Numerical methods for tensor networks study of spin systems and gauge theories

Raghav G. Jha[a]

Perimeter Institute for Theoretical Physics, Waterloo, Ontario N2L 2Y5, Canada

E-mail: rjha1@perimeterinstitute.ca

Abstract: In these lecture notes, we discuss numerical details of coarse-graining tensor algorithms for studying classical statistical systems and describe tensor formulation of two-dimensional pure gauge theory based using Wilson's lattice action for SU(2) gauge group. `Python` codes are also provided to allow for hands-on experience. This is very much work in progress. If you have comments, please email them to me.

Last edited: June 15, 2020

Contents

Contents

1 Introduction

The main problem of dealing with quantum many-body systems is the size of the Hilbert space. There is no hope to completely address the entire space even if we enter the NISQ era or even beyond it. Therefore, an understanding of the 'states that matters' becomes very important. In the last decade or so, it has been found that the ground states of local gapped Hamiltonian with short-range interactions are astonisghgly concentrated in a very tiny region of the Hilbert space which we hereafter call 'area-law' states. If one focusses just on this region and forget about the complete Hilbert space, one can still uncover the ground-state properties to reasonable accuracy. One of the guiding lights to identify and isolate 'area-law' states has been the notion of entanglement entropy. It has been shown that these special states follow a different scaling of the entropy than naively expected (volume-scaling) from a random state in the Hilbert space.

This problem is not so easy to deal for critical systems or systems where you have long-range interactions (say the Hamiltonian is next-to-next neighbour and even more non-local) but it seems that the fruit does not fall far from 'area-law' states even in those cases. So, one still has the advantage of not dealing with exponentially large Hilbert space $\mathcal{H}$.

```
from ncon import ncon
import numpy as np
```

Last edited: June 15, 2020

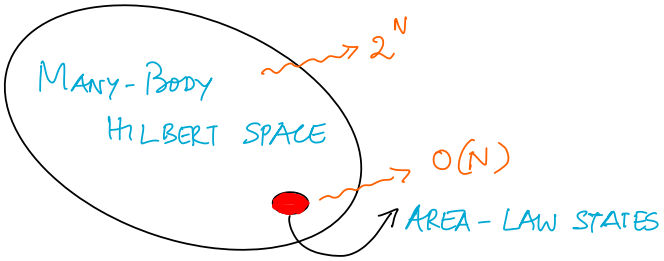

Figure 1. XX

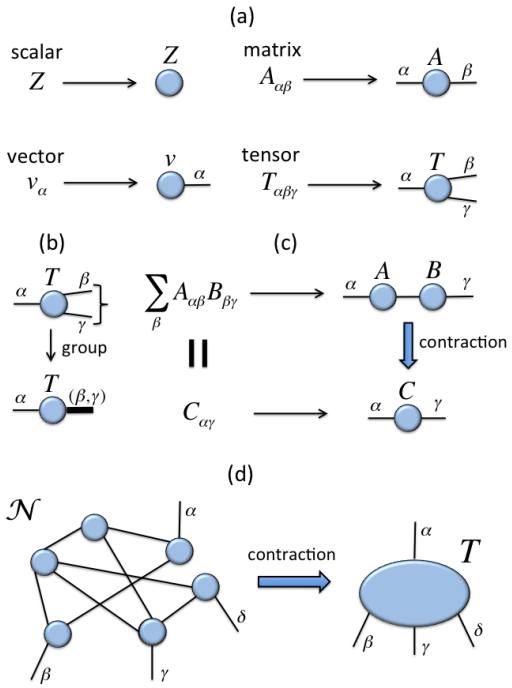

Figure 2. XX

```
"einsum"(Einstein's summation convention) is a very useful NumPy tool for
contracting tensors.  For example, matrix multiplication using this is -
np.einsum('ij,jk->ik', A, B) which is just C_ik = A_ij B_jk.  Dot product is
- np.einsum('i,i->', A, B). See this.
```

But "einsum" is slow. In these lecture notes, we will use NCON except Exercise 1 where we will use "einsum" once. Please download/copy NCON from here.

For example, $C_{ip} = A_{ijk}B_{pjk}$ can be implemented in Python using NCON as:

```
C = ncon((A, B),([-1,2,1], [-2,2,1]))
```

- Same positive integers are contracted indices while the order [-1, ···] stands for the ordering of the indices in the resulting tensor.

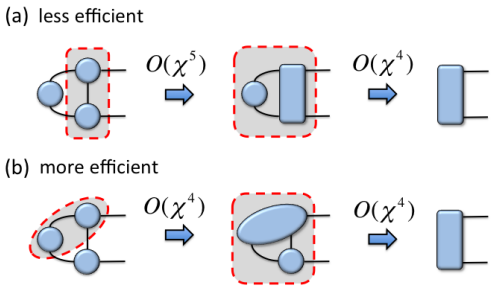

Figure 3. The figures are taken from [1]

Exercise 1: Consider the Hamiltonian of three spins ($N = 3$) quantum Ising model given by:

$$H = \sigma_1^x \otimes \sigma_2^x + \sigma_2^x \otimes \sigma_3^x + \sigma_3^x \otimes \sigma_1^x + h\left(\sigma_1^z + \sigma_2^z + \sigma_3^z\right)$$

Since $\dim(\mathcal{H}) = 8$, use exact diagonalization and compute ground state energy for various h.

Exercise 1B: Construct the quantum Ising Hamiltonian for N spins and check that it reproduces the result from previous exercise when $N = 3$. Now use exact diagonalization and compute ground state energy for same values of h with $N = 7$ or $N = 8$. Please do not try to run with $N > 10$.

Exercise 2: Calculate the trace of product of four random 3×3 matrices using einsum and check that result the result agrees with that obtained from np.trace and np.dot. You can construct random matrices using: `A = np.random.rand(3,3)`

Exercise 3: Compute the rank-four tensor A_{rqba} which is equal to $B_{ijkl}C_{jiqr}D_{lkab}$ using NCON where all indices run from $1 \cdots 3$. Draw a tensor diagram of this contraction. You can choose the tensors to be random like before.

1.1 Singular Value Decomposition (SVD)

The SVD of an $m \times n$ real or complex matrix $\mathbf{M}$ is a factorization of the form $U\Sigma V^\dagger$, where U is $m \times m$ real or complex unitary matrix, Σ is $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and V is an $n \times n$ real or complex unitary matrix. The diagonal entries Σ_{ii} of Σ are known as the singular values of $\mathbf{M}$. The number of non-zero singular values is equal to the rank of $\mathbf{M}$.

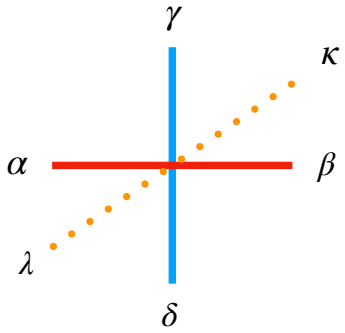

Figure 4. Diagrammatic representation of a rank-six tensor, $T_{\alpha\beta\gamma\delta\kappa\lambda}$ which can serve as a fundamental tensor of some 3d classical statistical system.

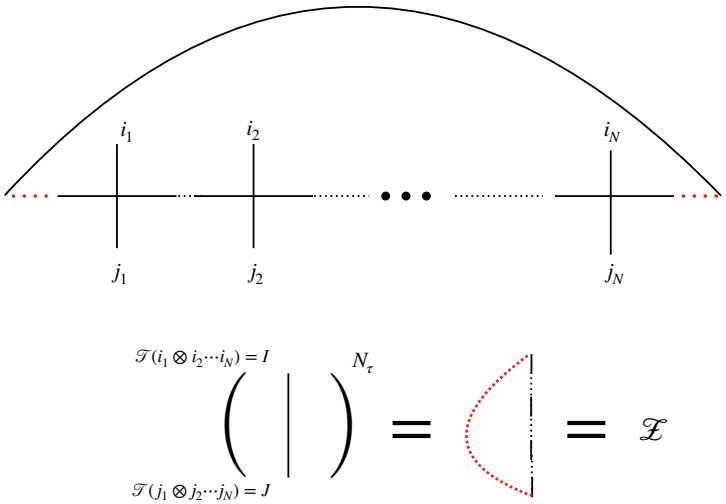

Figure 5. The schematic diagram which shows how we can coarse-grain the tensor along one direction and construct the transfer matrix and partition function.

Mathematical applications of the SVD include computing the pseudoinverse, matrix approximations, and determining the rank, range, and null space of a matrix. The SVD is also extremely useful in all areas of science, engineering, and statistics, such as signal processing, least squares fitting of data, and process control.

SVD is an integral step in all tensor network coarse-graining algorithm. It helps in reducing the ever-growing size of the fundamental tensor such that meaningful computations can be carried on classical computers. There are other alternatives which have been explored - such as randomized SVD. We will only use SVD in these lectures.

2 Example I – Exact diagonalization

Here we will discuss the method of exactly diagonalizing the quantum hamiltonian for 3 spins and plot the ground state energy as a function of magnetic field. here

3 Example II – 2d Ising model

The exact result obtained is [2]

Let us define $\kappa = \frac{\sinh(2\beta)}{2\cosh^2(2\beta)}$ and then the free energy density is given by: $f = -\frac{1}{\beta}\left(\log(2\cosh(2\beta)) - \kappa^2\, {}_4F_3\left(1,1,1.5,1.5;2,2,2;16\kappa^2\right)\right)$

The critical temperature is given by:

$T_c = \frac{2}{\log(1+\sqrt{2})} = 2.26918531421$ i.e. $\beta_c \approx 0.44069$

which is obtained by solving

$\sinh(2\beta_c)^2 = 1$.

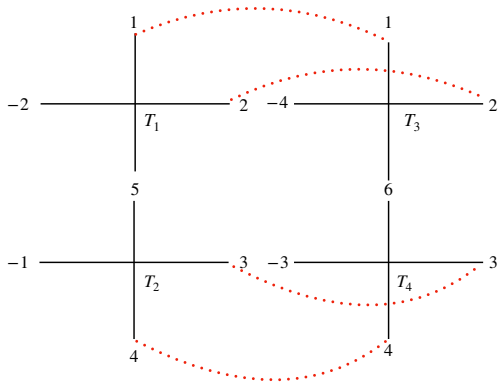

Figure 6. One step of coarse-graining along a specific direction. The `ncon` equivalent of this diagram is `ncon([T1,T2,T3,T4],[[-2,1,2,5],[-1,5,3,4],[-4,1,2,6],[-3,6,3,4]])`

In this example, we will reproduce Fig.(3) of arXiv version here[3].

The fundamental tensor T for the 2d classical Ising model can be written as:
$T_{ijkl} = W_{pi}W_{pj}W_{pk}W_{pl}$ `where` W `is given by:`
`W = np.array([[`$\sqrt{\cosh\beta}$`,`$\sqrt{\sinh\beta}$`],[`$\sqrt{\cosh\beta}$`,-`$\sqrt{\cosh\beta}$`]])`

The code is pasted below. You can refer to it if you are stuck. You will also need to use a simple numerical differentiation code to compute $\frac{-\partial \ln Z}{\partial\beta}$ from log of partition function. You can see it below:

2d ISING CODE

```
import sys
import math
from math import sqrt
import numpy as np
from scipy import special
from numpy import linalg as LA
```

```
import scipy as sp
import time
import datetime
from ncon import ncon
from matplotlib import pyplot as plt
# For T=2.0, f_2d_Ising = -1.02579
# -dlnZ/d beta = -1.7455677143228514
# beta_c= 0.4407 ~ sinh(2\beta_{c})^2 = 1
# With J=1

D=12
D_cut=12
Niters=20
Ns = int(2**((Niters)))
Nt = Ns
vol = Ns**2
numlevels = Niters
norm_all = [0 for x in range(numlevels+1)]

startTime = time.time()
print ("STARTED: " , datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

def tensorsvd(input,left,right,D):
    '''Reshape an input tensor into a rectangular matrix with first index
        corresponding
    to left set of indices and second index corresponding to right set of
        indices. Do SVD
    and then reshape U and V to tensors [left,D] x [D,right]
    '''
    T = input
    left_index_list = []
    for i in range(len(left)):
        left_index_list.append(T.shape[i])
    xsize = np.prod(left_index_list)
    right_index_list = []
    for i in range(len(left),len(left)+len(right)):
        right_index_list.append(T.shape[i])
    ysize = np.prod(right_index_list)
    T = np.reshape(T,(xsize,ysize))


    U, s, V = np.linalg.svd(T,full_matrices = False)

    if D < len(s): # Truncate if length exceeds input D
        s = np.diag(s[:D])
        U = U[:,:D]
        V = V[:D,:]
```

```
    else:
        D = len(s)
        s = np.diag(s)

    U = np.reshape(U,left_index_list+[D])
    V = np.reshape(V,[D]+right_index_list)
    return U, s, V

def Z2d_Ising(beta):

    a = np.sqrt(np.cosh(beta))
    b = np.sqrt(np.sinh(beta))
    W = np.array([[a,b],[a,-b]])
    out = np.einsum("ia, ib, ic, id -> abcd", W, W, W, W)
    return out

def Z2d_XY(beta, h):

    betah = beta*h
    for i in range (-Dn,Dn+1):
        L[i+Dn] = np.sqrt(sp.special.iv(i, beta))

    out = ncon((L, L, L, L),([-1],[-2],[-3],[-4]))
    # Alt: T = np.einsum("i,j,k,l->ijkl", L, L, L, L)
    for l in range (-Dn,Dn+1):
        for r in range (-Dn,Dn+1):
            for u in range (-Dn,Dn+1):
                for d in range (-Dn,Dn+1):
                    index = l+u-r-d
                    out[l+Dn][r+Dn][u+Dn][d+Dn] *= sp.special.iv(index, betah)

    return out

def CG_step(matrix, in2):

    T = matrix
    TI = in2
    AAdag = ncon([T,T,T,T],[[-2,1,2,5],[-1,5,3,4],[-4,1,2,6],[-3,6,3,4]])
    U, s, V = tensorsvd(AAdag,[0,1],[2,3],D_cut)
    A = ncon([U,T,T,U],[[1,2,-1],[2,-2,4,3],[1,3,5,-4],[5,4,-3]])
    AAAAdag = ncon([A,A,A,A],[[1,-1,2,3],[2,-2,4,5],[1,-3,6,3],[6,-4,4,5]])
    U, s, V = tensorsvd(AAAAdag,[0,1],[2,3],D_cut)
    AA = ncon([U,A,A,U],[[1,2,-2],[-1,1,3,4],[3,2,-3,5],[4,5,-4]])
    maxAA = np.max(AA)
    AA = AA/maxAA # Normalize by largest element of the tensor
    return AA, maxAA
```

```
if __name__ == "__main__":

    beta = np.arange(0.42, 0.46, 0.0005).tolist()
    Nsteps = int(np.shape(beta)[0])
    f = np.zeros(Nsteps)

    for p in range (0, Nsteps):

      T = Z2d_Ising(beta[p])
      #T = Z2d_XY(beta[p], 0.0)
      norm = LA.norm(T)
      T /= norm
      Tim = T
      Z = ncon([T,T,T,T],[[7,5,3,1],[3,6,7,2],[8,1,4,5],[4,2,8,6]])
      C = 0
      N = 1
      C = np.log(norm)
      Free = -(1.0/beta[p])*(np.log(Z)+4*C)/(4*N)

      for i in range (Niters):

          T, norm = CG_step(T, Tim)
          C = np.log(norm)+4*C
          N *= 4.0
          Free = -(1.0/beta[p])*(np.log(Z)+4*C)/(4*N)
          if i == Niters-1:

            Z1 = ncon([T,T],[[1,-1,2,-2],[2,-3,1,-4]])
            Z = ncon([Z1,Z1],[[1,2,3,4],[2,1,4,3]])
            Free = -(np.log(Z)+4*C)/(4*N)
            f[p] = Free

    dx = beta[1]-beta[0] # Asssuming equal spacing
    dfdx = np.gradient(f, dx)
    d2fdx2 = np.gradient(dfdx, dx)
    plt.plot(beta, f, marker="*", color = "r")
    plt.title('Free energy of classical 2d Ising model', fontsize=20)
    plt.xlabel('Inverse Temperature, $\beta$', fontsize=16)
    plt.ylabel('f', fontsize=16)
    plt.show()
    # Now plot internal energy 'E'
    plt.plot(beta, dfdx, marker="+", color = "b")
    plt.title('Internal energy of classical 2d Ising model', fontsize=20)
    plt.xlabel('Inverse Temperature, $\beta$', fontsize=16)
    plt.ylabel('E', fontsize=16)
    plt.show()
```

```
    plt.plot(beta, -d2fdx2, marker="*", color = "g")
    plt.title('Specific heat of classical 2d Ising model', fontsize=20)
    plt.xlabel('Inverse Temperature, beta', fontsize=16)
    plt.ylabel('SH', fontsize=16)
    plt.show()
    print ("Specific heat peaks at $\beta$ = ",
        beta[int(np.array(-d2fdx2).argmax())])
    print ("COMPLETED: " , datetime.datetime.now().strftime("%Y-%m-%d
        %H:%M:%S"))
```

4 Example II – 2d classical XY model

In this exercise, we will construct tensor formulation of classical XY model in two dimensions with $h = 0$ and calculate the free energy to reproduce the plot given in 7 from [4].

We start with the fundamental tensor (four legs) which sits on each lattice site and tiles the lattice.

$$T_{i,j,k,l} = \sqrt{I_l(\beta)I_r(\beta)I_u(\beta)I_d(\beta)}I_{i+k-j-l}(\beta h), \tag{4.1}$$

where indices (i, j, k, l) denote the four legs of the tensor. The length of each leg, called bond dimension, is infinite in principle from the character expansion formula. The coefficient $I_n(\beta)$ decreases exponentially with increasing n. Thus we can truncate the series and approximate $T_{i,j,k,l}$ by a tensor with finite bond dimension D with high precision. This leads to a finite-dimensional tensor representation for the partition function

$$Z = \mathrm{Tr}\prod_s T_{i_s,j_s,k_s,l_s}. \tag{4.2}$$

A bond links two local tensors. The two bond indices defined from the two end points are implicitly assumed to take the same values. For example, if the bond connecting i and j along the x direction, then $r_i = l_j$. The trace is to sum over all bond indices.

Magnetization is defined as,

$$m = \frac{1}{\beta}\frac{\partial \ln Z}{\partial h} = \frac{I_{l+u-r-d-1}(\beta h) + I_{l+u-r-d+1}(\beta h)}{2I_{l+u-r-d}(\beta h)} \tag{4.3}$$

5 Example III – Entanglement Entropy

Consider dividing a Hilbert space, $\mathcal{H}$ into a product of two spaces as, $\mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_B$ corresponding to sub-systems A and B. Then the subsystem A can be described by,

$$\rho_A = \mathrm{Tr}_B\ \rho = \sum_i \langle\psi_B^i|\rho|\psi_B^i\rangle \tag{5.1}$$

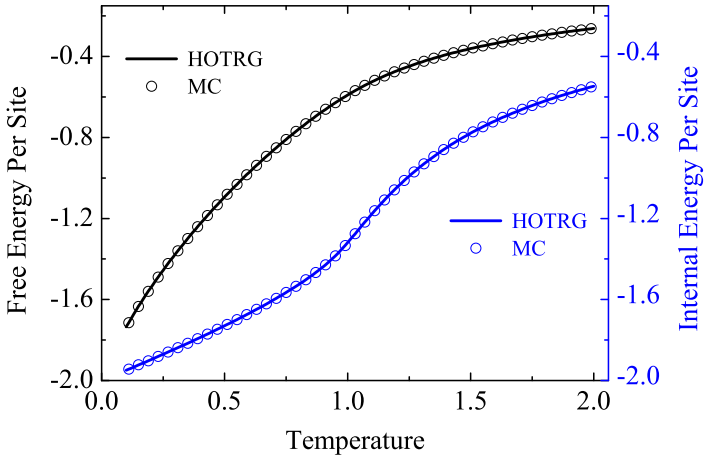

Figure 7. The free and internal energy of the 2d XY model.

where the Tr is only over the $\mathcal{H}_B$. Then the entanglement entropy (also von Neumann, sometimes also bipartite entanglement entropy) entropy is defined as,

$$S_A = -\mathrm{Tr}_A\Big(\rho_A \ln \rho_A\Big) \tag{5.2}$$

Some nice properties of S_A are mentioned below:

- $S_A(\rho_A)$ is maximum when the state is maximally entangled. In such a case, $S_A(\rho_A) = \ln(\dim\mathcal{H}_\mathcal{A})$
- If ρ_A is a pure state (*i.e.* $\rho^2 = \rho$), then $S_A = 0$
- S_A is constant under change of basis (unitary), *i.e.* $S_A(\rho_A) = S_A(U\rho_A U^\dagger)$

Consider a state like,

$$|\psi\rangle = \cos\theta|\downarrow\uparrow\rangle + \sin\theta|\uparrow\downarrow\rangle \tag{5.3}$$

Then we can write, $\rho_A = \mathrm{Tr}_B\ \rho$ as,

$$\rho_A = \cos^2\theta|\downarrow\rangle\langle\downarrow| + \sin^2\theta|\uparrow\rangle\langle\uparrow| \tag{5.4}$$

which gives entropy as,

$$S_A = -\cos^2\theta \mathrm{log}\cos^2\theta - \sin^2\theta \mathrm{log}\sin^2\theta \tag{5.5}$$

Note that at $\theta = \pi/4$, the entropy is maximum and the corresponding state is maximally entangled.

EE CODE

```
# This code generates a random state of "N" qubits and
# then computes the reduced density matrix of first "p" qubits.
# Then it calculates the entanglement entropy. Note that
# the entropy will be "p".

import sys
import math
from math import *
import numpy as np
from scipy import special
from numpy import linalg as LA
from numpy.linalg import matrix_power
from numpy import ndarray
import time

N=24
p=4   # Becomes expensive with increasing $p$. Don't try p>10

Psi = np.random.randn(2**N)
# 2^N random coefficients
Psi = Psi/LA.norm(Psi)

A = Psi.reshape(2**p, 2**(N-p))
Rho = np.dot(A, np.transpose(A).conj())

def comEE(Rho):
   u,v = LA.eig(Rho)
   chi = u.shape[0]
   #print ("Shape of u", np.shape(u)) # 2^p
   #print ("Shape of v", np.shape(v)) # 2^p x 2^p
   EE = 0
   for n in range (0 , chi):
      if u[n] > 0:
         EE += -u[n] * math.log(u[n],2)
   return EE

if __name__ == "__main__":

   entropy = comEE(Rho)
   print ("Entanglement Entropy is", entropy)

# S_exact = -rho log2(rho) = -1/d * ln (1/d) summed 'd' times i.e. log2(d) =
    log2(2^p) = p
```

Last edited: June 15, 2020

6 Homework problem!

It is also possible to formulate the tensor network for Wilson's action for SU(2) gauge group in two dimensions. This was done in [5]. Try to do this. You can also refer to my GitHub page to see the code if you want here.

Last edited: June 15, 2020

References

[1] R. Orus, "Exploring corner transfer matrices and corner tensors for the classical simulation of quantum lattice systems," *Phys. Rev.* **B85** (2012) 205117, arXiv:1112.4101 [cond-mat.str-el].

[2] G. M. Viswanathan, "The hypergeometric series for the partition function of the 2d ising model," *Journal of Statistical Mechanics: Theory and Experiment* **2015** no. 7, (Jul, 2015) P07004. http://dx.doi.org/10.1088/1742-5468/2015/07/P07004.

[3] H.-J. Liao, J.-G. Liu, L. Wang, and T. Xiang, "Differentiable programming tensor networks," *Phys. Rev. X* **9** (Sep, 2019) 031041. https://link.aps.org/doi/10.1103/PhysRevX.9.031041.

[4] J. F. Yu, Z. Y. Xie, Y. Meurice, Y. Liu, A. Denbleyker, H. Zou, M. P. Qin, and J. Chen, "Tensor Renormalization Group Study of Classical XY Model on the Square Lattice," *Phys. Rev.* **E89** no. 1, (2014) 013308, arXiv:1309.4963 [cond-mat.stat-mech].

[5] A. Bazavov, S. Catterall, R. G. Jha, and J. Unmuth-Yockey, "Tensor renormalization group study of the non-Abelian Higgs model in two dimensions," *Phys. Rev.* **D99** no. 11, (2019) 114507, arXiv:1901.11443 [hep-lat].

Last edited: June 15, 2020