Università degli Studi di Padova

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Corso di Laurea in Physics of Data

Information Theory and Computation Final project

Authors

Marco Agnolon, Tommaso Tabarelli

Abstract

Generative models that creates a probability distribution from the input data are becoming very important in Machine Learning (ML) since they data-driven and therefore completely autonomous; also their reconstructing power, intended as the capability to generate a distribution similar to that of the data, is an important property to take into account.

The main purpose of this project is to develop a Fortran algorithm for Unsupervised Learning (UL) using the innovative approach of Tensor Networks (TN). Its scope will be learning how to represent MNIST images using Matrix Product States (MPS). Inspired by the probabilistic interpretation of quantum mechanics, MPS TN creates a probabilistic representation of known configurations that are the training set images. From this representation it should be able to generate new configurations (images) from scratch and to reconstruct corrupted ones. The algorithm is fully unsupervised since no labels are used for distinguish the images in input.

The results will be compared with another classical fully unsupervised algorithm that creates a probability distribution in the encoded space: a denoising autoencoder implemented using Pytorch.

A.y. 2019-2020

THEORY

Introduction

Tensor networks are a class of tensor decomposition. They are used to rewrite high order tensor into a contracted network of low order ones. This allows to reduce the high dimensionality of data they represent and retrieving tensor components in a polynomial cost for manipulating.
They are based on the idea of *coarse graining* that comes from statistical mechanics and it is used to study very large system composed of many particles. This concept is very powerful since it expresses the possibility of conserving the large scale properties of a system marginalizing over its smallest scales. Then the procedure is repeated considering each time a new small scale and marginalizing over it. The procedure, step by step, exploits a new comprehension of the system from a larger point of view. This is called Renormalization Group, inside the quantum mechanics framework it could be conjugated in different ways, one of those is the Density Matrix Renormalization Group (DMRG).

This latter approach resembles the one we used in our implementation of the unsupervised algorithm.

In order to parametrize the system wave function, many tensor network structures are available. We decide to use the simplest one, that is the Matrix Product State (MPS). It is a one dimensional approximation of a high-order tensor. It consists of a one dimensional chain of three dimensional tensors that can be contracted in order to exploit their representational power. The lateral dimension regulate the links between different tensors, whereas the central dimension is the one that takes into account the information coming from the input data.

Compared to the classical generative approaches, MPS presents an explicit probability density that is clearly tractable (while standard algorithms usually exploit some aprroximations of it). Moreover we expect a polynomial scaling of computational resources when increasing the size of the dataset.

The task

The goal of an unsupervised algorithm is to build a probability distribution over the sample space that takes into account the most important features of the data in input. This probability distribution must be as close as possible to the probability distribution of the samples. Once the training has been performed, new samples can be generated using the learned data distribution.

First of all let us consider the data we decided to take into account. They come from the MNIST dataset which is composed of 70000 images of dimension 28x28 binary pixels; they represent handwritten digits from 0 to 9. We are going to use a smaller sub-sample (1000 images) of the entire dataset for computational resources issues.
Each of them is reshaped into a one dimensional vector of size 784 = 28x28. This vector contains only zeros or ones. The aim is to find the function $\Psi(\nu)$ representing the configuration of the given samples, where $\nu = (\nu_1, \nu_2, ..., \nu_{784}) \in \{0,1\}^{\otimes 784}$.
The probability of a given configuration is:

$$P(\nu) = \frac{|\Psi(\nu)|^2}{Z} \tag{1}$$

where $Z = \sum_{\nu \in V} |\Psi(\nu)|^2$, that is sort of partition function and $V = \{0,1\}^{\otimes 784}$.
Then we parametrize the wave function using MPS:

$$\Psi(\nu_1, \nu_2, ..., \nu_{784}) = Tr\left(A^{(1)\nu_1} A^{(2)\nu_2} \dots A^{(784)\nu_{784}}\right) \tag{2}$$

where each $A^{(k)\nu_k}$ is a b_{k-1} by b_k matrix and $b_0 = b_{784} = 1$. Thus, the MPS is composed of 784 three dimensional tensors of dimensions b_{k-1} x 2 x b_k, they become matrices if the index correspondending to the inserted pixel is fixed (in fact, it is of size 2, one for value 0 and another for value 1). Each tensor stands for a different pixel of the input images that is represented by the middle index of the tensors. The correlation between them are explained by the bond dimensions between the tensors.

We want the probability distribution learned to be as close as possible to the one of the data. In order to evaluate their similarities we use the Maximum Likelihood Estimator which defines a negative log-likelihood used as loss fuction:

$$L = -\frac{1}{T} \sum_{\nu \in T} ln(P(\nu)) \tag{3}$$

where T is the training set size.
The objective of the algorithm is to minimize this quantity with respect to the elements of the tensors. Therefore we compute its derivative:

$$\frac{\partial L}{\partial A^{(k,k+1) w_k w_{k+1}}_{i_{k-1} i_{k+1}}} = \frac{Z'}{Z} - \frac{2}{T} \sum_{\nu \in T} \frac{\Psi'(\nu)}{\Psi(\nu)} \tag{4}$$

where $Z' = 2 \sum_{\nu \in T} \Psi'(\nu) \Psi(\nu)$.

The procedure

The focus will be to minimize the loss updating properly the elements of the tensors. These elements are initialized randomly, then the MPS is left-canonicalized in order to have orthonormal matrices (exept for the rightmost one, whose elements are divided by the matrix norm). A "training step" consists

of merging in sequence (from right to left and back) consecutive couples of tensors into a 4-order tensor, called *merged tensor*.

Since we want to reduce the loss, we perform a gradient descent algorithm updating the merged tensor for each couple of consecutive tensors. At each iteration a fraction of the gradient is subtracted to the merged tensor fed with the corresponding values of input pixels.

Then the resulting updated merged tensor is transformed in a matrix of dimensions which corresponds to the non-contracted tensors dimensions multiplied by themselves (tipically, 2× bonds dimensions, the factor "2" representing the input dimension). This matrix is now reduced applying an SVD procedure which keeps the singular values that are greater than a certain fractions of the biggest one (this fraction is the *cutoff parameter*). The new retrieved matrices coming from the SVD are thresholded using the number of singular values kept. This two new matrices will reshaped to be the updated tensors that will replace the previous contracted ones.

In this way the bond between the new tensors may change its size in order to capture correlations between the corresponding pixels and increasing the likelihood. This size will match the number of singular values kept.

Once one couple of tensors is updated, the next one is considered and another two tensors are updated. A minimum and maximum dimension for the bonds are fixed initially, specially the maximum dimension is important to prevent overfitting since it is proved that a MPS is able to approximate each probability distribution if an infinite size for bonds is allowed.

This updating procedure performs several *gradient descent* steps for each considered couple of tensor. Their bonds are updated and also the values inside them. Then the SVD will keep only the most important features found in the GD procedure in order to discover the most important ones.

Generation

Once the training has been performed and we obtain all the tensors with proper bond dimensions and values, they can be easily used in the generating procedure, which is quite straightforward.

Two different tasks can be accomplished:

- Generating new images from scratch;
- Reconstructing corrupted image (half occluded in our case).

They both rely on the same generative procedure: the algorithm samples a new value for the next pixel giving the knowledge of the values of the already sampled ones. We need to compute the probability for a pixel to assume value 0 or 1; this probability depends on the values assumed by all the "previous" pixels. The generation (in our case) goes always from right to left, the only difference between the two approaches is that in the former no pixels are fixed at the beginning (it means that the image is initially blank), the first sampled is the 784th; on the other hand, in the latter the right half of the pixel are fixed (from the 393th to 784th) and the generation starts from the 392th given the knowledge of the fixed ones.

$$P(\nu_{k-1}|\nu_k,\nu_{k+1},...,\nu_{784}) = \frac{P(\nu_{k-1},\nu_k,\nu_{k+1},...,\nu_{784})}{P(\nu_k,\nu_{k+1},...,\nu_{784})} \tag{5}$$

Therefore, when generating, we compute the two probabilities in r.h.s. of Eq. 5 using Eq. 1, where the $\Psi(\nu)$ is computed multiplying all the tensor in the right side of the sampled one; finally we obtain the conditional probability of having value ν_{k-1} for the $(k-1)$th pixel. If we consider $\nu_{k-1} = 0$, thus when generating a random number, if it is less than the compute probability the pixel takes value 0 otherwise 1.

CODE DEVELOPMENT

ULTN

Description

We decide to consider a small subset of the entire dataset composed of only 1000 different samples. We import it with Python and convert it into a binary file made of 0s and 1s. Thus, it is a matrix 1000x784. This is the file that Fortran will read as input.

The Fortran program is made of three different files:

- "mod_utility.f90": contains all user defined types, functions and subroutines;
- "ULTN_train.f90": calls the subroutines for training;
- "ULTN_gener.f90": calls the subroutines for both generating from scratch and reconstruction.

Following the calls in "ULTN_train.f90":

- CHECK_BONDS: it is a check in order to perform correctly the left canonicalization of the tensor;
- LEFT_CANO: adjust all the tensors but the last one, in order to have them normalized, since at beginning they are filled with random numbers;
- INIT_CUMULANTS: this routine initializes the cumulants, their utility will be explain later on;

- DO cycle where repeating GRADIENT_DESCENT: performs the training repeating the gradient descent for the desired number of iteration.

As one may have noticed, the structure of the *Main* is very simple, the first three routines are used to initialize, prepare and check the tensors and the cumulants. The last routine, instead, contains all the procedures implemented in the algorithm.

So let us understand properly what GRADIENT_DESCENT routine does. It contains inside two main cycles, the first goes from 783 to 1 whereas the second goes from 1 to 783. They are symmetric because they perform the tensors update in both right-to-left and left-to-right ways, scrolling on each couple of tensors.

Inside each cycle the function calls another routine that merges the two considered tensor into a bigger one.

There a new loop begins. This loop perform a fixed number of GD steps (default 10) where the $\Psi(\nu)$ and the $\Psi'(\nu)$ are computed. This allows to compute also the gradient of the loss as represented in Eq. 4. The merged tensor is now updated using the gradient. The loop will return the updated version of the merged tensor that tries to minimize the loss.
Thus, it will be split calling the routine that performs the SVD and keep the proper number of variables. Therefore, at the end, each step of the two main loops inside the Gradient descent will return an updated version of the tensors composing the MPS chain.
Finally the cumulants update is performed in order to have them ready for the next step.

The trickiest part consist of find a easy way to compute the $\Psi(\nu)$ without tracing over each tensor as described in Eq. 2. This is performed using the cumulants. They are 784 matrices and they are initialized and updated using the following rules:

- $cumulant[1] = ones((1000,1))$
- $cumulant[784] = ones((1,1000))$
- $cumulant[j] = A(1)\cdots A(j-1)\ if\ 1 < j \leq k$ or $cumulant[j] = A(j+1)\cdots A(784)\ if\ k < j < 784$ where k is the bond we are merging in that specific step of the loop and A are the tensors.

Therefore, cumulants are fundamental since they keep trace of the multiplications between the tensor both on the left and on the right with respect to the merging ones. This allows, when computing the $\Psi(\nu)$ in a specific step, to use only the merged tensor and its adjacent cumulants, because they contains all the information coming from the multiplication between all the other tensors.

For what concerns the generating procedure, the *"ULTN_gener.f90"* file calls two different subroutines, one for generating from scratch and the other that tries to reconstruct an half corrupted image.
The former tries to generate an image relying on only the information contained in the tensors, so from the learned probability distribution. Let us focus on the second one: we give as input an image coming from a test set (different from the training one), we consider fixed the last 392 pixel and we start to generate the 392th. Then we compute the accuracy counting how many pixels of the original image the algorithm is able to guess.

Parameters

The training program needs some parameters as input, they are contained in a structure called *params*:

- Nx: number of training images, default = 1000;
- *space_size*: value representing the input size (28x28), default =784;
- *descend_steps*: number of steps for gradiend descent, default = 10;
- *init_bond_dim*: starting dimension of each bond, default =2;
- *mini_bond*: minimum bond dimension allowed, default =2;
- cut_off: threshold for SVD procedure, default $= 10^{-7}$;
- *maxi_bond*: maximum bond dimension allowed;
- *learning_rate*: gradient descent learning rate.

The default parameters are chosen to be balance between computation time and accuracy, because once the dimensions of the bonds increases also the time to compute them increases. The cutoff and the maxi bond parameters regulate the dimension of the bonds, whereas the other are limited in order to reduce the computation time.

The only two free parameters are the maxi bond dimension and the learning rate. We implement different choices for them since the former it is very useful to prevent overfitting: once it increases also the representational power of the MPS increases but this might lead to poor performances in reconstructing the test images. The latter parameter is the key one since it regulates the speed of learning and it could prevent falling into local minima. We choose to use a decreasing learning rate: at every GD step the initial value of 0.001 is multiplied by 0.9.

Denoising autoencoder

In order to check the results of the reconstruction performed over corrupted images we decide to implement a denoising autoencoder with Pytorch. This is a neural network composed of two different structures: encoder and decoder. The first take the input and encodes it in a lower dimensional space. Our previous experiments suggest that the best number of dimension for the encoded space is 10 since the MNIST contains 10 different digits. Once the encoder has performed this job, the decoder takes its results and transform it in a new image: it should be as similar as possible to the original one. We implement two losses between the images the Mean Square Error and the Binary Cross Entropy loss. The network parameters are trained in order to minimize it. In order to maximize its performances we train it using 60000 samples.

Therefore we have two different networks ("int.ipynb" and "real.ipynb") that take as input a binary pixels image and outputs different images: the first outputs an image with real values for pixel (that will be transformed into binary ones with a threshold in 0.5), the second a binary image.

The denoising autoencoder differs from what we have just described due to the fact that it takes as input also corrupted images, it encodes and decodes them and finally it compares the resulting image with the not-corrupted version of the input. This leads to better generalization property of the network and allows it to learn how to reconstruct corrupted images.

The performances of the denoisers are evaluated against 2000 corrupted test samples with an accuracy measure.

RESULTS

We trained the MPS with different parameters:

Attempt	maxibond	GD steps	time (min)
First	20	71	66
Second	100	50	1261
Third	400	8	1497

Table 1: Table shows number of epochs and time duration of different training processes.

We increase the allowed maximum dimension of tensor bonds, noticing that the computation time increased exponentially with them; as a consequence the possible number of gradient descent iteration that could be done in the same amount of time drastically decreased.

Fig. 1 shows that the more the bond dimensions increases the faster the loss decreases during the training iterations. Furthermore, Fig. 2 shows that the bond dimensions tend to become larger in the area of the images that is usually occupied by the digits, thus in the centre; indeed, the relations between different images pixels are more complex to learn in that area if compared with the image boundary areas; here the pixels usually assume *0* value for every image and thus the relation patterns are easier to learn.

Using the same argument, it is interesting to notice that the three cases shown in Fig. 2 present different behaviours. In the first case (max bond dimension = 20) almost every bond saturates to 20, while allowing a larger maximum bond dimension makes the saturated bonds decrease. This can be related to the fact that the incresing complexity of the patterns to learn is better captured by larger bond dimensions; hence when these ones are smaller than the proper dimension to represent the local complexity, the tensor bonds are more likely to saturate to the maximum values allowed.

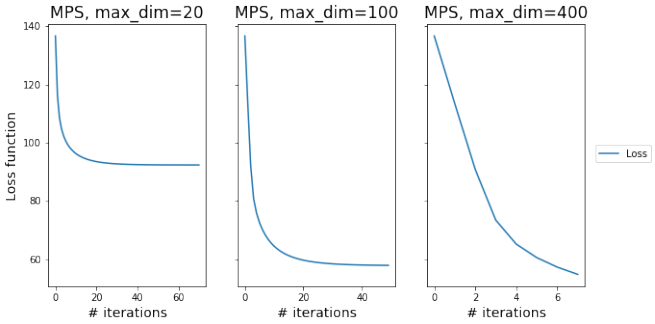


Figure 1: Figure shows the loss w.r.t. the nuber of iteration for different training cases.

The *generation from scratch* procedure creates images that are not recognizable as shown in Fig. 3. The difficulty in generating the images *ex novo* can be due to the fact that the complexity of the patterns to learn (in terms of correlations between pixels that can be very distant in the grid) would require a much larger local bond dimension w.r.t. the maximum one chosen for training. Indeed, as one can notice from Fig. 3, the images generated using TNs with larger bonds present more defined shapes (although they are not clearly recognizable as digits).

Looking at the reconstructed images shown in Fig. 4 one may notice that larger bond dimensions imply better reconstruction capabilities, although the digits are always hard to properly recognize.

To further understand this point, we tried to generate images using a *standard approach*, i.e. two different *Denoising Autoenconders* (DAEs). Their reconstruction performances are clearly better than those of MPSs, as shown in Fig. 5. To quantitatively sustain this statement, we evaluated the reconstruction accuracy on 2000 images different from those in the

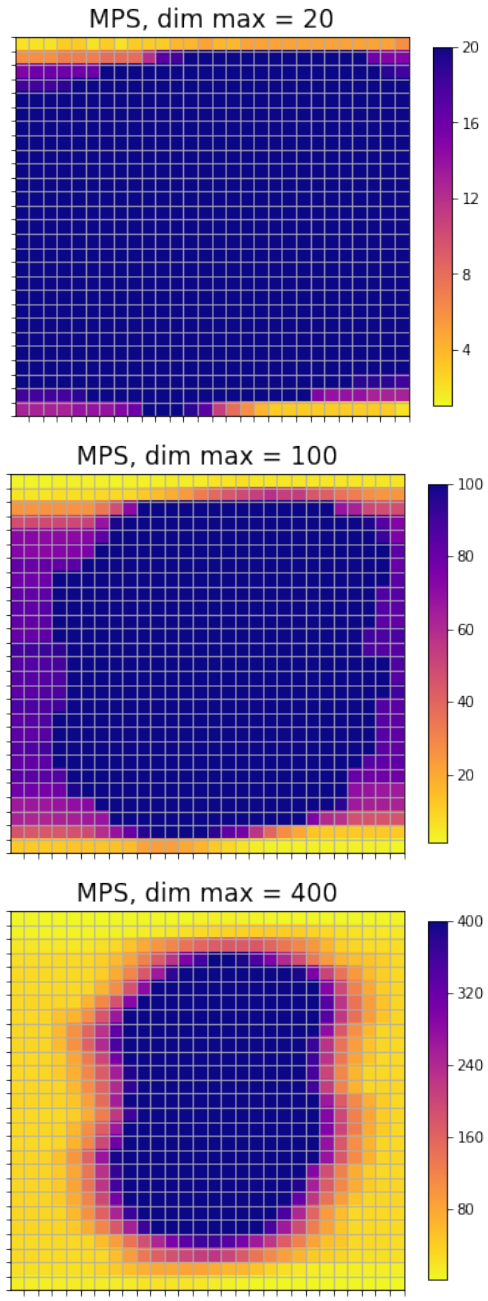


Figure 2: Figure shows the bond dimension of the MPS tensors. Each pixel represents the bond dimension of the left leg of the tensor associated to the identical coordinate in the original image.

training set for all MPS TNs and DAEs. The results are the following.

Approach	Accuracy
TN(20)	83.2%
TN(100)	83.0%
TN(400)	83.3%
DAE(MSE)	88.6%
DAE(BCE)	88.4%

Table 2: Table shows number of epochs and time duration of different training processes.

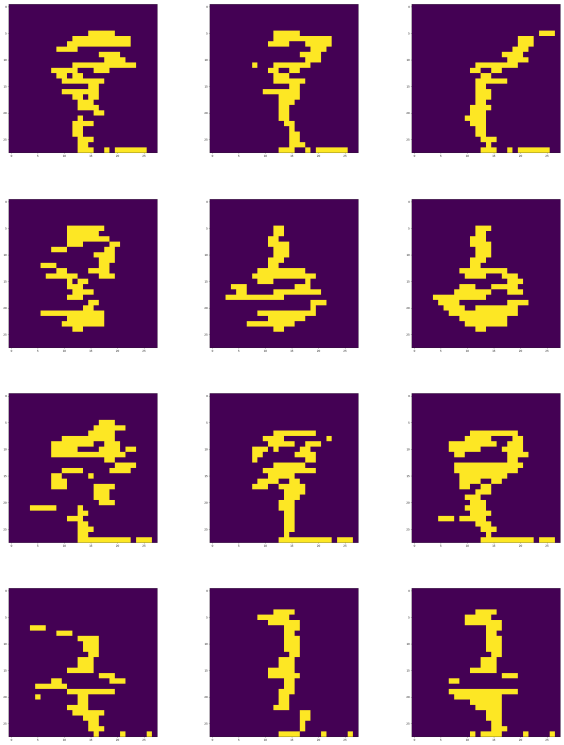


Figure 3: Figure shows some examples of images generated from scratch. Images in same column are generated using same tensor network. The only parameter that change is the maximum dimnesion of bonds. From left to right, respectively: 20, 100, 400.

CONCLUSIONS

As presented in the *Results* section, the DAEs generally perform better than the MPS. However, as explained in the *Theory* section, the MPS TNs have a lot of interesting features and in principle should be a preferable choice with respect to the standard approaches. In order to support this argument, as shown in Table 1, one may notice that MPS TNs requires a lot of computational resources and time to be trained and for this reason we choose to train them on a set of only 1000 samples, while DAEs could be trained in a similar amount of time on a set of 60000 samples. This could be the reason why TNs performances in our case turn out to be poorer than DEAs' ones. From this point of view, the performances of the TNs have really high possibilities of improvement (supposing to have sufficient computational resources and time).

SELF-EVALUATION

To obtain a better performing TNs a search on parameters can be done in order to find the optimal ones. This clearly requires lot of time and computational power.

Another possible improvement should be to study

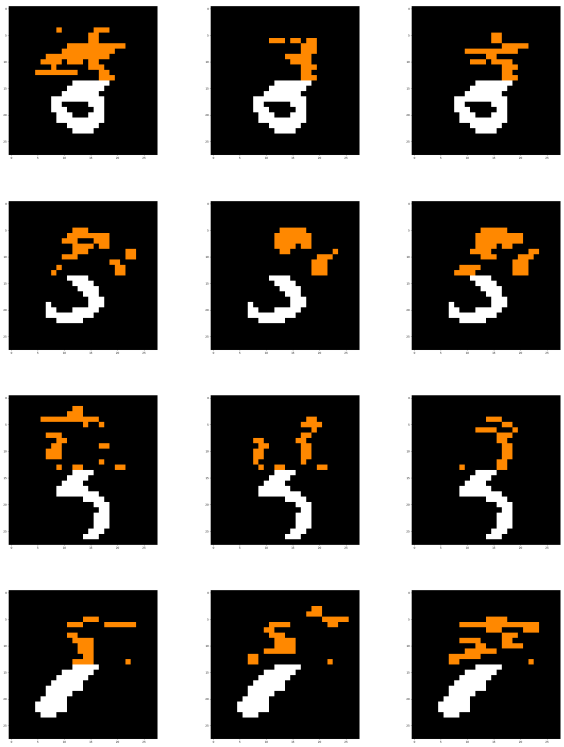
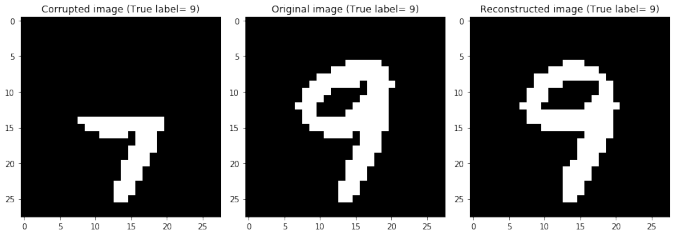
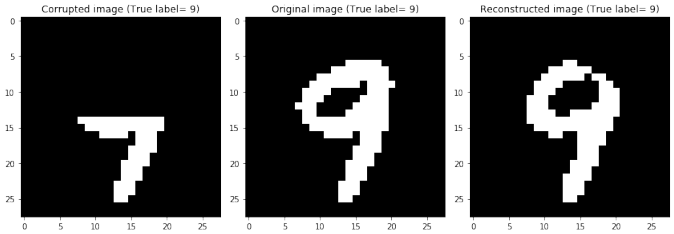


Figure 4: Figure shows some examples of reconstructed images. White pixels came from the original images while orange ones are those reconstructed by the tensor networks. Images in same column are generated using same tensor network. The only parameter that change is the maximum dimnesion of bonds. From left to right, respectively: 20, 100, 400.



(a) Reconstructed image using DAE with MSE loss.



(b) Reconstructed image using DAE with binary cross-entroyp loss.

Figure 5: Figure shows reconstructed images using denoising autoencoders (DAE).

a better implementation and try to use parrallel computing where possible.

Furthermore, to improve the final result the TNs can be allowed to have a greater rappresentation capability by increasing the maximum bonds dimension of the tensors. Of course, the drawbacks of this approach would be a more time demanding training process.

Correctness

The program works properly.

To achieve this result, we constantly tested the code while writing it.

During debugging we used to print to terminal some variables to check their correctness, even by hand evaluations if necessary (for example, when dealing with tensor contractions we checked manually that the results were as expected).

To better understand were the errors were hidden, we looked for and used many different *compiling flags* that enabled several warnings and even *run-time warnings*. When all issues were fixed we disabled all the flags because we noticed that they havily slowed the execution of the program.

Stability

Loops are made only on integers (usually the extreme values of the loop variables are the same used to initialize the dimension of the objects).

Memory error can arise (depending on the machine the program is run on) if the allowed maximum tensor bonds leads to exceed the RAM capacity.

Accurate discretization

An issue we encountered was caused by the fact that we started programming using REAL*4 Fortran type, but we soon noticed that we must dealt with number that could reach orders of magnitude of 10^{-200}, which made the REAL*4 variables to overflow causing program crashes (it reaches $10^{-}39$ as lower limit). We then decided to switch to use REAL*8 type for floating point values.

The tensors are randomly initialized, but this does not represent a problem since LEFT_CANO subroutine is called before performing the gradient descent procedure.

Flexibility

We tried to comment as much as possible the files to have a clear and understandable code.

Efficiency

In order to improve our program performances, we compiled it using the *-Ofast* flag.

All subroutines but that for the SVD procedure are implemented by hand. They may be improved.

For what concerns memory and time usage, they depend on the parameters passed.