Quantum Information and Computing

Prof. Simone Montangero, Dr. Giuseppe Magnifico

Faorlin, Marcomini, Piccinelli

Università degli Studi di Padova - Physics of Data

March 7, 2022

Why this project?

- To **learn** about and become more familiar with Tensor Networks methods.
- To continue into **thorny search** for distinction between pure randomness (intrinsic in the Nature of a physical process) and the result of complex mathematical algorithms.
- **Distinction** between classes of randomness has profound and ubiquitous implications.
- To work on a **final project** for the course in Quantum Information and Computing.

Outline

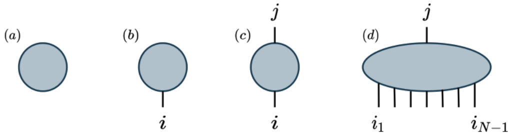
- Introduction
- Theoretical framework
    - Tensor Networks
    - Tensor Network Machine Learning
    - Tools: correlation map, overlap and entropies
- Simulations
- Results
- Comparison with standard ML algorithms
- Conclusion and outlook

Project goals

1. **Study** the Tree Tensor Network (TTN) representation of quantum states and their application to **machine-learning** problems.
2. Use the available TTN code to study and **improve the classification** of pseudo-random (with and without artificial correlations) and quantum-random sequences of numbers. **Analyse** in details the **classification errors** on the training set, the cross-validation set and the test set to understand possible problems of **underfitting** (high bias) and **overfitting** (high variance). Identify possible strategies to **improve the performances** (additional features, regularization of the cost function, etc.).
3. Consider **another learning algorithm**, such as Support Vector Machines or Neural Networks: by using available implementations (TensorFlow or similar), perform the classification on the same dataset, analysing the performances.
4. **Compare** the results obtained with the two algorithms.

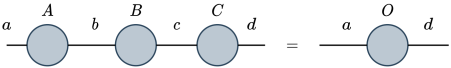
Tensor Networks

An N-rank tensor is an object with N indices, $T_{\alpha_1 \dots \alpha_N}$, and we can use the **tensor diagram formalism** to represent them.



A scalar, vector, matrix and a N-rank tensor.

A **link** defines a contraction of different indices.



$$\sum_{b,c} A_{a,b}\, B_{b,c}\, C_{c,d} = A_{a,b}\, B_{b,c}\, C_{c,d} = O_{a,d}$$

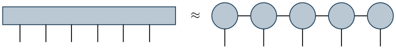
Tensor Networks - Applications

Gist: We efficiently represent a large tensor by approximating it via factorization to a contracted product of smaller lower-order tensors.

Some **benefits** are:

1. Exponential reduction in memory;
2. Exponential speed-up of computations (addition, product);
3. Theoretical insight and interpretation.

The Matrix Product State (MPS) is a **factorization** of a tensor with N indices into a chain-like product of three-index tensors. The MPS is one of the best understood and most successful tensor network architectures, for which many efficient algorithms have been developed.

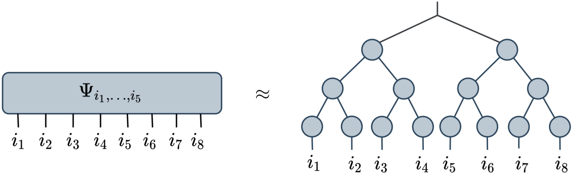


The main control parameter is the **bond dimension** χ, which interpolates between the **mean-field approximation** and the **exact representation** of the many body state. If a modest χ yields a good enough representation, we obtain a **massive compression**:

$$\mathcal{O}(d^N) \longrightarrow \mathcal{O}(Nd\chi^2)$$

Tensor Networks - Tree Tensor Network

However, MPS show some limitations when dealing with **periodic** systems [Silvi et al., 2019]: to address this issue we introduce a specific loop-free tensor network: the **binary Tree Tensor Network** (bTTN).


$\Psi_{i_1,\ldots,i_5}$
i_1 i_2 i_3 i_4 i_5 i_6 i_7 i_8
$\approx$
i_1 i_2 i_3 i_4 i_5 i_6 i_7 i_8

$$|\psi\rangle = \sum_{i_1,\ldots,i_8} \Psi_{i_1,\ldots,i_8} |i_1,\ldots,i_8\rangle \to \texttt{bTTN} \tag{1}$$

Like the MPS, this network is built on top of a one-dimensional lattice of **physical links** (with local dimension *d*) but treats open-boundary and periodic-boundary one-dimensional systems on equal footing. In **our work**, we use such architecture with 32 physical indices on which we clamp a **representation** of the number sequences.

Tensor Network Machine Learning

TTNs turn out to be also a very natural way to parametrize ML models:

1. **Rich theoretical understanding** of the properties of tensor networks (linear operations). This facilitates interpretability and generalization.
2. Training the model scales **linearly** in the training set size [Stoudenmire and Schwab, 2016].
3. **Adaptivity.** Dimensions of tensor indices internal to the network are modified during training to concentrate resources on the data most useful for learning.
4. **Hierarchy.** The shape of the tree is reminiscent of a hierarchical structure like the one of the brain, where information is processed via progressive complex abstract representations.

As firstly described in [Stoudenmire and Schwab, 2016], we are going to solve a **binary classification** problem where each piece of data $\boldsymbol{x}$ is mapped to each label ℓ through a linear decision function

$$f^{\ell}(\boldsymbol{x}) = W^{\ell} \cdot \phi(\boldsymbol{x}) = \left\langle W^{\ell} \middle| \phi(\boldsymbol{x}) \right\rangle$$

where W is the **optimized weight tensor** and $\phi(\cdot)$ is a multi-dimensional **feature map** that maps each piece of input data $\boldsymbol{x}$ to a **unentangled wave-function** of a N-body quantum system.

Tensor Network Machine Learning

The feature map does not depend on the label. When input values are numbers in $[0,1]$, the most common choice is the following **spin-map**:

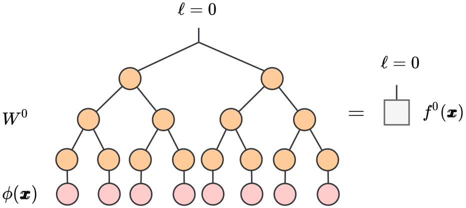
$$\phi^{s_j}(x_j) = \left[\cos\left(\frac{\pi}{2}x_j\right), \sin\left(\frac{\pi}{2}x_j\right)\right]. \tag{2}$$

Instead, we approximate W^ℓ as a bTTN. Since all links are normalized, upon contraction between W^ℓ and $\phi(\boldsymbol{x})$ the result $f^\ell(\boldsymbol{x})$ is vector of **unitary norm**, meaning that its components are the **amplitude probabilities** for $\phi(\boldsymbol{x})$ to belong to each class ℓ. As a result, we assign the sample $\boldsymbol{x}$ to the class with the highest value of $\left|f^\ell(\boldsymbol{x})\right|$.

In our case, the optimized weights tensor will play the role of two wave functions:

$$\left|W^\ell\right\rangle \equiv \left|\psi^\ell_{\mathrm{TTN}}\right\rangle.$$

Over them we compute **observables** and entropies of interest.



TTN optimization - Gauging routine

Loop-free tensor networks (LFTN), i.e. TN states whose network graph contains no cycles, can exploit **gauge transformations**.

Gauge transformations

The whole **freedom** we have in choosing the tensor elements to encode a specific ansatz state $|\Psi\rangle$ in a LFTN with given link dimensions is entirely determined by **link-local gauge transformations**.

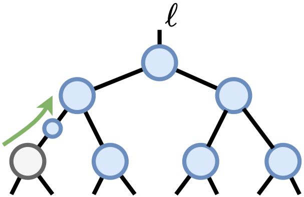
Smart gauging establishes rules on the network which can tremendously reduce the number of necessary contractions, thereby enhancing computational speed [Silvi et al., 2019].

We transform a bTTN into **central gauge**, so called because it is always defined with respect to a **single selected tensor** of the tree.

Both QR and the SVD decomposition are suited to carry out this task but the QR decomposition can be performed in roughly 2/3 of the time of the SVD, while exhibiting the same scaling of $\mathcal{O}(\chi^{3.8})$.

TTN optimization - Polar decomposition

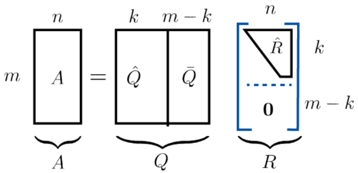
The procedure is carried out iteratively for each node of the TTN: we show the operations in diagrammatic form for the **top tensor** in the animation below.



Such decomposition is possible because every matrix can be decomposed in an **unitary** and a **non-unitary** part [Montangero, 2018],

$$A = QR$$

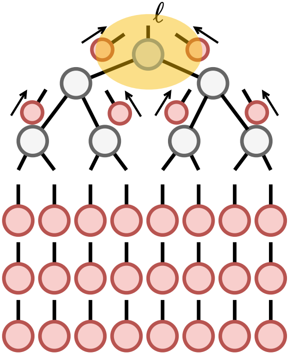
where Q is a unitary matrix and R a positive-semidefinite matrix.



TTN optimization

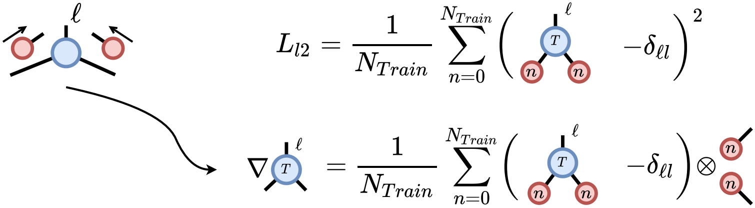
When optimizing each tensor we follow a **(stochastic) gradient descent**-like routine: we first tried with the **conjugate gradient method** but failed to retrieve previous results.

- All the contractions of the samples with the tree represent a vector that gets propagated via matrix-vector contraction towards the node that “retains all relevant information”.



TTN optimization - Top tensor update

- We define a cost function, e.g. the MSE loss L_{l2}. We update the value of the tensor following the (negative value of the) gradient.

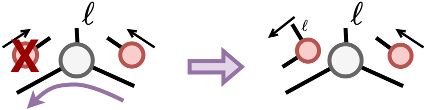


ℓ
$L_{l2} = \frac{1}{N_{Train}} \sum_{n=0}^{N_{Train}} \Big($
ℓ
T
n
n
$-\delta_{\ell l} \Big)^2$
∇
T
ℓ
$= \frac{1}{N_{Train}} \sum_{n=0}^{N_{Train}} \Big($
ℓ
T
n
n
$-\delta_{\ell l} \Big) \otimes$
n
n

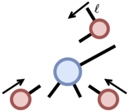
Weights are then updated according to the standard gradient descent rule. A tunable parameter allows to combine the gradient of the loss together with a L2 regularization term.

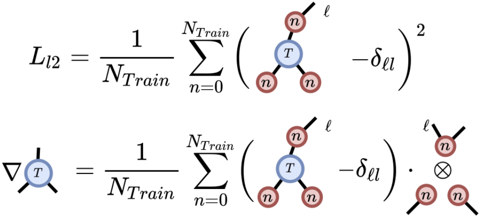
TTN optimization - General node update

- This steps are repeated for every tensor node. However, in this case the propagation follows slightly differently:



while for the update it holds:




$L_{l2} = \frac{1}{N_{Train}} \sum_{n=0}^{N_{Train}}$
n
T
ℓ
$-\delta_{\ell l}$
2
∇
T
$= \frac{1}{N_{Train}} \sum_{n=0}^{N_{Train}}$
$-\delta_{\ell l}$
$\cdot$
$\otimes$

Tools - Correlation map

As a special case of expectation value that can be computed from a TN, the correlation between two site i and j is defined as

$$C_{ij}^{\ell\ell'} = \left\langle \Psi_{\mathrm{TTN}}^{\ell} \middle| \mathcal{O}^i \mathcal{O}^j \middle| \Psi_{\mathrm{TTN}}^{\ell'} \right\rangle \tag{3}$$

In particular, the correlations of interest in the case under study, are between σ_z single site operators, i.e. the Pauli matrices.

Please notice that the wave functions associated to the weight tensor are in principle **not normalized**. As a result, we modified the code to solve this issue and obtain normalized correlation values:

$$C_{ij}^{\ell\ell'} = \frac{\left\langle \Psi_{\mathrm{TTN}}^{\ell} \middle| \sigma_z^i \sigma_z^j \middle| \Psi_{\mathrm{TTN}}^{\ell'} \right\rangle}{\sqrt{\left\langle \Psi_{\mathrm{TTN}}^{\ell} \middle| \Psi_{\mathrm{TTN}}^{\ell} \right\rangle \left\langle \Psi_{\mathrm{TTN}}^{\ell'} \middle| \Psi_{\mathrm{TTN}}^{\ell'} \right\rangle}} \tag{4}$$

In this way it is possible to compute also cross correlations for $\ell \neq \ell'$.

Tools - Entropies

Given a LFTN we can consider the **bi-partition of the network**, corresponding to the bi-partition of the physical system. Contracting the two sub-networks we can compute the **von Neumann entropy** between the two.

Given a density matrix ρ_A

$$\rho_A = \mathrm{Tr}_B \left|\Psi\right\rangle \left\langle\Psi\right| = \sum_\alpha p_\alpha \left|\Psi_\alpha^A\right\rangle \left\langle\Psi_\alpha^A\right| ;$$

The entropy $S(A)$ of sub-network A is defined as

$$S(A) = -\mathrm{Tr}(\rho_A \log \rho_A) = -\sum_\alpha p_\alpha \log p_\alpha \tag{5}$$

This entropy measures the amount of entanglement between the partition A and the rest B and it is known as **entanglement entropy**.

We can compute the entanglement entropy for all the possible bipartitions of the network, that is for every link connecting two sites (including physical sites).
Conceptually, this will represent the quantity of information contained in that sub-branch of the tree tensor network [Trenti, 2020].

A new spin map

While feeding train data to the tensor network, one must be careful about their representation in the feature space of dimension d. In particular, for numerical stability and optimal description such representation should satisfy [Stoudenmire and Schwab, 2016]:

$$\sum_s |\phi^s(x)|^2 = 1 \qquad \int_0^1 \bar{\phi}^s(x)\phi^{s'}(x)d_\mu(x) = \delta_{s,s'} \tag{6}$$

I.e., it must be normalized and have orthogonal components (as physical wave-functions). For $d = 2$ and $x_j \in [0,1]$ a common choice is:

$$\phi^{s_j}(x_j) = \left[\cos\left(\frac{\pi}{2}x_j\right), \sin\left(\frac{\pi}{2}x_j\right)\right] \tag{7}$$

We now present and motivate our proposal for a **new spin map**. PRNs are usually generated by modulo-N operations in the form:

$$x_j \equiv f\left(x_{<j}\right) \text{ mod N} = f\left(x_{<j}\right) - nN \text{ where } n = \lfloor f\left(x_{<j}, \ldots\right)/N \rfloor \tag{8}$$

where $f\left(x_{<j}\right)$ is a generic function of $x_{j-1}, \ldots x_0$ and for a LCG $f\left(x_{<j}\right) = ax_{j-1} + c$.

A new spin map

We aim to modify the map in Eq. 2, by **removing** the effect of the modulo operation, and **focusing more on the generation** function. Setting $f \equiv f\left(x_{<j}\right)$, we look for a solution in the form $\left[\cos\left(\alpha x_j\right), \sin\left(\alpha x_j\right)\right]$ such that:

$$\phi^0\left(x_j\right) \overset{!}{=} \phi^0\left(f\right) \implies \cos\left(\alpha x_j\right) = \cos\left(\alpha f - \alpha n N\right) \overset{!}{=} \cos\left(\alpha f\right) \implies \exists k \in \mathbb{N} : \alpha n N = 2k\pi$$

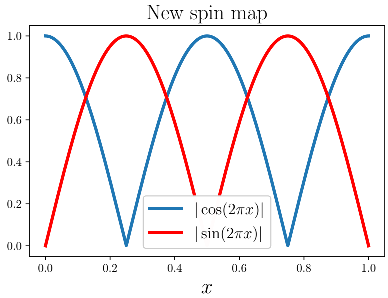
In Eq. 2 we had $\alpha = \pi/2$, and the problem might have no solution (e.g., odd n and N). Whereas, one may notice that for $\alpha = 2\pi$ the condition turns to $k = nN \in \mathbb{N}$, which is always satisfied. As a result, we propose the new map:

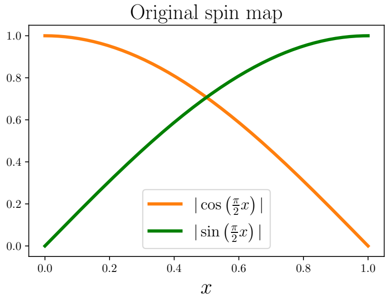
$$\phi^{s_j}\left(x_j\right) = \left[\left|\cos\left(2\pi x_j\right)\right|, \left|\sin\left(2\pi x_j\right)\right|\right] \tag{9}$$

where the modulus is introduced to keep the map components in $[0,1]$. It can be verified that such map satisfies the requirements of unitary norm and orthogonality of components.

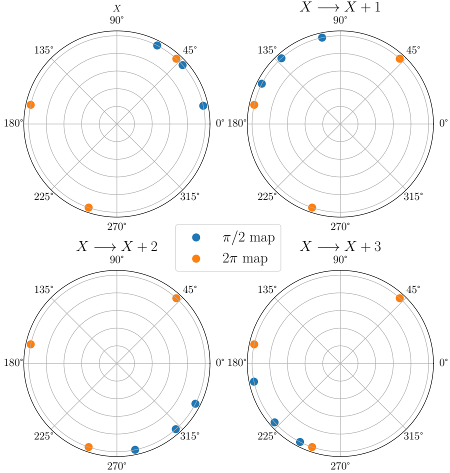
A new spin map - Spin map comparison

A visual comparison of the two maps: we **sacrifice bijectivity** in order to be more robust to periodic behavior, i.e. translations of the input.





A new spin map - Spin map comparison



Representation of $\theta \cdot \boldsymbol{X}$ for $\boldsymbol{X} = [0.1324, 0.4652, 0.6982]$: projections onto Cartesian axes represent features.

- $\theta = \pi/2$ in the old map
- $\theta = 2\pi$ in the new map

One can see that the new map is invariant for translations of integer numbers ($1, 2, 3$ displayed).

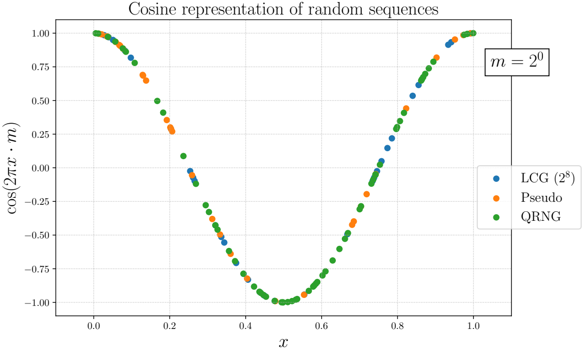
A new spin map - Into the code

Below the simple edit we performed **on the code** to test our proposal map. Further development could bring to permanent implementation among other feature maps already in use.

```
DO jj = 1, src
!  ->  new map: arg = 2*pi*array(ii,kk)/(max_val)
!  ->  old map: arg = pi*array(ii,kk)/(2*max_val)
arg = 2*pi*array(ii,kk)/(max_val)
tens%elem(jj) = sqrt(1.0*binomial(src-1,jj-1)) * &
!  ->  new map: ABS(COS(arg)**(src-jj)) * ABS(SIN(arg)**(jj-1))
!  ->  old map: (cos(arg)**(src-jj)) * (sin(arg)**(jj-1))
ABS(COS(arg)**(src-jj)) * ABS(SIN(arg)**(jj-1))
```

New spin map - Higher orders

A similar approach can be used to look for a specific periodicity in a sequence of data: in fact, by considering a generic map in the form $x \mapsto \cos(2\pi x \cdot m)$, if data are generated by an LCG of period m they will be **mapped into value 1 regardless of their distribution**. In principle, a **sequence of maps** like this could be used instead of a spin map to feed a TN.



Simulations hardware and resources

All the simulations that we will show have been carried out on a cluster made available by CloudVeneto, based on 500 gigabytes of RAM, NVIDIA RTX GPU and 100 physical cores (we used 15 of them).

Cut-off: some observations

A standard cut-off of $\chi = 25$ has been selected for most simulations. Over 160k train samples of size 32, the expected computational time for one epoch is ~ 70 min.
The computational resources at our disposal seem not to bear the computational load of simulations with $\chi \geq 70$.

Results - Roadmap

We report below an outline of our journey and the most important steps.

1. Test of trivial models (pseudo-random vs sequences of identical numbers)
2. Test of strongly correlated models (in-sequence repetition of values)
    - New spin map proposal
3. Test on QRNG and Linear Congruential Generators (LCGs)
    - Short period: $\leq 2^5$
    - Mid period: $\leq 2^{20}$
    - Large period: up to $\sim 2^{30}$
4. Test on QRNG and pseudo-random (Python built-in)
5. Test on QRNG and mixture of LCGs
6. Test on accuracy, scaling the bond dimension
7. Test on accuracy in time and frequency domain
8. Study of the overlap of TTN quantum states
9. Development of Fully Connected Neural Networks and repetition of study cases

The repetition method

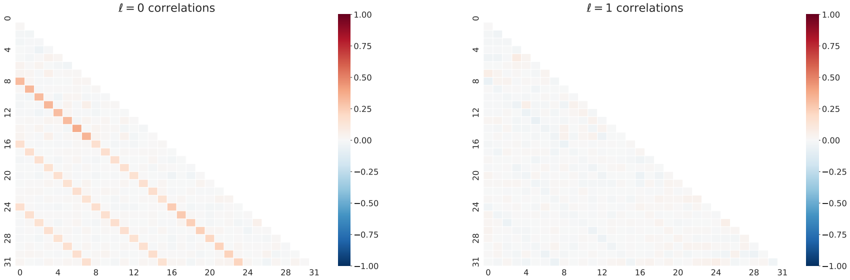
Keeping in mind that LCG generate numbers with a more or less long period, our sequences will be composed of **repeated numbers** with different periods $\mathcal{P}$. For example, for $\mathcal{P} = 4$:

$$0.1384, 0.8543, 0.223, 0.476, 0.1384, 0.8543, 0.223, 0.476, 0.1384, 0.8543, 0.223, 0.476, \ldots$$

The other type of sequences will be generated with the well-known general-purpose PRNG *Marsenne Twister* ($\mathcal{P} = 2^{19937} - 1$) or with a QRNG.

$\mathcal{P} = 8$

We now present some results in terms of **correlations** computed on the trained tensor networks[1]. Observing the rightmost graph it is possible to observe the emergence of the **same (anti-correlated) pattern** in **blue**.

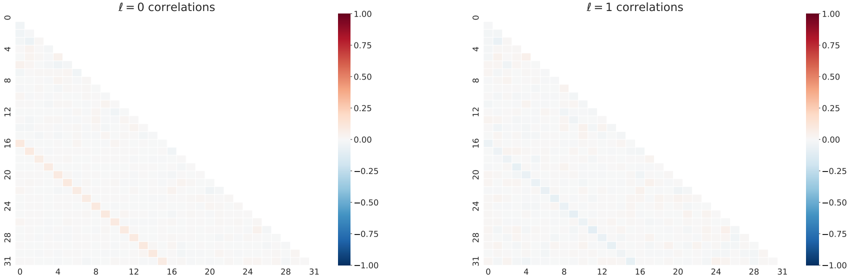


N_{tot}	$\ell = 0$	$\ell = 1$	N_{pass}	χ	$\%_{train}$	$\%_{test}$
2000	$\mathcal{P} = 4$	MT	12	25	$> 95\%$	$> 95\%$

[1] As shown in Eq. (4).

$\mathcal{P} = 16$

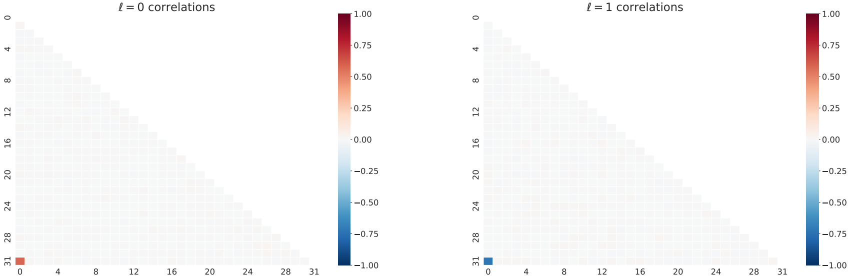
We increase the period length.



N_{tot}	$\ell = 0$	$\ell = 1$	N_{pass}	χ	$\%_{train}$	$\%_{test}$
2000	$\mathcal{P} = 16$	MT	12	25	$> 95\%$	$> 95\%$

$\mathcal{P} = 31$

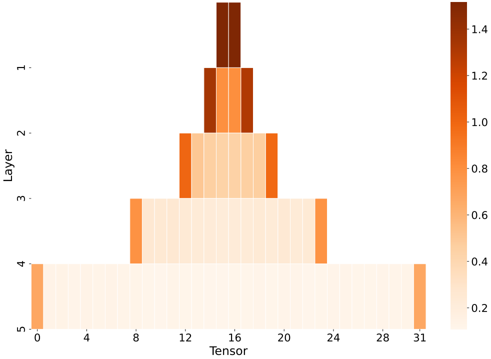
We increase the period length and the dataset size. It is curious to notice how for this case **the network seeks anticorrelations** between the first and last site, rather than absence of correlation, in order to distinguish the classes.



N_{tot}	$\ell = 0$	$\ell = 1$	N_{pass}	χ	$\%_{train}$	$\%_{test}$
200000	$\mathcal{P} = 31$	MT	12	25	$> 95\%$	$> 95\%$

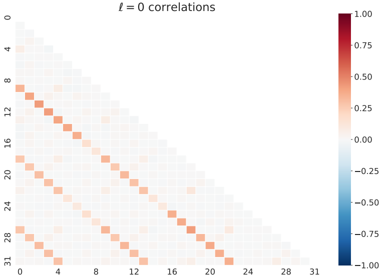
$\mathcal{P} = 31$

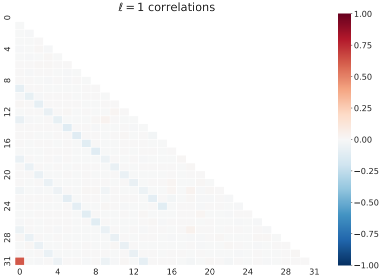
- In most of cases, entropy plots are far from immediate and informative;
- Here we see which are the tensors carrying the **largest amount of information** for the final classification.



$\mathcal{P} = 9$ vs $\mathcal{P} = 31$

On the right graph, the network classifies $\ell = 1$ both by exploiting a positive correlation (red-dots) and the inverse of the information used in $\ell = 0$. Exchanging the labels, we obtained a mirrored result, meaning that the networks is trained **symmetrically**.

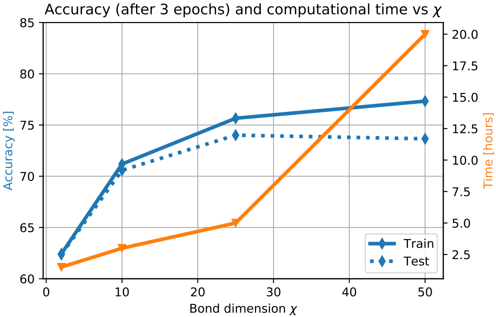




N_{tot}	$\ell = 0$	$\ell = 1$	N_{pass}	χ	$\%_{train}$	$\%_{test}$
200000	$\mathcal{P} = 9$	$\mathcal{P} = 31$	12	25	$> 95\%$	$> 95\%$

$\mathcal{P} \in [1, \ldots, 32]$ (changing χ)

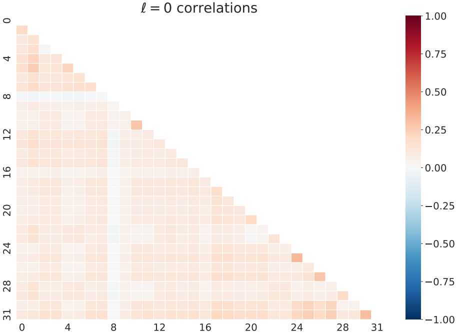
- We have chosen to study a case where the test accuracy is **mediocre**;
- The choice of $\chi = 25$ is now motivated.



N_{tot}	$\ell = 0$	$\ell = 1$	N_{pass}	χ
186000	$\mathcal{P} \in [1, \ldots, 32]$	QRN	12	$2, 10, 25, 50$

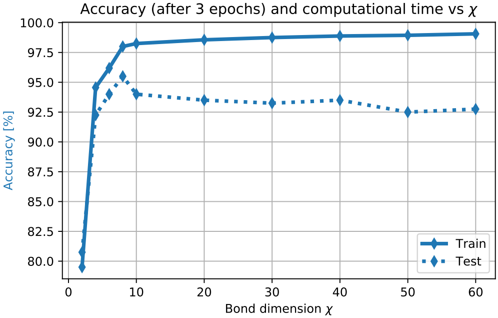
$\mathcal{P} \in [1, \ldots, 32]$ (changing χ)

The animation below shows the correlation maps for $\chi = 2$, $\chi = 10$, $\chi = 25$, $\chi = 50$.



$\mathcal{P} = 4$ (changing χ)

- A **small** bond dimension is sufficient to maximize test accuracy;
- For a larger bond dimension we can observe **overfitting**.



N_{tot}	$\ell = 0$	$\ell = 1$	N_{pass}	χ
2000	$\mathcal{P} = 4$	QRN	12	2, 4, 6, 8, 10, 20, 30, 40, 50, 60

$\mathcal{P} = 4$ (changing χ)

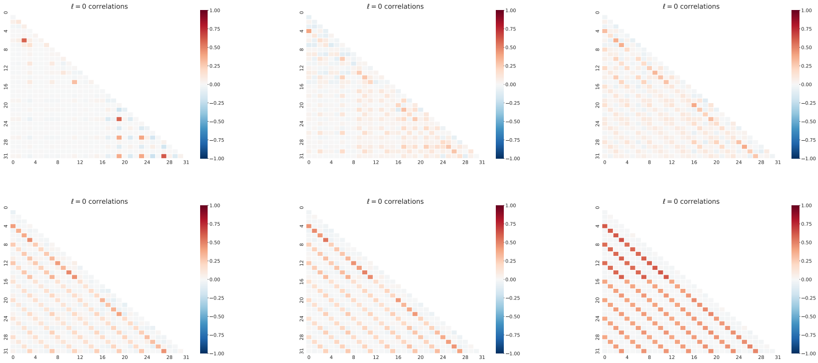


Figure: From top left: $\chi = 2$, $\chi = 4$, $\chi = 6$, $\chi = 8$, $\chi = 10$, $\chi = 50$.

Discriminative power: overlap

The contraction $\langle \Psi^0_{\mathrm{TTN}} | \Psi^1_{\mathrm{TTN}} \rangle$ defines the **overlap** between trained tensors over different labels. In an ideal case, we would expect such states to be **orthogonal**: $\left\langle \Psi^{\ell}_{\mathrm{TTN}} \middle| \Psi^{\ell'}_{\mathrm{TTN}} \right\rangle = \delta_{\ell,\ell'}$. Indeed, this has been observed for the most simple cases such as QRN vs. binary sequences ($\langle \Psi^0 | \Psi^1 \rangle = \mathcal{O}\left(10^{-5}\right)$, test accuracy of 100%).
We thus expect a correlation between the **classification ability** of the TTN and the **overlap** between the two network states (e.g., for $\langle \Psi^0_{\mathrm{TTN}} | \Psi^1_{\mathrm{TTN}} \rangle = \pm 1$ we expect a dumb classifier).

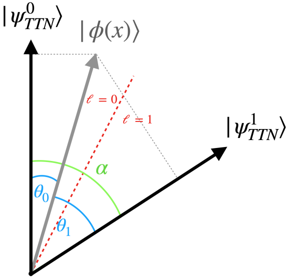
However, we found cases with high test accuracy ($> 95\%$) for which $\langle \Psi^0_{\mathrm{TTN}} | \Psi^1_{\mathrm{TTN}} \rangle \approx -0.8$.

The output of the TTN classification is:

$$\begin{aligned} f^{\ell}(\boldsymbol{x}) &= \left[\left\langle \psi^0_{\mathrm{TTN}} \middle| \phi\left(\boldsymbol{x}\right)\right\rangle, \left\langle \psi^1_{\mathrm{TTN}} \middle| \phi\left(\boldsymbol{x}\right)\right\rangle\right] \\ &\propto \left[\cos\left(\theta_0\right), \cos\left(\theta_1\right)\right] \end{aligned}$$

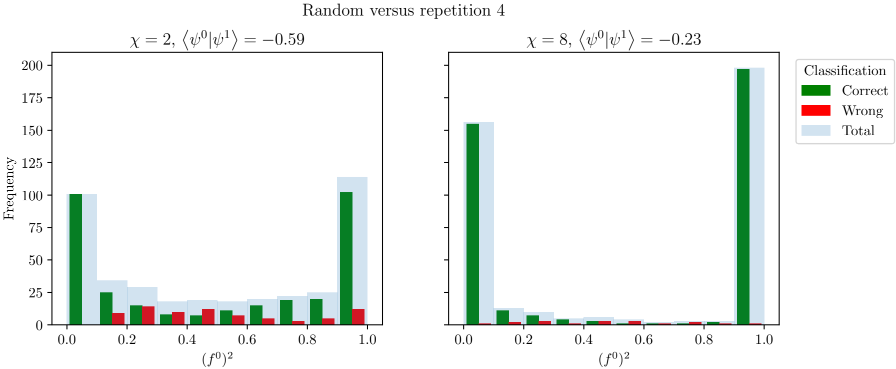
$$\text{prediction} = \max_{\ell} \left| f^{\ell}(\boldsymbol{x}) \right|$$

As a result, an accuracy of 100% can be achieved despite the angle α between network states being $\neq \pi/2$.



Discriminative power: overlap

We **proved** the previous assumptions for the previous $\mathcal{P} = 4$ classification problem, varying χ. In **both cases** below, a test accuracy of $\sim 95\%$ is achieved.



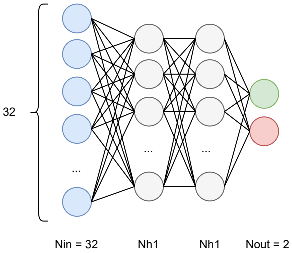
When the overlap is larger, the network is **less confident in its guesses**, even though these are generally correct.

It is worth mentioning that in all our simulations the overlap between TN states is negative.

Fully Connected Network (FCN)

To compare the performances we decided to use a Fully Connected Architecture (**FCN**).

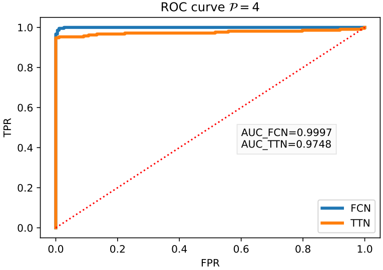
We build, using `PyTorch`, a FCN with the architecture reported on the right. Using `Optuna`, we perform a random grid search (RGS) over **500 models**, in order to find the most suitable one able to accomplish a specific task. In particular, the search is performed over the hyper-parameters reported in the table below.
The optimizer chosen is **Adam**, with $L2$ regularization.
The output layer is activated by a **Softmax function** in order to have a normalized probability distribution.

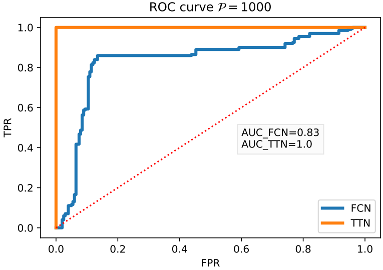


```
params = {"module__act_func"                  : [nn.ReLU(), nn.LeakyReLU()],
          "module__Nh1"                       : [24, 16, 8, 4],
          "module__Nh2"                       : [24, 16, 8, 4],
          "batch_size"                        : [64,128],
          "lr"                                : loguniform.rvs(1e-4, 1e-2, size = 20),
          'optimizer__weight_decay'           : loguniform.rvs(1e-5, 1e-2, size = 20) }
```

Comparison TTN and FCN

We report the Receiver Operating Characteristic (**ROC**) curves to evaluate the performances of the two classifiers (TTN and FCN).



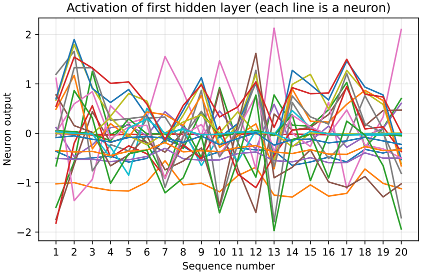


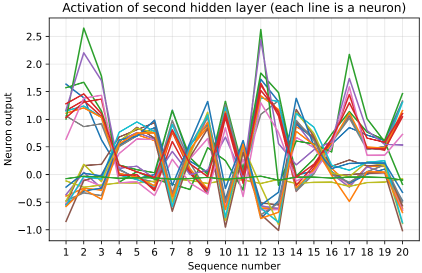
With $\mathcal{P} = 4$ the FCN performs slightly better than the TTN. In the next slide we are going to see some attempts at explainability.

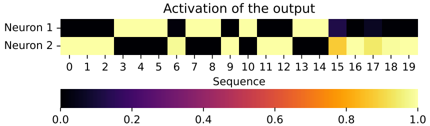
Here we see that for longer periods the TTN performs better than the FCN.

Comparison TTN and FCN - Receptive fields

We study, for $\mathcal{P} = 4$, the **receptive fields** of the network layers to understand if the activation of neurons is in some way **related** to the sequence under examination. It is very difficult to derive meaningful information.


Activation of first hidden layer (each line is a neuron)

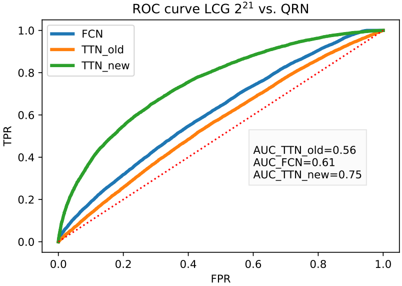

Activation of second hidden layer (each line is a neuron)


Activation of the output

Comparison TTN and FCN

We test the classification ability of the two models and two **different** spin maps:

- The FCN has the **same architecture** as above and the best set of hyper-parameters is found after a RGS over **500 models**.
- A TTN with **our map** proposal achieves the best performance;
- If we change the seed for the generation of $\ell = 0$ and we test the tree we obtain a dumb classifier (**lack of generalization**).



N_{tot}	$\ell = 0$	$\ell = 1$	N_{pass}	χ	$\%_{train}$	$\%_{test}$[2]
200000	$\mathcal{P} = 2^{21}$	MT	12	25	$\sim 70\%$	$\sim 70\%$

[2] on the best model TTN_new (green curve)

Discrete cosine transform

We try to exploit one of the main weaknesses of PRNGs, namely the **periodicity**: we perform a spectral analysis via **discrete cosine transform** (DCT), a common technique in the area of digital processing for the purposes of pattern recognition.

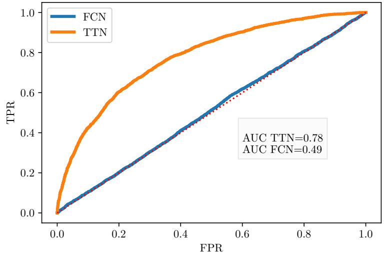
We generate an LCG with period 2^{20} over 2^{15} sequences: for a sequence length of $N = 2^5$ no repetitions are present.

Our analysis has been carried out training the TTN for:

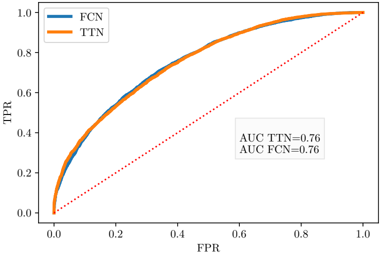
1. LCG vs. QRN as a benchmark for subsequent tests (vanilla): we obtain a slight overfit for an accuracy over the test set of $\sim 70\%$.
2. DCT of the LCG vs. QRN, normalizing each dataset to its maximum. We obtain a discrete distinguisher, with a test accuracy of $\sim 68\%$.

Both trials have been compared with a FC model, performing again a randomized hyperparameters search.
Interestingly, the NN reaches similar result for the case of point 2, while it performs significantly worse for point 1: this is highly significant, since the TN sees the period only once.

Discrete cosine transform



ROC curve vanilla LCG 2^{20} vs. QRN



ROC curve DCT of LCG 2^{20} vs. QRN

The information on point 2 cannot be know in a real-case scenario: we normalize the sequences either locally (rescale in $[0,1]$ to each sequence maximum) or globally (rescale in $[0,1]$ to the maximum across sequences): we obtain a dumb distinguisher for both cases.

Keeping in mind that the DCT concentrates information in the **low frequencies** we subtract the mean to each sequence, keeping the local normalization: again, the TN overfits the data.

Other takeaways

- We investigate the impact of **L2 regularization** for different problems by setting the strength coefficient to $0, 0.1, 0.01$ and 0.001. No noticeable differences are found.
- We change the **number of epochs** according to the task: generally, after 3 epochs results are stable enough, for $N_{pass} = 12, \chi = 25$. Our analyses are carried out over a standard of 5 epochs.
- We test the boundaries of our implementation by distinguishing with QRNG and LCGs of **large period**: positive results can be found for periods of 2^{20} (all train and test sequences different from each other) and 2^{21} (each sequence is repeated on average 1.5 times). While this is promising, performing classification on LCG data generated with different seed with pre-trained TTN returns 50% error rate. Therefore, we conclude that the model lacks of generalization. Enlarging the period to 2^{22}, the TTN cannot distinguish QRNG and LCG.

Conclusion

- We focused on the **functioning and the potential of TTNs**, exploring a plethora of scenarios in growing complexity and proposing both problems and solutions of interest.
- We have been able to setup and work with a machine on the cloud, automating the execution of our code via bash and Python scripts. We also adapted the TTN code in Fortran, readjusting it from a model to classify the Fashion MNIST to our case and fixing some minor bugs.
- We observed how TNs **can detect** correlations **beyond the dimension of input sequences**, with better performances with respect to NNs. However, the lack of generalization suggest us that TN might be learning actual numbers (overfitting) and not the relations between them.

  Still, the ability of this networks to memorize sequences proposed as input only once is impressive.
- We set the bases for an ulterior development of this architecture and further investigations of the problems we studied.

Outlook

Possible future work could include the following proposals:

- Introduce **higher dimensional feature maps** and compare results with the ones of simple data pre-processing.
- Try more **advanced classic machine learning** models. For instance, in a CNN we may observe that the receptive fields in each hidden layer recall the periodicity of the sequences.
- Work more on **predictability**. Following [Feng and Hao, 2020], we could study how a TN performs in predicting **the next outcome** in a sequence of numbers.

References

Feng, Y. and Hao, L. (2020).
Testing randomness using artificial neural network.
IEEE Access, 8:163685–163693.

Montangero, S. (2018).
Introduction to tensor network methods numerical simulations of low-dimensional many-body quantum systems.

Silvi, P., Tschirsich, F., Gerster, M., Jünemann, J., Jaschke, D., Rizzi, M., and Montangero, S. (2019).
The Tensor Networks Anthology: Simulation techniques for many-body quantum lattice systems.
SciPost Phys. Lect. Notes, page 8.

Stoudenmire, E. and Schwab, D. J. (2016).
Supervised learning with tensor networks.
Advances in Neural Information Processing Systems, 29.

Trenti, M. (2020).
Tensor Networks Machine Learning algorithms for event recognition in High Energy Physics.
Master thesis, Università degli Studi di Padova.