

UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia “Galileo Galilei”

Corso di Laurea in Fisica

Tesi di Laurea Triennale

Quantum random numbers characterization via quantum-inspired machine learning

Relatore

Prof. Simone Montangero

Laureando

Tommaso Faorlin

Anno Accademico 2019/2020

Abstract

The generation of good random numbers impacts basic research and applications beyond pure academic interests: random numbers are required for countless applications, such as cryptography and simulations. For most applications, it is of outmost importance to know if a set of numbers is truly random, pseudo-random or contains some residual correlations. Tensor networks are powerful data structures that spring from quantum many-body physics and are now increasingly applied to machine learning applications. This thesis plans to explore the intersection between these two fields, applying quantum-inspired machine learning to random number generation. We aim to perform and characterise novel statistical checks comparing and characterising the statistical quality of different number sets: correlated, pseudo-random, and quantum-random.

Contents

Introduction	vii
1 Tensor networks	1
1.1 Tensors	1
1.2 Tensors manipulation	1
1.2.1 Outer product	1
1.2.2 Contraction	2
1.3 Tensor networks	2
1.3.1 Matrix Product State (MPS)	3
1.3.2 Tree Tensor Network (TTN)	3
1.3.3 Tensor network differentiation	3
1.3.4 Accessible information in a tensor network	4
2 Tensor Network Machine Learning	5
2.1 Machine Learning (ML)	5
2.1.1 Supervised Learning	5
2.1.2 Loss Function	6
2.1.3 Backpropagation	6
2.2 Tensor Network Machine Learning (TNML)	7
2.2.1 Framework	7
2.2.2 Encoding Data in a TN	7
2.2.3 Multiple Label Classification	7
2.2.4 MPS approximation	8
2.2.5 Weights optimisation algorithm	8
3 Random and pseudo-random number generation	9
3.1 Pseudo Random Number Generators (PRNGs)	9
3.2 Truly Random Number Generators (TRNGs)	9
3.2.1 Quantum Random Number Generators (QRNGs)	10
4 Test on random numbers	11
4.1 Pseudo Random vs Correlated sequences	11
4.1.1 Data preparation and model description	11
4.1.2 Confidences	11
4.1.3 bTTN model validation	12
4.1.4 Correlations and Entropies	13
4.2 Quantum Random vs Pseudo Random	14
Conclusions	15

Introduction

Nowadays, in spite of the fact that randomness and uncertainty are often viewed as an obstacle, many fields such as cryptography, Monte Carlo simulations or research methods in general, requires random numbers. There exists different types of random numbers generators and the one commonly used are capable of generating sequences of Pseudo Random Numbers (PRNs). Indeed, since classical computers are only able to execute algorithms, every result they can generate is deterministic. For this reason, classical computers are not able to generate truly random sequences of numbers, although for many applications using sequences generated by PRNGs is sufficient. In some case, however, it is necessary to use truly random numbers and so unpredictability is a fundamental requisite [1,2].

A different but fascinating topic that is emerging these days concerns the interface between machine learning and the physical sciences. Biologically inspired Artificial Neural Networks (ANNs) are a well known model for supervised learning and they are of increasing interest in several fields such as research and industry. ANNs describe a functional mapping containing many parameters, which are optimised during the training procedure, able to classify an input according to the class it belongs. Only in the last few years, some profound connections between Machine Learning (ML) and physics, especially quantum physics, have been shown in the two verses [3,4]. In particular, it has been shown that Tensor Networks (TN), which are a particular numerical method originated from quantum physics that plays the role of the biological originated ANNs, can be applied to solve ML tasks [5]. By using a TN approach in ML, new information can be obtained in comparison with a traditional ANNs approach, as we will see in Chapters 1 and 4.

In this thesis, after a concise overview over Tensor Network Machine Learning and Random Number Generation, a link between these two topics is presented. The aim of this work is to present a TN approach for the supervised learning problem of characterising the statistical quality of different number sets: correlated, pseudo-random and quantum-random. Simulations are carried out on a Binary Tree Tensor Network (bTTN) coded in Fortran by Timo Felser and Marco Trenti and running on a INFN's CloudVeneto remote instance.

Chapter 1

Tensor networks

In this chapter we are going to introduce tensor networks and briefly describe the differences between a Tensor Network approach to machine learning and another based on Neural Networks, in terms of accessible information [6, 7].

1.1 Tensors

A tensor is a multidimensional array. There are sundry ways to define it, using different level of abstraction and different languages. For our purposes, a N -rank, m -dimensional tensor T consists of a set of numbers T_{i_1, \dots, i_N} , where each index i_j has m possible values.

As shown in Fig. 1.1, a tensor is represented by a circle with lines coming out of it. Each line, also called edge, represents an index and has a proper dimension. To take an example, a rank-1 tensor describing the position in space of one object, is depicted as a circle with a 'three dimensional' line departing from it [6, 7].

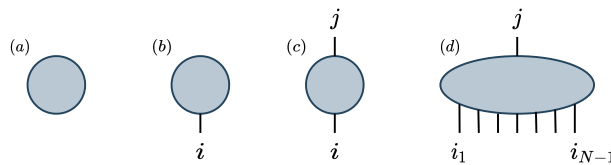


Figure 1.1: Examples of tensors in graphical representation: (a) Scalar (rank 0), (b) Vector ψ_i (rank 1). (c) Matrix $O_{i,j}$ (rank 2). (d) N -rank tensor ψ_{i_1, \dots, i_N} .

1.2 Tensors manipulation

The graphical notation introduced above, offers many advantages over traditional index notation, among which a lighter way to represent multi dimensional object manipulations. Later on, we will see that tensors are constantly manipulated and so why it is important to adopt a compact notation.

1.2.1 Outer product

The outer product between two tensors A and B is the element-wise product of the values of each tensor component. In diagrammatic notation (see Eq. 1.1), it is represented by two circles being placed next to each other and treated as a compound of the two.

$$A_{i_1 \dots i_r} B_{j_1 \dots j_s} = \begin{array}{c} \text{---} \\ | \\ \text{---} \end{array} \left(\begin{array}{c} \text{---} \\ | \\ \text{---} \end{array} \right) \begin{array}{c} \text{---} \\ | \\ \text{---} \end{array} \left(\begin{array}{c} \text{---} \\ | \\ \text{---} \end{array} \right) = \begin{array}{c} \text{---} \\ | \\ \text{---} \end{array} \left(\begin{array}{c} \text{---} \\ | \\ \text{---} \end{array} \right) \begin{array}{c} \text{---} \\ | \\ \text{---} \end{array} \left(\begin{array}{c} \text{---} \\ | \\ \text{---} \end{array} \right) = (A \otimes B)_{i_1 \dots i_r j_1 \dots j_s} \quad (1.1)$$

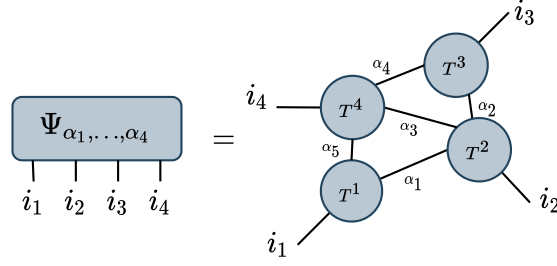


Figure 1.2: The coefficient of a many-body wave function with 4 constituents on the left can be decomposed into $Q = 4$ smaller tensors which are connected through $L = 5$ auxiliary links. The original tensor $\Psi_{\alpha_1, \dots, \alpha_4}$ can be obtained back by contracting tensors over virtual indices, following Eq 1.3.

1.2.2 Contraction

Any connection between two circles (shown on a more complex TN in Fig. 1.2) represents an index contraction and following Einstein's convention for indices, repeated indices are implicitly summed

$$\begin{array}{c} a \quad A \quad b \quad B \quad c \quad C \quad d \\ \text{---} \text{---} \text{---} \text{---} \end{array} = \begin{array}{c} a \quad O \quad d \\ \text{---} \text{---} \end{array} \quad (1.2)$$

$$\sum_{b,c} A_{a,b} B_{b,c} C_{c,d} = A_{a,b} B_{b,c} C_{c,d} = O_{a,d} \quad (1.3)$$

The result of the contraction is a matrix $O_{a,d}$ by the time there are only two indices left out of the sum and so the final tensor has rank 2.

An operation such as tensor trace (Eq. 1.4) can be plainly expressed by connecting a line departing from a tensor to that tensor itself. Partial trace operation follows naturally: given a tensor A having two indices m, n with the same dimension $d_m = d_n = d$ the latter is defined as the joint summation

$$\begin{array}{c} A \\ \text{---} \end{array} \text{---} \text{---} := \sum_i \begin{array}{c} A \quad i \\ \text{---} \end{array} \text{---} \text{---} = \sum_{i=1}^d A_{ii} \quad (1.4)$$

1.3 Tensor networks

Linked tensors can be arranged in several ways in space, and different spatial arrangements identify different classes of tensor networks (TNs). TNs have been developed to efficiently represent a quantum wavefunction of a quantum many-body system on a classical computer. Given a quantum system with size N (e.g. a lattice of N fermions) the most general quantum many-body pure state that describes the whole system can be written as

$$|\psi\rangle = \sum_{i_1, \dots, i_N} \Psi_{i_1, \dots, i_N} \left(\bigotimes_{j=1}^N |i_j\rangle \right) = \sum_{i_1, \dots, i_N} \Psi_{i_1, \dots, i_N} |i_1, \dots, i_N\rangle \quad (1.5)$$

where $\{|i_j\rangle\}_j$ is a canonical basis of the subsystem's j Hilbert space \mathcal{H}_j and the Hilbert space of the whole system corresponds to the tensor product of all the subspaces $\mathcal{H} = \mathcal{H}_1 \otimes \dots \otimes \mathcal{H}_N$. Supposing that each local state space is d -dimensional, the complete representation seeks for d^N coefficients. The complex amplitudes Ψ_{i_1, \dots, i_N} of this wave function define an exponentially large tensor and determines the state uniquely since by taking its squared module, $P_{i_1, \dots, i_N} = |\Psi_{i_1, \dots, i_N}|^2$ the probability to find the system in the configuration i_1, \dots, i_N is obtained. Using TN formalism it is possible to re-write this tensor as the contraction of a set of smaller tensors by expanding it over auxiliary virtual indices

$$\Psi_{i_1, \dots, i_N} = \sum_{\alpha_1, \dots, \alpha_L} T_{\{i\}_1\{\alpha\}_1}^1 T_{\{i\}_2\{\alpha\}_2}^2 \dots T_{\{i\}_Q\{\alpha\}_Q}^Q \quad (1.6)$$

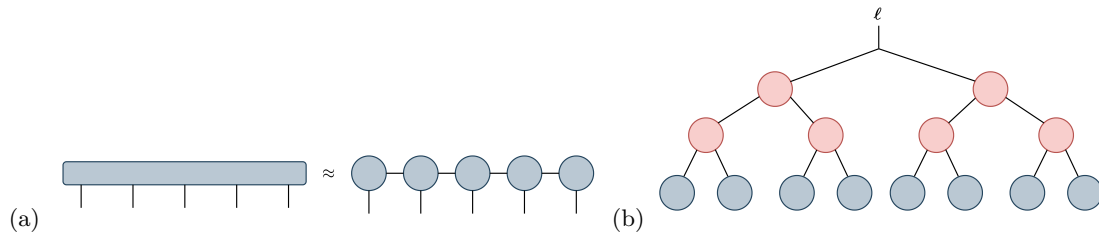


Figure 1.3: (a) A MPS decomposition, also named tensor train. A sign of approximation is used since the operation involves an information compression. (b) Tree tensor network. Tensors composing the bottommost layer take data as input and are called *input data*. The uppermost dangling link ℓ will become the prediction of the network.

for a network with Q nodes and L links. Links are also called edges, and they are dangling edges if only one side is connected to a tensor. For us, a dangling edge (e.g. i_2 in Fig. 1.2) is a physical index since it defines the amplitudes tensor. On the other hand, edges that connect two tensors are called virtual indices and each tensor can have any number of connected links $\{\alpha\}_l$ and any number of free legs over physical indices $\{i\}_q$. An example of how a tensor can be split into smaller ones is shown in Fig. 1.2. The following are two examples of tensor networks.

1.3.1 Matrix Product State (MPS)

A first crucial example for one-dimensional quantum systems, is the Matrix Product State (MPS) tensor network, depicted in Fig. 1.3a. Each physical index has a dimension d , the number of local configurations and each virtual index α has a dimension χ , which is known as bond dimension: a knob we can tune (hyper-parameter of the TN) in order to control the MPS approximation. In fact, allowing χ to grow exponentially, we pass from a trivial approximation to the most exact one (d^N) of a quantum state. It is worth to observe that, using a MPS, a rank- N tensor with d^N components is approximated with a train of lower order tensors with $N \cdot d \cdot \chi^2$ coefficients instead. The dimensional scaling is then linear rather than exponential in the input size N . From the MPS ansatz, the Projected Entangled Pair States (PEPS) can be deftly constructed in order to deal with two dimensional quantum systems.

1.3.2 Tree Tensor Network (TTN)

In this work, a binary Tree Tensor Network (bTTN) (depicted in Fig. 1.3b) is used, whom tensors are connected according to a hierarchical tree structure. The lattice is entirely composed of tensors with three links and each layer has half tensors and double the tensor of the upper an bottom layer respectively. The choice of a bTTN is a natural mirror of how a group of simple information (contained in the bottommost layer) combine to form more complex concept.

1.3.3 Tensor network differentiation

As we will discuss in chapter 2, any kind of TN (but also more classical Neural Networks) should perform an operation called gradient descent, in order to optimise the entries of the tensors with respect to a given input. We therefore need to define a tensor network differentiation. Without surprise, the derivative of any linear function of tensors (Eq. 1.6) with respect to a particular tensor T^* , is given by the tensor network where the tensor T^* has been removed

$$\frac{\partial}{\partial T^*} \left[\text{Diagram with } T^* \text{ tensor} \right] = \text{Diagram without } T^* \text{ tensor} \quad (1.7)$$

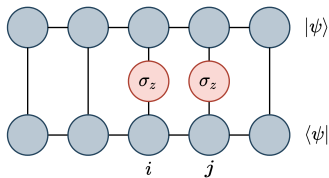


Figure 1.4: Example of a two sites correlation on a MPS following Eq. 1.8.

1.3.4 Accessible information in a tensor network

With the basic knowledge introduced so far, one can compute quantities of interest on a TN.

Expectation values

Given a local observable acting on a local Hilbert space \hat{O} , or a tensor product of those $\otimes_i \hat{O}_i$, one can compute efficiently its expectation value in a state. An interesting expectation value that can be calculated is a two sites (i and j) correlation, using a product of two σ_z Pauli matrices, defined as

$$C_{i,j} = \langle\psi|\sigma_i^z\sigma_j^z|\psi\rangle \quad (1.8)$$

We use this particular spin operator since we encode data in the bTTN using a local spin map (see Section 2.2.2). In case of maximum correlation or anti-correlation, the information contained on one feature site can be obtained by the other one and thus can be neglected. In figure 1.4 is depicted an example of such calculation, where the two Pauli operators are inserted in the contraction between an MPS $|\psi\rangle$ and its complex conjugate $\langle\psi|$. An important thing to notice is that, in this way, also state norm or state overlap can be calculated.

Entropy

Von Neumann defined the entropy S_V of a generic quantum state density matrix ρ by the formula

$$S_V(\rho) = -\text{Tr}(\rho \log \rho) \stackrel{(a)}{=} -\sum_i \lambda_i \log \lambda_i \quad (1.9)$$

Trace operation is invariant with basis changing and $\rho = \rho^\dagger$ is hermitian, so we can rewrite it in a basis in which it is diagonal [8]. Doing that, we find Shannon's definition of entropy, calculated on the eigenvalues (a). When dealing with tensor networks (pure states) S is the entanglement entropy and reflects the shared information between two TNs bipartitions.

Given a pure state $|\psi\rangle_{AB} \in \mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_B$, it can be decomposed according to Schmidt decomposition [8] into

$$|\psi\rangle_{AB} = \sum_{\alpha} \lambda_{\alpha} |\alpha_A\rangle |\alpha_B\rangle \quad (1.10)$$

where $\{|\alpha_A\rangle\}$ and $\{|\alpha_B\rangle\}$ are orthonormal states of systems A and B respectively and λ_i are the Schmidt-coefficients (non zero normalised singular values of the decomposition). The entanglement entropy is then defined as

$$S = -\sum_{\alpha} \lambda_{\alpha}^2 \ln \lambda_{\alpha}^2 \quad (1.11)$$

The minimal entropy $S = 0$ is obtained when, for example, $\lambda_1 = 1$ and in this case we can completely separate the two bipartitions as they share no information. An higher S means that some information is shared. In fact, in a machine learning framework, if the features in one bipartition provide no valuable information for the classification task, the entropy on that bipartition is 0, and it could be discarded with no loss of information.

Chapter 2

Tensor Network Machine Learning

As technology advances, an enormous amount of data is generated every day, and the capability to manipulate them efficiently is increasingly important. In this framework, Machine Learning (ML) algorithms have been developed with the aim of finding patterns in massive amount of data. In this chapter we will talk about a subclass of ML called supervised learning and subsequently understand how to accomplish supervised algorithm tasks using TNs notions, introduced above.

2.1 Machine Learning (ML)

The main goal of a ML algorithm is being able to label correctly an unknown input set of data. The choice of the label depends on an inferred function that take as an input a set of data \mathbf{x} and, on the basis of a set of parameters θ , provides the best label for \mathbf{x} . The choice of the optimal set of parameters is done during the so-called training of the algorithm.

2.1.1 Supervised Learning

Given sufficient labelled examples, a supervised-learning system would learn to recognize inside patterns and uses them to predict the values of the label on additional unlabelled data. Artificial Neural Networks (ANNs) are a human-brain inspired model used in supervised learning that can learn by processing labelled examples, forming probability-weighted associations between inputs and results. ANNs are essentially made of nodes (neurons) and weights (axons), respectively circles and lines in Fig. 2.1. Nodes and weights are further organised in layers: the input layer welcomes the input data, the output layer returns the label prediction and in between the two there are one or more (Deep Neural Networks (DNNs)) hidden layers.

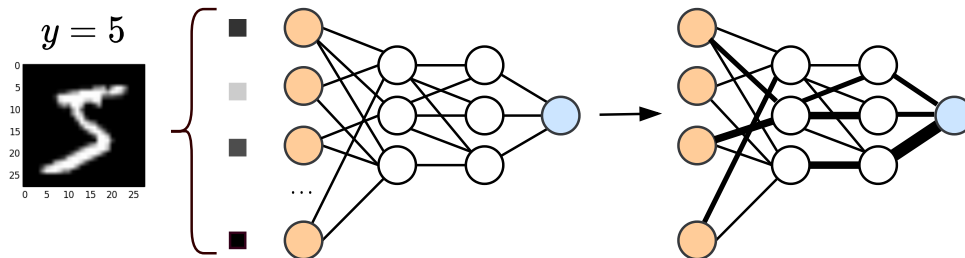


Figure 2.1: A two-hidden-layers DNNs: orange neurons compose the input layer and the sky blue neuron is the output layer. Labelled data in the training set \mathcal{S} (images of number for example) are given in input in order to optimise the model. Here is depicted the idea of training: we go from a blank model, with weights (black lines) of the same sizes to a trained one, which has links of different thicknesses.

The neuron \mathcal{N} is intended as a cell that receive a linear combination of the inputs x and weights w , then through an activation function σ outputs a number.

$$\mathcal{N}(x, w, b) = \sigma(x \cdot w + b) \in \mathbb{R} \tag{2.1}$$

where b is the bias and σ is usually a non linear map that can be chosen among different ones, such as sigmoid or ReLU. Each possible configuration of the network's weights is part of the hypothesis class set \mathcal{H} .

Formally, in a supervised learning framework we have a domain set \mathcal{X} that contains all the possible instances $\mathbf{x} \in \mathcal{X}$ usually treated as a vector of features. Each instance has its own label $y \in \mathcal{L}$ so that when forming the training set \mathcal{S} we are selecting an instance-label pair that belongs to the domain points $\mathcal{S} \subset (\mathcal{X} \times \mathcal{L}) : \mathcal{S} = ((\mathbf{x}_1, y_1) \dots (\mathbf{x}_m, y_m))$. Ideally, we assume there exist a probability distribution \mathcal{D} over the domain \mathcal{X} that generates the instances, and a labelling function $f : \mathcal{X} \rightarrow \mathcal{L}$, both not known a priori. Then, through the training procedure, we use ML to find a good prediction rule $h \in \mathcal{H}$, which means obtaining a good approximation of f and \mathcal{D} . The output of the entire training process is then a trained model able to classify correctly different unlabelled, see Fig. 2.2.

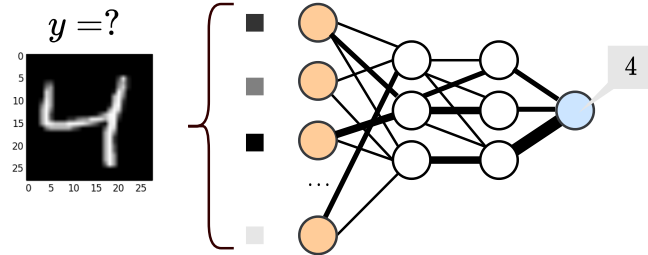


Figure 2.2: A correctly trained model should be able to map the features vector $\mathbf{x} \in \mathcal{X}$, (the one containing the value of pixels of the image representing the number '4') to its right label $\ell = 4 \in \mathcal{L}$.

2.1.2 Loss Function

The training process in a ML algorithm makes use of a loss function (or cost function) λ . It's a method of evaluating how well a specific prediction rule h reproduce the ideal labelling function f . If predictions deviates too much from actual results, loss function would cough up a very large number. Gradually, with the help of some optimisation function, the loss function is able to reduce the error in prediction. In fact, an optimisation process aims to minimise the training error, defined by means of different loss functions λ as:

$$L_{\mathcal{S}}(h) = \frac{1}{N_t} \sum_{i=1}^{N_T} \lambda_i(h, (\mathbf{x}, y))_t \quad (2.2)$$

where N_T is the number of training data. There exists a lot of different loss function but one of the most commonly used and also highly expressive to understand the key idea is the squared loss

$$\lambda_i(h, (\mathbf{x}, y)) = \lambda_{sq,i}(h, (\mathbf{x}, y)) = (h(\mathbf{x}_i) - y_i)^2 \quad (2.3)$$

used for the penalisation of large errors in regression but also in classification.

2.1.3 Backpropagation

Defined the hypothesis class \mathcal{H} , the training set $\mathcal{S} \in \mathcal{X} \times \mathcal{L}$ and specified the loss function λ_i in the training error, the learning procedure (or weights optimisation) is led by the so called backpropagation. It is an algorithm that uses the gradient descent to update the parameters of the network in order to minimise the training error (Eq. 2.2). Chosen an initial predictor $h_0 \in \mathcal{H}$ (usually a random set of parameters), it iteratively updates the parameters according to the following equation

$$\begin{aligned} \alpha_t &= \eta \nabla_h L_{\mathcal{S}}(h_t) \\ h_{t+1} &= h_t - \alpha_t \end{aligned} \quad (2.4)$$

where $\nabla_h L_{\mathcal{S}}(h_t)$ is the gradient of the training error and we have introduced the learning rate η that controls how big a step we should take in the direction of the gradient at time step t . We won't discuss here how to perform more complex gradient descents as we will use only the ideas from the vanilla gradient descent for developing a TNs analog.

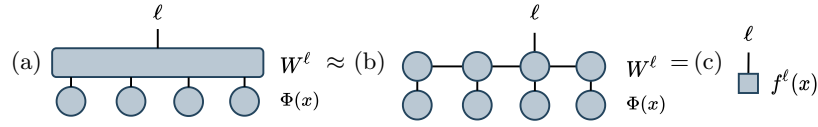


Figure 2.3: A multiple label classification model (a) approximated using an MPS topology (b). The contraction of the weight tensor with a feature vector is the decision function $f^\ell(x)$ (c), as said in Eq. 2.5. The label index ℓ is placed arbitrarily on one of the MPS tensors but can be moved to other locations. [9]

1.3.4) for all labels ℓ and we further classify the input \mathbf{x} by choosing the label ℓ for which $|f^\ell(\mathbf{x})|$ is largest.

2.2.4 MPS approximation

Once the weights have been organised into a TN structure through W^ℓ , a way to handle the curse of dimensionality must be found. In fact, taking the spin map $\phi^{s_j}(x_j) : \mathbb{R} \rightarrow \mathbb{R}^2$ ($d = 2$), the components of W^ℓ scales exponentially in the input size N : $N_\ell 2^N$. As N grows, the tensor becomes quickly unmanageable. The strategy pursued to solve this nuisance is to represent this tensor as a TN, because we can approximate the exponentially large set of components in terms of a much smaller set of parameters whose number grows only polynomially in the dimension of the input space. As depicted in Fig. 2.3, for MNIST classification W^ℓ has been written as a MPS tensor network

$$W_{s_1 s_2 \dots s_N}^\ell = \sum_{\alpha_1 \dots \alpha_{N-1}} T_{s_1 \alpha_1}^1 T_{s_2 \alpha_1 \alpha_2}^2 \dots T_{s_j \alpha_j \alpha_{j+1}}^{\ell; j} \dots T_{s_N \alpha_{N-1}}^N \quad (2.9)$$

so that we deftly pass from $N_\ell 2^N$ to $N_\ell N 2 \chi^3$ components of W^ℓ (where χ is the already discussed bond dimension, see Section 1.3.1 and Fig. 2.3). The label index ℓ is placed arbitrarily on one of the MPS tensors but can be moved to other locations [9].

2.2.5 Weights optimisation algorithm

According to Sections 2.1.2 and 2.1.3 we have to implement the cost function into the TN and minimise it by optimising each tensor. Picking always the square loss as loss function and defined the training set $\mathcal{S} \in \mathcal{X}$ with m features \mathbf{x}_i with their label y_i , then in TN formalism Eq. 2.2 become

$$L_{\mathcal{S}}(h) = \frac{1}{2} \sum_{i=1}^m \sum_{\ell} (f^\ell(\mathbf{x}_i) - \delta_{y_i}^\ell)^2 \quad (2.10)$$

where $\delta_{y_i}^\ell = 1$ is a Kronecker's delta that is 1 if $y_i = \ell$ and 0 in the other cases. Each tensor can be optimised by minimising this cost function. Given, for example, the tensor T^* and following Section 1.3.3, we calculate the gradient of Eq. 2.10 with respect to this tensor

$$\frac{\partial}{\partial T^*} L_{\mathcal{S}} = \sum_{i=1}^m \sum_{\ell} (f^\ell(\mathbf{x}_i) - \delta_{y_i}^\ell) \frac{\partial f^\ell(\mathbf{x}_i)}{\partial T^*} = \sum_{i=1}^m \sum_{\ell} (f^\ell(\mathbf{x}_i) - \delta_{y_i}^\ell) \tilde{T}^* = \hat{T}^* \quad (2.11)$$

The update rule, where $\alpha \in \mathbb{R}$ is the learning rate hyper-parameter, is then

$$T_{new}^* = T_{old}^* - \alpha \hat{T}^* \quad (2.12)$$

The optimisation algorithm then sweeps back and forth along the TN in order to do this to all the tensors involved. A complete sweep constitutes an epoch of the training and after each epoch we can measure the overall performances by computing the accuracy over the training set \mathcal{S} and over the validation set \mathcal{T} .

Chapter 3

Random and pseudo-random number generation

Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.

Various techniques used in connection with random digits, *John Von Neumann*, 1951.

Commonly, randomness is the lack of pattern or predictability in events of any kind. It was only in the sixteenth century, with the advent of calculus, that randomness was defined more specifically and included in a new formal framework, namely probability theory. Although randomness and uncertainty are often viewed as an obstacle, today random numbers have a fundamental role in many fields such as cryptography, Monte Carlo simulations [2] and research methods in general [1]. Being aware that randomness is a broad argument, the purpose of this chapter is to briefly illustrate the differences between two ways in which random numbers can be generated (PRNGs vs QRNGs) and to give the key ideas on two ways to generate them exploiting quantum mechanical properties (Section 3.2.1).

3.1 Pseudo Random Number Generators (PRNGs)

A PRNG, also known as deterministic random bit generator (DRBG), is nothing more than a mathematical formula producing periodic sequences of numbers (PRNs) that are completely determined by the initial state called seed. Although the output binary sequences are usually perfectly balanced between 0 and 1, the algorithm used to generate numbers inserts strong long-range correlation, which can undermine cryptographic security. Furthermore, the knowledge of the arithmetical algorithm and the particular seed allow to reproduce a certain sequence of numbers, since there exists a one-to-one correspondence between the two.

If a large amount of these numbers is needed, computers are involved and they admirably do this job by making use of Pseudo Random Number Generators (PRNGs).

In this work, the tool used for generating PRNs is Python's `random` module, which uses the Mersenne Twister PRNG algorithm as its core generator [11]. Its name derives from the fact that its period length (\mathcal{P}) is chosen to be a Mersenne prime, specifically $\mathcal{P} = 2^{19937} - 1$.

3.2 Truly Random Number Generators (TRNGs)

For many application, unpredictability is a fundamental requisite; therefore, in literature can be found plenty of ways to generate truly random numbers [12]. Here, we briefly explain how quantum mechanical systems can be used to generate truly random numbers [13], motivated by the fact that quantum random numbers (QRNs) will be used in the final chapter.

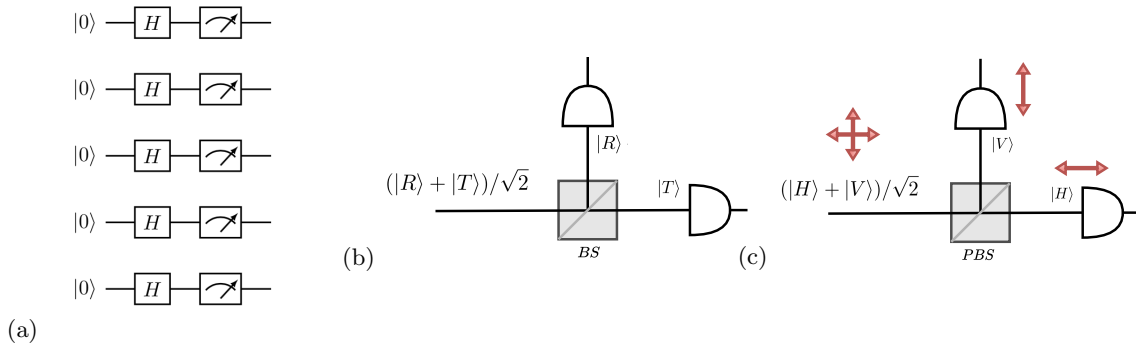


Figure 3.1: (a) Circuit implemented in the five-qubit IBM quantum computer: Hadamard gate are applied to a $|0\rangle$ quantum state to have a quantum superposition, which is then measured giving a string of five truly random bit. (b) After passing through a symmetric beam splitter (BS), a photon exists in a superposition of transmitted (T) and reflected (R) paths $(|R\rangle + |T\rangle)/\sqrt{2}$. (c) Random bits generation exploiting single photon polarization degree of freedom.

3.2.1 Quantum Random Number Generators (QRNGs)

In quantum mechanics, a two-level system can be prepared in a superposition of the (measurement) basis states. According to Born's rule, the measurement outcome of this particular quantum state can never be predicted better than blindly guessing. Therefore, the nature of inherent randomness in quantum measurements can be exploited for generating true random numbers.

In this work we used two different types of QRNGs: one developed by the Information Engineering Department (DEI) at the University of Padova (UniPD) [14] and a five-qubit quantum computer of the IBM Q Experience, through the open-source framework for quantum computing Qiskit [15]. We see here briefly an implementation based on a single photon detector which is at the base of the more complex platform developed at DEI [14]. As a matter of interest, besides optical systems, there have also been proposals for QRNG based on radioactive decays or atomic quantum systems.

IBM Quantum Computer

The IBM Quantum Experience provides the possibility to use one of his quantum computer through Qiskit, an open source software development kit for working with quantum computers at the level of pulses, circuits and algorithm. Among all the different backends, we chose the five-qubit quantum computer named `ibmq_london`, we applied to each qubit a Hadamard gate to obtain quantum superposition (in the Bloch sphere, we are mapping the vector $|0\rangle$ in the Z basis to the vector $|+\rangle$ in the X basis)

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = |+\rangle \quad (3.1)$$

and then we measured all the five registers thus obtaining 5 truly random bits (see Fig. 3.1a).

Single Photon Detector

High-quality optical components can be found with a relative low costs and together with the potential of chip-size integration, most of today's practical QRNGs are implemented in photonic systems. For example, a photon can be encoded with information about a two-level quantum system, that is a qubit by definition, using its polarization. A single photon can be described as having horizontal or vertical linear polarization, or a superposition of the two (see Fig. 3.1c). A polarising beam splitter (PBS) then transmits the horizontal and reflects the vertical polarisation. For random bit generation, the photon is measured by two single-photon detectors (SPDs). There exists also path-based QRNG (see Fig. 3.1b), where a photon, after passing through a symmetric beam splitter (BS), exists in a superposition of transmitted (T) and reflected (R) paths.

Chapter 4

Test on random numbers

In this chapter we apply TNML to solve the problem of distinguishing between correlated, pseudo-random, truly random sequences of numbers. For this purpose we build and train a bTTN. The work is divided in two main parts: we first perform a distinction between PRNs and correlated sequences. Then, we also describe and try to address a similar approach to the distinction between sequences of PRNs and QRNs.

4.1 Pseudo Random vs Correlated sequences

4.1.1 Data preparation and model description

The dataset $\mathcal{X} \times \mathcal{L}$ (feature space times label space) consists of 100 thousand PRNs sequences and 100 thousand sequences of correlated numbers. Both sequences are made of numbers containing the label information ($\ell = 0$ for PRNs and $\ell = 1$ for correlated) and 32 real numbers $x_j \in [0, 1]$ with four decimal digits. PRNs sequences are created as described in Sec. 3.1.

On the other hand, correlated numbers are created in eight equiprobable different ways, where each one is randomly chosen by means of another PRNG. It is important to point out that each kind of correlation we introduce is characterised by a certain periodicity \mathcal{P} and that six out of eight different ways correlate numbers with the same fixed period (that will be visualised later on in Fig. 4.3 when computing sites correlations). By correlating numbers we mean, for example, that we take two numbers randomly and the third as the sum, or difference, or multiplication of the previous two, until 32 is reached (in this case we say that the sequence has $\mathcal{P} = 3$). The two out of eight procedures remaining correlate numbers with $\mathcal{P} = 2$.

We define the training set $\mathcal{S} \subset (\mathcal{X} \times \mathcal{L})$, which contains a fraction of 4/5 of the entire dataset, while the remaining 1/5 forms the test set $\mathcal{T} \subset (\mathcal{X} \times \mathcal{L}) : \mathcal{S} \cap \mathcal{T} = 0$. Sequences of numbers are then mapped to the feature space using the same spin map described in Eq. 2.6. The weight tensors W^ℓ , written by using a bTTN topology (Section 1.3.2), are then optimised using a conjugate gradient descent, which is a more sophisticated version of the vanilla method described in Sec. 2.1.3 and Sec. 2.2.5. Notice that there exist other stochastic gradient descent that can be explored, but we leave their exploration as an outlook for our work.

4.1.2 Confidences

We cast the classification problem to a probability calculus on a quantum state, written in a fixed basis. In a quantum mechanical framework, as stated also in Section 2.2.3, our trained network can be seen as the quantum superposition state $|\psi_{TTN}\rangle = c_0 |\psi_{TTN}^0\rangle + c_1 |\psi_{TTN}^1\rangle$ where $c_0^2 + c_1^2 = 1$. In this case, $|\psi_{TTN}^0\rangle$ and $|\psi_{TTN}^1\rangle$ play the role of two optimised weight tensors W^0 and W^1 . Given a sequence $\mathbf{x} \in \mathcal{T}$, we define the confidence as the probability $P^\ell(\mathbf{x}) \in [0, 1]$ that \mathbf{x} is a correlated or pseudo random number sequence, respectively, namely

$$P^\ell(\mathbf{x}) = \frac{|\langle \Phi(\mathbf{x}) | \psi_{TTN}^\ell \rangle|^2}{\sum_{\ell=0}^1 |\langle \Phi(\mathbf{x}) | \psi_{TTN}^\ell \rangle|^2} \quad (4.1)$$

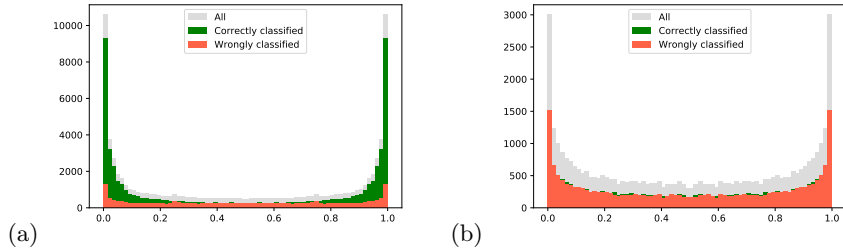


Figure 4.1: Confidences in prediction. In red are reported wrongly classified sequences, in green the one classified correctly, and in grey all the sequences contained in the test set \mathcal{T} . In the left side (a) we have a well trained bTTN classifier. Here, max bond dimension is $\chi = 25$ and for each tensor the number of optimization steps is $N_{pass} = 12$. On the right (b), instead, is shown the confidences plot of a bad model, that is, the one trained for the distinction between QRNs and PRNs (see Section 4.2), with the same hyper-parameter.

where we normalised the results since the output of the contraction is not normalised. Since Eq. 4.1 gives a confidence both for $\ell = 0$ and $\ell = 1$ and the two classification labels are exactly the boundaries of the interval in which probability is normalised, we plot in grey in Fig. 4.1 all the confidences for $\ell = 0$ (e.g., if $P^0(\mathbf{x}) = 0.12$ we know for certain that \mathbf{x} is associated to $\ell = 1$ with $1 - P^0(\mathbf{x}) = 0.88$). All the confidences have subsequently been compared with the true labels of the sequences and thus the green and orange histograms are built. Our bTTN model shows a flatter distribution in terms of wrongly classified data (e.g., when a PRNs sequence with $\ell = 0$ is classified as $\ell = 1$). Remarkably though, the model behaves differently when dealing with correctly classified sequences, indicating with the two peaks at the boundaries that the bTTN is very confident when predicting the correct label.

4.1.3 bTTN model validation

In order to validate our classification model we made use of two well-known tools of predictive analytics and performance measurements: the confusion matrix and the Receiving Operator Characteristic (ROC) curve. The first one, reported on Fig. 4.2a, shows that the model is able to associate a sequence to its correct label. In order to check the ability of the bTTN to classify whether a sequence is made of PRNs or contains correlations, we consider the ROC curve. Therefore, we define TP (TN) and FP (FN) as the number of the true positive (negative) and the false positive (negative) classifications made by the machine after the training. Then, we plot the rate of correctly tagged sequences (defined as True Positive Rate $TPR = TP/(TP + FN)$) against the rate of wrongly tagged sequences (defined as False Positive Rate $FPR = FP/(FP + TN)$). By changing the acceptance probability threshold we obtain the orange curve shown in Fig. 4.2a. In order to quantify the efficiency of our classifier we compare it with the so-called line of no-discrimination, which represents the ROC of a randomly guessing classifier. To compare different classifiers, it can be useful to summarize the performance of each classifier into a single measure. One approach is to calculate the Area Under the Curve (AUC). It is equivalent to the probability that a randomly chosen positive instance is ranked higher than a randomly chosen negative instance. In other words, higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s. In our case, $AUC = 0.818$, so we have another way to validate our model.

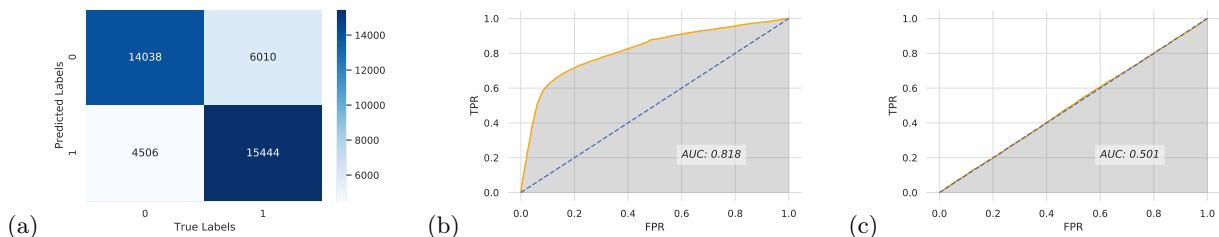
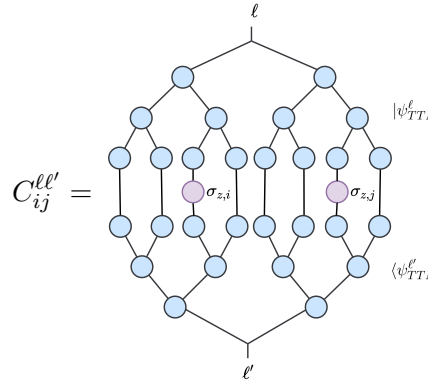


Figure 4.2: Confusion matrix (a) and ROC curve (b) of the PRNs vs correlated distinction (orange), compared with the line of no-discrimination (dotted-navy-blue line). ROC curve (c) of the QRNs vs PRNs distinction, discussed in Section 4.2.

4.1.4 Correlations and Entropies

Thanks to the bridge between TNML and quantum physics and according to what said in Section 1.3.4, one can compute quantities of interest on a TN and relate them to the classification problem under investigation. We start by computing the correlations contained in the TN after the training procedure, over pair of sites of the bottommost layer. In the following graphs we are calculating the expectation value of the product of two σ_z over a pair of sites i and j according to Sec. 1.3.4, and this can be represented as

$$C_{ij}^{\ell\ell'} = \frac{\langle \psi_{TTN}^{\ell'} | \sigma_{z,i} \sigma_{z,j} | \psi_{TTN}^{\ell'} \rangle}{\sqrt{\langle \psi_{TTN}^{\ell'} | \psi_{TTN}^{\ell'} \rangle \langle \psi_{TTN}^{\ell} | \psi_{TTN}^{\ell} \rangle}} \quad (4.2)$$


where the normalisation is required because tensors are not already normalised. By doing this operation, over all the possible combinations of two sites, we obtain Figs. 4.3. The first thing that catches the eye in Fig. 4.3a is that the periodicity of the sequences used during the training emerge in the correlations.

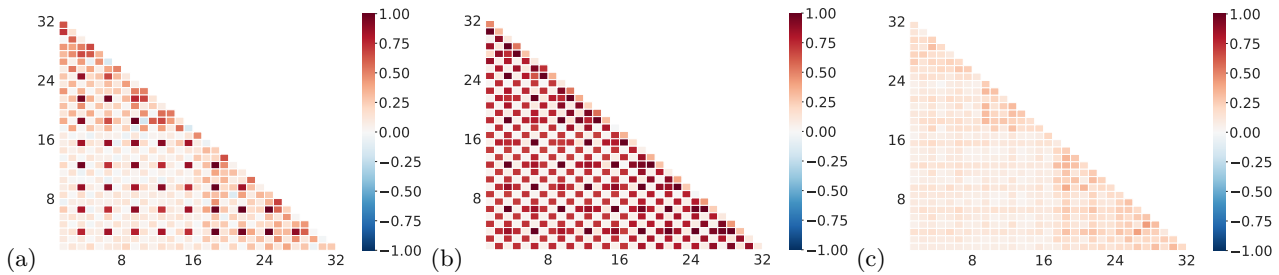


Figure 4.3: For all the figures, in the x-axis we put $|\psi_{TTN}^{\ell'}\rangle$, in the y-axis $\langle \psi_{TTN}^{\ell'}|$ and in the z-axis the result of $C_{ij}^{\ell\ell'}$. Two-sites correlations on the correlated sequences bTTN (a) ($\ell = 1$) trained on 200 thousand numbers sequences with $\mathcal{P} = 3$. Two-sites correlations on the correlated numbers sequences bTTN (b) ($\ell = 1$) and on the PRNs sequences bTTN (c) ($\ell = 0$) both trained on 1 million of the same sequences used in (a). The increase of the dataset size allows the less-probable correlation with $\mathcal{P} = 2$ to be observed. ($\chi = 25$, $N_{pass} = 12$).

Other simulations are made varying only the dataset size. In particular, the biggest one involves 1 million of sequences with $\mathcal{P} = 3$, and the result on correlations is reported in Fig. 4.3b,c. On heatmap b, we can observe how the less-probable correlations with $\mathcal{P} = 2$ emerge. The additional 16×16 triangles that appears (Fig. 4.3a) under the main pattern, are not related to sequences correlations and might be due to noise introduced by the hierarchical structure of the bTTN. In fact, as the dataset enlarges (Fig. 4.3c), triangles become smaller, indicating that the noise affects gradually lower tensors.

Another way to quantify correlations in number sequences is computing the entropy of the weight states. In Fig. 4.4 the entropy of each tensor belonging to the five different layers of the bTTN. The bottommost layer of the two graphs displays the entropies of the features, which reflects the information of the input vector connected below. Higher layers correspond instead to the mutual information of the input features connected from below. In the ML context, if the features in one bipartition provide no valuable information for distinguishing between the given classes, the entropy is zero. On the other hand, S increases the more information between the two bipartitions are exploited. The entropy information can be useful to optimise the learning procedure: whenever $S = 0$, the feature can be discarded with no loss of information for the classification and a new, more efficient model with features and tensors can be introduced. On the contrary, whenever a bipartition entropy

is high, highlights which features are important for the correct predictions. In short, if the entropy of a feature bipartition is low, we can discard one of them providing negligible loss of information.

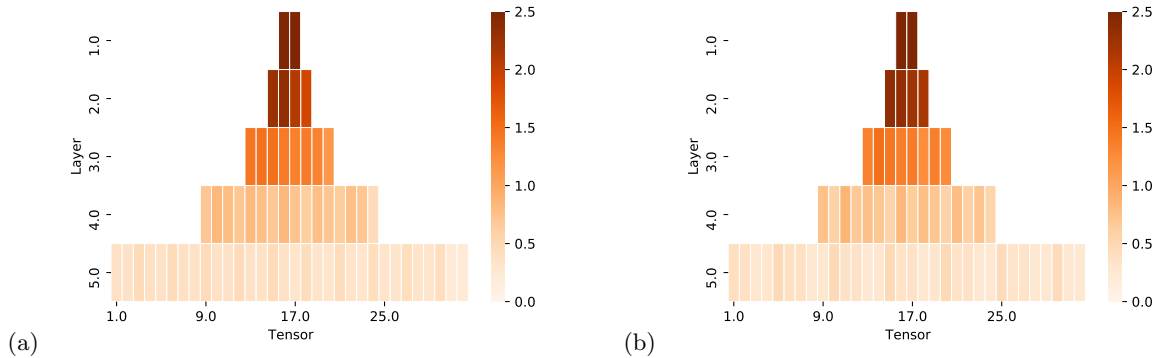


Figure 4.4: Single-tensor entropies. On the left, entropies of a bTTN trained on sequences with $\mathcal{P} = 3$. Same on $\mathcal{P} = 5$ on the right. It is interesting to observe how entropy increases climbing the hierarchical structure and how the sequences period is reflected on the feature layer (5.0).

We can observe how in the subjacent layer of both (a) and (b) in Fig. 4.4 entropies are periodic. This highlights the fact that the bulk of information is stored in the feature that contains the sum of the previous 2 (4) features in case of $\mathcal{P} = 3$ ($\mathcal{P} = 5$).

4.2 Quantum Random vs Pseudo Random

As a final step, we have tried to train our machine in order to distinguish between sequences of pseudo-random and truly random numbers. The sequences we have used are those taken generated according to Sec. 3.1 and Sec. 3.2.1.

The first performed comparison was between QRNs and correlated numbers sequences. The result is similar to the one between PRNs and correlated number sequences in terms of test accuracy (an average of 74.38% and 74.69% in five epochs for QRNs and PRNs vs correlated number sequences respectively) and in terms of correlations and entropies. This result shows how QRNs and PRNs perform in the same way against correlated number sequences.

Finally, we tried to distinguish QRNs from PRNs. An initial approach, following Section 4.1.1 where correlated number sequences are replaced with QRNs sequences and PRNs sequences remains the same, returned a model unable to distinguish between the two kind of sequences, or rather the test accuracy with which the bTTN tries to associate a sequence to its correct label is less than 50%.

The second approach was to build PRNs sequences collecting one number every 50000 generated. In this way, a larger amount of number is exploited to find differences from the QRNs in terms of long range correlation, but the model was still unable to recognise and correctly label the sequences. In Fig. 4.1b we plot the confidence graph related to this last case: the network has been able to find a wrong guessing frequency on the train set \mathcal{S} and the results on the test set \mathcal{T} shows two bins in 0 and 1 with the same height for wrongly and correctly classified data. Furthermore, in Fig. 4.2 we plot the ROC curve of this model. The curves confirm that our classifier model is not able to distinguish between truly random and pseudo random numbers sequences. A last comment can be made on a last approach we pursued: in this case, each feature in the PRNs sequences is defined as the sum of 50000 PRNs and, after a proper normalisation between 0 and 1, each of these has been used to form 50000 sequences 64 numbers long. In this way, we obtained a good distinction between the two classes (around 90% test accuracy) but we may have mistakenly inserted biases within the dataset. Anyway, the research is still proceeding in this direction.

Conclusions

In this work we have shown a working application of a tensor network machine learning algorithm. Thanks to the properties of tensor network machine learning algorithm and its connection of tensor network with quantum physics, we have been able to cast a machine learning classification task to a probability calculus on TNs quantum states. This bridge allowed us to calculate correlations and entropies on TNs quantum states, associated to the classification weights, making us able to understand the emerging patterns in terms of the correlation period inserted within the sequences. These new operations represent a huge step in terms of interpretability in machine learning and bring with them the possibility of opening doors on new statistical analysis on number sequences.

We have been able to reach a solid distinction between correlated sequences and pseudo (quantum) random numbers and these results can be used, among the many different possibilities, to investigate sequences containing longer correlation periods.

The comparison between quantum and pseudo random numbers is instead much harder and this is due to multiple factors. First of all, it is important to point out that the tensor network machine learning approach used in this work is very recent and the code implementation, made by the Quantum Theory Group of the University of Padua during the last year, is still under development. For example, we are still being limited to a bounded number of features: we cannot create sequences of as many number as we want. This is different from using an already well-known and tested model such as the AlexNet Convolutional Neural Network (CNN) or others. Moreover, better statistical gradient descent are known and can be implemented in our code to make it perform better.

Furthermore, Python's Marsenne Twister pseudo random number generator is tested to generate strong random numbers within the huge period, so it is in principle a difficult challenge to distinguish a very small set of them contained in \mathcal{P} from quantum random numbers. Remarkably though, during the distinction between correlated number sequences and pseudo random numbers in Sec. 4.1.4 (precisely in Fig. 4.3), we have observed how less-probable correlations contained in number sequences emerge when increasing the input size and so, an outlook for the future in the distinction quantum vs pseudo random number sequences would be to be able to use a larger amount of numbers. We could not use as many numbers as we wanted also because of the trade-off between the dataset size and the provided computing power.

Also a comparison with traditional deep neural networks should be done to see if they perform differently, as happened in [4], where the two different approaches result in similar outcomes in terms of prediction performances but the underlying information used by the two discriminators was inherently different.

Bibliography

- [1] M. Stipčević, “Quantum random number generators and their use in cryptography,” in *2011 Proceedings of the 34th International Convention MIPRO*, pp. 1474–1479, 2011.
- [2] N. Metropolis and S. Ulam, “The monte carlo method,” *Journal of the American Statistical Association*, vol. 44, no. 247, pp. 335–341, 1949. PMID: 18139350.
- [3] S. D. Sarma, D.-L. Deng, and L.-M. Duan, “Machine learning meets quantum physics,” *Physics Today*, vol. 72, pp. 48–54, 2019.
- [4] M. Trenti, L. Sestini, A. Gianelle, D. Zuliani, T. Felser, D. Lucchesi, and S. Montangero, “Quantum-inspired Machine Learning on high-energy physics data,” 4 2020.
- [5] D. Bzdok, M. Krzywinski, and N. Altman, “Machine learning: supervised methods,” *Nat Methods* 15, 5–6, 2018.
- [6] P. Silvi, F. Tschirsich, M. Gerster, J. Junemann, D. Jaschke, M. Rizzi, and S. Montangero, “The tensor networks anthology: Simulation techniques for many-body quantum lattice systems,” *arXiv: Quantum Physics*, p. 008, 2017.
- [7] S. Montangero, *Introduction to Tensor Network Methods, Numerical Simulations of Low-Dimensional Many-Body Quantum Systems*. Springer Nature Switzerland AG, 2018.
- [8] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.
- [9] E. Stoudenmire and D. Schwab, “Supervised learning with quantum-inspired tensor networks,” 05 2016.
- [10] A. Milsted, M. Ganahl, S. Leichenauer, J. Hidary, and G. Vidal, “TensorNetwork on TensorFlow: A Spin Chain Application Using Tree Tensor Networks,” *arXiv e-prints*, p. arXiv:1905.01331, May 2019.
- [11] M. Matsumoto and T. Nishimura, “Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator,” *ACM Trans. Model. Comput. Simul.*, vol. 8, p. 3–30, Jan. 1998.
- [12] M. Herrero-Collantes and J. C. Garcia-Escartin, “Quantum random number generators,” *Rev. Mod. Phys.*, vol. 89, p. 015004, Feb 2017.
- [13] X. Ma, X. Yuan, Z. Cao, B. Qi, and Z. Zhang, “Quantum random number generation,” *npj Quantum Information*, vol. 2, p. 16021, 2016.
- [14] M. Avesani, D. Marangon, G. Vallone, and P. Villoresi, “Source-device-independent heterodyne-based quantum random number generator at 17 gbps,” *Nature Communications*, vol. 9, 12 2018.
- [15] O. Hansha, “qRNG — Quantum Random Number Generator,” Jan. 2020.