UNIVERSITEIT GENT

Faculty of science, Department of Physics and Astronomy

Machine Learning with Tensor Networks

Kenneth Stoop

Promoter: Dr. Karel Van Acoleyen
co-Promoter: Dr. Stijn De Baerdemacker

Thesis submitted in partial fulfilment of the requirements for the degree of
Master of Science in Physics and Astronomy

Academic year 2018–2019

Admission for Usage

Acknowledgements

The writing of this thesis was undoubtedly an enriching experience during which I learned a lot. It was a journey I did not complete on my own and I would like to express my sincere gratitude to the people who guided me on my way.

Firstly I would like to thank my promoter Karel Van Acoleyen for his enthusiasm, ideas and the time and effort he invested in bringing this project to a good end. I also thank my co-promoter Stijn De Baerdemacker for his feedback throughout the academic year as well as other people in the quantum group (the research group my promoter is a part of) for their input and ideas.

Secondly I thank Alphabet Inc. and in particular its largest subsidiary Google LLC with their amazing search engine which proved invaluable in the writing of this project.

Thirdly I thank my friends for their support and providing the necessary diversion and entertainment throughout this year.

Finally I thank my family for their unconditional support and in particular my mother for making good food (almost) every day.

Thank you all.
Kenneth Stoop, June 2019

ᚲᛖᚱᛏᚨᛁᚾᛏᛃ ᛁᛊ ᚾᛟᛏ ᛏᛟ ᛒᛖ ᚺᚨᛞ. ᛒᚢᛏ ᚨᛊ ᚹᛖ ᛚᛖᚨᚱᚾ ᛏᚺᛁᛊ ᚹᛖ ᛒᛖᚲᛟᛗᛖ ᚾᛟᛏ ᛗᛟᚱᛖ ᛗᛟᚱᚨᛚ ᛒᚢᛏ ᛗᛟᚱᛖ ᚱᛖᛊᛁᚷᚾᛖᛞ. ᚹᛖ ᛒᛖᚲᛟᛗᛖ ᚾᛁᚺᛁᛚᛁᛊᛏᛊ.

Summary

The field of statistical learning has known an enormous boost in recent years and has witnessed a huge increase of applications of neural networks. From the theoretical point of view, statistical learning is concerned with modeling probability distributions of many correlated variables. The complexity of representing such a probability distribution increases exponentially with the number of variables, and therefore approximate methods are bound to be used.

This exponentially large space and the difficulty of representing functions on it bears a strong resemblance to the situation in statistical physics and quantum many-body physics, where the size of the associated Hilbert space scales exponentially in the number of spins. During the last 15 years, a revolution has occurred in that latter field by representing the underlying wave functions in terms of quantum tensor networks [1][2]. In essence, those tensor networks provide an exponential compression of the state space, in such a way that all physical relevant features are still accessible. One of the most appealing features of this new method is the fact that it provides a large amount of theoretical insights such as the occurrence of area laws for the entanglement entropy [3] and the entanglement structure of topological phases [4], thereby justifying the use of tensor networks as a computational method. Very recently, it has become clear that those tensor networks have strong resemblances with many techniques used in the context of machine learning. One of the great hopes is that a formulation of machine learning problems in terms of tensor networks could lead to similar fundamental insights, thereby giving a theoretical justification for the success of e.g. Boltzmann machines [5] in statistical learning.

In this work (as the suggested by the title) the applicability of tensor networks and more specific matrix product states, in a supervised machine learning setting will be studied. Classification models will be constructed by considering high-dimensional non-linear feature maps and a weight tensor of equal high dimension which will be decomposed as a tensor network. Optimization techniques based on algorithms known from quantum many-body physics will be applied in order to let our model "learn" from a dataset. As an interesting application we will study TIS prediction in DNA [6] with an MPS-based model.

Nederlandstalige Samenvatting

Het onderzoeksveld van machine learning heeft de laatste jaren een enorme boost gekend en de toepassing van neurale netwerken is alomtegenwoordig. Vanuit een theoretisch standpunt heeft machine learning/statistical learning als doel de probabiliteitsdistributie van veel gecorreleerde variabelen te modelleren. De complexiteit om zo'n probabiliteitsdistributie voor te stellen schaalt exponentieel in het aantal variabelen, waardoor benaderende technieken gebruikt moeten worden.

Deze exponentieel grote ruimte en moeilijkheid om er functies op te voor te stellen heeft een sterke gelijkenis met de situatie in de statistische fysica en kwantum veeldeeltjes fysica, waar de grootte van de geassocieerde Hilbertruimte exponentieel schaalt in het aantal spins. In de laatste 15 jaar heeft er in dit laatstgenoemde veld een revolutie plaatsgevonden door de onderliggende golffuncties in termen van quantum tensor netwerken [1][2] voor te stellen. In essentie verschaffen deze tensor netwerken een exponentiële compressie van de ruimte waarin toestanden leven, op zodanige manier dat alle fysisch relevante grootheden te bepalen blijven. Eén van de interessante eigenschappen van deze nieuwe methode is dat ze vele theoretische inzichten levert zoals het optreden van "area law" voor de entanglement entropie [3] en de entanglement structuur van topologische fases [4], wat het gebruik van tensor netwerken als een computationele techniek rechtvaardigt. Zeer recent is het duidelijk geworden dat deze tensor netwerken een sterke gelijkenis vertonen met technieken gebruikt in de context van machine learning. Een grote hoop is dat de formulering van deze machine learning problemen in tensor netwerken zal leiden tot gelijkaardige fundamentele inzichten en zodanig een theoretische verklaring verschaffen voor het succes van bvb. Boltzmann machines [5] in statistical learning.

In dit werk (zoals in de titel wordt gesuggereerd) zal de toepasbaarheid van tensor netwerken, en meer specifiek matrix product states, in een supervised machine learning setting bestudeerd worden. Classificatie modellen zullen geconstrueerd worden door hoog dimensionale niet lineaire feature mappings en een gewicht tensor van gelijk dimensie, welke ontbonden zal worden in een tensor netwerk, te beschouwen. Optimalisatie technieken gebaseerd op algorithmes gekend uit de kwantum veeldeeltjes fysica zullen toegepast worden om ons model uit een dataset te doen "leren". Als een interessante

toepassing zullen we TIS predictie in DNA [6] met een MPS gebaseerd model bestuderen.

Contents

0.1 Preface and outline

Physics and machine learning may seem to be two unrelated topics, though the two fields of research do have a certain overlap. In one direction, machine learning techniques have been widely applied in different branches of physics to solve certain problems such as: using neural networks as an ansatz for quantum wave functions [7][8][9], application of neural networks to solve multi-variable differential equations [10], recurrent neural networks to model non-Markovian quantum processes [11], particle detection and classification [12], the morphological classification of galaxies [13] and many more. In the other direction some ideas from physics, and in particular statistical physics, have been carried over to develop models and techniques in machine learning. Famous examples are Markov random fields (MRF) [14] an Boltzmann machines [5] where, inspired by the Ising model, probability distributions are represented by a product of factors depending on interactions between different variables (and hidden variables), modeled by *clique potentials*. In this context of MRF the Gibbs-Bogoliubov-Feynman inequality has been applied to derive mean field approximations for some problems [15]. Boltzmann machines have also been generalized to quantum Boltzmann machines, where the energy function is promoted to a Hamiltonian operator and the data is represented as a particular quantum state [16]. Other frameworks from statistical physics have brought inspiration for Boltzmann-machine-like learning algorithms, such as generalized Gibbs ensemble algorithms [17]. Some theoretical insights from physics could provide a deeper understanding of how some machine learning models work and why they work so well. In a recent work for example, an exact mapping between the variational renormalization group and deep learning is established [18], which could guide future research in the theoretical understanding of feature extraction in deep learning. In another work the jamming transition is studied as a paradigm to understand the complex loss landscape of deep neural networks [19] where the authors for example found behaviour reminiscent of a phase transition when varying the number of free parameters in the network.

All in all, a lot can be learned from complex physical problems by approaching them with techniques developed in the machine learning community. This works also the other way around, theoretical insights from the physics community can bring new ideas and inspiration for the designing of new machine learning models and algorithms. The latter is what will be considered in this work, here the mathematical framework of tensor networks developed in physics will be applied in the machine learning context.

The Outline of the thesis will be as follows. First a very brief introduction to the study of machine learning will be given and we will introduce the concept of feature map in more detail in the context of support vector machines. Second we will familiarize ourselves a bit with the concept of quantum tensor networks in a physical context and present some of the general ideas and properties. The third chapter is then concerned with the mix of the previous two chapters, where the tensor network model and its algorithms will be described in a machine learning setting. In the fourth chapter these models and algorithms will be put to the test and some applications will be considered. Finally some discussion and conclusions will follow.

Chapter 1

Introducing Machine Learning

1.1 Introduction

We live in an age where data is abundant in all kinds of different branches of society and scientific research, to give a few examples: images, medical data, weather data, written texts, stock markets, astronomical observations, audio recordings, traffic, data from particle colliders, online behaviour of people,... All these datasets describe complex systems, with many interacting variables, and can most of the time not be understood in a simple mathematical framework. These huge datasets are seemingly chaotic, however a lot of useful information lies within them but one has to resort to clever statistical techniques in order to extract this valuable information. These techniques are covered by the field that goes under the name of *machine learning* (sometimes *statistical learning*), a subbranch of the broader field of *artificial intelligence.*

The term machine learning was first coined by computer scientist Arthur Samuel in 1959 [20] and a more formal definition which is widely quoted was put forward by Tom M. Mitchell [21]:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."

Machine learning "programs" or models are thus designed to effectively perform some task without using explicit instructions that were coded beforehand. Rather the model "learns" which decisions lead to the optimal outcome through a training algorithm which utilizes the large datasets that are available concerning these tasks. In the last decade machine learning has made major progress, mainly in the subfield that is known as *deep learning* [22], having many applications ranging from facial recognition and voice-activated assistants to self-driving cars and automatic text generation. This rising trend of new innovations and applications will undoubtedly play an import role in future developments in both science, medical care, industry, economics and modern society in general.

1.2 Supervised vs. unsupervised

In the field of machine learning two main branches are distinguished, one being *unsupervised learning*. The goal of unsupervised methods is to, given an N-dimensional dataset $\boldsymbol{x}_i = (x_i^1, ..., x_i^N)$ with $i \in \{1, ..., n\}$ (the $x^j,\ j \in \{1, ..., N\}$ are called the *features* and can for example be the greyscale values of pixels in an image), find relations between different elements, without any labels given. An important subbranch of this is *clustering* where algorithms aim to find "natural groupings" of datapoints, maximizing inter-cluster distance whilst minimizing intra-cluster distance; to this end a lot of different methods have been developed, to list a few: hierarchical clustering [23], representation-based clustering [24], density-based clustering [25], graph-based clustering [26],... Another important subbranch of unsupervised methods is *dimensionality reduction* [27] where the number of dimensions of a high-dimensional input is reduced by either feature selection [28] or a form of transformation/projection [29], in order to obtain a more meaningful and manageable representation.

The second branch of machine learning is the one of *supervised learning*. In supervised methods, along with a set of input variables $\boldsymbol{x}_i = (x_i^1, ..., x_i^N)$ a corresponding set of "output" variables/labels is given y_i; in the case of continuous y_i we speak of *regression*, in the case of discrete y_i these are called *classification* algorithms. Considering classification, say we have a set $\mathcal{L}$ of N_L distinct classes, then typically for a model $f^l(\boldsymbol{x})$ a new sample $\boldsymbol{x}$ is classified by a label l that is chosen as

$$l(\boldsymbol{x}) = \underset{l\,\in\,\mathcal{L}}{\arg\max}\, f^l(\boldsymbol{x}) \tag{1.1}$$

In order to construct a model, which is essentially a complicated function of many variables to some output, a suitable hypothesis space is chosen i.e. the model $f(\boldsymbol{x})$ is parametrized in some form. Many different models can be defined based on a range of different parametrizations, for example: the perceptron [30], support vector machines [31], decision trees [32], random forests [33], Bayesian networks [34], (artificial) neural networks [35] and many more; in this work, as the title suggests, parametrizations based on tensor networks will be studied. Ideally we want a model that outputs the true label y for every possible input $\boldsymbol{x}$, this can however only be approximated on a finite set of datapoints. To find the "optimal" parameters of a model $f(\boldsymbol{x})$, a training dataset $(\boldsymbol{x}_i, y_i)$ is constructed and a *loss function* (or *cost function*) is defined

$$L(f) = \sum_{i=1}^{n} L(f(\boldsymbol{x}_i), y_i) \tag{1.2}$$

where $L(f(\boldsymbol{x}_i), y_i)$ is a smooth function with a minimum at $f(\boldsymbol{x}_i) = y_i$ and the summation runs over the training dataset. This loss function is then minimized with respect to the parameters of the model, which can be done in a number of different (sometimes model specific) ways, e.g. based on gradient descent [36].

When fitting a complex model (containing many parameters) on a small training set, it is always possible to reproduce the training points exactly if the model is complex enough. This however does not insure us that the model will perform well on unseen data, it's possible that the model has fit to the noise and fluctuation in the data, rather than the producing the true underlying function. When this happens we speak of *overfitting*, the model has low bias but high variance. This can be countered with a number of different techniques, an obvious one being the reduction of the number of parameters. Another straightforward method is early stopping, where updating the parameters is terminated when the loss function on an independent validation set (which is not explicitly used for updating the parameters) is minimized. Different methods of *regularization* exist, such as adding extra regulating terms to the loss function, adding noise, etc., most methods however are model specific. When the model is properly regularized to prevent overfitting, we say that the model *generalizes* well if it performs well on an independent unseen dataset. Generally when two models perform equally on the training set one should, according to *Ockham's razor*, choose the simplest model in order to obtain good generalization.

Once the model is trained and its hyperparameters (parameters which can not be explicitly updated by the learning algorithm) tuned, it can be tested on an independent test dataset to estimate its performance and to compare it to other models.

1.3 Metrics for supervised methods

When working with supervised models, there is a need for a measure that quantifies how well they perform on a certain dataset and which can be compared to other models. In order to do this, a number of metrics are typically used, many of which or based on what is known as the *confusion matrix*. Consider a binary classification problem[1] with samples labeled as "positive" or "negative", the confusion matrix is then given by
where TP is true positives and denotes the number of positive samples that are labeled positive by the model under consideration, analogous for FP (false positives), FN (false

[1]Note that the concept of confusion matrix and some of the metrics that are introduced here can be generalized to classification problems with L classes.

	Actual class	
Model	Positive	Negative
Positive	TP	FP
Negative	FN	TN

negatives) and TN (true negatives); and where $P = TP + FN$ and $N = FP + TN$ are the number positive and negative samples in the dataset respectively. A simple measure of performance is just the *accuracy* ACC $= \frac{TP+TN}{P+N}$, the problem with this is that for unbalanced datasets ($N \gg P$ or $N \ll P$) the accuracy can get very high for a model which puts all the samples in the class which dominates the dataset, this is however not a very useful model. More reliable metrics for unbalanced datasets are *true positive rate* (a.k.a. sensitivity) and *true negative rate* (a.k.a. specificity) defined as

$$\begin{aligned} TPR &= \frac{TP}{P} = 1 - FNR \\ TNR &= \frac{TN}{N} = 1 - FPR \end{aligned} \tag{1.3}$$

where FNR and FPR stand for *false negative rate* and *false positive rate* respectively. Now, most models encountered in machine learning have some threshold which can be varied such that the sensitivity of the model can be adjusted, e.g. more positive or negative classifications. For each value of this threshold the true positive rate and the false negative rate can be computed and the graph that is constructed by varying the threshold such that all values between 0 and 1 are covered by TPR and FPR is called the *receiver operating characteristic* (ROC) [37], an example is given below[2].

In the worst case (random guessing) the ROC is just a straight line going from (0,0) to (1,1), in the ideal case it is a function that is one everywhere. An often used metric that is derived from the ROC-curve is the value of FPR at $TPR = 0.80$, denoted as Se80 or $FPR.80$, and is ideally as small as possible. This however contains information from only one point of the ROC-curve, a better and more robust measure is the area under the ROC-curve (AUC), which is equal to one for a perfect model.

Note that these metrics used to evaluate a model are not the same as a loss function that is used for optimization, the above mentioned metrics are non-differentiable and can thus not be used for gradient descent-based algorithms. Often however, the loss function can be used as a metric to evaluate the model. An example of an often used loss function is the quadratic loss (mean squared error)

[2]https://www.ncss.com/software/ncss/roc-curves-ncss/

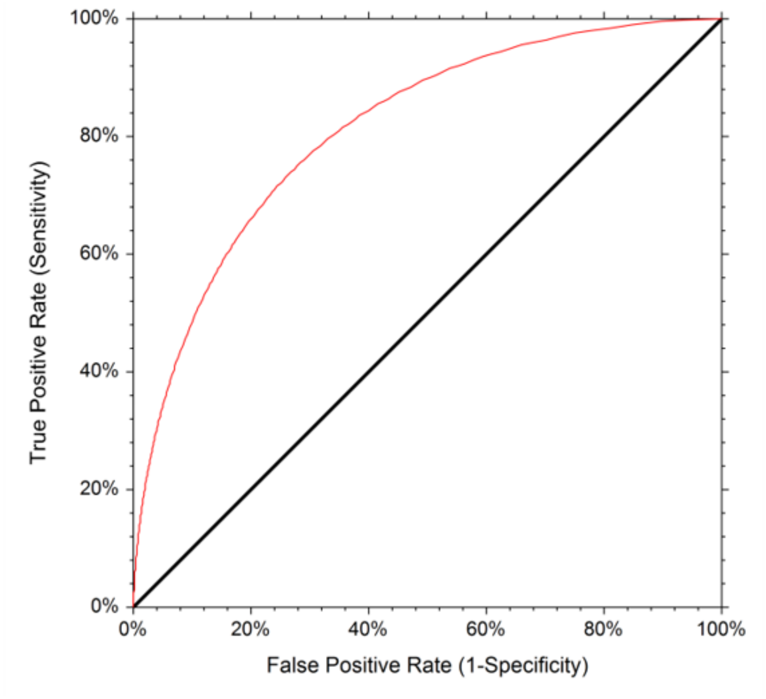


Figure 1.1: An example of a receiver operating characteristic.

$$L(f) = \frac{1}{n}\sum_{i=1}^{n}(f(\boldsymbol{x}_i) - y_i)^2 \tag{1.4}$$

Another common example for binary classification is the cross entropy (equivalent to minus the log-likelihood) with $y_i \in \{0, 1\}$ and where the output of the model is assumed to be a probability (i.e. $0 < f(\boldsymbol{x}_i) < 1$)

$$L(f) = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(f(\boldsymbol{x}_i)) + (1 - y_i)\log(1 - f(\boldsymbol{x}_i)) \tag{1.5}$$

Cross-entropy quantifies in a sense the ”overlap” between two distributions, in this case between the true underlying distribution and the approximated one.

1.4 Kernel based methods

In literature, kernel methods are combined with all kinds of (supervised as well as unsupervised) machine learning methods [38]. In this section a short introduction is given to kernel methods in the context of support vector machines (SVM), where they are most used.

Consider a binary classification problem with two possible class labels $y_i \in \{-1, 1\}$. A well known supervised model is a support vector machine which tries to find a hyperplane $\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$ that separates datapoints in two classes, maximizing the margin between

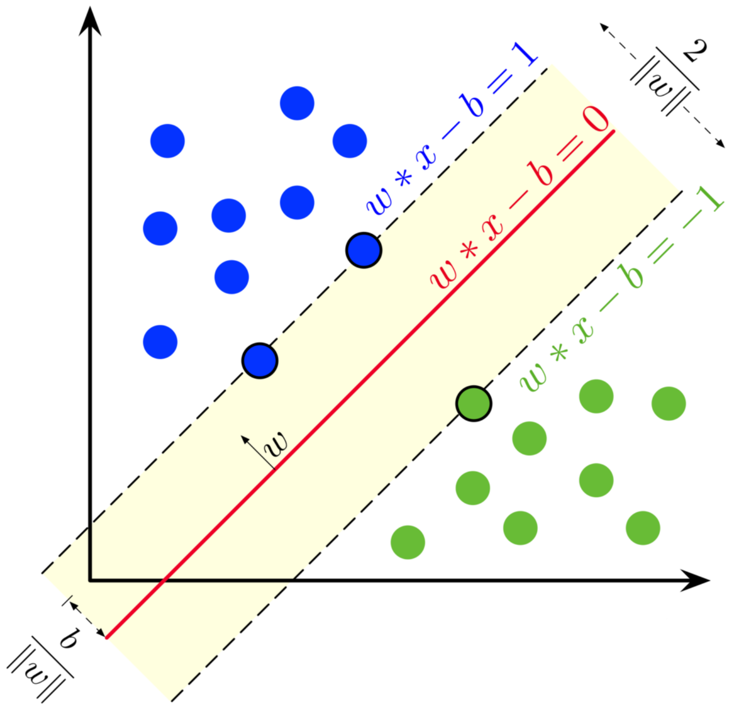


Figure 1.2: Classification with a SVM.

the two groups of points (figure 1.2). Classification is done in the following way: $y_i = 1$ if $\boldsymbol{w} \cdot \boldsymbol{x}_i + b \geq 1$ and $y_i = -1$ if $\boldsymbol{w} \cdot \boldsymbol{x}_i + b \leq -1$; which can be rewritten as the following constraint

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) - 1 \geq 0 \quad \forall i \tag{1.6}$$

The hyperplanes $\boldsymbol{w} \cdot \boldsymbol{x} + b = 1$ and $\boldsymbol{w} \cdot \boldsymbol{x} + b = -1$ are defined by the data points that lie upon them, these data points are called the *support vectors* and fully define the SVM model. The distance between these two hyperplanes is given by $\frac{2}{||\boldsymbol{w}||}$, maximizing the margin between the two classes thus comes down to minimizing $||\boldsymbol{w}||$ under constraint (1.6). We can define the following loss function (Lagrangian) to be minimized with respect to the parameters

$$L = \frac{1}{2}||\boldsymbol{w}||^2 - \sum_i \lambda_i (y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) - 1) \tag{1.7}$$

where we introduced the Karush–Kuhn–Tucker multipliers $\lambda_i \geq 0$ [39] (Lagrange multipliers for inequality constraints). The task is thus to minimize L with respect to the parameters $\boldsymbol{w}$ and b while requiring that the derivatives $\frac{\partial L}{\partial \lambda_i} = 0 \;\; \forall i$, this defines a convex quadratic optimization problem. Once the model is trained and its parameters found, classification on a test set is done as $\hat{y}_j = \text{sgn}(\boldsymbol{w} \cdot \boldsymbol{x}_j + b)$. As a side note, until now we have defined an optimization problem for the case of linear separable problems. When the classification problem is not perfectly linear separable, slack variables can be introduced and a corresponding term can be added to (1.7) in order to approximate the ideal case [40].

The framework defined above can be generalised to non-linear problems. By transforming the input $\boldsymbol{x} = (x_1, ..., x_N)$ to a high-dimensional space via a *feature map* $\phi(\boldsymbol{x}) : \mathbb{R}^N \mapsto \mathbb{R}^M$, for example a quadratic mapping (with $M = \frac{(N+2)(N+1)}{2}$)

$$\phi(\boldsymbol{x}) = (1, \sqrt{2}x_1, ..., \sqrt{2}x_N, x_1^2, ..., x_N^2, \sqrt{2}x_1x_2, ..., \sqrt{2}x_{N-1}x_N), \tag{1.8}$$

the problem can become linear separable and a dividing hyperplane can be found (illustrated in figure 1.3). The hyperplane $\boldsymbol{W} \cdot \phi(\boldsymbol{x}) = 0$ (the bias parameter b is included in $\boldsymbol{W}$ through the first component of the feature map) in the new high-dimensional feature space will then represent a more general surface in the original input space, for the quadratic feature map this will be a general conic section.

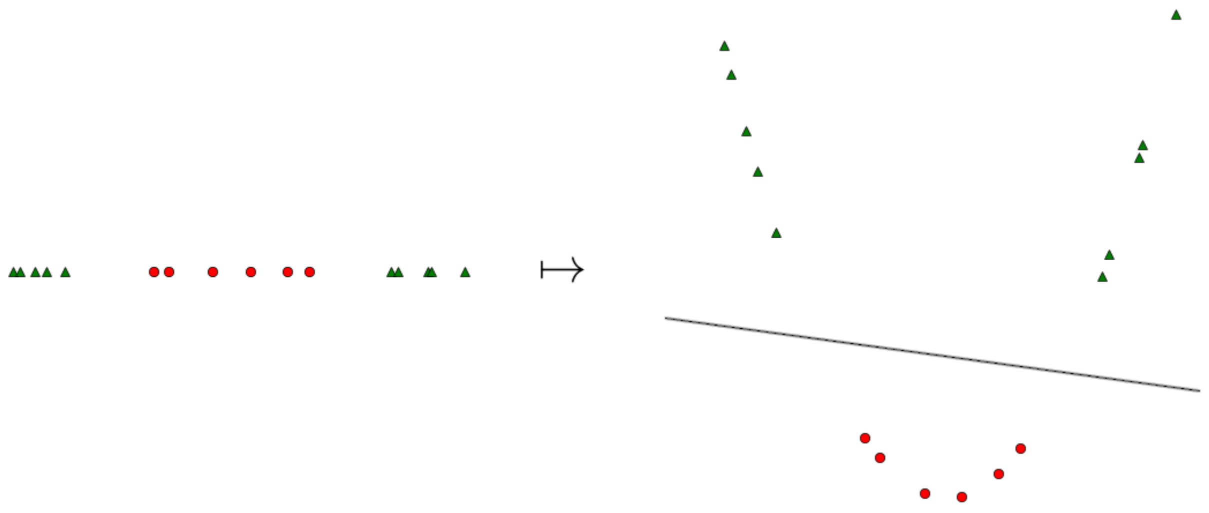


Figure 1.3: Non-linear separable points become linear separable with the feature map $x_i \mapsto (x_i, x_i^2)$.

The new feature space has a high dimension, for the quadratic feature map this is just of the order $M \sim N^2$, but it can be much larger or even infinite. Working explicitly in this new space would in those cases be computationally very demanding or even impossible. In order to cope with this, let us define the *Wolfe dual* [41] optimization problem of (1.7), with constraints $\nabla_{\boldsymbol{w}} L = 0$ and $\frac{\partial L}{\partial b} = 0$ (see appendix for more information):

$$\widetilde{L} = \sum_i \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i \cdot \boldsymbol{x}_j \tag{1.9}$$

which is to be maximized with respect to the λ_i with the constraints

$$\sum_i \lambda_i y_i = 0 \quad \text{and} \quad \lambda_i \geq 0. \tag{1.10}$$

From the *dual variables* λ_i the original parameters can be retrieved as

$$
\begin{aligned}
\boldsymbol{w} &= \sum_i \lambda_i y_i \boldsymbol{x}_i \\
b &= \boldsymbol{w} \cdot \boldsymbol{x}_k - y_k
\end{aligned}
\tag{1.11}
$$

with $\boldsymbol{x}_k$ a data point on the margin's boundary, i.e. a support vector ($\lambda_k \neq 0$). In the new feature space defined by $\phi(\boldsymbol{x})$ the Lagrangian becomes

$$
\widetilde{L} = \sum_i \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j \phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_j) \tag{1.12}
$$

Here $\frac{n^2}{2}$ dot products need to be computed[3] each requiring M calculations. Note that only inproducts of the feature vectors appear and that in principle the exact form of the map $\phi(\boldsymbol{x})$ needs not to be known. For the quadratic mapping (1.8) it can be shown that $\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y}) = (1 + \boldsymbol{x} \cdot \boldsymbol{y})^2$, i.e. needing only N computations instead of $M \sim N^2$. In general we can define a (positive semi-definite[4]) *kernel function* such that $k(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})$, examples are:

- the polynomial kernel: $k(\boldsymbol{x}, \boldsymbol{y}) = (1 + \boldsymbol{x} \cdot \boldsymbol{y})^p$ with $p \in \mathbb{N}$
- the Gaussian (RBF) kernel: $k(\boldsymbol{x}, \boldsymbol{y}) = e^{-\gamma ||\boldsymbol{x} - \boldsymbol{y}||^2}$
- the Laplacian kernel: $k(\boldsymbol{x}, \boldsymbol{y}) = e^{-\gamma ||\boldsymbol{x} - \boldsymbol{y}||}$

Once $\widetilde{L}$ is optimized and the λ_i found, classification of new data is obtained as

$$
\hat{y}_j = \text{sgn}\Big(\sum_i^n \lambda_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}_j)\Big). \tag{1.13}
$$

Evaluation of data points thus, in principle scales with the size of the training set n, however as there are often not too many support vectors ($\lambda_i \neq 0$) this is a reasonably fast computation. By working with $k(\boldsymbol{x}, \boldsymbol{y})$ for the optimization, we avoid working explicitly with the feature map $\phi(\boldsymbol{x})$ and the space to which it maps; this method is sometimes called "the kernel trick". Note that although it is avoided to work with the high (possibly infinite) dimension M, the algorithm now scales quadratic in the number of training samples instead of linear.

[3] n denotes the number of training samples.

[4] A positive-definite kernel is in this context defined in the following way. A symmetric function $k : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$, where $\mathcal{D}$ is assumed to be a nonempty set, is called a positive-definite (p.d.) kernel on $\mathcal{D}$ if

$$
\sum_i^n \sum_j^n c_i c_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0
$$

for any $n \in \mathbb{N}, \boldsymbol{x}_i \in \mathcal{D}\ \forall i$ and $c_i \in \mathbb{R}\ \forall i$. In machine learning literature $k(\boldsymbol{x}, \boldsymbol{y})$ is said to be p.d. if equality in the above equation implies $c_i = 0\ \forall i$, if this is not necessarily the case $k(\boldsymbol{x}, \boldsymbol{y})$ is called positive semi-definite (p.s.d.).

Chapter 2

Tensor Networks

2.1 Introduction and justification

Consider a general quantum many-body system consisting of N particles, its description is governed by the following wave function

$$|\Psi\rangle = \sum_{i_1,\dots,i_N} C_{i_1,\dots,i_N} |i_1\rangle \otimes \dots \otimes |i_N\rangle \tag{2.1}$$

where each $|i_n\rangle$ denotes the basis of an individual particle with label n. Assuming that each of these particles live in a local Hilbert space $\mathcal{H}$ of dimension d, the dimension of the full Hilbert space $\mathcal{H}^{\otimes N}$ is d^N. The coefficient tensor $C_{i_1,\dots,i_N}$ thus has d^N entries and a full description of the wave function $|\Psi\rangle$ –and thereby the system– is intractable. A generic way to deal with this is to approximate the tensor $C_{i_1,\dots,i_N}$ (and thus the wave function) by a contracted network of lower order tensors [1][2] (figure 2.1). Tensor networks (TN) provide an exponential compression of the state space, in such a way that all physical relevant features are still accessible.

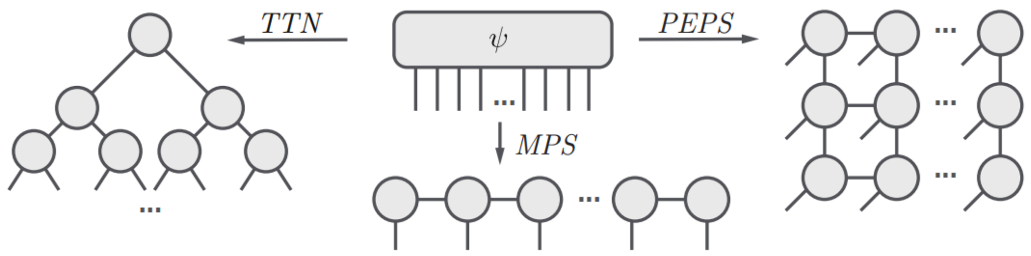


Figure 2.1: A few examples of tensor network decompositions [42]. More on the diagrammatic notation later.

Of course there is much more to it than just an exponential reduction in the number of parameters. One of the most appealing features of this new method is the fact that it provides a large amount of theoretical insights. When studying a quantum many-body system, specifying the coefficients of the wave functions in given local basis doesn't give a lot of insight in the entanglement/correlation structure of the particles in the system. The structure of entanglement will depend on the system under consideration, e.g. the dimensionality of the system, criticality and correlation length etc. Tensor networks allow to represent a quantum state in a way where this information is explicit and directly available, they form an entanglement representation and different networks are suited for different kinds of physical states.

The main reason for the success of tensor networks in quantum many-body physics is the locality of interactions between the constituents of the system. Consider a system which is divided into two subsystems, A and B. For a general quantum state picked at random from the huge Hilbert space $\mathcal{H}^{\otimes N}$, the entanglement entropy S between subsystem A and subsystem B will (most likely) scale as the volume of a subregion. However, the relevant states encountered in nature are a result of a Hamiltonian that acts locally which has an important consequence. One can prove that for a gapped Hamiltonian with local interactions, the low-energy states obey the so called *area law for entanglement entropy* [3][43]. This means that, for a physically relevant state, the entanglement entropy between two subsystems of this state scales as the area of the boundary between these two subsystems $S \sim \partial A$ instead of the volume as for general states. The low-energy states of a local Hamiltonian are thus heavily constrained in the sense that they must obey the area law and they cover only a tiny exponentially small subspace of the unmanageable large many-body Hilbert space (figure 2.2). Moreover one can prove that by evolving a many-body state for a time $O(\text{poly}(N))$ with a local Hamiltonian, the manifold of states that can be reached is also exponentially small [44]. The vast majority of the Hilbert space will thereby only be reached in an exponential amount of time; or given an initial (area law) state, the greatest part of the Hilbert space will be practically unreachable. It turns out that this exponentially small manifold of relevant states is well described by the framework of tensor networks.

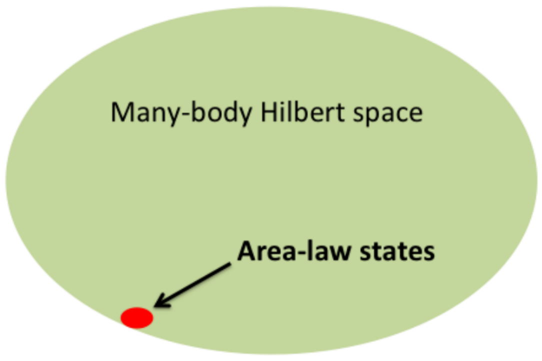


Figure 2.2: The relevant (area law) states reside in an exponentially small corner (manifold) of the gigantic many-body Hilbert space [1].

2.2 Diagrammatic notation

For our purpose, a *tensor* is just defined as a multidimensional array of complex numbers, labeled by a set of indices, and will be denoted by a capital letter. The number of indices is called the *order* (sometimes rank or degree) of the tensor. A contraction over a pair of repeated indices will for the remainder be denoted according to the Einstein summation convention[1]

$$A_{\mu\nu} = \sum_{\alpha} Z_{\mu\alpha} E_{\alpha\nu} =: Z_{\mu\alpha} E_{\alpha\nu}. \tag{2.2}$$

Following the convention of tensor network diagram notation, a tensor will be depicted by some shape like a circle or square (which can have a different meaning depending on the context) and a number of legs corresponding to the indices of the tensor. Two connected legs between two tensors denote a contraction between the corresponding tensors (figure 2.3).

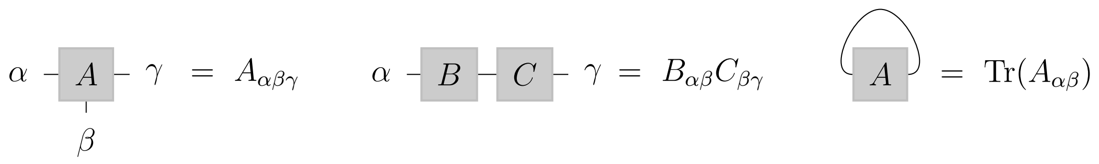

α
A
γ
β
$= A_{\alpha\beta\gamma}$
α
B
C
γ
$= B_{\alpha\beta}C_{\beta\gamma}$
A
$= \mathrm{Tr}(A_{\alpha\beta})$

Figure 2.3: Diagrammatic notation for tensors and contractions.

Using these tensor network diagrams gives more insight and intuition in calculations and algorithms involving a lot of tensors. When studying tensor networks in physics, the internal indices over which one contracts are called the *virtual indices* and the range over which they vary is called the *bond dimension* D. The external indices corresponding to the physical degrees of freedom are called the *physical indices*.

To illustrate that tensor networks obey an area law consider a system described by a tensor network where each tensor corresponds to a physical particle (for example an MPS or PEPS, see later). Let us define a finite subsystem A within this system and call the remainder of the system B, with a boundary that cuts L virtual indices of bond dimension D (figure 2.4). The wave function can be written in the following Schmidt form (see appendix)

$$|\Psi\rangle = \sum_{\alpha}^{D^L} \lambda_\alpha |\psi_\alpha^A\rangle \otimes |\psi_\alpha^B\rangle \tag{2.3}$$

where α denotes the combined index of the L indices that connect subsystem A with B

[1]The indices of the tensor being either in subscript or superscript won't have any meaning in this work and are mostly just placed as they are for convenience.

and can take on D^L values. The reduced density matrix of the inner part A (which is diagonalized by the Schmidt decomposition) is given by

$$\rho_A = \mathrm{Tr}_B(|\Psi\rangle\langle\Psi|) = \sum_{\alpha}^{D^L} \lambda_\alpha^2 |\psi_\alpha^A\rangle\langle\psi_\alpha^A| \tag{2.4}$$

We can derive an upper bound for the entanglement entropy of subsystem A. Assuming that A and B are maximally entangled, we have $\lambda_\alpha = \lambda, \forall\alpha$ and the density matrix is proportional to the identity $\rho_A = \lambda^2\mathbb{I}$ with $\lambda^2 = \frac{1}{D^L}$. The entanglement entropy is now given by

$$\begin{aligned} S_{\text{upper}} &= -\mathrm{Tr}(\rho_A \log \rho_A) \\ &= -\sum_{\alpha}^{D^L} \lambda_\alpha^2 \log(\lambda_\alpha^2) \\ &= L \log D \end{aligned} \tag{2.5}$$

This shows that the entanglement entropy of a state modeled by a PEPS can at most scale with the area of the boundary L (with $D \sim$ constant), and if one wants the entropy S to scale as the volume, it is necessary that $D \sim \exp L$.

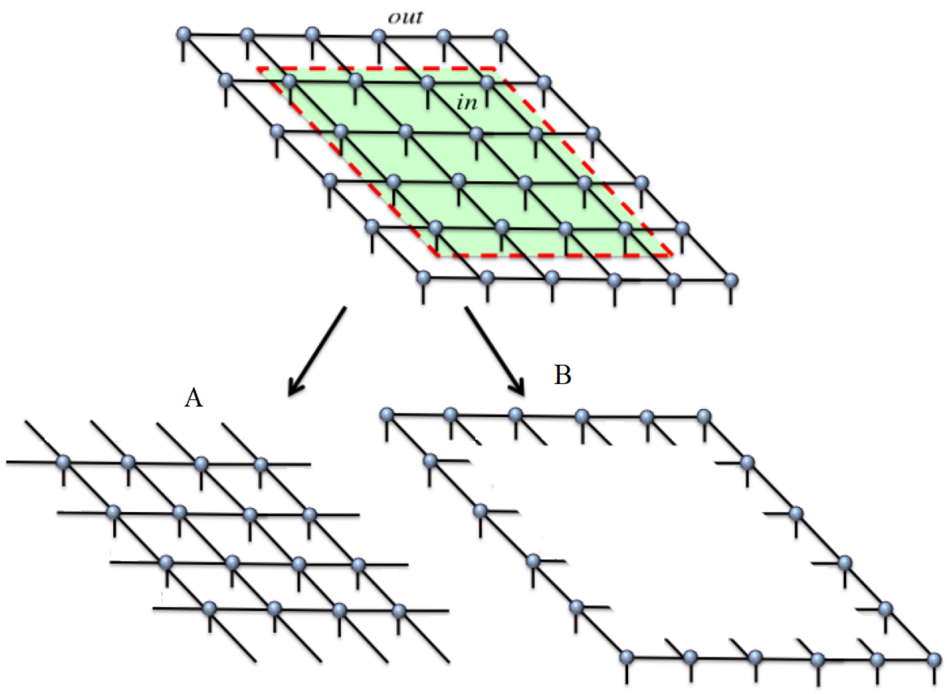


Figure 2.4: A system described by a PEPS divided into two subsystems A and B.

2.3 Matrix product states

Consider a finite one-dimensional lattice of particles of length N, a general wave function has the form (2.1) where the label i_n denotes the state of the particle on position n along the chain. Let us now illustrate how the general state tensor $C_{i_1,...,i_N}$ can be written as a *matrix product state* (MPS) [45][46], where $i_\mu \in \{1, ..., d_\mu\}$. Let us first unfold the tensor in a matrix $C[1]_{i_1}^{(i_2,...,i_N)}$ with row index i_1 and collective column index $(i_2, ..., i_N)$. Next a singular value decomposition (SVD, see appendix) is applied to this matrix, resulting in

$$\begin{aligned} C[1]_{i_1}^{(i_2,...,i_N)} &= U_{i_1}^{\alpha_1} \Lambda_{\alpha_1}^{\alpha_2} (V^\dagger)_{\alpha_2}^{(i_2,...,i_N)} \\ &= A[i_1]^{\alpha_1} C[2]_{\alpha_1}^{(i_2,...,i_N)} \end{aligned} \tag{2.6}$$

where we defined $A[i_1]^{\alpha_1} := U_{i_1}^{\alpha_1}$ and $C[2]_{\alpha_1}^{(i_2,...,i_N)} := \Lambda_{\alpha_1}^{\alpha_2} (V^\dagger)_{\alpha_2}^{(i_2,...,i_N)}$ and a bond dimension $D_1 = d_1$ is obtained. Now the matrix $C[2]_{\alpha_1}^{(i_2,...,i_N)}$ is reshaped into $C[2]_{(\alpha_1,i_2)}^{(i_3,...,i_N)}$ and an SVD used on it

$$\begin{aligned} C[2]_{(\alpha_1,i_2)}^{(i_3,...,i_N)} &= U_{(\alpha_1,i_2)}^{\alpha_2} \Lambda_{\alpha_2}^{\alpha_3} (V^\dagger)_{\alpha_3}^{(i_3,...,i_N)} \\ &= A[i_2]_{\alpha_1}^{\alpha_2} C[3]_{\alpha_2}^{(i_3,...,i_N)} \end{aligned} \tag{2.7}$$

where $A[i_2]_{\alpha_1}^{\alpha_2} := U_{(\alpha_1,i_2)}^{\alpha_2}$ and $C[3]_{\alpha_2}^{(i_3,...,i_N)} := \Lambda_{\alpha_2}^{\alpha_3} (V^\dagger)_{\alpha_3}^{(i_3,...,i_N)}$ and with bond dimension $D_2 = d_1 d_2$. These decompositions can be repeated until the following decomposition of the original tensor is obtained (figure 2.5)

$$C_{i_1,...,i_N} = A[i_1]^{\alpha_1} A[i_2]_{\alpha_1}^{\alpha_2} ... A[i_N]_{\alpha_{N-1}} \tag{2.8}$$

with the bond dimensions given by $D_\mu = \prod_{\nu=1}^{\mu} d_\nu$. The result obtained so far is exact, but not very useful yet, as the bond dimensions scale exponentially: $D_\mu \geq (\min_\nu d_\nu)^\mu$. Suppose however we consider states for which the entanglement rank across any bisection of the chain is bounded. In particular, suppose that only D of the Schmidt weights (singular values) were non-zero. Then the MPS form van be used to take advantage of this property by truncating the Λ matrix. The states described by this approximation are the area law states mentioned earlier and the description in terms of a (truncated) MPS containing only $O(ND^2d)$ parameters is exact in these cases.

The MPS as derived here (successive Schmidt decompositions) is said to be in canonical form. Note that due to the $A[i_\mu]$ being unitary for $\mu < N$, the normalization of the wave function is entirely contained in the last tensor $A[i_N]$

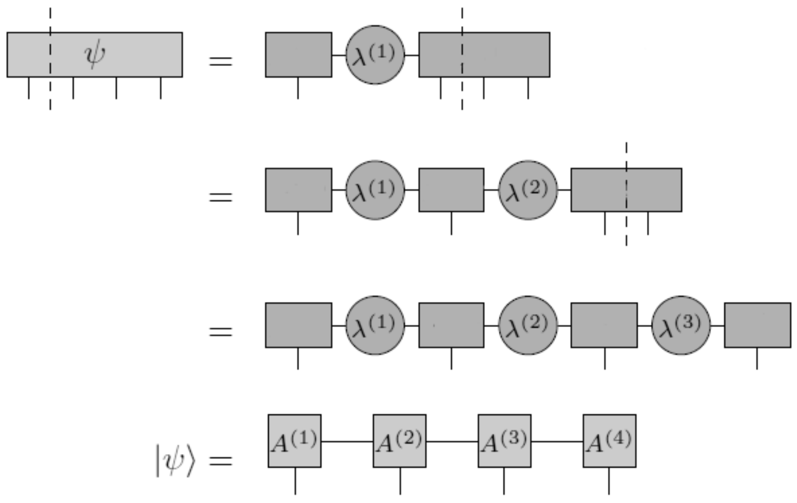


Figure 2.5: MPS decomposition of a general state tensor [47].

$$
\begin{aligned}
\langle\Psi|\Psi\rangle &= \sum_{\{i_\mu\}} |C_{i_1,\dots,i_N}|^2 \\
&= \sum_{\{i_\mu\}} A[i_N]^\dagger \dots A[i_2]^\dagger A[i_1]^\dagger A[i_1] A[i_2] \dots A[i_N] \\
&= \sum_{i_N} A[i_N]^\dagger A[i_N]
\end{aligned}
\tag{2.9}
$$

Generally speaking, we say that an MPS is in the left-canonical form (also left-orthogonal) if for all tensors but the last $\sum_{i_\mu} A^\dagger[i_\mu]A[i_\mu] = \mathbb{I}_D$, or equivalently the left-unfoldings[2] of these tensors are semi-unitary/left-unitary (figure 2.6 left). Analogous, the MPS is said to be in the right-canonical form (also right-orthogonal) if for all tensors but the first, we have $\sum_{i_\mu} A[i_\mu]A^\dagger[i_\mu] = \mathbb{I}_D$ (figure 2.6 right). An MPS is said to be in a mixed canonical form if for some site μ, the tensors on the left are in left-canonical form and the tensors on the right in right-canonical form, the MPS is then also said to be μ-orthogonal.

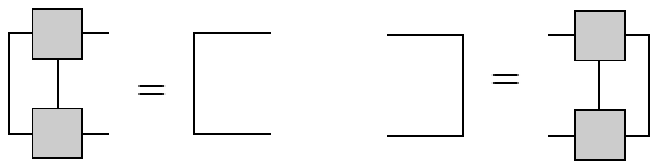


Figure 2.6: TN notation for left-canonical (left) and right-canonical (right) forms.

MPS have some interesting properties, let us very shortly summarize some of these.

[2]The left-unfolding of an MPS tensor is given by $A[i_\mu]^L = A^{\alpha_\mu}_{(\alpha_{i_{\mu-1}}, i_\mu)} \in \mathbb{R}^{D_{\mu-1}d_\mu \times D_\mu}$ and equivalently the right-unfolding $A[i_\mu]^R = A^{(\alpha_\mu, i_\mu)}_{\alpha_{i_{\mu-1}}} \in \mathbb{R}^{D_{\mu-1} \times d_\mu D_\mu}$.

The MPS considered until now were finite with open boundary conditions and each tensor could be different, of course they could be made translational invariant by defining a unit cell of tensors and repeating these, taking the MPS to the thermodynamic limit or imposing periodic boundary conditions by taking the trace

$$C_{i_1,\dots,i_N} = \mathrm{Tr}(A[i_1]^{\alpha_1}_{\alpha_0} A[i_2]^{\alpha_2}_{\alpha_1} \dots A[i_N]^{\alpha_N}_{\alpha_{N-1}}) \tag{2.10}$$

An interesting property of MPS is that they are dense, i.e. they can represent any quantum state of the many-body Hilbert space by increasing the value of the bond dimension D. However, as one could expect, to cover all the states in the Hilbert space D needs to be exponentially large in the system size. A related property is that MPS are finitely-correlated, i.e. the correlation functions of an MPS always decay exponentially with the separation distance. This means that the correlation length of these states is always finite and therefore MPS are unable to reproduce the properties of critical or scale-invariant systems, where the correlation length is known to diverge. Lastly MPS make computations and simulations feasible, not only because of the exponential compression of the state vector, but also because operations on them can be efficiently carried out and because they can efficiently be contracted. For example the inproduct between two MPS can exactly be computed in $O(NdD^3)$ time, the same is true for expectation values of local operators such as correlation functions, energies and local order parameters. An overview of different operations on tensors in the MPS format and their complexities is given in table 2.1.

Operation	Resulting bond dimension	Complexity
$C = \text{const} \cdot A$	$D(C) = D(A)$	$O(dD(A))$
$C = A + \text{const}$	$D(C) = D(A) + 1$	$O(NdD(A)^2)$
$C = A + B$	$D(C) \leq D(A) + D(B)$	$O(NdD(C)^2)$
$C = A \odot B$	$D(C) \leq D(A)D(B)$	$O(NdD(A)^2D(B)^2)$
$\langle A\|A\rangle$	-	$O(NdD(A)^3)$

Table 2.1: Different operations of tensors in the MPS form and their complexities [48]. With $D(T)$ the bond dimension/TT-rank of a tensor T in MPS form is denoted.

Finally it is worth mentioning that in the mathematics community matrix product states are also known as tensor trains (TT) [48]. In this context the effective bond dimension is called the TT-rank of a tensor, it is the number of non-zero singular values of the SVD of any μ-unfolding of the tensor: $\mathbb{R}^{d_1 \times d_2 \times \dots \times d_N} \rightarrow \mathbb{R}^{(d_1, d_2 \dots, d_\mu) \times (d_{\mu+1}, \dots, d_N)}$ (i.e. reshaping the tensor in a matrix). For more information we refer to [48][49].

2.4 Finding a satisfactory MPS

Until now we discussed how an MPS decomposition is obtained and why they are interesting. In this section we will shortly describe two well known algorithms of optimizing matrix product states, namely DMRG (density matrix renormalization group) [50] and TEBD (time-evolving block decimation) [51].

2.4.1 DMRG

Suppose one wants to find the ground state of some Hamiltonian H, according to the variational principle, for a given state $|\Psi\rangle$ it will always be true that

$$\frac{\langle\Psi|H|\Psi\rangle}{\langle\Psi|\Psi\rangle} \geq E_0 \tag{2.11}$$

and the ground state can be approached from above by minimizing the left-hand side. This minimization is typically done for some parameterized family of states, in this case matrix product states, giving us the following optimization problem

$$\min_{|\Psi\rangle\in\text{MPS(D)}}(\langle\Psi|H|\Psi\rangle - \lambda\langle\Psi|\Psi\rangle) \tag{2.12}$$

Ideally, the above minimization should be done simultaneously over all the free parameters of the TN state, hence over all the coefficients of all the tensors for all sites. However this is normally quite difficult to implement and not particularly efficient. Instead of this, the strategy is to proceed tensor by tensor. This is, to minimize with respect to one tensor while keeping the others fixed, then move to another tensor and repeat the minimization, and so on. In practice one sweeps over all the tensors several times until the desired convergence in expectation values is attained. When updating tensor A in the MPS, this can be written as

$$\min_{A}(A^\dagger H_{\text{eff}} A - \lambda A^\dagger N A) \tag{2.13}$$

where H_{eff} and N are the effective Hamiltonian and the normalization matrix respectively (i.e. the environment of tensor A, see figure 2.7). Minimization then results in a generalized eigenvalue problem

$$\begin{aligned} &\frac{\partial}{\partial A^\dagger}(A^\dagger H_{\text{eff}} A - \lambda A^\dagger N A) = 0 \\ \iff\ & H_{\text{eff}} A = \lambda N A \end{aligned} \tag{2.14}$$

This technique of sweeping over the MPS and optimizing each tensor locally is known as DMRG and all kinds of variations on this algorithm exist. An example being two-site DMRG, where two neighbouring tensors are contracted into a bond tensor, this bond tensor is then optimized and again split up in two separate tensors by means of an SVD, this gives control over the bond dimension growth and a quantification of truncation errors.

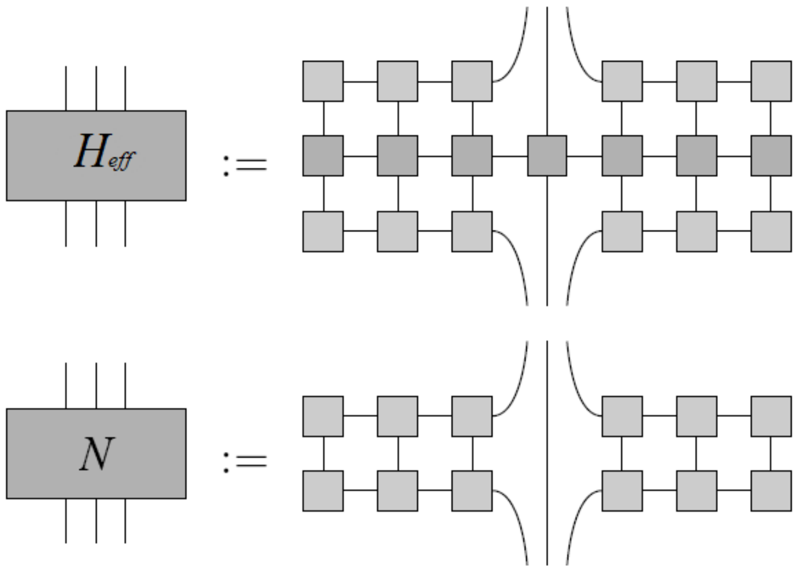


Figure 2.7: The definition of the effective Hamiltonian and normalization (environments) [47].

2.4.2 TEBD

A second way of finding the ground state of a Hamiltonian H is imaginary time evolution. By evolving some arbitrary quantum state $|\Psi(0)\rangle$ at $t = 0$ (with non-zero overlap with the ground state) in imaginary time $\tau = -it$, one obtains the following in the limit $\tau \to \infty$

$$\begin{aligned} e^{-itH}|\Psi(0)\rangle &= \sum_n c_n e^{-\varepsilon_n \tau}|\Psi_n\rangle \\ &\underset{\tau\to\infty}{=} c_0 e^{-\varepsilon_0 \infty}|\Psi_0\rangle \end{aligned} \tag{2.15}$$

i.e. only the ground state remains (as $\varepsilon_0 \leq \varepsilon_n, \forall n$), apart from a normalization constant. The idea now is to find a way to implement such an evolution on MPS, for a given quantum Hamiltonian. A way to achieve this is to split the time-evolution operator into small imaginary-time steps,

$$e^{-\tau H} = (e^{-\delta\tau H})^M \tag{2.16}$$

with $M = \frac{\tau}{\delta\tau} \gg 1$. Next, let us assume for the sake of simplicity that the Hamiltonian is the sum of two-body nearest-neighbor terms, $H = \sum_{\langle i,j\rangle} h_{ij}$. Using the Suzuki-Trotter expansion [52] up to first order, the evolution operator can be approximated as

$$e^{-\delta\tau H} = \prod_{\langle i,j\rangle} e^{-\delta\tau h_{ij}} + O(\delta\tau^2) \tag{2.17}$$

Each of the terms in the multiplication above is called a two-body gate, in a notation that is reminiscent from that used in the language of quantum circuits, given by $g_{ij} = e^{-\delta\tau h_{ij}}$. The imaginary-time evolution operator (2.16) can then be approximated by repeating the operator $U(\delta\tau) = \prod_{\langle i,j\rangle} g_{ij}$ M times. There are different orders in which these individual two-body gates g_{ij} can be applied each time step, for example sequential or, most commonly, in parallel (see figure 2.8).

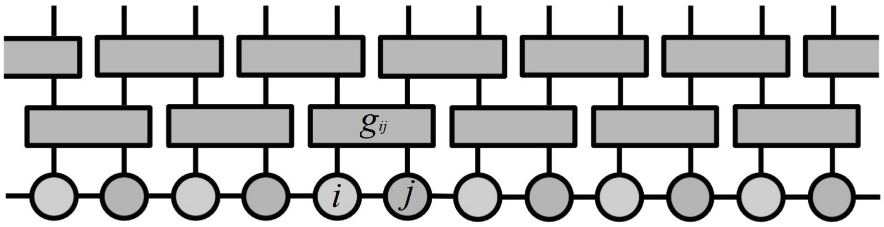


Figure 2.8: Application of local imaginary time evolution operators.

After one local time step, two neighbouring tensors have to be separated again, which is done with an SVD (figure 2.9), this gives control over the bond dimension growth as in the case of DMRG2. This algorithm of doing small steps in imaginary time is known as TEBD and a lot of variations exist on this.

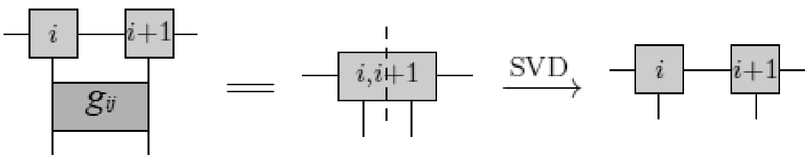


Figure 2.9: One evolution step followed by an SVD.

2.5 Other tensor networks

In the previous section we discussed the widely used matrix product states, there exist however a lot of other tensor network architectures, designed to tackle different kinds of problems. A well known class of tensor networks are *projected entangled pare states* (PEPS) [53][54], which are a generalization of MPS to higher spatial dimensions (figure 2.10). They can ofcourse also be defined on other lattices than the square lattice, such as hexagonal and triangular lattices.

PEPS are, as MPS, dense in the sense that they can represent any state in the large Hilbert space by increasing the bond dimension, needing exponentially large bond dimensions to cover the whole space. Nevertheless, one expects the bond dimension to be

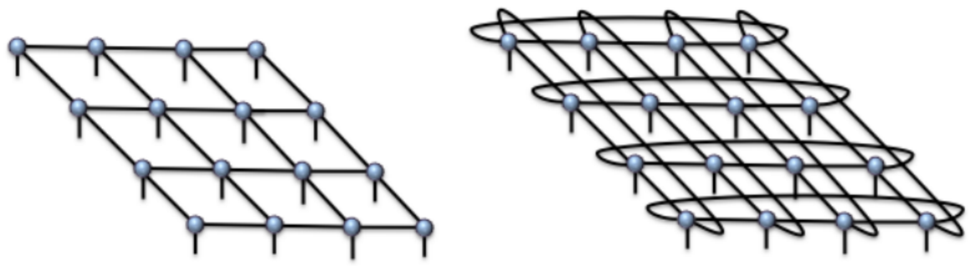


Figure 2.10: Projected entangled pare states on a square lattice with open boundary conditions (left) and periodic boundary conditions (right) [1].

reasonably small for low-energy states of interesting quantum systems and, as we showed earlier, they describe states that obey an area low for the entanglement entropy. Contrary to MPS, PEPS can capture two-point correlation function that decay polynomially, i.e. have infinite correlation length, even for the small bond dimension of $D = 2$. This is an import property if one wants to describe critical behaviour. A problem that didn't appear in the case of MPS is that the exact contraction of PEPS is an exponentially hard problem, taking $O(e^N)$ time to compute [55]. There exist however approximate methods for contracting PEPS with high accuracy [56].

Another famous and widely used type of tensor network is the *multi-scale entanglement renormalization ansatz* (MERA) [57][58] (figure 2.11 right) which is constructed by successive applications of coarse-graining. This type of tensor network is, contrary to MPS, able to model critical behaviour in one dimension and to capture the resulting power law decay of the correlations. A variation on this is the *tree tensor network* (TTN) [59], which does not contain the unitary disentangler operations as MERA does (figure 2.11 left).

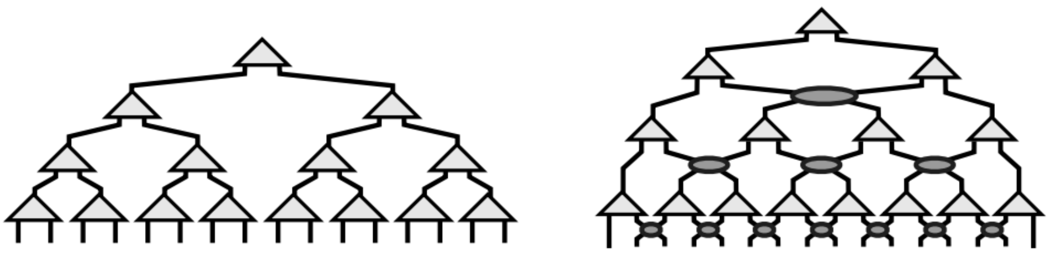


Figure 2.11: The tree tensor network (left) and the multi-scale entanglement renormalization ansatz (right).

Chapter 3

Machine learning with tensor networks

The problem of the exponential size of the many-body Hilbert space and difficulty representing a wave function in it is reminiscent of the problem in machine learning. In the field of statistical learning the goal is to model the joint probability distribution $P(x_1, ..., x_N, y)$ (or some classification function $f : (x_1, ..., x_N) \mapsto y$) of many correlated variables, the complexity of which also scales exponentially with the number of variables, and as is the case in quantum many-body physics approximate methods are bound to be used. A well known method to represent the complicated function $f(x_1, ..., x_N)$ is to decompose it as a (artificial) neural network which are able to represent very non-linear functions and are widely used in all kinds of applications, for more information we refer to the wide literature that is available [60][61][62][63][64].

Inspired by the success of tensor network decompositions to capture the relevant correlations in quantum many-body physics, machine learning models based on tensor network decompositions to describe the correlation structure in data will be studied. Only very recently it has been put forward that TN's could be viable candidate in the field of big data science and statistical learning and in the next section a short overview will be given of the work that is already done on this topic.

3.1 Overview of TN's in machine learning

In the context of (deep) neural networks tensor networks have been used in different ways, e.g. to improve the performance and computational cost and to better understand deep architectures. TN's have for example been applied to significantly compress the weight matrices of fully connected layers in deep architectures while preserving the expressive power of the layer [65]. This in turn makes it possible to consider much wider layers and construct larger models than was possible before and could promise to be of great importance for future development of neural network models. In another work a common TN-based language is established between deep learning architectures and TN quantum many-body wave functions and quantum entanglement is put forth a natural quantifiers of correlations in deep networks [66][67]. A deep convolutional arithmetic circuit (ConvAC) for example is equivalent to a tree tensor network (figure 3.1) and insights from the field of physics can guide the design of a network that is meant to characterize certain correlations among subsets of input variables.

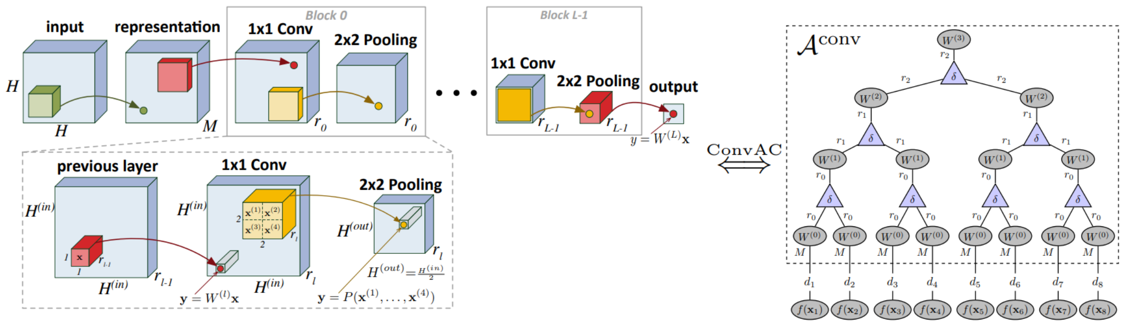


Figure 3.1: Equivalence between a deep convolutional arithmetic circuit and a tree tensor network (TTN) [66].

The field of artificial neural networks and deep learning has made great empirical advancements in the last decade although theoretical answers to the key questions of how and why these models work so well remain to be stated [68][69]. This common language between many-body physics and neural networks could offer theoretical insights from the former field to be carried over to the latter field and provide the theoretical answers sought after. Also in the other direction, this common framework could guide research in the field of quantum many-body physics and guide the development of more general wave function ansatzes based on the success of neural networks in the field of machine learning. The authors in [67] have for example shown that a wave function can be constructed which supports an entanglement entropy scaling as the volume with a number of parameters that is the square root of the number required by Restricted Boltzmann Machines (RBM) [70]. The same theoretical analysis can be made for recurrent neural networks (RNN) by mapping them to a (MPS-based) tensor network [67][71].

In the branch of graphical models and Markov random fields (MRF) [72] the joint probability distribution is represented as a product of factors depending on a smaller number of parameters

$$P(x_1, ..., x_N) = \prod_{c\in\zeta} \phi_c(x_c) = \frac{1}{Z}e^{-H(x_1,...,x_N)} = \frac{1}{Z}e^{-\sum_c V_c(x_c)} \tag{3.1}$$

where $c \in \zeta$ are called the *cliques* and $V_c(x_c)$ the clique potentials, depending on a small neighbourhood of variables x_c, a common example is the one with only nearest neighbour interactions based on the Ising model. To this representation corresponds a graph, described in the more general framework of graphical models [73], when this graph contains cycles the estimation of the partition function Z and the search of the most probable configuration (MAP-inference) becomes challenging. In [74] this problem is tackled by decomposing, not the joint probability distribution $P(\boldsymbol{x})$, but the energy tensor as a tensor train, which is shown to have a low TT-rank (i.e. bond dimension) in contrast to $P(\boldsymbol{x})$ which has an exponentially large TT-rank. This decomposition allows for efficient operations on the tensors and inference in the model.

TN's and more particularly MPS, have been deployed to handle feature extraction in high-dimensional datasets. When analyzing high order datasets, for which the data is represented by a tensor $X^k \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ (where $k = 1, ..., n$ denotes the training sample), one can do a Tucker decomposition (TD) [75], also known as a high order singular value decomposition (HOSVD), to reduce the dimensionality of multidimensional data. However, the algorithm to compute this decomposition is higher-order orthogonal iteration (HOOI) [76] for which the computational cost per iteration scales exponentially in N, moreover the TD preserves the high order structure of the data and is still exponential in N. Another way of decomposing the data X^k is an MPS decomposition, as is shown in [77][78], see figure 3.2.

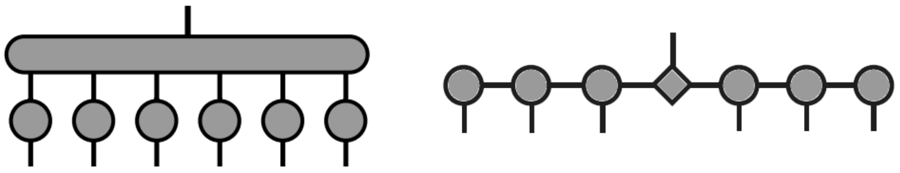


Figure 3.2: Tucker decomposition (left) versus MPS decomposition (right) for high dimensional data.

MPS decompositions have computational savings compared to HOOI for TD and do not require an iterative algorithm to be constructed, additionally MPS performs better than the TD for relevant feature extraction and classification. Tensor network decomposition have also been proven a useful tool in big data analytics and data mining and allow solving a wide class of big data optimization problems that would otherwise be intractable, for an overview we refer to [79].

As a form of generative unsupervised learning, (double layered) MPS have been applied to represent the joint probability distribution of data [80]. This is done by constructing a wave function as an MPS and writing the probabilities as the square of this wave function, i.e. working in the L_2-norm:

$$P(\boldsymbol{x}) = \frac{\Psi^2(\boldsymbol{x})}{Z} \tag{3.2}$$

where the wave function is restricted to values in $\mathbb{R}$ and where $Z = \sum_{\boldsymbol{x}} \Psi^2(\boldsymbol{x})$ denotes the normalization. To approximate the true underlying probability distribution with this parameterization, the log-likelihood is maximized on some training set by means of two-site DMRG

$$L = \sum_{i=1}^{n} \log \left(\frac{\Psi^2(\boldsymbol{x}_i)}{Z} \right) \tag{3.3}$$

The authors worked in the canonical gauge, which makes the expression for the partition function Z somewhat simpler. With the trained MPS one can for example generate new samples by computing marginal probability distributions, which can easily be done in the MPS format. One can also reconstruct samples for which only part of the feature values x^j are given, this can for example be used for image reconstruction (see figure 3.3) and denoising.

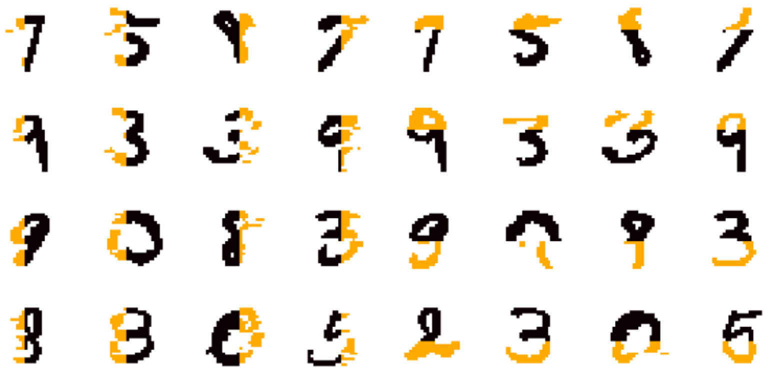


Figure 3.3: Image completion on a test set sampled from the MNIST dataset with an double layered MPS based model [80].

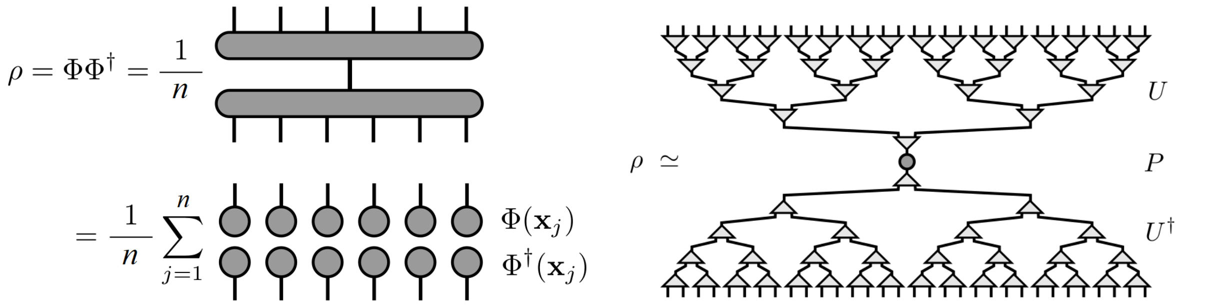
Another interesting approach is worked out in [81], where coarse-graining based on hierarchical tree tensor networks is studied as a means of learning relevant features. The input is mapped to a feature map with a tensor product structure (see later) $\Phi^{\boldsymbol{s}}(\boldsymbol{x}) = \phi^{s_1}(x_1) \otimes ... \otimes \phi^{s_N}(x_N)$ and a covariance matrix is constructed

$$\rho^{\boldsymbol{s}\boldsymbol{s}'} = \frac{1}{n} \Phi_j^{\boldsymbol{s}} \Phi_j^{\dagger \boldsymbol{s}'} \tag{3.4}$$

where $\Phi_j^{\boldsymbol{s}} = \Phi^{\boldsymbol{s}}(\boldsymbol{x}_j)$ and the index $j = 1, ..., n$ runs over the training samples. The dimension of the input (i.e. the feature map) is step by step lowered by locally projecting by means of *isometries* to the most relevant features (corresponding to the highest singular values) while keeping the fidelity as high as possible

$$F = \mathrm{Tr}(\rho) = \frac{1}{n}\Phi_j^s \Phi_j^{\dagger s} \tag{3.5}$$

This is a form of unsupervised learning, the resulting covariance matrix is given in figure 3.4



$\rho = \Phi\Phi^\dagger = \frac{1}{n}$
$= \frac{1}{n}\sum_{j=1}^{n}$
$\Phi(\mathbf{x}_j)$
$\Phi^\dagger(\mathbf{x}_j)$
$\rho \simeq$
U
P
$U^\dagger$

Figure 3.4: On the left the original covariance matrix, on the right the resulting approximate diagonalization of $\rho = UPU^\dagger$ with u a tree tensor network.

Once the isometries are determined, they can be used for feature extraction (partial coarse graining) with a supervised model on top (for example an MPS as shown in figure 3.5, see also later). For a more detailed explanation of how and why this works we refer to the original paper [81].

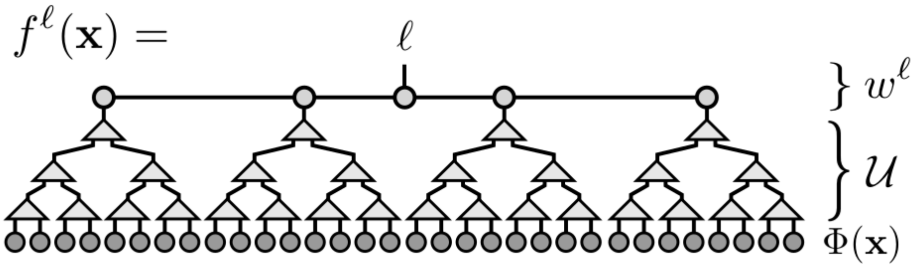


Figure 3.5: Partial unsupervised feature compression combined with a supervised (MPS) tensor network on top.

More general models have been proposed and tested in [82], where the concept of tensor networks is generalized to describe also graphical models (like RBM's) and to incorporate the non-linear copy-operation (which can in general not be represented by a tensor) for information reuse (figure 3.6).
Different types of TN's were combined, such as string-bond states (SBS) and entangled plaquette states (EPS) to construct supervised models, they where also combined with convolutional layers known from convolutional neural networks and were tested and compared on the task of image classification. Another new concept was introduced, instead to define the local feature maps $\phi^s(x)$ manually beforehand, the continuous variables are

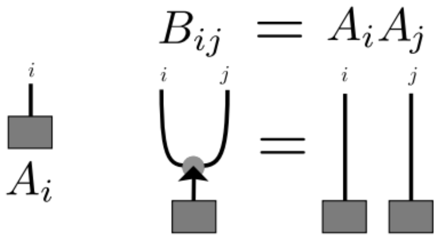


Figure 3.6: The copy-operation in diagrammatic notation.

discretized and the feature map is parameterized by a *feature tensor* to obtain more general non-linear feature maps.

We have only discussed a few instances of tensor networks being used in a machine learning context and new work on this is constantly being published. Finally it is worth mentioning that some of the TN-based methods and models mentioned above can be realized on a quantum computer and can benefit from a quantum algorithmic speedup and could guide future development of *"quantum machine learning"* (i.e. machine learning on a quantum computer) [82][83].

3.2 Supervised learning with TN's

3.2.1 Defining the model

The main method that will be studied in this project is based on [84][85] and aims to directly construct and optimize a supervised model constructed from a tensor network (more particularly an MPS). Consider the following classification function

$$f(\boldsymbol{x}) = \langle W|\Phi(\boldsymbol{x})\rangle \tag{3.6}$$

where $\Phi(\boldsymbol{x})$ is a high dimensional non-linear feature map. As mentioned earlier the dimension of the new feature space can be very large or even infinite, which makes the construction and optimization of the weight-tensor W intractable. As we described, one way to deal with this is to work only with the inproducts of the feature vectors and thus avoiding to construct the parameter-tensor W or even knowing the explicit form of the mapping $\Phi(\boldsymbol{x})$. Here a different approach will be considered, the tensor W will be explicitly constructed and optimized by decomposing it a a tensor network. As explained this decomposition provides an exponential compression of the large tensor. In quantum many-body physics TN's have proven a powerful framework that, despite of the exponential reduction in number of parameters, is still able to capture the correlation structure of systems with many entangled particles. Here we will investigate whether TN's remain a powerful tool in the case of statistical learning and if they are able to model the correlation pattern in data. This approach has some advantages compared to the kernel trick described earlier, the first one being the scaling of the training algorithm, in the dual representation used for the kernel trick, this scales as n^2, working directly with W allows the algorithm to scale only as n. A second advantage is that in the tensor network formulation of the classification model, a interpretation can be given to the tensors in the network and it can be viewed as extracting hidden information from the input. The TN form of W also allows for adaptive optimization, changing the bond dimension with respect to the resources needed to model the correlations in a certain patch of input data. Finally the tensor network decomposition can be viewed as an additional form of regularization next to the choice of feature map.

In order to construct a TN supervised model, a special kind of feature maps are used, the global feature map acting on $\boldsymbol{x} = (x^1, x^2, ..., x^N)$ will be constructed as the tensor product of local feature maps, of dimension d, which depend on only one variable [86] (figure 3.7):

$$\Phi(\boldsymbol{x}) = \phi^{s_1}(x^1) \otimes \phi^{s_2}(x^2) \otimes ... \otimes \phi^{s_N}(x^N) \tag{3.7}$$

with $s_i \in \{1, ..., d\}$ $\forall i$, the resulting feature map is of dimension d^N.

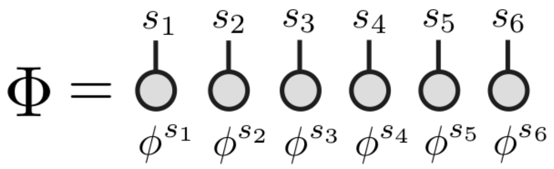

s_1 s_2 s_3 s_4 s_5 s_6
$\Phi =$
ϕ^{s_1} ϕ^{s_2} ϕ^{s_3} ϕ^{s_4} ϕ^{s_5} ϕ^{s_6}

Figure 3.7: The global feature map as the tensor product of local feature maps in TN notation.

An example of a (normalized: $\sum_s |\phi^s(x)|^2 = 1$) local feature map for continuous variables is

$$\phi^s(x) = \sqrt{\frac{(d-1)!}{(s-1)!(d-s)!}} \Big(\cos(\frac{\pi}{2}x) \Big)^{d-s} \Big(\sin(\frac{\pi}{2}x) \Big)^{s-1} \tag{3.8}$$

where $x \in [0, 1]$. A special case being $d = 2$, as used in [84]

$$\phi^s(x) = \begin{pmatrix} \cos(\frac{\pi}{2}x) \\ \sin(\frac{\pi}{2}x) \end{pmatrix} \tag{3.9}$$

which can be viewed as the normalized wave function of a qubit. Another example of a local feature map is the following, as introduced in [85]

$$\phi^s(x) = \begin{pmatrix} 1 \\ x \end{pmatrix} \tag{3.10}$$

and the global feature map $\Phi(\boldsymbol{x}) = \phi^{s_1}(x^1) \otimes ... \otimes \phi^{s_N}(x^N)$ can be given an interpretation as containing all possible interactions between features/variables to every order

$$\begin{aligned} \langle W|\Phi(\boldsymbol{x})\rangle = W_{00...0} &+ W_{10...0}x^1 + ... \\ &+ W_{110...0}x^1x^2 + W_{101...0}x^1x^3 + ... \\ &+ ... \\ &+ W_{11...1}x^1x^2...x^N \end{aligned} \tag{3.11}$$

As mentioned earlier, it is also possible to construct a more general feature map for continuous variables as a feature tensor [82]. In order to do this, the continuous variables $x \in [a, b]$ are discretized in $B = \frac{b-a}{\epsilon}$ bins and each bin of size ϵ corresponds to some output value

$$\phi^s(x) = F \cdot \begin{pmatrix} \pi_{[a,a+\epsilon[}(x) \\ \pi_{[a+\epsilon,a+2\epsilon[}(x) \\ \vdots \\ \pi_{[b-\epsilon,b]}(x) \end{pmatrix} \tag{3.12}$$

where $\pi_{[\alpha,\beta]}(x)$ is the rectangular function which is equal to one in the interval $[\alpha, \beta]$ and zero elsewhere, the matrix $F \in \mathbb{R}^{d\times B}$ maps this one-hot encoding to the corresponding feature vector of dimension d which is used as the input for the tensor network. These feature tensors can be trained together with the complete TN model or they can be pre-trained in combination with a linear classifier. The local feature maps mentioned until now are for continuous variables, when dealing with variables that can only take on values from some discrete set (categorical variables) one can for example use a one-hot representation, i.e. vectors with a length equal to the number of possible values in the discrete set and one of its elements equal to 1 and all others zero.

$$\phi^s(x) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{pmatrix}, \quad \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \quad \begin{pmatrix} 0 \\ 0 \\ 1 \\ \vdots \end{pmatrix}, \quad ... \tag{3.13}$$

With this tensor product feature map, a classification model is constructed as $f(\boldsymbol{x}) =$

$\langle W|\Phi(\boldsymbol{x})\rangle$, where in this work, the weight tensor W will be decomposed as matrix product state (figure 3.8). This 1-dimensional structure makes the model suitable for 1D machine learning tasks, but it has been illustrated to be powerful even for 2D tasks such as image classification [84].

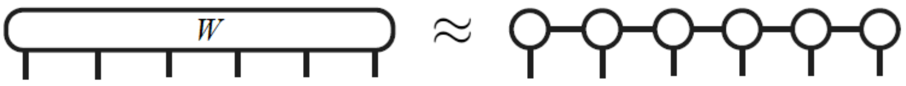


Figure 3.8: The weight tensor W as an MPS.

Note that here, contrary to MPS in quantum many-body physics, it is not a probability distribution constructed from the square of the norm of the wave function that is considered, but a general unnormalized classification function. The entries of the MPS tensors are not constrained in any sense and are able to take on all values in $\mathbb{R}$.

The classification function $f(\boldsymbol{x}) = \langle W|\Phi(\boldsymbol{x})\rangle$ described above has just one real number as output and can be used for binary classification problems. If the problem at hand has N_L different classes, the classification can straightforwardly be generalized to

$$f^l(\boldsymbol{x}) = \langle W^l|\Phi(\boldsymbol{x})\rangle \tag{3.14}$$

i.e. $f^l : \mathbb{R}^N \to \mathbb{R}^{N_L}$ and where the index l denotes the outputs for the different classes and W^l can be a set of N_L different independent tensor networks or where the l index can be attached to a single tensor in the tensor network.

3.2.2 Training the model

DMRG with gradient descent

In order to train the model defined in the section above (figure 3.8 + 3.7) one can resort to different optimization schemes. One can for example use a simple gradient descent based on one-stie or two-site DMRG [84] to update the tensors in the MPS construction of the classification function. Let us shortly describe the two-site DMRG algorithm in the context of supervised learning for a binary classification problem. Consider the general loss function

$$L(f) = \frac{1}{n}\sum_{i=1}^{n} L_i(f(\boldsymbol{x}_i), y_i) \tag{3.15}$$

which we want to minimize with respect to some training set. As in DMRG, each tensor in the MPS is updated after one another and one sweeps back and forth along the MPS until convergence is reached. Consider the tensors on site j and $j+1$, these can be contracted into a single bond tensor $B^{\alpha_{j-1}\alpha_{j+1}}_{s_j s_{j+1}} = A^{\alpha_{j-1}\alpha_j}_{s_j} A^{\alpha_j \alpha_{j+1}}_{s_{j+1}}$ and the classification function can be written as

$$f(\boldsymbol{x}) = B^{\alpha_{j-1}\alpha_{j+1}}_{s_j s_{j+1}} \widetilde{\Phi}^{\alpha_{j-1}\alpha_{j+1}}_{s_j s_{j+1}}(\boldsymbol{x}) = \langle B | \widetilde{\Phi}(\boldsymbol{x}) \rangle \tag{3.16}$$

where $\widetilde{\Phi}(\boldsymbol{x})$ denotes the input $\Phi(\boldsymbol{x})$ projected through the wings of the MPS. The gradient of the loss function with respect to this bond tensor is then given by

$$\frac{\partial L}{\partial B} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial L_i}{\partial f(\boldsymbol{x}_i)} \widetilde{\Phi}^{\alpha_{j-1}\alpha_{j+1}}_{s_j s_{j+1}}(\boldsymbol{x}_i) \tag{3.17}$$

and the bond tensor is updated in the following way

$$B \to B - \alpha \frac{\partial L}{\partial B} \tag{3.18}$$

with α the *learning rate* of the algorithm. See figure 3.9 for an overview of this updating scheme in TN notation.

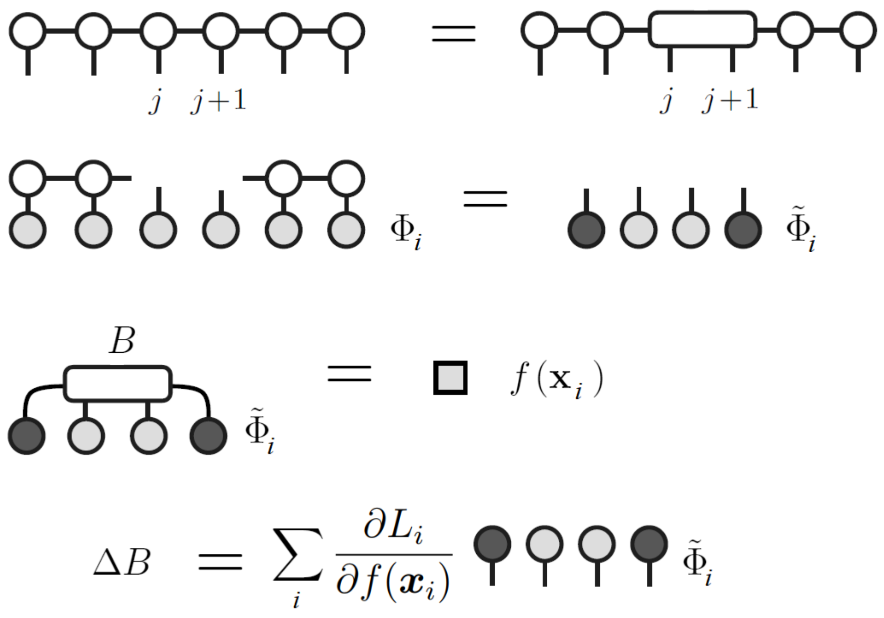


Figure 3.9: Different steps in the two-site DMRG updating scheme with gradient descent.

After this update, the tensor B is split up again into two separate tensors by means of an SVD, for which $B^{\alpha_{j-1}\alpha_{j+1}}_{s_j s_{j+1}}$ is first reshaped into a matrix with collective row index (α_{j-1}, s_j) and column index (α_{j+1}, s_{j+1}). The SVD is the given by

$$\begin{aligned} B^{(\alpha_{j+1},s_{j+1})}_{(\alpha_{j-1},s_j)} &= U^{\alpha_j}_{(\alpha_{j-1},s_j)} \Lambda^{\alpha'_j}_{\alpha_j} (V^\dagger)^{(\alpha_{j+1},s_{j+1})}_{\alpha'_j} \\ &= A^{\alpha_{j-1}\alpha_j}_{s_j} A^{\alpha_j \alpha_{j+1}}_{s_{j+1}} \end{aligned} \tag{3.19}$$

where we redefined $A_{s_j}^{\alpha_{j-1}\alpha_j} = U_{(\alpha_{j-1},s_j)}^{\alpha_j}$ and $A_{s_{j+1}}^{\alpha_j\alpha_{j+1}} = \Lambda_{\alpha_j}^{\alpha_j'}(V^\dagger)_{\alpha_j'}^{(\alpha_{j+1},s_{j+1})}$, see also figure 3.10. Whether the singular values are contracted with the left or the right tensor is a bit arbitrary here, we will see however that this has an effect on the stability and convergence of the algorithm. It is in this step that one can vary the bond dimension adaptively, by retaining only the D largest singular values (larger than some threshold ϵ) and setting the others equal to zero. Next the tensors on sites $j+1$ and $j+2$ are contracted and updated and so forth. To obtain the projected input $\widetilde{\Phi}_{s_{j+1}s_{j+2}}^{\alpha_j\alpha_{j+2}}(\boldsymbol{x}_i)$ for the new sites $j+1$ and $j+2$, the input from the left of the previous projected input $\widetilde{\Phi}_{s_js_{j+1}}^{\alpha_{j-1}\alpha_{j+1}}(\boldsymbol{x}_i)$ can be contracted with $A_{s_j}^{\alpha_{j-1}\alpha_j}$ (see figure 3.10). Doing this assures that the complete contraction of the input through the MPS is not repeated over and over again, which allows the complete algorithm to scale only linear in the size of the input N (the intermediate projections from the right have to be kept in memory when they are computed ofcourse). The scaling of the described algorithm is $O(d^3D^3nN)$ with D the largest bond dimension, d the dimension of the physical index, N the size of the input and n the size of the training set.

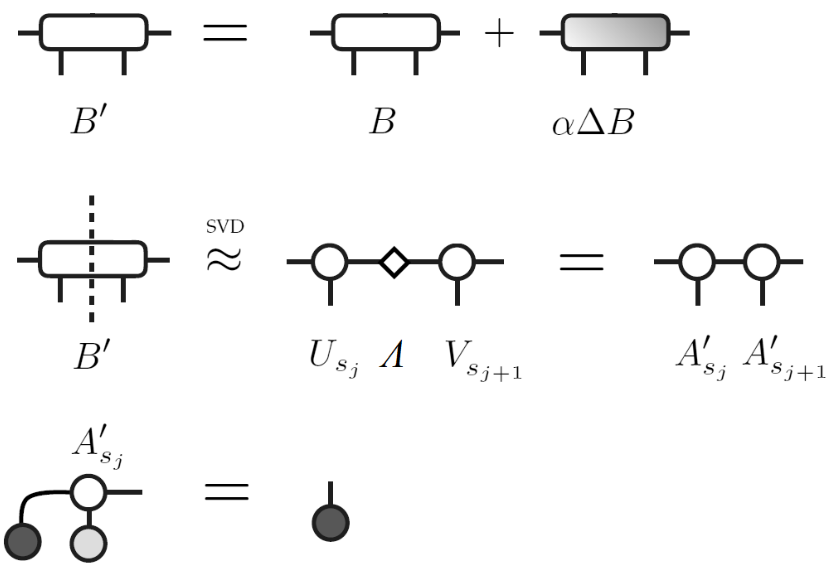


Figure 3.10: Steps of splitting the bond tensor in the two-site DMRG updating scheme.

DMRG with the Levenberg-Marquardt algorithm

A variation on this sweeping algorithm is to apply the *Levenberg-Marquardt algorithm* each step instead of the simple gradient descent step. Let us replace the bond tensor B at site j by a new estimate $B+\delta$, to first order the classification function varies as

$$f(\boldsymbol{x}_i, B+\delta) \approx f(\boldsymbol{x}_i, B) + \langle\widetilde{\Phi}(\boldsymbol{x_i})|\delta\rangle \tag{3.20}$$

with the gradient just the projected input as before:

$$\widetilde{\Phi}(\boldsymbol{x_i}) = \frac{\partial f(\boldsymbol{x}_i, B)}{\partial B} \tag{3.21}$$

Considering a quadratic loss function, under the variation δ it becomes

$$L(B+\delta) \approx \sum_{i=1}^{n} (f(\boldsymbol{x}_i, B) + \langle \widetilde{\Phi}(\boldsymbol{x_i}) | \delta \rangle - y_i)^2 \tag{3.22}$$

Writing this in matrix notation where the rows represent the different training samples $\boldsymbol{x}_i$, this can be recast as

$$\begin{aligned} L(B+\delta) &\approx ||\boldsymbol{f}(B) + \langle \boldsymbol{\Phi} | \delta \rangle - \boldsymbol{y}||^2 \\ &= (\boldsymbol{f}(B) + \langle \boldsymbol{\Phi} | \delta \rangle - \boldsymbol{y})^\top (\boldsymbol{f}(B) + \langle \boldsymbol{\Phi} | \delta \rangle - \boldsymbol{y}) \\ &= (\boldsymbol{f}(B) - \boldsymbol{y})^\top (\boldsymbol{f}(B) - \boldsymbol{y}) - 2(\boldsymbol{f}(B) - \boldsymbol{y})^\top \langle \boldsymbol{\Phi} | \delta \rangle + (\langle \boldsymbol{\Phi} | \delta \rangle)^\top \langle \boldsymbol{\Phi} | \delta \rangle \end{aligned} \tag{3.23}$$

where $\langle \boldsymbol{\Phi} | \delta \rangle$ denotes the inproduct of δ with each row of $\boldsymbol{\Phi}$. Taking the derivative with respect to δ and setting it to zero results in

$$\delta = (\boldsymbol{\Phi}^\top \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\top (\boldsymbol{y} - \boldsymbol{f}(B)) \tag{3.24}$$

More generally this is replaced by a damped version such that the update δ_j is given by

$$\delta = (\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \lambda \mathbb{I})^{-1} \boldsymbol{\Phi}^\top (\boldsymbol{y} - \boldsymbol{f}(B)) \tag{3.25}$$

with λ a tunable parameter. For $\lambda = 0$ this is just the Gauss–Newton algorithm, in the limit of large λ this update equals the gradient descent from before.

Riemannian optimization for MPS

A different way of optimizing an MPS with respect to some loss function was described in [85] which makes use of the following. One can show [87] that the following set

$$\mathcal{M}_{\boldsymbol{D}} = \{X \in \mathbb{R}^{d_1 \times d_2 \times \ldots \times d_N} | \text{TT-rank}(X) = \boldsymbol{D}\} \tag{3.26}$$

with $\boldsymbol{D} = (D_1, D_2, ..., D_{N-1})$ the effective bond dimensions of the MPS/TT (i.e. the number of non-zero singular values between any partition), forms a smooth submanifold of $\mathbb{R}^{d_1 \times d_2 \times \ldots \times d_N}$ of dimension

$$\text{dim}\mathcal{M}_{\boldsymbol{D}} = \sum_{\mu} D_{\mu-1} d_\mu D_\mu - D_\mu^2 \tag{3.27}$$

Together with the standard Euclidean inner product $\langle X, Y \rangle = X^{i_1 i_2 \ldots i_N} Y_{i_1 i_2 \ldots i_N}$ which induces an Euclidean metric on the embedding space $\mathbb{R}^{d_1 \times d_2 \times \ldots \times d_N}$, $\mathcal{M}_{\boldsymbol{D}}$ forms a Riemannian manifold. Knowing this, one can resort to *Riemannian optimization* to optimize the weight tensor W in the MPS format (from now on assumed to have a constant bond dimension $D_\mu = D \ \forall \mu$), for which the following steps are repeated until convergence:

1. The total gradient $\frac{\partial L}{\partial W}$ is projected on the tangent space of $\mathcal{M}_{\boldsymbol{D}}$ at the point W, denoted by $T_W\mathcal{M}_{\boldsymbol{D}}$. This projection is called the *Riemann gradient* and is due to linearity of the projection operator given by

$$P_{T_W\mathcal{M}_{\boldsymbol{D}}}\left(\frac{\partial L}{\partial W}\right) = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial L_i}{\partial f(\boldsymbol{x}_i)}P_{T_W\mathcal{M}_{\boldsymbol{D}}}(\Phi(\boldsymbol{x}_i)) \tag{3.28}$$

In [88] it was shown that that the TT-rank of the projection of a general tensor $Z \in \mathbb{R}^{d_1\times d_2\times \ldots \times d_N}$ on the tangent space $T_W\mathcal{M}_{\boldsymbol{D}}$ is bounded by

$$\text{TT-rank}(P_{T_W\mathcal{M}_{\boldsymbol{D}}}(Z)) \le 2\text{TT-rank}(W) = 2D \tag{3.29}$$

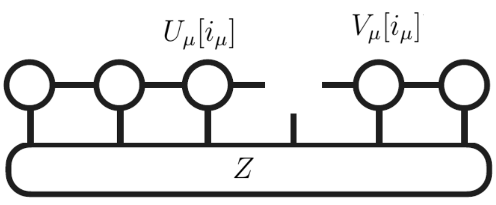
independent of Z. The same authors also proposed an algorithm to project a general tensor Z on the tangent space of $\mathcal{M}_{\boldsymbol{D}}$ at a point W, which scales as $O(ND^2\text{TT-rank}(Z)^2)$. We will use the form introduced in [89], with the weight tensor $W = A_1[i_1]A_2[i_2]...A_N[i_N]$ the projection is given by $P_{T_W\mathcal{M}_{\boldsymbol{D}}}(Z) = Y_1[i_1]Y_2[i_2]...Y_N[i_N]$ with

$$Y_1[i_1] = \begin{pmatrix}\delta U_1[i_1] & U_1[i_1]\end{pmatrix}, Y_\mu[i_\mu] = \begin{pmatrix} V_\mu[i_\mu] & 0 \\ \delta U_\mu[i_\mu] & U_\mu[i_\mu]\end{pmatrix}, Y_N[i_N] = \begin{pmatrix} V_N[i_N] \\ \delta U_N[i_N]\end{pmatrix} \tag{3.30}$$

where $U_\mu[i_\mu]$ are the left-orthogonal tensors and $V_\mu[i_\mu]$ the right-orthogonal tensors of W. The $\delta U_\mu[i_\mu]$ obey the gauge condition $(U_\mu^L)^\top \delta U_\mu^L = 0$ for $\mu = 1, ..., N-1$ and are given by

$$\begin{aligned} \delta U_\mu^L &= (\mathbb{I}_{Dd} - P_\mu)(\mathbb{I}_d \otimes W_{\le \mu-1})^\top Z^{<\mu>} W_{\ge \mu+1}, \quad \text{for } \mu = 1, ..., N-1 \\ \delta U_N^L &= (\mathbb{I}_d \otimes W_{\le N-1})^\top Z^{<N>} \end{aligned} \tag{3.31}$$

here P_μ is the projection on the range of U_μ^L i.e. $P_\mu = U_\mu^L (U_\mu^L)^\top$ and $Z^{<\mu>} \in \mathbb{R}^{(d_1 d_2 \ldots d_\mu)\times(d_{\mu+1}\ldots d_N)}$ the μth unfolding of the tensor Z. The following notation was used $W_{\le\mu} = U_1[i_1]U_2[i_2]...U_\mu[i_\mu] \in \mathbb{R}^{d_1 d_2 \ldots d_\mu \times D}$ and $W_{\ge\mu} = V_\mu[i_\mu]V_{\mu+1}[i_{\mu+1}]...V_N[i_N] \in \mathbb{R}^{D\times d_\mu d_{\mu+1}\ldots d_N}$. In TN notation this equation becomes more clear:

$(\mathbb{I}_d \otimes W_{\leq \mu-1})^\top Z^{<\mu>} W_{\geq \mu+1} \quad =$ 

2. Next a step along the Riemann gradient is taken:

$$W \rightarrow W - \alpha P_{T_W \mathcal{M}_{\boldsymbol{D}}}\left(\frac{\partial L}{\partial W}\right) \tag{3.32}$$

   Adding two tensors A and B in MPS form can be done by defining a new MPS, C with tensors given by

$$C_1[i_1] = \begin{pmatrix} A_1[i_1] & B_1[i_1] \end{pmatrix}, C_\mu[i_\mu] = \begin{pmatrix} A_\mu[i_\mu] & 0 \\ 0 & B_\mu[i_\mu] \end{pmatrix}, C_N[i_N] = \begin{pmatrix} A_N[i_N] \\ B_N[i_N] \end{pmatrix} \tag{3.33}$$

   It is clear that $C_1 C_2 ... C_N = A_1 A_2 ... A_N + B_1 B_2 ... B_N$. Multiplying an MPS with a scalar is trivial, one can for example just multiply one of the tensors in the MPS with it.

3. Due to the step, the new W no longer lies on the manifold $\mathcal{M}_{\boldsymbol{D}}$, it has a larger bond dimension (with upper bound $2D$). One can retract back to the lower dimensional manifold by doing a sweep of SVD's and attaining only the D largest singular values (a *TT-rounding*).

For an illustration of these three steps, see figure 3.11.

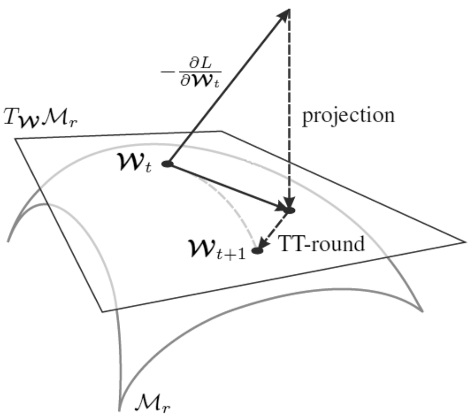

$-\frac{\partial L}{\partial \mathcal{W}_t}$
$T_{\mathcal{W}}\mathcal{M}_r$
projection
$\mathcal{W}_t$
$\mathcal{W}_{t+1}$
TT-round
$\mathcal{M}_r$

Figure 3.11: Visual representation of Riemannian gradient descent [85].

3.3 Number-state preserving TN's for supervised learning

Only in the last weeks of the writing of this thesis a newly published paper [90] came to our attention, describing a more restricted class of tensor networks as models for supervised learning. This method will be presented in more detail below.

In the methods of constructing classifiers from TN's and optimizing them with gradient descent based algorithms described above, the tensors in the network have no constraints whatsoever, their entries can take on all values in $\mathbb{R}$ (as opposed to $\mathbb{C}$ in the description of quantum many-body systems). In [90] the author proposes to build supervised models from a restricted class of tensors, namely *number-state preserving tensors.* The idea is to obtain more sparse tensors/models, which allow more interpretability and are better suited for the problem of classical data than their less restricted quantum counterparts. Moreover working with number-state preserving tensors makes contracting and optimizing more complicated tensor networks such as MERA computationally feasible, which is not the case for a networks constructed from general tensors.

3.3.1 Number-state preserving tensor networks

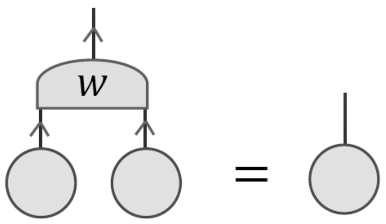
Imagine a lattice of N sites, with each site living in a local Hilbert space of dimension d. The basis states can be labeled by integers $|\phi\rangle \in \{|0\rangle, |1\rangle, ..., |d-1\rangle\}$ which are interpreted as particle numbers and are represented by unit vectors (i.e. one-hot representation)

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \quad ... \tag{3.34}$$

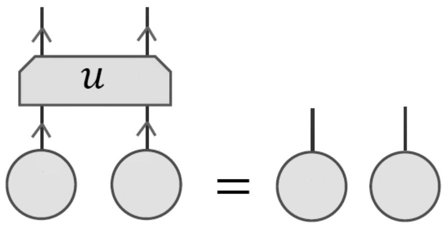
A number-state on the lattice is then defined as a product state with a well-defined particle number

$$|\Phi\rangle = |\phi_1\rangle|\phi_2\rangle...|\phi_N\rangle, \quad \text{or} \quad \Phi = \phi_1 \otimes \phi_2 \otimes ... \otimes \phi_N. \tag{3.35}$$

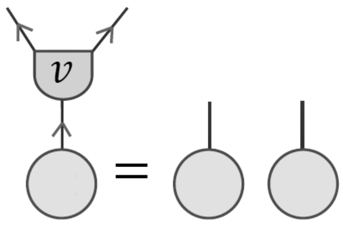
Now consider transformations of number states on an input lattice to states on an output lattice, which will be represented by oriented tensors with incoming and outgoing legs, corresponding with the input and output lattice. Such an oriented tensor is defined as number-state preserving if it maps any number-state on the input lattice to another number-state on the output lattice. It is then immediately clear that networks constructed from such tensors as a whole (with the outputs of one tensor correctly matched with the inputs of others) are number-state preserving. A few examples of number-state preserving tensors are given below (with $d = 2$). The matrix representation is obtained when taking the outgoing legs together as a collective row index and the incoming legs as the collective column index. The identification of the rows and columns of this matrix with the output and input makes it clear that in order for the mapping to be number-state preserving, each column of the corresponding matrix must have at most 1 non-zero entry. Note that this definition also allows mappings with columns containing only zeros, corresponding to mapping some number-states to the null state.



$$\begin{array}{l} |0\rangle|0\rangle \mapsto |0\rangle \\ |0\rangle|1\rangle \mapsto |1\rangle \\ |1\rangle|0\rangle \mapsto |1\rangle \\ |1\rangle|1\rangle \mapsto |0\rangle \end{array}, \qquad w = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$



$$\begin{array}{l} |0\rangle|0\rangle \mapsto |0\rangle|0\rangle \\ |0\rangle|1\rangle \mapsto |1\rangle|0\rangle \\ |1\rangle|0\rangle \mapsto |1\rangle|0\rangle \\ |1\rangle|1\rangle \mapsto |1\rangle|1\rangle \end{array}, \qquad u = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



$$\begin{array}{l} |0\rangle \mapsto |0\rangle|0\rangle \\ |1\rangle \mapsto |1\rangle|1\rangle \end{array}, \qquad v = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$$

One can thus construct number-state preserving versions of well known tensor networks such as MPS and MERA. Note however that these number-state preserving tensor networks do not remain number-state preserving if the orientation of the indices is reversed; in that case these reversed networks can generate superpositions and entangled states. In the remainder only *unital* number-state preserving tensors will be considered, these are tensors where the non-zero entries are exactly one and where each column of the corresponding input-output matrix has exactly one non-zero element (of which the above mentioned mapping are examples). These unital tensors map incoming states to outgoing states of the same normalization and phase.

3.3.2 Supervised number-state preserving model

In order to construct a supervised model, each sample $\boldsymbol{x}_i = (x_i^1, x_i^2, ..., x_i^N)$ of a dataset are mapped to a product state with definite particle particle number $|\Phi(\boldsymbol{x}_i)\rangle$, i.e. of the form (3.35) with basis (3.34); continuous variables can be discretized to this end. The corresponding labels y_i are also mapped to a number-state $|y_i\rangle$ (one-hot representation) with a dimension equal to the number of classes. Next a tensor network $\mathcal{T}$ is constructed out of number-state preserving tensors such that the output is a vector $|f(\boldsymbol{x}_i)\rangle = \mathcal{T}|\Phi(\boldsymbol{x}_i)\rangle$ which is equal to $|y_i\rangle$ in the ideal case. Since only unital number-state preserving mappings are considered, the inproduct $\langle y_i|f(\boldsymbol{x}_i)\rangle$ is either unity or zero, corresponding with correct or incorrect classification. The number of correct classified samples is then given by

$$C = \sum_{i=1}^{n} \langle y_i|\mathcal{T}|\Phi(\boldsymbol{x}_i)\rangle \tag{3.36}$$

which will be used as cost/loss. The tensors in the tensor network $\mathcal{T}$ thus should be optimized as to maximize C.

Now imagine we want to update a tensor A in the network, with an environment $\mathcal{T}_A$ (the fully contracted TN, including the contraction between the output index and the label state $|y_i\rangle$, with tensor A left out and summed over the n training samples). The number of correct classified samples is then given by the contraction of A with its environment:

$$C = \langle A|\mathcal{T}_A\rangle \tag{3.37}$$

In order to maximize this scalar product, one should place a one in each column of the matrix representation of A, in the location corresponding with the maximum element of each column in the environment $\mathcal{T}_A$. Let us give a example by considering the matrix representation of an environment and the matrix that maximizes (3.37):

$$\mathcal{T}_A = \begin{pmatrix} 3 & 11 & 29 & 53 \\ 2 & 19 & 31 & 43 \\ 7 & 13 & 37 & 41 \\ 5 & 17 & 23 & 47 \end{pmatrix} \quad \Rightarrow \quad A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{3.38}$$

The number of correct labels in this example is $C = 53 + 19 + 7 + 37 = 116$.

This method of optimization thus directly jumps to the optimal single tensor A, it is however still possible to get trapped in a local maximum of the complete tensor network. To avoid this, some randomness will be introduced in the optimization procedure, inspired by Monte Carlo methods. Rather than jumping to the optimal tensor, sub-optimal solutions will be allowed with a probability decaying exponentially in comparison with how far this solution lies from the optimal one. To this end a difference matrix Ω is defined, obtained by subtracting from each column of $\mathcal{T}_A$ the maximum of that column

$$\Omega_{ij} = (\mathcal{T}_A)_{ij} - \max_i (\mathcal{T}_A)_{ij} \tag{3.39}$$

This difference matrix is then used to construct a matrix of transition probabilities P, defined element-wise as

$$P_{ij} = \frac{e^{\Omega_{ij}/T}}{\sum_i e^{\Omega_{ij}/T}} \tag{3.40}$$

with T a hyperparameter that controls the amount of randomness that is introduced, for $T \to 0$ P tends to the optimal solution for A and for $T \to \infty$ all transition probabilities will be the same and the process is completely random. For the example given above, the difference matrix and transition probabilities are given by (with $T = 3$)

$$\Omega = -\begin{pmatrix} 4 & 8 & 8 & 0 \\ 5 & 0 & 6 & 10 \\ 0 & 6 & 0 & 12 \\ 2 & 2 & 14 & 6 \end{pmatrix} \qquad \Rightarrow \qquad P = \begin{pmatrix} 0.13 & 0.04 & 0.06 & 0.84 \\ 0.10 & 0.58 & 0.11 & 0.03 \\ 0.51 & 0.08 & 0.82 & 0.02 \\ 0.26 & 0.30 & 0.01 & 0.11 \end{pmatrix} \tag{3.41}$$

The transition matrix is then used to perform a stochastic update of A, the probabilities in each column of P determine how likely the unity of each column will be set to that particular location.

As a final remark, we mention that the number-state preserving tensors allow the environments $\mathcal{T}_A$ to be determined efficiently for more complex tensor networks such as MERA, which would be computationally unfeasible in the general case. The requirement for this to be possible is that the maximum width of the causal cones in the network is not too large. The tensor network we will consider is an MPS, for which the exact procedure of computing the environments in the case of number-state preserving tensors is not of great importance (remember $\mathcal{T}_A = \sum_i \widetilde{\Phi}(\boldsymbol{x}_i)$). For a detailed explanation of how these contractions can be carried out efficiently in the case of number-state preserving tensors for general networks, we refer to the original paper [90].

Chapter 4

Applications

The models and optimization algorithms described in this work were implemented in python and standard packages such as numpy and scipy were used.

4.1 TIS prediction

4.1.1 TIS in DNA

DNA (deoxyribonucleic acid, see figure[1] 4.1) is a large molecule composed of two interconnected strands forming a double helix and carries the genetic code used in the growth, development, functioning, and reproduction of all known living organisms. The two strands (known as polynucleotides) are made up of four nucleotides, each consisting out of a sugar (called deoxyribose), a phosphate group and one of four nitrogen-containing nucleobases: adenine (A), cytosine (C), guanine (G) or thymine (T). The nucleotides are joined to one another by covalent bonds between the sugar of one nucleotide and the phosphate of the next, the two separate strands are in turn bound by hydrogen bonds between the nitrogenous bases, according to base pairing rules (A with T and C with G), to form the double helix structure.

One could say that DNA contains all the biological information needed for the functionality of lifeforms. This information is extracted and used to create proteins (providing structure to cells etc.), in a process which is described by the framework known as *the central dogma of molecular biology* [91]. The conversion of DNA to proteins takes place in two main steps, the first being *transcription*. In transcription the information contained in a section of DNA is replicated in the form of a newly assembled piece of mRNA (messenger RNA), see figure[2] 4.2. RNA (ribonucleic acid) is, just like DNA, built up from nucleotides, based on the nitrogenous bases: adenine (A), cytosine (C), guanine (G) and

[1]https://commons.wikimedia.org/wiki/File:DNA_chemical_structure.svg

[2]https://www.khanacademy.org/science/biology/gene-expression-central-dogma/transcription-of-dna-into-rna/a/overview-of-transcription

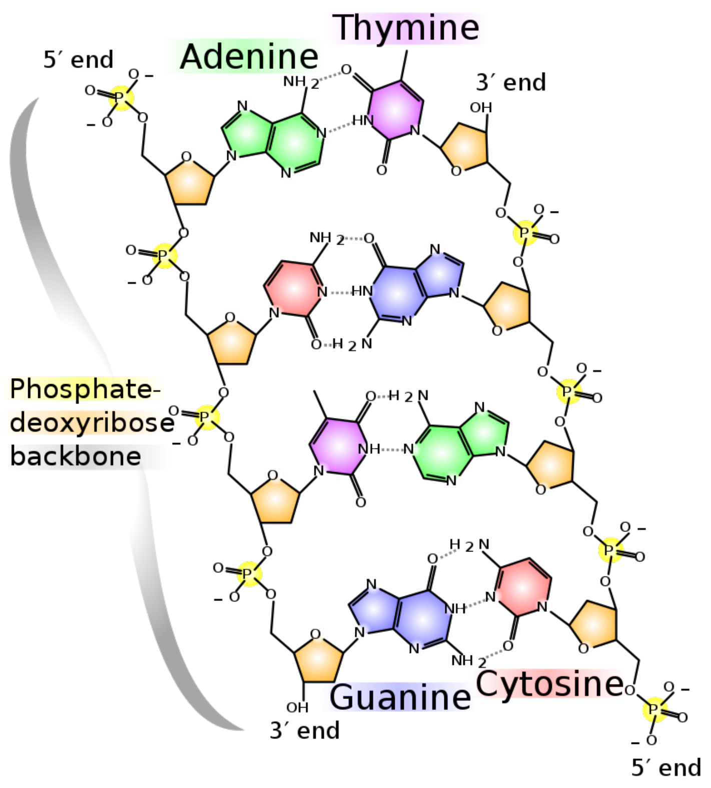


Figure 4.1: Chemical structure of DNA.

uracil (U), but is unlike DNA, more often found as a single strand folded onto itself. After this transcription, a process that goes under the name of splicing takes place, where introns (non-coding DNA) are removed en exons are joined together[3].

Once this piece of (single strand) mRNA is created, it can proceed to the second step, known as *translation* (figure[4] 4.3). This takes place in a structure known as a ribosome, which organize translation and catalyze the reaction that joins amino acids to make a protein chain. Now the mRNA is read in triplets of nucleotides, known as *codons*, whilst the ribosome runs along the mRNA strand. Codons carry the information of how the proteins should be build; as they consist of 3 nucleotides there are 64 distinct codons, 61 decoding 20 different amino acids and 3 of them are stop codons: UAG, UAA and UGA (corresponding to TAG, TAA and TGA in DNA respectively). A stop codon signals when the translation into proteins should be terminated. The reading of the codons in the mRNA is done by means of a molecule called tRNA (transfer RNA). Each tRNA strand is folded into itself in a complicated shape, with in the middle an anticodon which will attach to the corresponding codon in the mRNA (according to the base pairing rules: A to U and C to G). At the other end of the tRNA, the fitting amino acid is attached

[3]The intron part in a strand of DNA can constitute more than 98% for humans.

[4]https://www.khanacademy.org/science/biology/gene-expression-central-dogma/translation-polypeptides/a/translation-overview

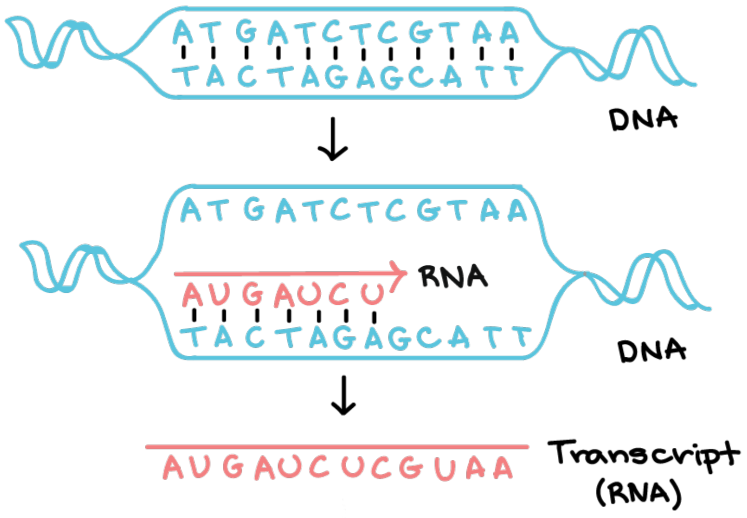


Figure 4.2: Transcription: information from DNA is replicated in a piece of pre-mRNA, after which splicing takes place.

and at each step of reading a new codon, the amino acid is joined with the rest of the chain of amino acids, ultimately resulting in a protein.

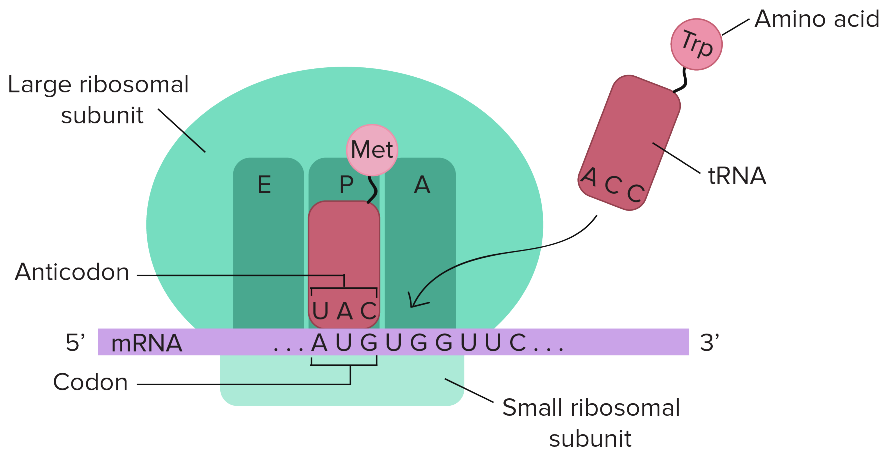


Figure 4.3: Translation of the mRNA into a protein taking place in the ribosome.

Where the ribosome should attach first and thus where translation is initiated is indicated by a start codon AUG, corresponding to ATG in DNA before transcription. This site goes by the original name of *translation initiation site* (TIS). The correct identification of these true TIS (in the non-transcript DNA) is a challanging problem [6] as many more pseudo-TIS occur. These pseudo-TIS can arise in different ways, the first being the large fraction of introns in the DNA. Introns are non-coding (”junk” DNA) and carry no meaning, a lot of false ATG codons can appear in these parts. A second way pseudo-TIS appear is the misalignment of the codons, for example in the sequence TCG A**AT G**AC, where an overlap between two real codons is read as a start codon. We thus have a classification problem, with as input the genomic data[5] of a DNA sequence and as output one of two classes: true-TIS or pseudo-TIS.

4.1.2 MPS for TIS prediction

So the goal is, given some context around a candidate TIS, to determine whether it is a true TIS or not (pseudo-TIS). The region preceding the ATG (i.e. on the left) is also called the upstream u and the region following after the ATG-codon (i.e. on the right) is called the downstream d, u and d can be considered hyperparameters (with some upper bound) that determine how much of the context around the TIS is fed into the model. Some examples are given below, the first three being true-TIS and the last three being pseudo-TIS.

...CCTAATCCCGCCTTGGCC **ATG** AGGGAGATCGTGCTCACG...
...TCCCTACCAACCGACACC **ATG** AACACCATCGTCTTCAAC...
...GCGCGGCGCCCGCGGAGC **ATG** GCGGACCGCAGCCTGGAG...
...CTACTGCCTTTCCTGAGA **ATG** GGCAGGAGCCACCTGCAG...
...GCGGCCCTCCCCTCGCTC **ATG** CCCTCCCGCCCCACGCAG...
...GCGGCCCTCCCCTCGCTC **ATG** CCCTCCCGCCCCACGCAG...

Two datasets are used in this experiment, the first one is compiled from the consensus CDS (CCDS) database[6] and was used to train the model. The CCDS project is a collaborative effort to identify a core set of human protein coding regions that are consistently annotated and of high quality. Annotation updates represent genes that are defined by a mixture of manual curation and automated computational processing. The CCDS database was downloaded, which contains 13 917 genes, and the pseudo-TIS were defined as all ATG-codons appearing in a window of 1000 upstream and 1000 downstream of the actual TIS of each gene. Furthermore, all genes from chromosome 21 were discarded, to ensure a non-biased validation on this chromosome (see further). The second dataset consists of human chromosome 21, for which the sequence and annotation were downloaded from Ensembl[7] (based on NCBI build 36 version 1) and was only used to test the model. This dataset contains 294 genes, 258 of which have a consensus TIS (i.e. the triplet ATG). These ones were chosen as the positive examples of the dataset, while the remaining ATGs were included as negative examples.

Before throwing an MPS at this problem, let us analyze the data by looking at the *mutual information* (see appendix) between each pair of sites in the context around the

[5]Working on a genomic scale means the TIS identification is done before the DNA is transcribed, i.e. before the introns are removed.
[6]http://www.ncbi.nlm.nih.gov/CCDS/
[7]http://www.ensembl.org

TIS (figure 4.4). It is clear that the collection of DNA sequences containing a true TIS (right) has a lot more structure than the ones where it's just a pseudo-TIS (left). In the true TIS case, a clear signal with period 3 is visible behind the ATG codon, a consequence of codons being represented by triplets of nucleotides. For pairs in front of the ATG a rather random signal is found, which is to be expected in the upstream as it most likely consists of introns.

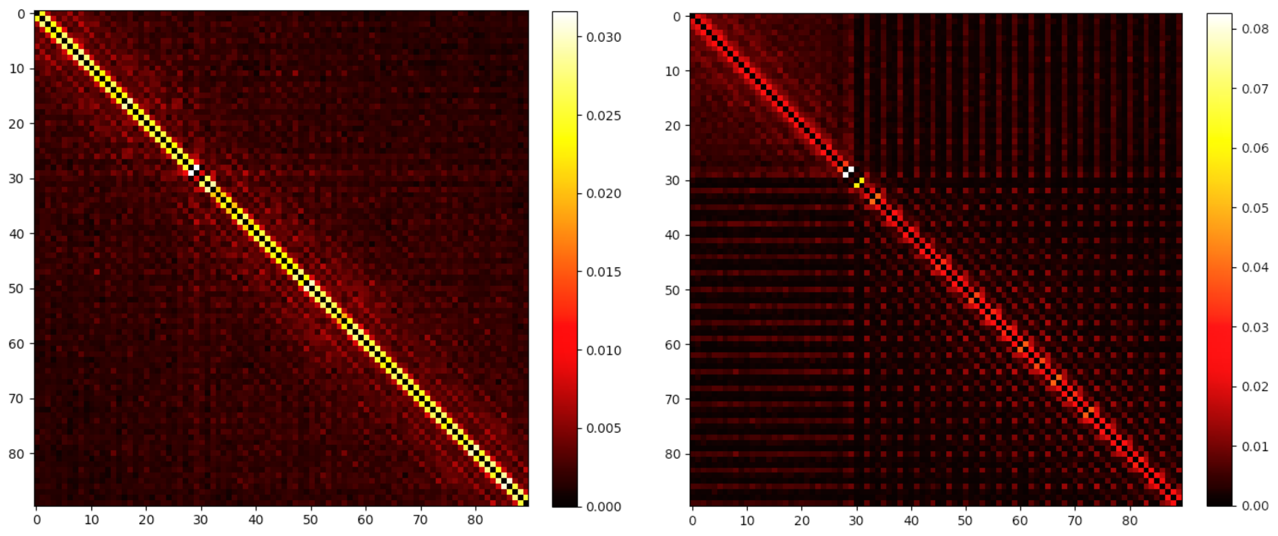


Figure 4.4: The mutual information between each pair of sites in the DNA sequence around the ATG codon, located at site 30 (the ATG is left out). On the left the mutual information in the pseudo-TIS context and on the right the mutual information in the true TIS context.

In order to construct an MPS-based model, the labels are defined as $y_i = 1$ for true TIS and $y_i = 0$ for pseudo-TIS and the following local feature map is defined

$$\phi^s(A) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \phi^s(C) = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad \phi^s(G) = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad \phi^s(T) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{4.1}$$

4.1.3 Some stability issues

As a first method of optimization, one-site DMRG was used in combination with the loss function used in [84] and [85], namely the simple quadratic loss function

$$L = \frac{1}{2n}\sum_{i=1}^{n}(f(\boldsymbol{x}_i) - y_i)^2 \quad \Rightarrow \quad \frac{\partial L}{\partial A_{s_j}^{\alpha_{j-1}\alpha_j}} = \frac{1}{n}\sum_{i=1}^{n}(f(\boldsymbol{x}_i) - y_i)\widetilde{\Phi}_{s_j}^{\alpha_{j-1}\alpha_j}(\boldsymbol{x}_i) \tag{4.2}$$

The problem with this is that it is unstable and the gradient is prone to fluctuations which cause it to explode during sweeping. This can be caused by the fact that we are working with a product of many matrices. To illustrate this, consider the case of a function f decomposed as the product of N 1×1 matrices (i.e. scalars), with N large

$$f = a_1 a_2 ... a_N \tag{4.3}$$

which we want to minimize by varying one scalar a_i at a time. Assume that the a_i are initialized with a constant value c with some Gaussian noise added on top, such that $f(0) \sim 1$

$$f(0) = a_1(0) a_2(0) ... a_N(0), \tag{4.4}$$

Imagine that, at a given point during sweeping, the absolute value of $f(0)$ becomes large, of the order $R \gg 1$. The update for the first scalar is given by (in the following we will denote $a_i(0)$ with a_i)

$$\Delta a_1 = -\alpha \frac{\partial f}{\partial a_1} = -\alpha a_2 ... a_N \sim \frac{\alpha R}{a_1} \tag{4.5}$$

If N is large a_1 will be of the order $R^{\frac{1}{N}} \sim 1$ and thus by updating, $a_1(1)$ will become of order $\frac{\alpha R}{a_1}$ (if α is not to small, such that $\Delta a_1 > a_1$) and now f will be of order

$$f(1) \sim \frac{\alpha R^2}{a_1^2}. \tag{4.6}$$

Repeating this, sweeping to the right, we get the following (using $\Delta a_n \sim \frac{\alpha f(n-1)}{a_n}$ and $f(n) \sim \frac{\Delta a_n f(n-1)}{a_n}$)

$$\begin{aligned}
&\Delta a_1 \sim \frac{\alpha R}{a_1} \Rightarrow f(1) \sim \frac{\alpha R^2}{a_1^2} \\
&\Delta a_2 \sim \frac{\alpha^2 R^2}{a_1^2 a_2} \Rightarrow f(2) \sim \frac{\alpha^3 R^4}{a_1^4 a_2^2} \\
&\Delta a_3 \sim \frac{\alpha^4 R^4}{a_1^4 a_2^2 a_3} \Rightarrow f(3) \sim \frac{\alpha^7 R^8}{a_1^8 a_2^4 a_3^2} \\
&... \\
&\Delta a_n \sim \frac{(\alpha R)^{2^{n-1}}}{a_1^{2^{n-1}} a_2^{2^{n-2}} ... a_n} \Rightarrow f(n) \sim \frac{1}{\alpha} \frac{(\alpha R)^{2^n}}{a_1^{2^n} a_2^{2^{n-1}} ... a_n^2}
\end{aligned} \tag{4.7}$$

Assuming $a_i \sim R^{\frac{1}{N}}$ for all i, the denominator becomes of order

$$\sim R^{\frac{1}{N}(2^n + 2^{n-1} + ... + 2)} = R^{\frac{1}{N}(2^{n+1} - 1)} \tag{4.8}$$

We thus see that after n updates, f scales in the following way

$$f(n) \sim \frac{1}{\alpha}\alpha^{2^n} R^{2^n - \frac{1}{N}(2^{n+1}-1)} \approx \frac{1}{\alpha}\alpha^{2^n} R^{2^n(1-\frac{2}{N})} \approx \frac{1}{\alpha}(\alpha R)^{2^n} \tag{4.9}$$

i.e. super exponential in the number of updates, and so will the gradient. Once f reaches a value slightly to large, the fluctuation grows and the MPS blows up. This effect is even worse when minimizing a quadratic loss

$$L = \frac{1}{2}(f-y)^2. \tag{4.10}$$

due to the extra factor $(f-y)$ in the gradient. The case of matrices is more complicated but similar effects take place.

We could set α small enough to keep the updates under control, but this in turn causes very slow convergence. It's also possible to add an explicit regularization term to the loss function to try to keep the tensors under control. For example of the form

$$L(f) = L_0(f) + \frac{\lambda}{2}\sum_{j=1}^{N}\sum_{s_j\alpha_{j-1}\alpha_j}(A_{s_j}^{\alpha_{j-1}\alpha_j})^2 \tag{4.11}$$

resulting in an extra weight decaying term in the gradient

$$A_{s_j}^{\alpha_{j-1}\alpha_j} \to (1-\alpha\lambda)A_{s_j}^{\alpha_{j-1}\alpha_j} - \alpha\frac{\partial L_0}{\partial A_{s_j}^{\alpha_{j-1}\alpha_j}} \tag{4.12}$$

This merely postponed the inevitable divergence of the gradient. Another possibility to counter this exploding gradient problem is to do a 'gauge transformation' each step. One can for example do an SVD after each update

$$A_{s_j}^{\alpha_{i-j}\alpha_j} = U\Lambda V^\dagger \tag{4.13}$$

where (α_{j-1}, s_j) is treated as a collective row index. Next one defines $A'_{s_j} = U$ (using only the first D columns, since Σ is a diagonal $Dd \times D$ matrix) and $A'_{s_{j+1}} = \Lambda V^\dagger A_{s_{j+1}}$, such that the matrices on the left are semi-unitary and all the large numbers are included in $A'_{s_{j+1}}$ (via Λ), which is the next tensor to be updated. The gradient of $A'_{s_{j+1}}$ is thus confined, since it doesn't directly depend on $A'_{s_{j+1}}$ itself. Testing this gauge showed that it resulted in vanishing gradients, which could somewhat be expected, since all the tensors on the left contain numbers smaller than one.

The trick that solved this exploding gradient problem was to use a different loss function that suppresses the exponential growth of fluctuations. The one we used is given by

$$L(f) = \frac{1}{2n}\sum_{i=1}^{n}\log((f(\boldsymbol{x}_i) - y_i)^2 + b) \tag{4.14}$$

with b a hyperparameter of order of magnitude 1. Since this is a smooth function its minimum is quadratic and this loss function will behave in the same way as the quadratic loss function for small deviations. The one-site gradient is now given by

$$\frac{\partial L}{\partial A_{s_j}^{\alpha_{j-1}\alpha_j}} = \frac{1}{n}\sum_{i=1}^{n} \frac{f(\boldsymbol{x}_i) - y_i}{(f(\boldsymbol{x}_i) - y_i)^2 + b} \widetilde{\Phi}_{s_j}^{\alpha_{j-1}\alpha_i}(\boldsymbol{x}_i). \tag{4.15}$$

For a large deviation from the minimum at y_i, the large values in $\widetilde{\Phi}(\boldsymbol{x}_i)$ are kept under control by the factor $\sim \frac{1}{(f(\boldsymbol{x}_i)-y_i)}$. For a small deviation from the minimum we indeed get the same gradient as in the case of the quadratic loss function.

4.1.4 Results of gradient descent for MPS

As an optimization scheme, two-site DMRG is used in combination with the loss function defined in (4.14) as this gave the best results and good convergence. Here, after each update and SVD of the bond tensor, the singular value matrix is contracted with the tensor on the left (when sweeping to the right), in contrast to what is usually done when working in the canonical form. This is to avoid the vanishing gradient problem that is encountered when contracting the singular values on the right (in the direction of sweeping) resulting in orthogonal tensors everywhere. As orthogonal tensors contain numbers smaller or equal to one, the environment of the tensor to be updated becomes very small, i.e. the gradient is greatly suppressed (through $\widetilde{\Phi}$).

The MPS was initialized with the same constant value in all entries (i.e. an effective bond dimension of 1), such that $f \sim 0.1$, this gave better results than the initialization with Gaussian noise. The model was trained on 20000 training samples and validated on 2000 validation samples (both half true TIS half pseudo-TIS); an upstream of 30 and a downstream of 70 is considered. The validation set was used to determine when to stop training, namely when the loss on the validation set started rising again (this is sometimes referred to as *early stopping*). The trained model was then tested on 100000 test samples from the human chromosome 21 set (containing only 258 true TIS samples).

The evolution of the root-mean-square error[8] (RMSE) on the training and validation sets while sweeping is given in figure 4.5 below for two different sizes n of training set. One can observe that the loss has an ever decreasing trend on the training set, but that already after a few sweeps (if the learning rate α is chosen appropriately) the loss on the validation set reaches a minimum. This minimum in the validation loss then signals when to stop training, when going past this minimum one enters the overfitting regime. It is also apparent that for small training sets the data can be fitted really well, but evaluat-

[8]RMSE $= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(f(\boldsymbol{x}_i) - y_i)^2}$

ing on an independent dataset shows that there is a lack of generalizability. For larger training sets the loss on the training set and the validation set lies closer and the model has a better generalization. This behaviour is typical in machine learning and illustrates the need for data to suppress overfitting.

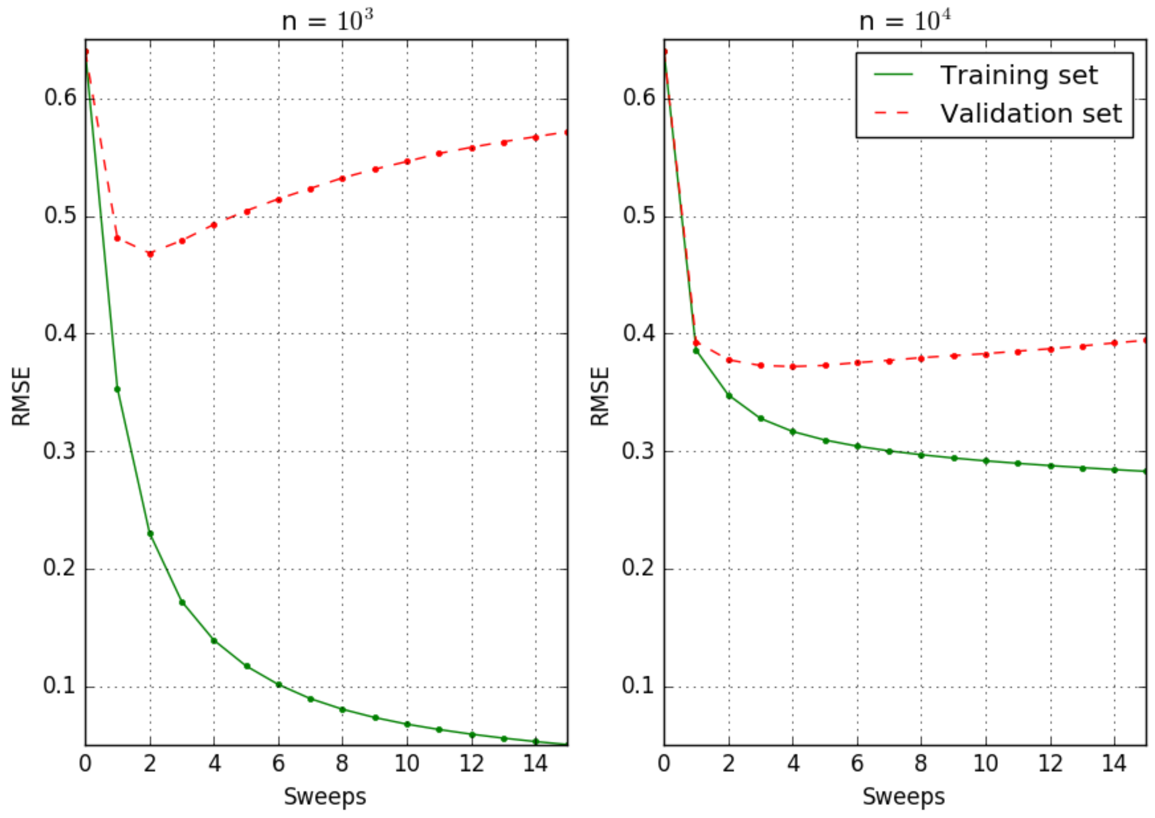


Figure 4.5: The evolution of the RMSE while sweeping over the MPS ($D = 5$) on the training set and validation set. On the left a training set of size $n = 10^3$, on the right a training set of size $n = 10^4$.

To quantify performance two metrics are considered: Se80, which is often used in the context of TIS prediction and the area under the ROC curve (AUC). In the table below these two metrics are given for models with different bond dimensions. It is apparent that the performance of the MPS at first increases with the bond dimension, but that this trend already ceases at a mere bond dimension of 6-7 (considering AUC as it is a more robust measure than Se80). Increasing the bond dimension further results in a better classification of the training set, but no further improvement on the independent test set. The test set accuracy even worsening with higher bond dimensions. We thus have a problem with the generalizability of the model for higher bond dimensions. The corresponding ROC curves for bond dimensions 1 to 7 are given in figure 4.6, which illustrates the initial improvements by increasing the bond dimension.

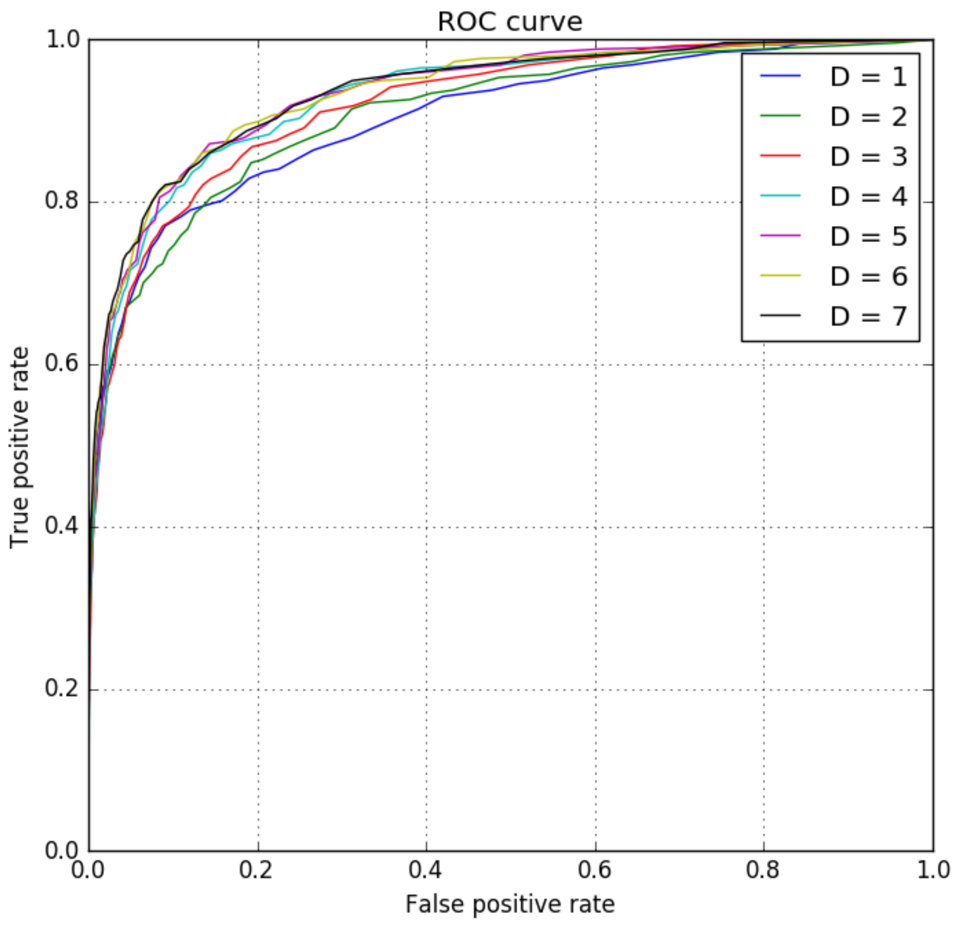


Figure 4.6: The ROC curves for MPS with different bond dimensions, trained on 20000 training samples and tested on 100000 test samples with $u = 30$ and $d = 70$.

Next two-site DMRG was used in combination with the Levenberg-Marquardt algorithm, as described in section 3.2.2. The performance on the training became very high very fast, especially for higher bond dimensions. On an independent validation set however the performance does not increase whatsoever when compared to a bond dimension of 1, unless the parameter λ is set high enough, making the algorithm effectively a slow converging simple gradient descent. The Levenberg-Marquardt algorithm (with small λ) thus quickly results in overfitting and is not very suited for this problem.

As a last method of optimizing, the Riemann optimization scheme was implemented, this however did not seem able to lower the training set loss at all for this problem of TIS classification.

Let us compare the MPS model with two models that were studied in the bioinformatics community. In an older work [6] a model (dubbed *StartScan*) for TIS prediction on the genomic scale was constructed by combining 3 conceptually simple models. The au-

Bond dimension D	Se80	AUC
1	0.153	0.9045
2	0.140	0.9088
3	0.122	0.9202
4	0.095	0.9298
5	0.083	0.9342
6	0.075	0.9348
7	**0.076**	**0.9359**
8	0.076	0.9332
9	0.076	0.9332
10	0.074	0.9325
20	0.094	0.9293
40	0.093	0.9259

Table 4.1: Se80 and AUC for different bond dimensions evaluated on an independent test set. The best model is obtained at a bond dimension of ~ 7, consistent with what was observed on the validation set when optimizing.

thors trained and tested their model on the same datasets that were used here. The best results were obtained for $u = 60$ and $d = 140$. In a more recent work [92] a convolutional neural network (CNN) was applied to this problem of TIS prediction (in combination with the same mapping used here (4.1)). Their model was trained and tested on the same datasets used in [6] with $u = 60$ and $d = 140$ and the authors achieved state-of-the-art performance. To compare the MPS model performance with these two models, we also considered $u = 60$ and $d = 140$ and trained the model on all positive samples form the CCDS dataset (13917) and an equal amount of negative samples from this dataset (10 % of the training set was exclusively used for validation). An optimum was found for a bond dimension of 6. The performance in terms of Se80 and AUC for the StartScan model and the CNN are given in the table below together with the results of the MPS. The ROC curves of the MPS in the CNN are plotted in figure 4.7.

approach	Se80	AUC
StartScan	0.125	
CNN	0.031	0.9757
MPS$_{D=6}$	0.069	0.9392

As can be seen, our MPS based model performs better than the StartScan model for the classification of hypothetical TIS, but it is clearly outperformed by the CNN.

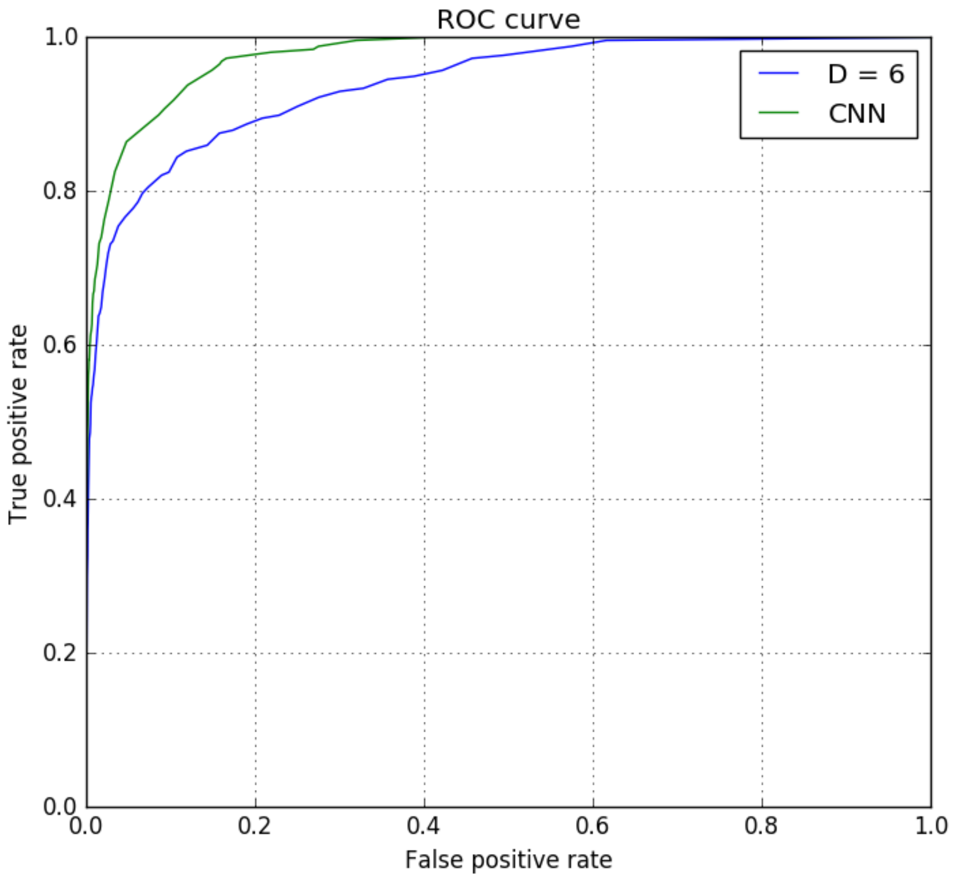


Figure 4.7: The ROC curves for the optimal MPS ($D = 6$) and the CNN from [92] tested on human chromosome 21.

Analyzing the optimized MPS

To analyze the obtained MPS by training we can look at the singular values of each tensor (after reshaping it into a matrix by taking the external index together with one of the virtual indices into a collective index). The ability to interpret theses SVD's depends on the optimization scheme used. Here one-site DMRG will be considered, as the repeated SVD's in two-site DMRG make the interpretation of the singular values of the tensors in the MPS somewhat difficult. The parameters used are: $D = 5$, $u = 50$, $d = 60$ and $n = 20000$. Let us first plot the first singular value for each site of the MPS, the result is given in figure 4.8. Two things are notable, firstly a signal with a period of 3 is apparent. This could be a result of the nucleotides coding information in groups of 3 (trinucleotides), these are the codons in the DNA (ATG, TAA,...). Secondly these patterns reach further in the downstream side than they do in the upstream side of the sequence, possibly due to the fact that the upstream is more likely to consist of introns.

Now let us consider the region around the ATG codon in more detail. To get an idea of the contribution made by each nucleotide (A, C, G and T) at the sites around the

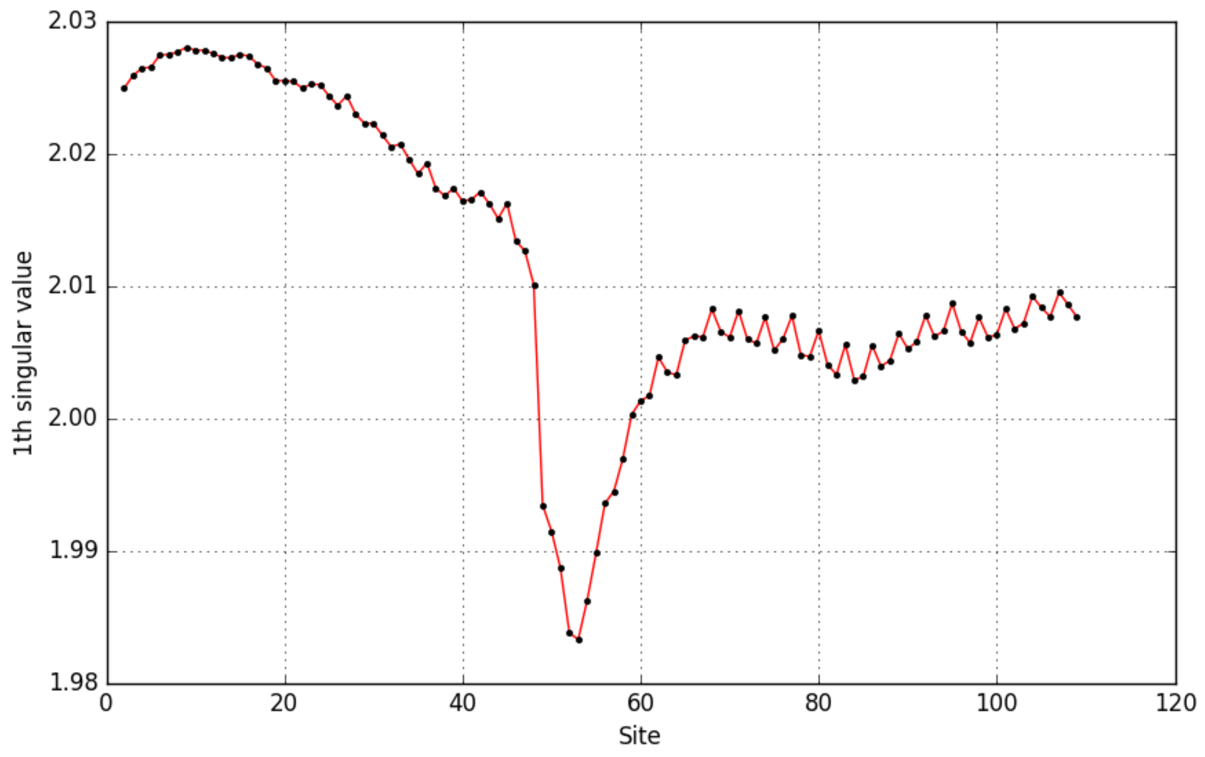


Figure 4.8: The first singular value of each MPS tensor, the ATG codon is located at site 51.

hypothetical TIS, the singular values of the tensors around the ATG codon are computed for each physical index of the tensor, i.e. $A^{\alpha\beta}_{s=N} = U\Lambda V^{\dagger}$ with $N \in \{A, C, G, T\}$ (such that the row corresponds with virtual index α and the columns with virtual index β). The result for the first singular value is shown in figure 4.9. This is exactly the *Kozak consensus sequence* [93] which is based on the probability of each nucleotide appearing around the TIS, the true Kozak sequence is shown in figure 4.10.

It is not very surprising that this Kozak sequence is retrieved with the one-site DMRG optimization, since it essentially generates a *position weigth matrix* model [94] (i.e. an effective bond dimension of one), which is exactly what defines the Kozak sequence.

Extending these methods to interpreting the more general learned MPS/TN through different singular value decompositions and/or transformations may be an interesting area for further research.

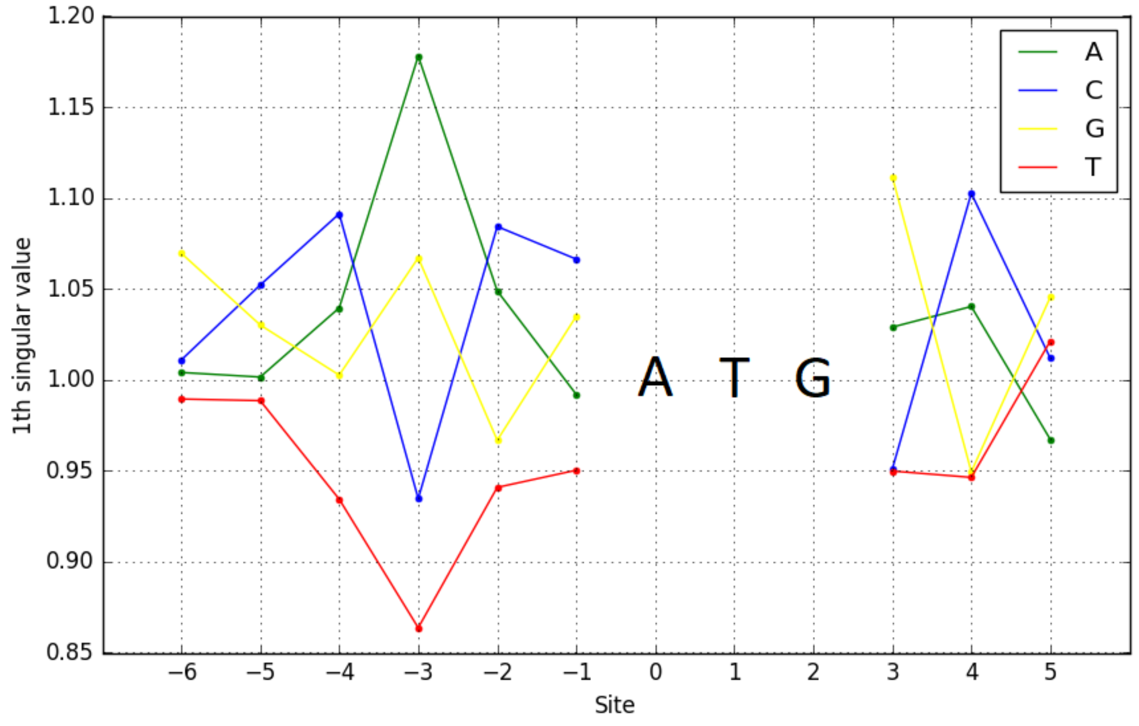


Figure 4.9: The first singular value of the MPS tensors around the TIS (located at sites 0, 1 and 2) for each physical index of the tensors.

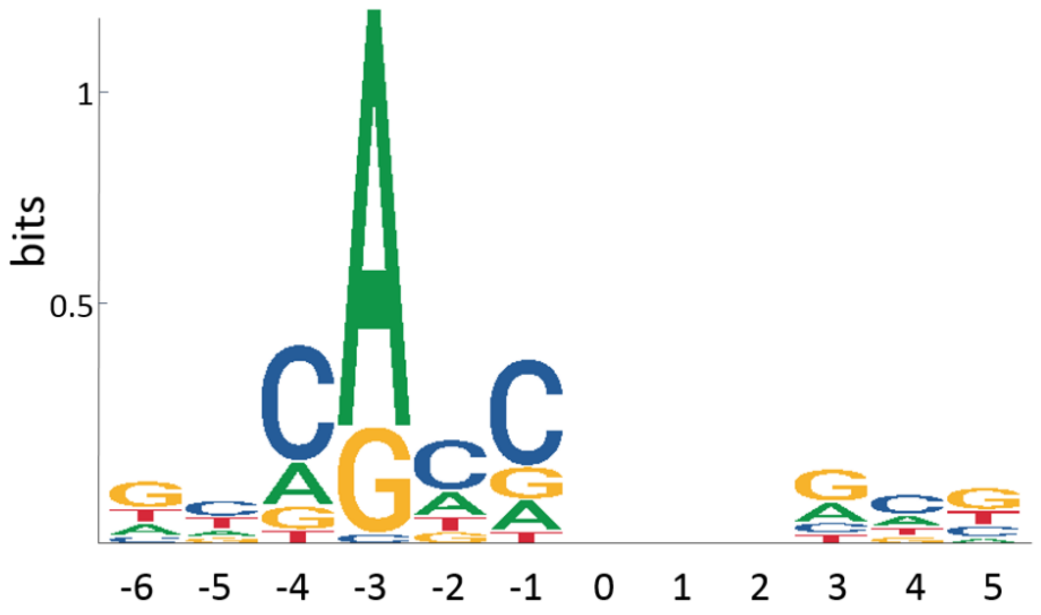


Figure 4.10: The true Kozak sequence in DNA, the height of the letters indicates how probable it is for the corresponding nucleotide to appear on that site.

Double layered MPS for TIS

A different model that was shortly considered is based on [80] where the joint probability distribution of the data is represented by the square of a wave function, which is constructed in MPS form (as described in section 3.1).

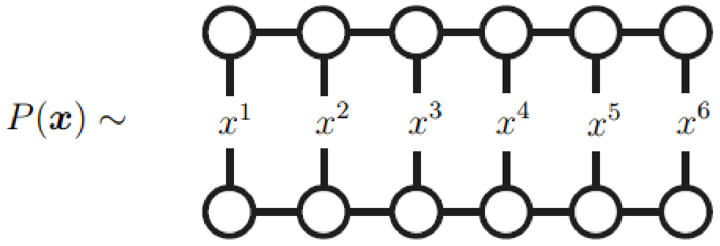


Figure 4.11: Representing the joint probability distribution as the square of the MPS wave function.

Two separate MPS can be constructed, one describing the probability distribution of true TIS sequences $P_+(\boldsymbol{x})$ and for the pseudo-TIS sequences $P_-(\boldsymbol{x})$

$$P_+(\boldsymbol{x}) = \frac{\Psi_+^2(\boldsymbol{x})}{Z_+}, \qquad P_-(\boldsymbol{x}) = \frac{\Psi_-^2(\boldsymbol{x})}{Z_-} \tag{4.16}$$

where Z_+ and Z_- are normalizations of these two probability distributions, obtained by contracting the MPS with itself. The log-likelihood can be maximized respectively to the true and pseudo training samples

$$L = \frac{1}{n_+} \sum_{i \in true} \log\left(\frac{\Psi_+^2(\boldsymbol{x}_i)}{Z_+}\right) + \frac{1}{n_-} \sum_{i \in pseudo} \log\left(\frac{\Psi_-^2(\boldsymbol{x}_i)}{Z_-}\right) \tag{4.17}$$

for which the gradient for a tensor on site j in the ($+$ and $-$) MPS is given by

$$\frac{\partial L}{\partial A_j} = \frac{2}{n} \sum_{i \in +/-} \frac{\Psi'(\boldsymbol{x}_i)}{\Psi(\boldsymbol{x}_i)} - \frac{Z'}{Z} \tag{4.18}$$

where the derivatives of the MPS are just the environments of tensor A_j as usual. One can work in the canonical form for MPS, which makes the expression of the normalization Z and its derivative somewhat simpler, namely $Z = \langle A_j | A_j \rangle$ and $Z' = A_j$. Once these MPS are trained and the probabilities $P_+(\boldsymbol{x})$ and $P_-(\boldsymbol{x})$ modeled, a new sample can be classified by giving it the following score

$$z = \log\left(\frac{P_+(\boldsymbol{x})}{P_-(\boldsymbol{x})}\right) \tag{4.19}$$

such that if $z > 0$ the sample is classified as true TIS and if $z < 0$ it is classified as pseudo-TIS.

Both MPS were trained on DNA sequences with upstream 6 and downstream 39 for different bond dimensions, again an initial increase in performance is observed with increasing bond dimension. On the validation set (part of CCDS) used while training the best performance found was for a bond dimension of 5, testing on human chromosome 21 dataset showed that it only improved slightly until a bond dimension of ~ 3. Altogether the performance was worse than in the case of the single layer MPS described in the previous subsection and the generalization to the test set was inferior. The ROC curves on the validation set and test set are given in figure 4.12.

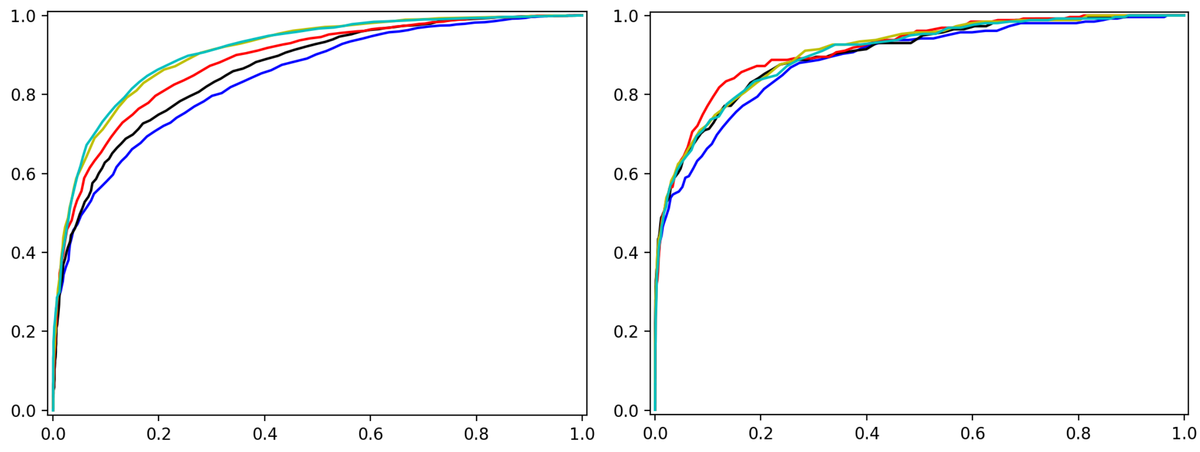


Figure 4.12: ROC curves for the validation set (left) and test set (right) for bond dimensions: 1 (blue), 2 (black), 3 (red), 4 (yellow) and 5 (green).

4.1.5 Number-state preserving MPS for TIS

To obtain more sparse and interpretable models, one can resort to the restricted class of number-state preserving tensors to build a model. Here this class of tensors will be used to construct an MPS for the problem of TIS prediction, the orientations of the number-state preserving tensors are given in the figure below.

Optimization is done by sweeping over the MPS and updating the tensors as described in section 3.3.2 with respect to some training set. Again an upstream of 30 and a downstream of 70 is considered; the model is trained on training set containing all positive samples from the CCDS dataset and an equal amount of negative samples (10% of the training set is used for validation). The tensors are initialized randomly. Next different bond dimensions were considered, due to the noise one the end result (as a consequence of the stochastic updates), the MPS was optimized 10 times for each bond dimension and the results are an average of these different runs. The best results were obtained for $T \approx 0.0001$ (the environment is defined as $\frac{1}{n}\sum_i \widetilde{\Phi}(\boldsymbol{x}_i)$ here). The results are summarized

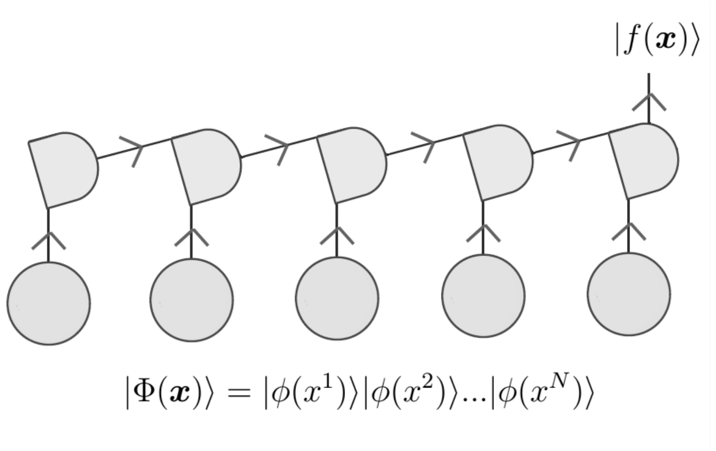

$|f(\boldsymbol{x})\rangle$
$|\Phi(\boldsymbol{x})\rangle = |\phi(x^1)\rangle|\phi(x^2)\rangle...|\phi(x^N)\rangle$

Figure 4.13: An MPS consisting of number-state preserving tensors for classification.

in the table below with in the fourth column the balanced accuracy[9] (BACC), note that for this model it is impossible to construct a ROC curve, as the output is either δ_{0l} or δ_{1l}. The test set used contains 100000 samples from the human chromosome 21 set (containing only 258 true TIS samples).

Bond dimension D	TPR	TNR	BACC
1	0.50	0.63	0.56
2	0.52	0.73	0.62
3	0.58	0.78	0.68
4	0.58	0.81	0.69
5	0.58	0.81	0.69
6	0.57	0.81	0.69
7	0.59	0.81	0.70
8	0.59	0.80	0.69
9	0.59	0.80	0.69
10	0.59	0.78	0.68
20	0.57	0.77	0.67

Table 4.2: True positive rate and true negative rate on a test set for a number-state preserving MPS with different bond dimensions.

It is clear that again, the initial increasing improvement with raising bond dimension ceases really early, moreover the performance is far worse than in the case of an unrestricted MPS, which can be seen when comparing the point $(FPR, TPR) = (0.2, 0.6)$ with the ROC curves in figure 4.6 (remember $FPR = 1 - TNR$).

[9] $BACC = \frac{TPR+TNR}{2}$

4.2 Hidden Markov Models

A well known statistical model for sequential data is the *hidden Markov model* (HMM) [95]. The idea is that one has, at every time step t, an unobservable (hidden) variable $H(t) \in \{h_1, h_2, ..., h_D\}$ that can be in D distinct states and an observable variable $X(t) \in \{x_1, x_2, ..., x_d\}$ that has d possible states to reside in. The hidden variable at time t influences the hidden variable at time $t+1$ through the *transition probabilities* $P(H(t+1) = \nu | H(t) = \mu) =: T_{\mu\nu}$. The state the hidden variable $H(t)$ is in at time t then determines the probability distribution of the observable states $X(t)$ at that time, through the *observation probabilities* $P(X(t) = s | H(t) = \mu) =: \epsilon_\mu^s$. The model then generates sequences of observable states $X(1)X(2)...X(t)$ by starting in some hidden state $H(1)$ (with a probability given by the *prior* $P(H(1) = \mu) =: \pi_\mu$) which determines the probabilities of the observable at that time and of the next hidden variable in the chain, and so on. In this model the first-order Markov assumption is made, i.e. the variables at time t only directly depend on variables from $t-1$. A graphical depiction of a HMM is given in figure 4.14, where the nodes represent the variables and an arrow going from a variable x to some variable y denotes the conditional dependence between these variables $P(y|x)$.

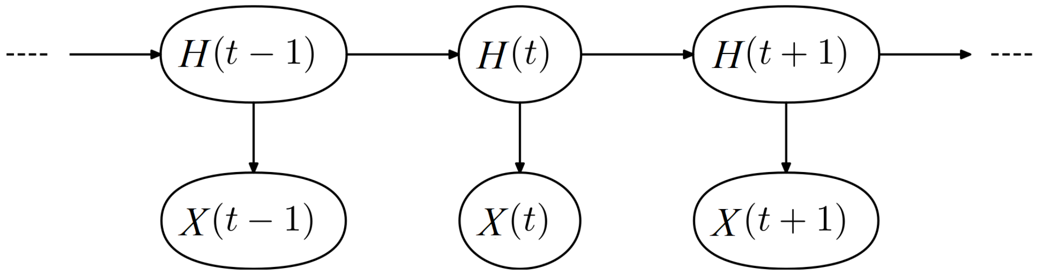


Figure 4.14: Graphical depiction of a (first-order) hidden Markov model.

Given a HMM one can compute different things, one being *scoring* of a sequence of observations $X(1)X(2)...X(t)$, i.e. evaluation of the probability of this sequence occurring given the models parameters. Given the structure of the model this can be done efficiently, the algorithm used to compute this is known as the *forward algorithm*. This is a part of a more general scheme known as the *forward-backward algorithm* which allows to compute the posterior marginals $P(H(t)|X(1)X(2)...X(N))$ of all hidden state variables given a sequence of observations. The scaling of this algorithm is $O(ND^2)$ with N the length of the sequence and D the number of possible hidden states. A second task one might want to perform with a HMM is *decoding*, i.e. determine the most likely sequence of hidden states $H(1)H(2)...H(t)$, given a sequence of observables. This is done with the *Viterbi* *algorithm* which also has a time complexity of $O(ND^2)$. Maybe the most important task to perform is, given a set of observations, to determine the parameters of the HMM: π, T and ϵ. The algorithm which achieves this goes under the name of the *Baum-Welch algorithm* [96], which is an iterative algorithm scaling as $O(ND^2)$ per step.

The attentive reader may have already seen that this hidden Markov model can be written as a translation invariant MPS with bond dimension D. The HMM depicted in figure 4.14 can be redrawn using the usual TN notation where the tensors reside on the edges of 4.14 and the legs of the tensors correspond to the variables. The result is given in figure 4.16 in terms of oriented tensors, where the tensor $\delta_{\mu\nu\lambda} = 1$ if $\mu = \nu = \lambda$ and zero elsewhere.

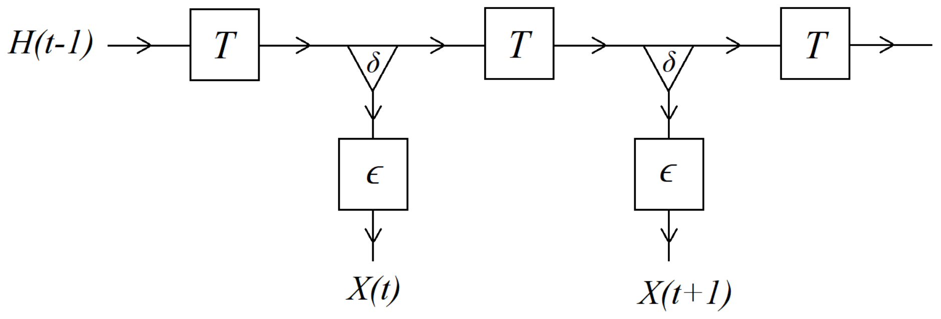


Figure 4.15: TN notation for a hidden Markov model.

The arrows on the legs of the tensors denote the conditionals dependence of the indices with outgoing arrows on the ones with ingoing arrows:

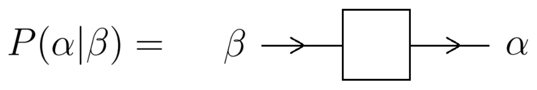


Figure 4.16: Arrow notation for conditional dependence in tensor networks.

The virtual indices now correspond with the states of hidden variables and contracting over them and thus summing them out gives us the total (marginal) probability distribution of a sequence of observables $P(X(1)X(2)...X(N))$. The MPS form of a HMM can be made more explicit by defining the MPS tensors as

$$A^s_{\mu\nu} = T_{\mu\lambda}\delta_{\lambda\sigma\nu}\epsilon^s_\sigma \tag{4.20}$$

resulting in the MPS shown in figure 4.17.
Note that the exact definition of these MPS tensors A in terms of the variables of the HMM is arbitrary, another choice is for example $A^s_{\mu\nu} = \delta_{\mu\sigma\lambda}T_{\lambda\nu}\epsilon^s_\sigma$. As is the case for general MPS, a certain gauge freedom remains (keeping positivity and normalization of

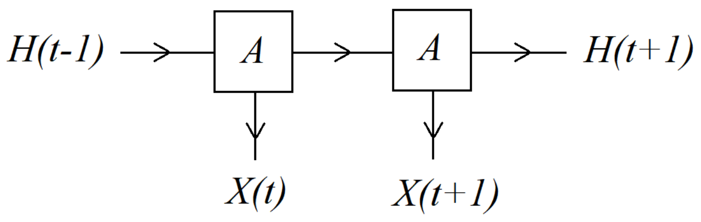


Figure 4.17: Explicit MPS form of a hidden Markov model.

the tensors in mind).

Now the question arises whether this identification of a HMM as an MPS leads to new insights and if, for example, the identification of more general graphical models (of which HMM's are a special case) with tensor networks allows any of the known algorithms for tensor networks to be carried over to these probabilistic models. Two-site DMRG for example can not be applied to a HMM as it does not insure positivity of the MPS due to the gradient (which can be fixed by choosing a good initialization) and the successive SVD's. One would need to work on the L_1-norm as opposed the L_2-norm (used in quantum physics), the MPS could then for example be optimized with algorithms based on linear programming. A lot of the known TN techniques would have to be converted to the L_1-norm (like for example a positive analogue for an SVD) and this forms a very interesting area of research.

As a small experiment, consider two datasets of sequences generated by two distinct HMM's, dubbed class A and class B. Both HMM's have 10 hidden states and 4 observable states. Let us test to what extend the generated sequences can be classified as A or B by an unrestricted MPS. Note that perfect classification of the samples as originating from either HMM A or HMM B is impossible, there is a theoretical limit to this. If non of the transition probabilities $T_{\mu\nu}$ or observation probabilities ϵ_μ^s are exactly zero, all possible sequences of observable states $X(1), X(2), ..., X(N)$ can originate both from A and B, albeit with different probabilities. A scoring of how likely a certain sequence $\boldsymbol{x}$ is one of the two types is the following

$$z = \log\left(\frac{P_A(\boldsymbol{x})}{P_B(\boldsymbol{x})}\right) \tag{4.21}$$

where $P_A(\boldsymbol{x})$ and $P_B(\boldsymbol{x})$ are the probabilities obtained by summing over the hidden variables in models A and B. A certain threshold can be chosen to classify a sample, for example if $z > 0$ then $\boldsymbol{x} \in A$ and if $z < 0$ then $\boldsymbol{x} \in B$. With these scores a ROC curve can be constructed, which can be seen as a theoretical upper bound to the best possible classification that can be obtained. This upper bound depends on the transition and observation probabilities and on the length of the sequences generated (see figure 4.18).

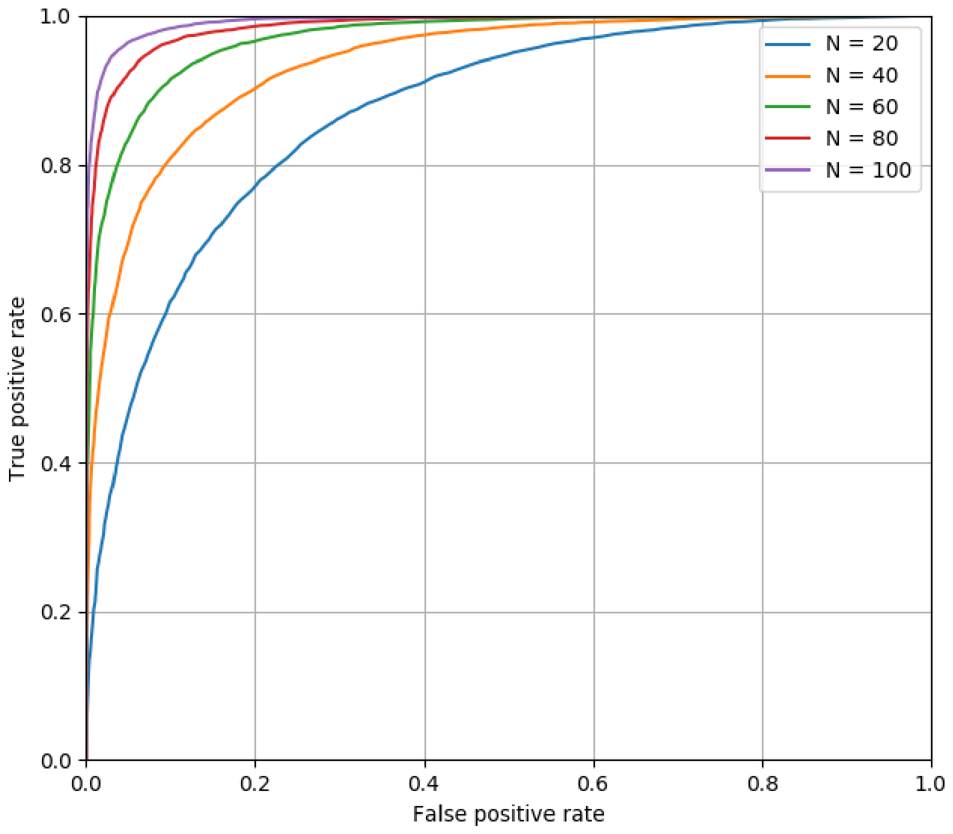


Figure 4.18: Upper bound on best classification possible of two distinct HMM's for different lengths N of the generated sequences.

MPS with different bond dimensions where trained on 50000 generated sequences of length 30, the results of testing on an independent set are summarized in the table below (4.2). It appears that already at a bond dimension of one, classification performance comes very close to the theoretical bound. The slightest improvement occurs towards a bond dimension of 5, after which the model begins to overfit and performance starts to lower. As a more extreme example, 2 different HMM's with different π, T and ϵ were considered with, for $N = 70$, a theoretical upper bound of AUC $= 0.9946$. Already with the simplest model with a bond dimension of 1, classification performance quasi equal to this upper bound was obtained, with AUC $= 0.9937$.

	Se80	AUC
$D = 1$	0.163	0.9010
$D = 2$	0.159	0.9010
$D = 3$	0.159	0.9020
$D = 4$	0.162	0.9027
$D = 5$	**0.158**	**0.9031**
$D = 6$	0.162	0.9019
$D = 7$	0.161	0.9007
$D = 8$	0.164	0.8999
$D = 9$	0.165	0.8990
$D = 10$	0.167	0.8986
HMM	**0.137**	**0.9147**

Table 4.3: Results of HMM (with 10 hidden variables) classification with an MPS.

Chapter 5

Conclusions and discussion

First experiments showed that in transferring the framework of matrix product states and the associated optimization algorithms from the squared MPS form in physics ($P(\boldsymbol{x}) = |\Psi(\boldsymbol{x})|^2$), to the (real valued) single layer MPS form for machine learning ($f(\boldsymbol{x}) = \Psi(\boldsymbol{x})$), a lot of numerical stability problems pop up. There are different methods of resolving these, some more effective than others and each having a slightly different impact on the end results. This is somewhat typical in designing machine learning models and algorithms, where through well argumented ideas and a lot of experimentation basic models evolved to state-of-the-art techniques. In this sense the methods presented in this work are still in an early stage when it comes to applying them in machine learning and a lot more sophisticated updating schemes than the relatively simple ones considered here could be converted to TN form. It was illustrated that the DMRG algorithm, which initially suffered from either exploding or vanishing gradients, could be made stable by a slightly different choice of loss function resulting in an gradient with a different dependence on the environment of the tensor to be updated. Of course there are different ways of fixing this problem, although this seemed to be the simplest and most effective way.

The classification problem which was first considered and used for initial testing of the methods and algorithms was TIS prediction in DNA. Training a single layered MPS as to maximize the overlap with the true-TIS samples from the training set $\langle W|\Phi_{true}\rangle \to 1$ and minimize the overlap with the pseudo-TIS samples $\langle W|\Phi_{pseudo}\rangle \to 0$ with 2-site DMRG showed that increasing the bond dimension resulted in better fitting on the training set and test set. This positive trend already halted at a mere bond dimension of $D \sim 6$ after which the performance on the independent test set slowly declined with increasing bond dimension as the model began to overfit more for a higher number of trainable parameters. Nonetheless with this small bond dimension of 6 the MPS model proved to perform better than the simple StartScan model described in [6] when trained on the same upstream and downstream, it is however clearly outperformed by the convolutional neural network described in [92]. It was also illustrated that some basic properties from the considered dataset could be extracted by looking at the SVD's of the tensors in the MPS, such as the nucleotides coding information in groups of 3 (codons) and the presence of the Kozak sequence around the ATG codon. This analysis of the MPS was however only possible if it was optimized with 1-site DMRG, generalizing this and developing techniques of interpreting general tensor networks trained on a specific dataset is an interesting area of further research.

A different angle of approaching the problem of TIS prediction, was a model more closely inspired by MPS in a quantum many-body context. Instead of constructing a unnormalized classification function, the joint probability distribution of the true TIS and pseudo-TIS data was modeled by representing it as the square of a normalized (real valued) wave function $P(\boldsymbol{x}) \sim \Psi^2(\boldsymbol{x})$. The log-likelihood with respect to a training set was maximized and a scoring could be extracted from these two separate distributions. This model had the same behaviour as the single layer MPS in the sense that initially the performance on a validation set improved with increasing bond dimension, but only to a bond dimension of $D \sim 5$. The model also seemed to have poorer performance on the human chromosome 21 test set. From these observations of the model overfitting for bond dimensions higher than 5-7 and failing to capture any more correlations in the data, it seems we have to conclude that MPS are not suited for the modelling of TIS in DNA and that there are perhaps longer distance correlations present in DNA as might be suggested from the mutual information between sites plotted in figure 4.4.

As a method of obtaining sparse and perhaps better models, a restricted class of tensors was considered, namely the one of number-state preserving tensors. The constrictions on the tensors allow for efficient contraction and optimization of more general tensor networks such as MERA. We limited ourselves to a number-state preserving MPS and showed that with these constrictions the MPS performed a lot worse than the unrestricted MPS. It thus seems that this restricted class of tensor networks is not as powerful as generic unrestricted tensor networks and is inefficient for certain classification tasks, contrary to what the author of [90] presented. Note that the classification tasks considered in [90] are fairly trivial and specific (bit sequence parity classification and remainder of division by 7 of binary strings classification), for which they achieved 100% accuracy, and might lend themselves better to these restricted models. Nevertheless these restricted tensors in combination with other TN architectures than MPS could prove to yield powerful models and a network such as MERA might be able to capture the correlations in a problem such as TIS prediction in DNA.

Finally hidden Markov models where considered and it was shown that they can be identified as a translation invariant MPS. This MPS is constricted in the sense that its entries are positive and the tensors are normalized according to the probabilities they represent. The identification of more general graphical probabilistic models [73] as tensor networks, where the hidden variables correspond to the virtual states in the network, could lead to some interesting insights and forms an interesting area of research. A small experiment was carried out by constructing two distinct HMM's, A and B, and looking to what extend samples generated by these models could be classified by and unrestricted MPS. It appears that in order to classify samples from an underlying HMM (which is not the same as modeling the actual joint probability distributions), the simplest MPS with a bond dimension of one suffices, which is essentially a position weight matrix model [94]. From this one might conclude that a HMM is not a good model for the context around a TIS in DNA, as we would be able to classify it with the best possible performance with an MPS with $D = 1$ if it were.

In this work we presented some models for supervised learning based on the framework of tensor networks widely used in the community of quantum many-body physics. We mainly focused on matrix product states and studied a few basic optimization schemes. This interdisciplinary field of machine learning and tensor networks set off only very recently and I believe a lot more than was presented here remains to be discovered.

Chapter 6

Appendix

6.1 Duality principle for optimization

Consider the following constraint optimization problem with *primal variable* $\boldsymbol{x} \in \mathbb{R}^n$

$$\begin{aligned} &\min_{\boldsymbol{x}} f(\boldsymbol{x}) \\ \text{subject to: } &h_j(\boldsymbol{x}) = 0, j \in \{1, ..., l\} \\ &g_k(\boldsymbol{x}) \leq 0, k \in \{1, ..., m\} \end{aligned} \tag{6.1}$$

the Lagrangian $L(\boldsymbol{x}, \boldsymbol{\gamma}, \boldsymbol{\lambda}) : \mathbb{R}^n \times \mathbb{R}^l \times \mathbb{R}^m \mapsto \mathbb{R}$ of this problem is defined as

$$L(\boldsymbol{x}, \boldsymbol{\gamma}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \sum_j \gamma_j h_j(\boldsymbol{x}) + \sum_k \lambda_k g_k(\boldsymbol{x}) \tag{6.2}$$

where $\boldsymbol{\gamma}$ and $\boldsymbol{\lambda}$ ($\lambda_i \geq 0$) are called the *dual variables* (or Lagrange multipliers). The *Lagrange dual function* $\Lambda(\boldsymbol{\gamma}, \boldsymbol{\lambda}) : \mathbb{R}^l \times \mathbb{R}^m \mapsto \mathbb{R}$ is defined as

$$\Lambda(\boldsymbol{\gamma}, \boldsymbol{\lambda}) = \inf_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\gamma}, \boldsymbol{\lambda}) \tag{6.3}$$

which is a concave function and yields a lower bound on the optimal value of the initial problem: $\Lambda(\boldsymbol{\gamma}, \boldsymbol{\lambda}) \leq f(\boldsymbol{x}^*)$. The dual optimization problem is thus defined as

$$\begin{aligned} &\max_{\boldsymbol{\gamma}, \boldsymbol{\lambda}} \Lambda(\boldsymbol{\gamma}, \boldsymbol{\lambda}) \\ \text{subject to: } &\lambda_i \geq 0 \end{aligned} \tag{6.4}$$

The difference between $\max_{\boldsymbol{\gamma}, \boldsymbol{\lambda}} \Lambda(\boldsymbol{\gamma}, \boldsymbol{\lambda})$ and $f(\boldsymbol{x}^*)$ is called the *duality gap* and for convex optimization problems under a constraint qualification, the duality gap is zero (strong duality).

For a convex problem with only inequality constraints, the Lagrangian dual problem is

$$\begin{aligned}
&\max_{\boldsymbol{\lambda}} \inf_{\boldsymbol{x}} \left(f(\boldsymbol{x}) + \sum_k \lambda_k g_k(\boldsymbol{x}) \right) \\
&\text{subject to: } \lambda_i \geq 0
\end{aligned} \tag{6.5}$$

Provided that the functions $f(\boldsymbol{x})$ and $g_k(\boldsymbol{x})$ are continuously differentiable, the infimum occurs where the gradient is equal to zero and the problem becomes

$$\begin{aligned}
&\max_{\boldsymbol{\lambda}} \left(f(\boldsymbol{x}) + \sum_k \lambda_k g_k(\boldsymbol{x}) \right) \\
\text{subject to: } &\nabla f(\boldsymbol{x}) + \sum_k \lambda_k \nabla g_k(\boldsymbol{x}) = 0 \\
&\lambda_i \geq 0
\end{aligned} \tag{6.6}$$

and is called the *Wolfe dual* problem. Applying this to the problem defined by a SVM in (1.7), the following conditions are obtained

$$\begin{aligned}
&\boldsymbol{w} = \sum_i \lambda_i y_i \boldsymbol{x}_i \\
&\sum_i \lambda_i y_i = 0 \\
&\lambda_i \geq 0, \quad \forall i
\end{aligned} \tag{6.7}$$

and inserting this into (1.7) results in the dual Lagrangian (1.9). For more information and proofs of the results above we refer to [97].

6.2 Singular value decomposition

Consider a matrix $M \in \mathbb{C}^{n\times m}$, the following factorization exists

$$M = U\Sigma V^\dagger \tag{6.8}$$

where $U \in \mathbb{C}^{n\times n}$ and $V^\dagger \in \mathbb{C}^{m\times m}$ are unitary matrices and Σ a $n \times m$ diagonal matrix containing non-negative real numbers σ_i which are called the *singular values*: they are uniquely determined and are by convention ordered in descending order. This factorization is called a *singular value decomposition* (SVD) and can be viewed as writing the matrix

M as a weighted sum of separable matrices[1]

$$M = \sum_i \sigma_i \boldsymbol{u}_i \otimes \boldsymbol{v}_i \tag{6.9}$$

where $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ are the orthonormal columns of the unitary matrices U and $V^\dagger$ and σ_i the singular values.

The number of singular values of a matrix determines the *rank* of that matrix. A matrix M with a rank r_0 can be approximated by a matrix $\widetilde{M}$ with rank $r < r_0$ defined by

$$\widetilde{M} = U\widetilde{\Sigma}V^\dagger \tag{6.10}$$

where $\widetilde{\Sigma}$ is the diagonal matrix containing only the r largest singular values of M. It can be shown that the matrix $\widetilde{M}$ defined above is the one that minimizes $||M - \widetilde{M}||_F$ (F stands for Frobenius norm) under the constraint $\text{rank}(\widetilde{M}) \leq r$, this is known as the Eckart–Young theorem [98].

In the field of quantum information theory, the decomposition as given in (6.9) is known as the *Schmidt decomposition* [99]. It is used to express a general pure bipartite state $|\psi\rangle_{AB} = \sum_{\mu,\nu} c_{\mu\nu}|\mu\rangle_A \otimes |\nu\rangle_B \in \mathcal{H}_A \otimes \mathcal{H}_B$ as

$$|\psi\rangle_{AB} = \sum_i^n \sigma_i |i\rangle_A \otimes |i\rangle_B \tag{6.11}$$

with $n = \min(\dim(\mathcal{H}_A), \dim(\mathcal{H}_B))$ and where $|i\rangle_A$ and $|i\rangle_B$ are orthonormal sets (not necessarily complete bases since the dimensions of $\mathcal{H}_A$ and $\mathcal{H}_B$ can differ) which depend on the particular state $|\psi\rangle_{AB}$. It is also clear that these (possibly incomplete) bases diagonalize the density matrices $\rho_A = \text{Tr}_B(|\psi\rangle\langle\psi|_{AB})$ and $\rho_B = \text{Tr}_A(|\psi\rangle\langle\psi|_{AB})$ and that, in the case where the dimensions are equal, they have the same non-zero eigenvalues. The number of non-zero singular values σ_i is called the *Schmidt number* and a bipartite state is said to be entangled (non-separable) if the Schmidt number is greater than 1. The (Von Neumann) entanglement entropy [100] for one of the reduced states is defined as

$$S = -\text{Tr}(\rho_A \log \rho_A) = -\text{Tr}(\rho_B \log \rho_B) = -\sum_i \sigma_i^2 \log(\sigma_i^2) \tag{6.12}$$

which is zero if $|\psi\rangle_{AB}$ is a product state (unentangled).

[1]A matrix A is said to be seperable if it can be written as an outer product of two vectors $A = \boldsymbol{u} \otimes \boldsymbol{v}$.

6.3 Mutual information

The *mutual information* between two random variables X and Y (with p_X and p_Y their respective distributions and p_{XY} the joint distribution) is defined as

$$\begin{aligned} I(X,Y) &= \sum_{x,y} p_{XY}(x,y) \log \left(\frac{p_{XY}(x,y)}{p_X(x)p_Y(y)} \right) \\ &= H(X) + H(Y) - H(X,Y) \end{aligned} \tag{6.13}$$

where $H(X)$ is the well known *Shannon entropy*:

$$H(X) = -\sum_x p(x) \log p(x) \tag{6.14}$$

The mutual information between two random variables is a non-negative and symmetric measure of how much information these two variables share, it quantifies the amount of information (in units of bits if one uses a base 2 logarithm) obtained about one random variable through observing the other random variable. For two independent random variables for example, one has $I(X,Y) = 0$.

Bibliography

[1] Román Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349:117–158, 2014.

[2] Frank Verstraete, Valentin Murg, and J Ignacio Cirac. Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems. *Advances in Physics*, 57(2):143–224, 2008.

[3] Michael M Wolf, Frank Verstraete, Matthew B Hastings, and J Ignacio Cirac. Area laws in quantum systems: mutual information and correlations. *Physical review letters*, 100(7):070502, 2008.

[4] Jutho Haegeman, Valentin Zauner, Norbert Schuch, and Frank Verstraete. Shadows of anyons and the entanglement structure of topological phases. *Nature communications*, 6:8284, 2015.

[5] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

[6] Yvan Saeys, Thomas Abeel, Sven Degroeve, and Yves Van de Peer. Translation initiation site prediction on a genomic scale: beauty in simplicity. *Bioinformatics*, 23(13):i418–i423, 2007.

[7] Giuseppe Carleo and Matthias Troyer. Solving the quantum many-body problem with artificial neural networks. *Science*, 355(6325):602–606, 2017.

[8] Zhih-Ahn Jia, Biao Yi, Rui Zhai, Yu-Chun Wu, Guang-Can Guo, and Guo-Ping Guo. Quantum neural network states. *arXiv preprint arXiv:1808.10601*, 2018.

[9] Dong-Ling Deng, Xiaopeng Li, and S Das Sarma. Quantum entanglement in neural network states. *Physical Review X*, 7(2):021021, 2017.

[10] Qianshi Wei, Ying Jiang, and Jeff ZY Chen. Machine-learning solver for modified diffusion equations. *Physical Review E*, 98(5):053304, 2018.

[11] Leonardo Banchi, Edward Grant, Andrea Rocchetto, and Simone Severini. Modelling non-markovian quantum processes with recurrent neural networks. *New Journal of Physics*, 20(12):123030, 2018.

[12] Dan Guest, Kyle Cranmer, and Daniel Whiteson. Deep learning and its application to lhc physics. *Annual Review of Nuclear and Particle Science*, 68:161–181, 2018.

[13] MC Storrie-Lombardi, O Lahav, L Sodre Jr, and LJ Storrie-Lombardi. Morphological classification of galaxies by artificial neural networks. *Monthly Notices of the Royal Astronomical Society*, 259(1):8P–12P, 1992.

[14] Ross Kindermann. Markov random fields and their applications. *American mathematical society*, 1980.

[15] Jun Zhang. Application of the gibbs-bogoliubov-feynman inequality in mean field calculations for markov random fields. In *Visual Communications and Image Processing'94*, volume 2308, pages 982–994. International Society for Optics and Photonics, 1994.

[16] Mohammad H Amin, Evgeny Andriyash, Jason Rolfe, Bohdan Kulchytskyy, and Roger Melko. Quantum boltzmann machine. *Physical Review X*, 8(2):021050, 2018.

[17] Tatjana Puškarov and Axel Cortés Cubero. Machine learning algorithms based on generalized gibbs ensembles. *Journal of Statistical Mechanics: Theory and Experiment*, 2018(10):103102, 2018.

[18] Pankaj Mehta and David J Schwab. An exact mapping between the variational renormalization group and deep learning. *arXiv preprint arXiv:1410.3831*, 2014.

[19] Mario Geiger, Stefano Spigler, Stéphane d'Ascoli, Levent Sagun, Marco Baity-Jesi, Giulio Biroli, and Matthieu Wyart. The jamming transition as a paradigm to understand the loss landscape of deep neural networks. *arXiv preprint arXiv:1809.09349*, 2018.

[20] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM JOURNAL OF RESEARCH AND DEVELOPMENT*, pages 71–105, 1959.

[21] T Mitchell. Machine learning'. mcgraw-hill, new york, 1997.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[23] Sakshi Patel, Shivani Sihmar, and Aman Jatain. A study of hierarchical clustering algorithms. In *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 537–541. IEEE, 2015.

[24] Hassan Ashtiani and Shai Ben-David. Representation learning for clustering: a statistical framework. *arXiv preprint arXiv:1506.05900*, 2015.

[25] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[26] Pasquale Foggia, Gennaro Percannella, Carlo Sansone, and Mario Vento. A graph-based clustering method and its applications. In *International Symposium on Brain, Vision, and Artificial Intelligence*, pages 277–287. Springer, 2007.

[27] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. Dimensionality reduction: a comparative. *J Mach Learn Res*, 10:66–71, 2009.

[28] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[29] Andrew Kusiak. Feature transformation methods in data mining. *IEEE Transactions on Electronics packaging manufacturing*, 24(3):214–221, 2001.

[30] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[31] Zhang Xuegong. Introduction to statistical learning theory and support vector machines. *Acta Automatica Sinica*, 26(1):32–42, 2000.

[32] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[33] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[34] Finn V Jensen et al. *An introduction to Bayesian networks*, volume 210. UCL press London, 1996.

[35] Simon Haykin. *Neural networks*, volume 2. Prentice hall New York, 1994.

[36] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[37] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[38] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220, 2008.

[39] Harold W Kuhn and Albert W Tucker. Nonlinear programming, in (j. neyman, ed.) proceedings of the second berkeley symposium on mathematical statistics and probability, 1951.

[40] Wei Chu, S Sathiya Keerthi, and Chong Jin Ong. A general formulation for support vector machines. In *Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP'02.*, volume 5, pages 2522–2526. IEEE, 2002.

[41] Philip Wolfe. A duality theorem for non-linear programming. *Quarterly of applied mathematics*, 19(3):239–244, 1961.

[42] J Biamonte and V Bergholm. Quantum tensor networks in a nutshell. *arXiv*, 1708, 2017.

[43] Jens Eisert, Marcus Cramer, and Martin B Plenio. Area laws for the entanglement entropy-a review. *arXiv preprint arXiv:0808.3773*, 2008.

[44] David Poulin, Angie Qarry, Rolando Somma, and Frank Verstraete. Quantum simulation of time-dependent hamiltonians and the convenient illusion of hilbert space. *Physical review letters*, 106(17):170501, 2011.

[45] Mark Fannes, Bruno Nachtergaele, and Reinhard F Werner. Finitely correlated states on quantum spin chains. *Communications in mathematical physics*, 144(3):443–490, 1992.

[46] A Klümper, A Schadschneider, and J Zittartz. Groundstate properties of a generalized vbs-model. *Zeitschrift für Physik B Condensed Matter*, 87(3):281–287, 1992.

[47] Jacob C Bridgeman and Christopher T Chubb. Hand-waving and interpretive dance: an introductory course on tensor networks. *Journal of Physics A: Mathematical and Theoretical*, 50(22):223001, 2017.

[48] Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[49] Wolfgang Hackbusch. *Tensor spaces and numerical tensor calculus*, volume 42. Springer Science & Business Media, 2012.

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

[51] Guifré Vidal. Efficient classical simulation of slightly entangled quantum computations. *Physical review letters*, 91(14):147902, 2003.

[52] Masuo Suzuki. M. suzuki, prog. theor. phys. 56, 1454 (1976). *Prog. Theor. Phys.*, 56:1454, 1976.

[53] Frank Verstraete and J Ignacio Cirac. Renormalization algorithms for quantum-many body systems in two and higher dimensions. *arXiv preprint cond-mat/0407066*, 2004.

[54] Frank Verstraete, Michael M Wolf, David Perez-Garcia, and J Ignacio Cirac. Criticality, the area law, and the computational power of projected entangled pair states. *Physical review letters*, 96(22):220601, 2006.

[55] Norbert Schuch, Michael M Wolf, Frank Verstraete, and J Ignacio Cirac. Computational complexity of projected entangled pair states. *Physical review letters*, 98(14):140506, 2007.

[56] Michael Lubasch, J Ignacio Cirac, and Mari-Carmen Banuls. Algorithms for finite projected entangled pair states. *Physical Review B*, 90(6):064425, 2014.

[57] Guifre Vidal. Entanglement renormalization. *Physical review letters*, 99(22):220405, 2007.

[58] Guifré Vidal. Class of quantum many-body states that can be efficiently simulated. *Physical review letters*, 101(11):110501, 2008.

[59] Y-Y Shi, L-M Duan, and Guifre Vidal. Classical simulation of quantum many-body systems with a tree tensor network. *Physical review a*, 74(2):022320, 2006.

[60] Simon Haykin. *Neural networks*, volume 2. Prentice hall New York, 1994.

[61] Laurene V Fausett et al. *Fundamentals of neural networks: architectures, algorithms, and applications*, volume 3. prentice-Hall Englewood Cliffs, 1994.

[62] Simon S Haykin, Simon S Haykin, Simon S Haykin, Kanada Elektroingenieur, and Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, 2009.

[63] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[64] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[65] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *Advances in neural information processing systems*, pages 442–450, 2015.

[66] Yoav Levine, David Yakira, Nadav Cohen, and Amnon Shashua. Deep learning and quantum entanglement: Fundamental connections with implications to network design. *arXiv preprint arXiv:1704.01552*, 2017.

[67] Yoav Levine, Or Sharir, Nadav Cohen, and Amnon Shashua. Bridging many-body quantum physics and deep learning via tensor networks. *arXiv preprint arXiv:1803.09780*, 2018.

[68] Henry W Lin, Max Tegmark, and David Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168(6):1223–1247, 2017.

[69] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

[70] Dong-Ling Deng, Xiaopeng Li, and S Das Sarma. Quantum entanglement in neural network states. *Physical Review X*, 7(2):021021, 2017.

[71] Valentin Khrulkov, Oleksii Hrinchuk, and Ivan Oseledets. Generalized tensor models for recurrent neural networks. *arXiv preprint arXiv:1901.10801*, 2019.

[72] Ross Kindermann. Markov random fields and their applications. *American mathematical society*, 1980.

[73] Daphne Koller, Nir Friedman, and Francis Bach. *Probabilistic graphical models: principles and techniques.* MIT press, 2009.

[74] Alexander Novikov, Anton Rodomanov, Anton Osokin, and Dmitry Vetrov. Putting mrfs on a tensor train. In *International Conference on Machine Learning*, pages 811–819, 2014.

[75] Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[76] Bernard N Sheehan and Yousef Saad. Higher order orthogonal iteration of tensors (hooi) and its relation to pca and glram. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 355–365. SIAM, 2007.

[77] Johann A Bengua, Ho N Phien, Hoang D Tuan, and Minh N Do. Matrix product state for feature extraction of higher-order tensors. *arXiv preprint arXiv:1503.00516*, 2015.

[78] Johann A Bengua, Ho N Phien, and Hoang D Tuan. Optimal feature extraction and classification of tensors via matrix product state decomposition. In *2015 IEEE International Congress on Big Data*, pages 669–672. IEEE, 2015.

[79] Andrzej Cichocki. Tensor networks for big data analytics and large-scale optimization problems. *arXiv preprint arXiv:1407.3124*, 2014.

[80] Zhao-Yu Han, Jun Wang, Heng Fan, Lei Wang, and Pan Zhang. Unsupervised generative modeling using matrix product states. *Physical Review X*, 8(3):031012, 2018.

[81] E Miles Stoudenmire. Learning relevant features of data with multi-scale tensor networks. *Quantum Science and Technology*, 3(3):034003, 2018.

[82] Ivan Glasser, Nicola Pancotti, and J Ignacio Cirac. Supervised learning with generalized tensor networks. *arXiv preprint arXiv:1806.05964*, 2018.

[83] William Huggins, Piyush Patel, K Birgitta Whaley, and E Miles Stoudenmire. Towards quantum machine learning with tensor networks. *arXiv preprint arXiv:1803.11537*, 2018.

[84] E Miles Stoudenmire and David J Schwab. Supervised learning with quantum-inspired tensor networks. *arXiv preprint arXiv:1605.05775*, 2016.

[85] Alexander Novikov, Mikhail Trofimov, and Ivan Oseledets. Exponential machines. *arXiv preprint arXiv:1605.03795*, 2016.

[86] Vladimir Vapnik. The nature of statistical learning theory, 2013.

[87] Sebastian Holtz, Thorsten Rohwedder, and Reinhold Schneider. On manifolds of tensors of fixed tt-rank. *Numerische Mathematik*, 120(4):701–731, 2012.

[88] Christian Lubich, Ivan V Oseledets, and Bart Vandereycken. Time integration of tensor trains. *SIAM Journal on Numerical Analysis*, 53(2):917–941, 2015.

[89] Michael Steinlechner. Riemannian optimization for high-dimensional tensor completion. *SIAM Journal on Scientific Computing*, 38(5):S461–S484, 2016.

[90] Glen Evenbly. Number-state preserving tensor networks as classifiers for supervised learning. *arXiv preprint arXiv:1905.06352*, 2019.

[91] Francis HC Crick. On protein synthesis. In *Symp Soc Exp Biol*, volume 12, page 8, 1958.

[92] Jasper Zuallaert, Mijung Kim, Yvan Saeys, and Wesley De Neve. Interpretable convolutional neural networks for effective translation initiation site prediction. In *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1233–1237. IEEE, 2017.

[93] Marilyn Kozak. An analysis of 5'-noncoding sequences from 699 vertebrate messenger rnas. *Nucleic acids research*, 15(20):8125–8148, 1987.

[94] Gary D Stormo, Thomas D Schneider, Larry Gold, and Andrzej Ehrenfeucht. Use of the 'perceptron'algorithm to distinguish translational initiation sites in e. coli. *Nucleic acids research*, 10(9):2997–3011, 1982.

[95] Lawrence R Rabiner and Biing-Hwang Juang. An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16, 1986.

[96] Lawrence Rabiner. First hand: the hidden markov model. *IEEE Global History*, 2013.

[97] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[98] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[99] Erhardt Schmidt. E. schmidt, math. ann. 63, 433 (1907). *Math. Ann.*, 63:433, 1907.

[100] Ingemar Bengtsson and Karol Życzkowski. *Geometry of quantum states: an introduction to quantum entanglement*. Cambridge university press, 2017.

List of Figures

List of Tables