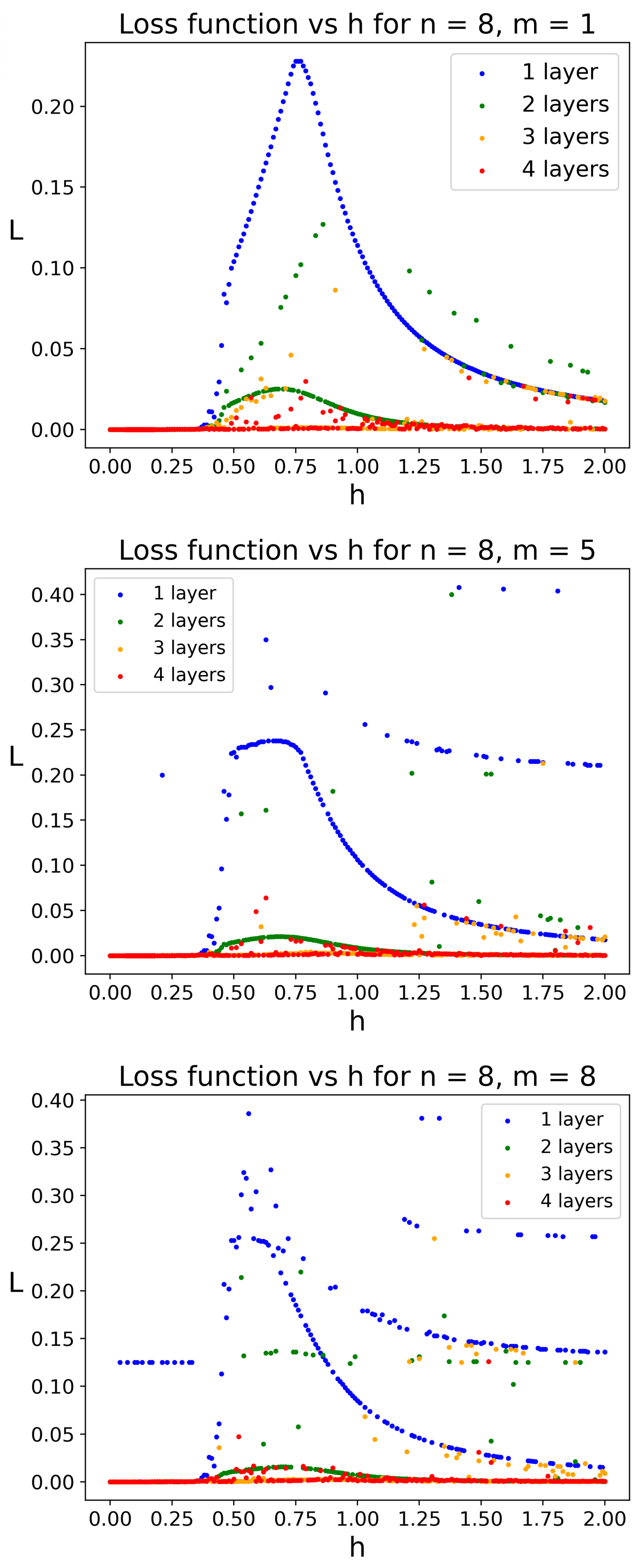

Figure 10: Minimum value achieved by the loss function for each value of h for different values of m (trash qubits) and k (number of layers).

7.3 $n = 15$ sites

For 15 qubits the phase transition is already at around $h = 0.75$ (figure 6). In figures 11, 12 and 13 we can see that local minima is making the plots not reliable. The scale of the 4 plots for every figure is the same, meaning that most of the time the global minimum is not being accomplished. Another observation worth mentioning is that, for 3 and 4 layers, the global minimum is quite low, meaning that most of the difficulty on decoding the states is not going to be solved by adding more layers. You could in principle obtain a very good decoding with just 3 layers (less than $L = 0.02$), but the local minima would give you a hard time.

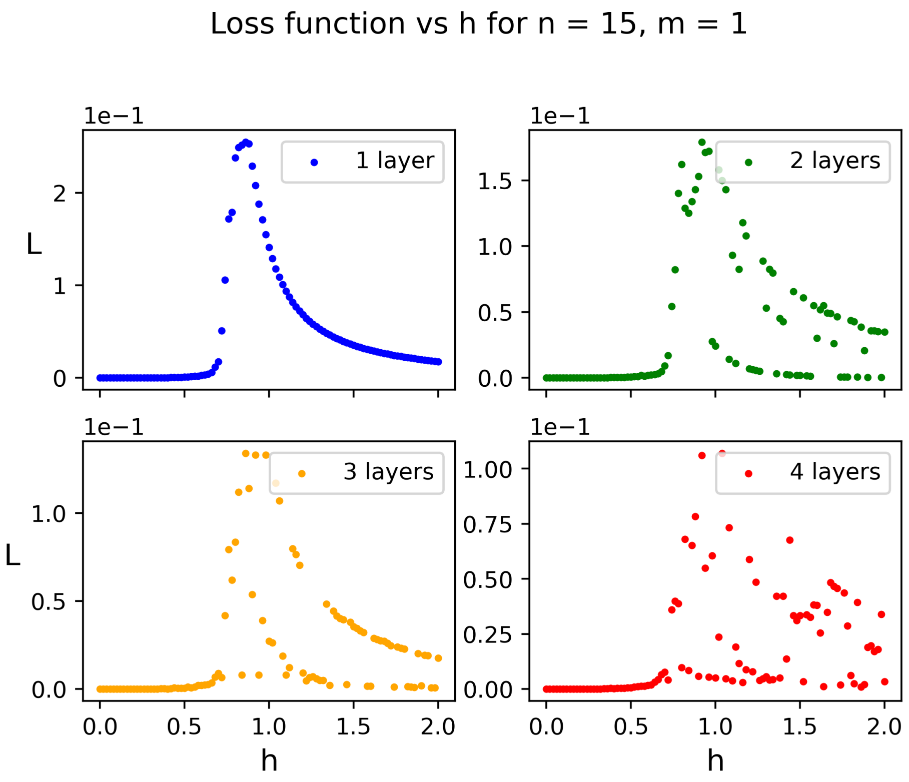

Figure 11: Minimum value achieved by the loss function for each value of h for $m = 1$ (trash qubits) and different values of k (number of layers).

7.4 $n = 21$ sites

For 21 sites the phase transition is almost at $h = 1$ and the entropy continues to follow it (figure 6). The problem here is that the long calculation times and the local minima makes the study unfeasible (see figure 14).

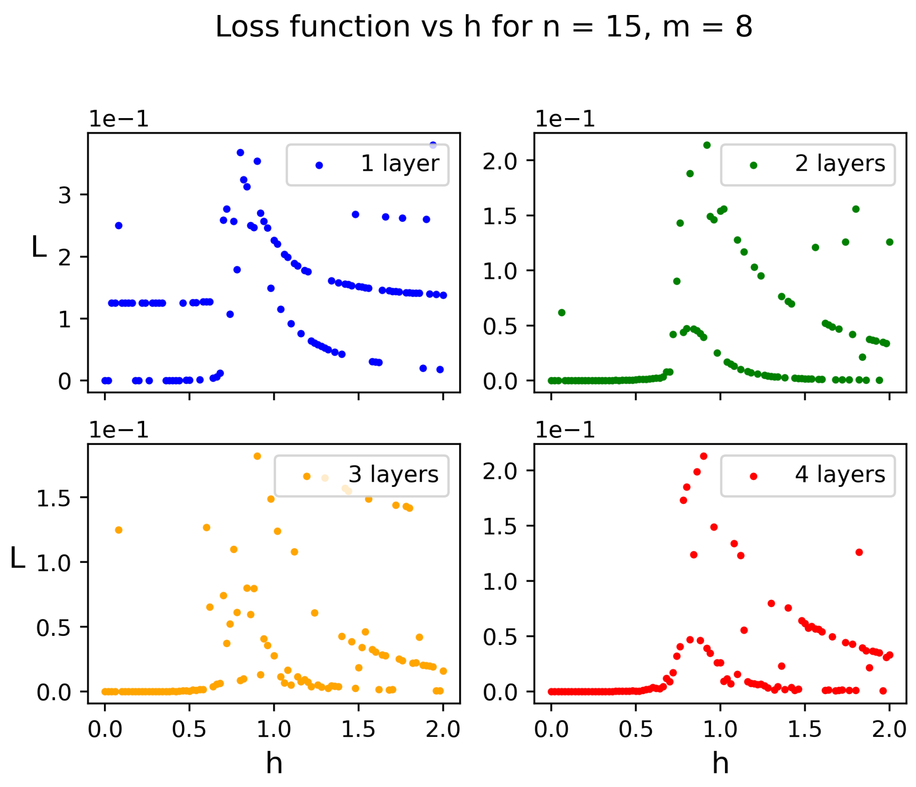

Figure 12: Minimum value achieved by the loss function for each value of h for $m = 8$ (trash qubits) and different values of k (number of layers).

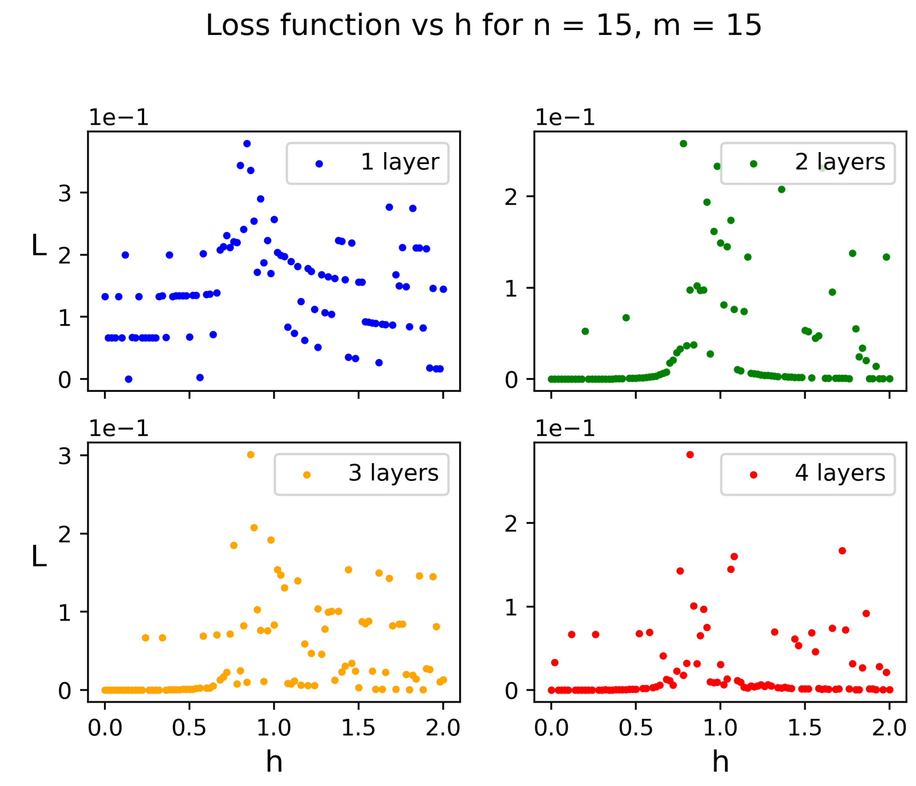

Figure 13: Minimum value achieved by the loss function for each value of h for $m = 15$ (trash qubits) and different values of k (number of layers).

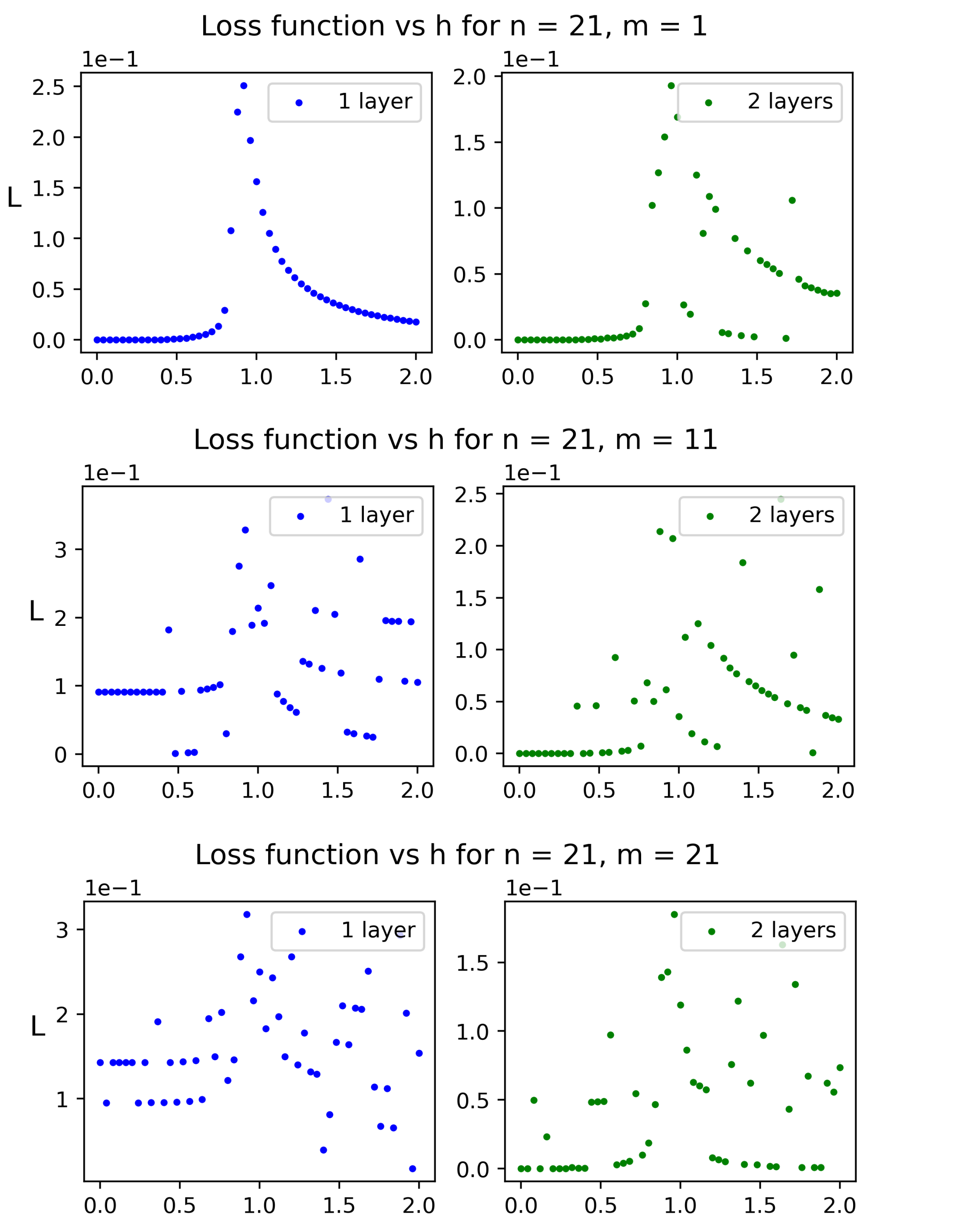

Figure 14: Minimum value achieved by the loss function for each value of h for different values of m (trash qubits) and k (number of layers).

7.5 Loss function vs number of sites

Although we can't make proper simulations beyond 20 qubits, we can calculate some points for more qubits. In figure 15 we can see the minimum Loss value versus the number of sites, up to 40 qubits. As expected, the larger the amount of qubits we want to decode, the worse the loss value will be (the plot has fixed parameters: $m = 1$ trash qubits, $k = 3$ layers, $h = 1.6$.). Despite that, the relationship is linear, which means that we can get a good enough decoding for a large number of qubits just with a small circuit (like the one on the figure 15, which has 3 layers).

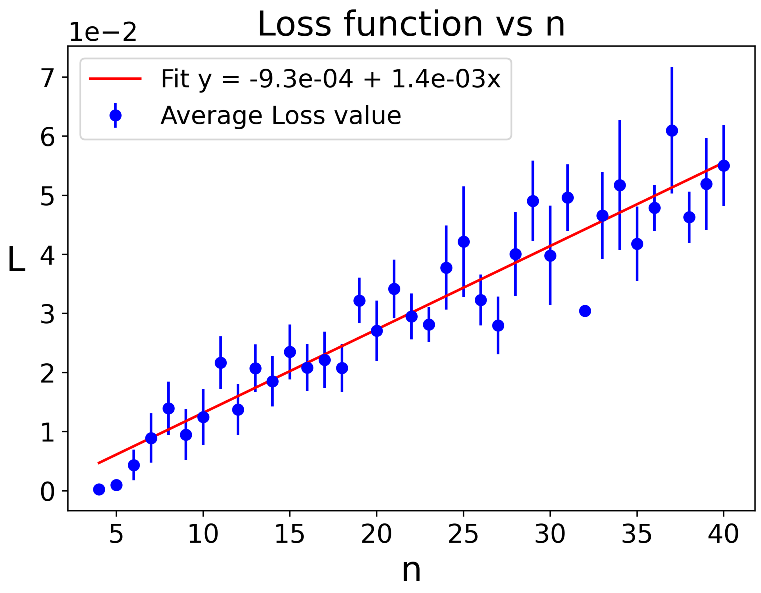

Figure 15: Minimum value achieved by the loss function for each value of n. Fixed parameters: $m = 1$ trash qubits, $k = 3$ layers, $h = 1.6$. for different values of m (trash qubits) and k (number of layers).

8 Execution times

One of the points of this thesis was to use tensor networks in order to work with lots of qubits. Our simulations couldn't get past 20 qubits, simply because of the time. On figure 16 we can see that the times and iterations needed to execute the Nelder Mead algorithm for each number of qubits follows an exponential trend instead of a linear one expected from an MPS. We can also see on figure 17 that the calculation of the loss function is not what is holding back the algorithm. The scaling with the number of sites is linear. This linearity relationship coupled with the exponential one from the iterations, makes clear that the problem with the times is on the workings of the Nelder Mead algorithm.

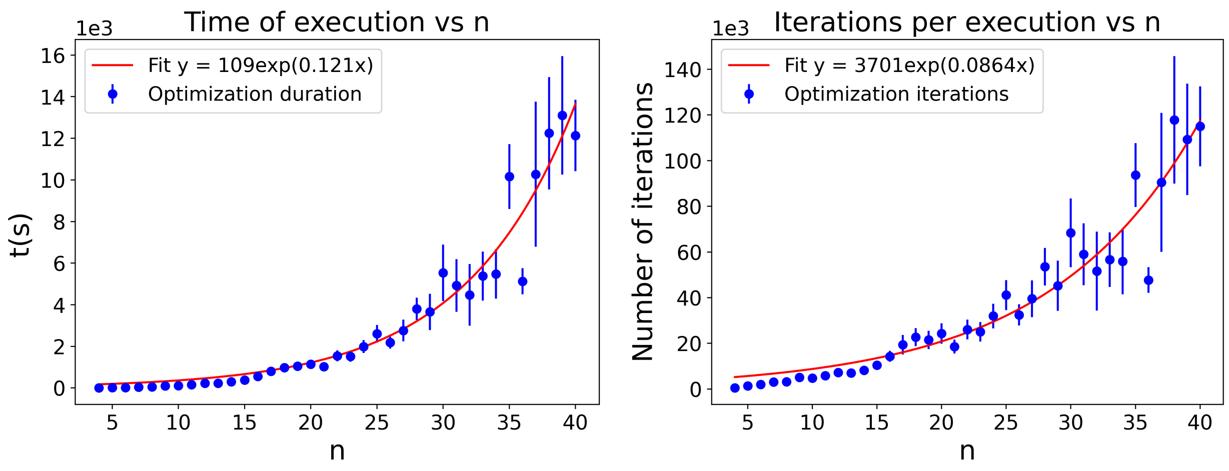

Figure 16: Single instances of a calculation for each number of qubits. This time is only of the Nelder Mead algorithm. Fixed parameters: $m = 1$, $k = 3$, $h = 1.6$.

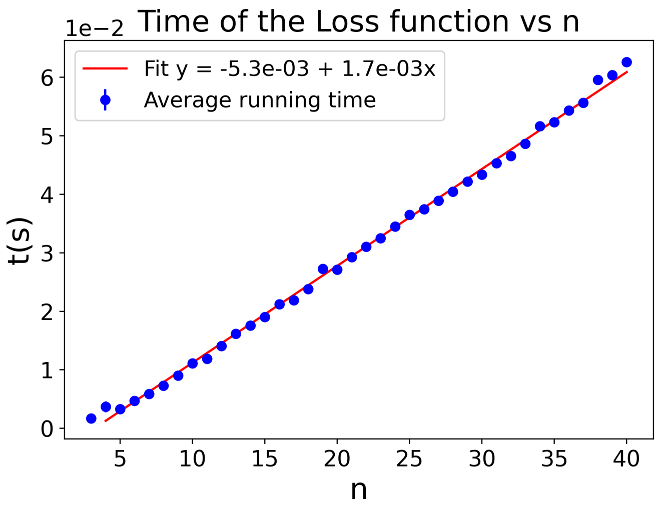

Figure 17: Single instances of a calculation for each number of qubits. This time is only of the Loss function. Fixed parameters: $m = 1$, $k = 3$, $h = 1.6$.

9 Next steps

The first thing worth putting time on is the efficiency of the algorithm. We would need to investigate with other algorithms like SPSA [Che21] or discover what is the problem with the current one [Meh20]. From here, it would be interesting to simulate large numbers of qubits and see the differences. It would also be interesting to try ground states of different Hamiltonians or even other entangled states not necessarily related to any Hamiltonians.

10 Conclusions

- The more entangled the state is, the longer the circuit needs to be in order to precisely decode the state.
- Nevertheless, increasing the number of layers will not directly result on a better decoding. As mentioned, local minima also increases and it will become harder to reach the global minimum. A lot of runs will be required in order to find the global minimum. It might also be impossible to have an accurate picture for complex systems and long circuits due to barren plateaus [MBS+18]
- Both n and k increase the local minima. Since the number of variational parameters on our quantum circuit is $n * k$, it is expected that increasing n or k will give us more local minima since the dimensionality of the search space also increases. [CSV+21].
- Interestingly, m also increases the local minima. The more trash qubits we try to decode, the easier it is to get stuck there. The value of the absolute minimum of the loss function L doesn't change as much with smaller or larger m, so in principle you would be able to obtain very good results with a big number of trash qubits. The problem comes when trying to get there. Since there are so many local minima, it would be harder to properly disentangle the state.
- As expected, n makes the loss function increase. The limit on how good a fixed number of layers perform is worse the larger the number of qubits we want to decode.

References

[Ba2] Mari Carmen Bañuls. Tensor network algorithms: a route map, 2022.

[BB17] Jacob Biamonte and Ville Bergholm. Tensor networks in a nutshell, 2017.

[Che21] Yiwen Chen. Theoretical study and comparison of spsa and rdsa algorithms, 2021.

[CSV$^+$21] M. Cerezo, Akira Sone, Tyler Volkoff, Lukasz Cincio, and Patrick J. Coles. Cost function dependent barren plateaus in shallow parametrized quantum circuits. *Nature Communications*, 12(1), mar 2021.

[ECP10] J. Eisert, M. Cramer, and M. B. Plenio. icolloquium/i: Area laws for the entanglement entropy. *Reviews of Modern Physics*, 82(1):277–306, feb 2010.

[MBS$^+$18] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 9(1), nov 2018.

[Meh20] V.K. Mehta. Improved nelder–mead algorithm in high dimensions with adaptive parameters based on chebyshev spacing points. *Engineering Optimization*, 52(10):1814–1828, 2020.

[Sch11] Ulrich Schollwöck. The density-matrix renormalization group in the age of matrix product states. *Annals of Physics*, 326(1):96–192, jan 2011.

[Sto21] Paolo Stornati. *Variational Quantum Simulations of Lattice Gauge Theories*. PhD thesis, Humboldt University of Berlin, 2021.

A Code

A.1 Main variables

- `nsites`: number of sites
- `h`: value of the ising model field
- `nqubits0`: number of trash qubits
- `nlayers`: number of layers of the quantum circuit
- `iter`: number of maximum iterations that the optimizer will make

- θ: array of the variational parameters of the quantum circuit
- ψ`0` : MPS of the ground state
- `qubit0_start`: position of the first trash qubit
- `qubit0_end`: position of the last trash qubit

A.2 Functions

`ising_hamiltonian(nsites; h)`
Defines the Hamiltonian for the ground state.

`layer(nsites, `θ`)`
Defines a layer of the quantum circuit

`variational_circuit(nsites, nlayers, `θ`)`
Defines the quantum circuit

`loss(`θ`, `ψ`0, nqubits0, nlayers, qubit0_start, qubit0_end)`
Defines the loss function L to be optimized

`ground_state(nsites, h)`
Calculates the ground state from the given hamiltonian

`optim_nelder(`ψ`0, nqubits0, nlayers, iter, qubit0_start, qubit0_end)`
Optimizes the loss function given a ground state

`main()`
Sets variables, creates files and calls the other functions

A.3 Pseudocode

```
main()
  loop for nsites, h, nqubits0, nlayers

    ground_state()
      ising_hamiltonian()

    optim_nelder()
      loss()
        variational_circuit()
          layer()
```

The main function starts looping through all the values set by the user. Then calculates the ground state (using the hamiltonian function) and starts with the optimization (optim nelder function). This function receives the ground state and calls the loss function, optimizing the parameters at every iteration until the value L is good enough to stop, or until the maximum number of iterations is reached.

The loss function calls the variational circuit function which calls the layer function and they build the quantum circuit. Then proceeds to apply this circuit to the ground state and lastly calculates the value L for the result before passing it to the optim nelder function.

A.4 Code of the program

```
using ITensors
using Random
using Optim

using BenchmarkTools
using Dates
using Printf

function ising_hamiltonian(nsites; h)
  ℋ = OpSum()
  for j in 1:(nsites - 1)
    ℋ += -1, "Z", j, "Z", j + 1
  end
  for j in 1:nsites
    ℋ -= h, "X", j
  end
  return ℋ
end

# A layer of the circuit we want to optimize
function layer(nsites, θ   )
  RY_layer = [("Ry", (n,), (θ=θ   [n],)) for n in 1:nsites]
  CX_layer = [("CX", (n, n + 1)) for n in 1:(nsites - 1)]
  return [RY_layer; CX_layer]
end

# The variational circuit we want to optimize
function variational_circuit(nsites, nlayers, θ   )
  range = 1:nsites
  circuit = layer(nsites, θ   [range])
  for n in 1:(nlayers - 1)
    circuit = [circuit; layer(nsites, θ   [range .+ n * nsites])]
  end
  return circuit
```

```julia
end

function loss(θ, ψ0, nqubits0, nlayers, qubit0_start, qubit0_end)
  nsites = length(ψ0)
  s = siteinds(ψ0)

  𝒰θ = variational_circuit(nsites, nlayers, θ)
  Uθ = ops(𝒰θ, s)
  global ψθ = apply(Uθ, ψ0; cutoff=1e-8)

  p1 = 0

  for j in qubit0_start:qubit0_end
    orthogonalize!(ψθ::MPS,j)
    Sz_j = op("Sz", s, j)
    ψθ_dag_j = dag(prime(ψθ[j]::ITensor, "Site"))
    p1 += 0.5 - scalar(ψθ_dag_j * Sz_j * ψθ[j]::ITensor)
  end

  return real.(p1)/nqubits0

end

function ground_state(nsites, h)

  #####################################
  # Calculate Ground State      ######
  #####################################

  s = siteinds("Qubit", nsites)
  ψ0 = MPS(s, "0")
  #ψ0 = randomMPS(ComplexF64, s; linkdims=bondim)

  ℋ = ising_hamiltonian(nsites; h=h)
  H = MPO(ℋ, s)

  sweeps = Sweeps(5)
  setmaxdim!(sweeps, 10)
  e_dmrg, ψ0 = dmrg(H, ψ0, sweeps)


  #####################################
  # Print initial wave function ######
  #####################################

  open(name_file_sumup, "a") do f
    write(f, "\np1_i = [")

    for j in 1:nsites

      orthogonalize!(ψ0,j)
      Sz_j = op("Sz", s, j)
      ψ0_dag_j = dag(prime(ψ0[j], "Site"))
      p = real.(round( 0.5 - scalar(ψ0_dag_j * Sz_j * ψ0[j]), digits = 3) )

      p_str = @sprintf "%4.3f" p

      if j != nsites
        write(f, "$p_str, ")
      else
        write(f, "$p_str")
      end

    end
    write(f, "]")
  end


  ################################################################
  # Calculate the entanglement entropy of the wave function ######
  ################################################################

  q_middle = trunc(Int, (nsites + 1)/2)

  orthogonalize!(ψ0, q_middle)
  U,S,V = svd(ψ0[q_middle], (linkind(ψ0, q_middle-1), siteind(ψ0, q_middle)))
  SvN = 0.0
  for n=1:dim(S, 1)
    p = S[n,n]^2
    SvN -= p * log(p)
  end


  #########################################################
  # Calculate the magnetization of the wave function ######
  #########################################################

  m = 0.0
  for j in 1:nsites
    orthogonalize!(ψ0,j)
    Z_j = op("Z", s, j)
    ψ0_dag_j = dag(prime(ψ0[j], "Site"))
    m += real.(round( scalar(ψ0_dag_j * Z_j * ψ0[j]), digits = 3) )
  end
  m /= nsites


  return ψ0, SvN, m

end

function optim_nelder(ψ0, nqubits0, nlayers, iter, qubit0_start, qubit0_end)

  ####################################
  # Optimization                ######
  ####################################

  nsites = length(ψ0)
  s = siteinds(ψ0)

  θ   ₀ = 2π * rand(nsites * nlayers)
  rest = Optim.optimize(θ    -> loss(θ   , ψ0, nqubits0, nlayers, qubit0_start,
       qubit0_end), θ   ₀, NelderMead(), Optim.Options(iterations = iter, g_tol
       = 8e-7))

  ####################################
  # Print final wave function   ######
  ####################################

  open(name_file_sumup, "a") do f
    write(f, "\np1_f = [")

    for j in 1:nsites
      orthogonalize!(ψθ   ::MPS,j)
      Sz_j = op("Sz", s, j)
      ψθ  _dag_j   = dag(prime(ψθ   [j]::ITensor, "Site"))
      p = real.(round( 0.5 - scalar(ψθ  _dag_j   * Sz_j * ψθ   [j]::ITensor),
           digits = 3) )

      p_str = @sprintf "%4.3f" p

      if j != nsites
        write(f, "$p_str, ")
      else
        write(f, "$p_str")
      end

    end
    write(f, "]")
    s_spaces_begin = "     , "^(qubit0_start - 1)

    if qubit0_end == nsites
      s_spaces_end = ""
      s_last = ""
      s_zeros = "0.000, "^(nqubits0 - 1) * "0.000"
    else
      s_spaces_end = "     , "^(nsites - qubit0_end - 1)
      s_last = "     "
      s_zeros = "0.000, "^(nqubits0)
    end


    write(f, "\nidea = [$s_spaces_begin$s_zeros$s_spaces_end$s_last]")
    write(f, "\ng_converged = $(rest.g_converged)")
    write(f, "\nmin_loss = $(rest.minimum)")
    write(f, "\niterations $(rest.iterations)/$iter")
  end

  open(name_file_plot, "a") do f
    write(f, "\n$(rest.g_converged) $(rest.minimum) $(rest.iterations)")
  end

  return nothing
end

function main()

  ####################################
  #   Parameters   ###################
  ####################################

  #Random.seed!(1234)

  #conf_        = [begin, end, runs,  step]
  conf_nsites   = [4,     4,    0,     1]
  conf_nqubits0 = [1,       1,    1,     0]
  conf_h        = [1.6,   1.6,    1,     0]
  conf_nlayers  = [3,       3,    1,     0]

  iter = 1000000000


  ####################################
  #   Code   #########################
  ####################################

  # Caculate Steps if missing ###

  if conf_nsites[4] == 0
    if conf_nsites[3] != 1
      conf_nsites[4] = (conf_nsites[2] - conf_nsites[1])/(conf_nsites[3] - 1)
    else
      conf_nsites[4] = conf_nsites[2]
    end
  end

  if conf_h[4] == 0
    if conf_h[3] != 1
      conf_h[4] = (conf_h[2] - conf_h[1])/(conf_h[3] - 1)
    else
      conf_h[4] = conf_h[2]
    end
  end

  if conf_nqubits0[4] == 0
    if conf_nqubits0[3] != 1
      conf_nqubits0[4] = (conf_nqubits0[2] - conf_nqubits0[1])/(conf_nqubits0[3]
           - 1)
    else
      conf_nqubits0[4] = conf_nqubits0[2]
    end
  end

  if conf_nlayers[4] == 0
    if conf_nlayers[3] != 1
      conf_nlayers[4] = (conf_nlayers[2] - conf_nlayers[1])/(conf_nlayers[3] - 1
          )
    else
      conf_nlayers[4] = conf_nlayers[2]
    end
  end


  # Caculate Runs if missing ###

  if conf_nsites[3] == 0
    conf_nsites[3] = trunc(Int, (conf_nsites[2] - conf_nsites[1] + 1)/
        conf_nsites[4] + 1)
  end

  if conf_h[3] == 0
    conf_h[3] = (conf_h[2] - conf_h[1])/conf_h[4] + 1
  end

  if conf_nqubits0[3] == 0
    conf_nqubits0[3] = trunc(Int, (conf_nqubits0[2] - conf_nqubits0[1])/
        conf_nqubits0[4] + 1)
  end

  if conf_nlayers[3] == 0
    conf_nlayers[3] = trunc(Int, (conf_nlayers[2] - conf_nlayers[1])/
        conf_nlayers[4] + 1)
  end

  # Choose the directory to save the files
  dir_pc = "C:/Users/Nosgraph/Documents/GitHub/TFM/Results raw/"
  dir_lap = "/home/user/Documents/TFM/TFM/Results raw/"

  dir = dir_lap

  if ispath(dir_pc)
    dir = dir_pc
  else
    dir = dir_lap
  end

  # Name the sumup file
  time_now = Dates.format(now(), "yy.mm.dd e, HH.MM.SS")
  global name_file_sumup = dir * time_now *  " - 0. Sumup.txt"


  # Name the plot file with the changes
  name_changes = "Time"

  if conf_nsites[3] != 1
    name_changes *= " vs nsites"
  else
    name_changes *= " vs nsites = $(conf_nsites[1])"
  end
  if conf_nqubits0[3] != 1
    name_changes *= " vs nqubits0"
  else
    name_changes *= " vs nqubits0 = $(conf_nqubits0[1])"
  end
  if conf_h[3] != 1
    name_changes *= " vs h"
  else
    name_changes *= " vs h = $(conf_h[1])"
  end
```

```
if conf_nlayers[3] != 1
  name_changes *= " vs nlayers"
else
  name_changes *= " vs nlayers = $(conf_nlayers[1])"
end

global name_file_plot = dir * time_now * " - " * name_changes * ".txt"

# Write the headers in the files
open(name_file_sumup, "a") do f
  write(f, "nsites = $conf_nsites\nnqubits0 = $conf_nqubits0\nh = $conf_h\
       nnlayers = $conf_nlayers\n\nchanges = $name_changes\niter = $iter\
       nmethod = $method\n\n")
end

open(name_file_plot, "a") do f
  write(f, "g_converged min_loss iter nsites nqubits0 h nlayers SvN time m")
end

#initialize some variables
ψ0 = 0
SvN = 0
m = 0
nsites_2 = 0
nqubits0_2 = 0
h_2 = 999
qubit0_start = 0
qubit0_end = 0

# Start loop for the calculations
for nsites in range(conf_nsites[1], conf_nsites[2], step = conf_nsites[4])
  max_nqubits0 = min(nsites, conf_nqubits0[2])
  for h in range(conf_h[1], conf_h[2], step = conf_h[4])
    for nqubits0 in range(conf_nqubits0[1], max_nqubits0, step = conf_nqubits0
         [4])
      for nlayers in range(conf_nlayers[1], conf_nlayers[2], step =
           conf_nlayers[4])

        nsites = trunc(Int, nsites)
        nqubits0 = trunc(Int, nqubits0)
        nlayers = trunc(Int, nlayers)

        # Save the qubits in the middle of the state (the ones we will turn to
             0)
        if nsites_2 != nsites || nqubits0_2 != nqubits0
          qubit0_start = trunc(Int, (nsites-nqubits0)/2 ) + 1
          qubit0_end = qubit0_start + nqubits0 - 1
        end

        # Calculate the ground state everytime there is a change
        if h_2 != h || nsites_2 != nsites
          ψ0, SvN, m = ground_state(nsites, h)
        end

        nsites_2 = nsites
        nqubits0_2 = nqubits0
        h_2 = h

        # Execute minimizer algorithm, save the time
        time = @elapsed optim_nelder(ψ0, nqubits0, nlayers, iter, qubit0_start
             , qubit0_end)


        # Write the last data on the files
        open(name_file_sumup, "a") do f
          write(f, "\nh = $h")
          write(f, "\nnlayers = $nlayers")
          write(f, "\nSvN = $SvN")
          write(f, "\nm = $m")
          write(f, "\ntime = $time s\n")
          end

          open(name_file_plot, "a") do f
            write(f, " $nsites $nqubits0 $h $nlayers $SvN $time $m")
          end
        end
      end
    end
  end

  return nothing

end


for j in range(1, 1, step=1)
  main()
end
```