CARL VON OSSIETZKY UNIVERSITÄT

BACHELOR THESIS

Machine Learning: Binary Non-negative Matrix Factorization

Author:
Diego YUS LÓPEZ

Supervisor:
Prof. Jörg LÜCKE

A thesis submitted in fulfillment of the requirements for the degree of Graduated of Physics

in the

Machine Learning Group
Department für Medizinische Physik und Akustik

September 2015

Declaration of Authorship

I, Diego Yus López, declare that this thesis titled, 'Machine Learning: Binary Non-negative Matrix Factorization' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ____

Date: ____

Acknowledgements

In first place, I really want to thank my project advisor, Prof. Jörg Lücke, whose descriptive and clarifying lectures got me so much interested in the totally unknown for me field of Machine Learning that I decided to write my thesis on that topic. Of course, I also want to thank him for suggesting me an appealing thesis topic and advising me during the thesis by giving many useful ideas on how to solve the arising problems, including the visualization functions, but also on how to focus the thesis itself.

On the academic side, I also want to thank PhD student Maryam Sadreddini, who was my closest contact during the thesis helping me with the implementation and whose explanations, given with infinite patience, helped me to get a deeper understanding of some concepts.

I also want to thank my family for their support during this international exchange.

. . .

Contents

For/Dedicated to/To my…

Chapter 1

Introduction

Machine Learning can be defined as “a field of computer science based on the implementation of algorithms that can learn and make prediction from data. These algorithms operate by building a model in order to make data-driven predictions or decisions. Machine Learning is employed in a range of computing tasks, such as spam filtering, search engines or computer vision for example”[1]. Machine Learning is mainly subdivided in three categories: Supervised Learning, Unsupervised Learning and Reinforcement Learning.

In the Supervised Learning, the system is already given some inputs (data) and the desired output (labels), being the task of the system to learn from these examples what the relation between these inputs and outputs is, so it can be further applied to new unlabeled data to obtain the correct outputs.
In contrast to the previous learning, in the Unsupervised Learning the system is only fed with the unlabeled data and its goal is to find some patterns or hidden structure in that data. Because the data is unlabeled, there is no error function to evaluate a potential solution, contrary to the Supervised Learning.
Reinforcement Learning is “learning by interacting with an environment. An RL agent learns from the consequences of its actions, rather than from being explicitly taught and it selects its actions on basis of its past experiences (exploitation) and also by new choices (exploration), which is essentially trial and error learning. The reinforcement signal that the RL-agent receives is a numerical reward, which encodes the success of an action’s outcome, and the agent seeks to learn to select actions that maximize the accumulated reward over time”[2].

[1] https://en.wikipedia.org/wiki/Machine_learning
[2] http://www.scholarpedia.org/article/Reinforcement_learning

In this thesis, the chosen approach will be the Unsupervised Learning, because it is considered a more general approach, given the fact that the labels are not always available. Unsupervised Learning itself is used in different applications, such as denoising, inpainting, or speech and handwriting recognition.

Also, when talking about such algorithms, another aspect has to be kept in mind: the difference between probabilistic and deterministic approaches. A deterministic approach is one in which the final result is determined since the beginning of the process, while in a probabilistic approach, as its name indicates, there is no fixed outcome, but different final results are possible, each one of them with a certain probability to occur. In our case, the algorithm will be probabilistic, because this approach resembles reality better than a deterministic one, having therefore a superior performance when analyzing real data.
This probabilistic approach can be subdivided into two categories: generative and discriminative. In the generative one, the one chosen in my thesis, the algorithm models how the data was generated, e.g. in order to classify a signal; in other words, it tells which category is most likely to generate this signal based on my generation assumptions. The discriminative algorithm however, does not care about how the data was generated, it simply categorizes a given signal. One of the reasons to choose a generative approach lies in the fact that, when applying it to artificial data, it is possible to know the exact result and therefore have an easier implementation.

From a general perspective, the aim of this bachelor thesis is to implement an algorithm that, by means of unsupervised learning using a probabilistic generative model, learns to identify some hidden structure in data. The model chosen is called Binary Non-Negative Matrix Factorization (BNMF), and it is closely related to the type of data to be analyzed, as it is explained in Chapter 2.
A mathematical method needs then to be applied to learn these parameters (hidden structure). In our case, the method chosen is called EM Algorithm, which is an iterative method that maximizes the likelihood function (in each iteration) with respect to the parameters to be learned in a two-step process. It is used where the model depends on unobserved latent variables, as it is the case. It is explained in detail and the equations derived to this particular model in Chapter 3. Besides, in this chapter some important procedures about the computer implementation of this EM algorithm are explained.
Finally, the learning algorithm will be tested in numerical experiments on artificial and real data to check its performance; the tests and the corresponding results are shown in Chapter 4 whereas the final conclusions are presented in Chapter 5.

Chapter 2

Generative Model

In a probabilistic generative approach there is no defined model of how the data to be analyzed is generated. However, a particular generative model with parameters has to be chosen so that we can define a model according to whom the data could have been generated. Choosing that model is not an easy or trivial question, and it is usually related to the way data looks like, existing many different generative models that have already been proposed, such as Sparse Coding or Mixture of Gaussians.

In our case, we suppose the data to be non-negative, as there is a significant number of real situations where this condition is accomplished, being the audio spectrograms (energy values of the different frequencies in which a sound is decomposed by the cochlea in the inner ear) one of the most important examples in Machine Learning. Because of the non-negativity of the data, it seems logical to choose a model in which the generated data according to this model satisfies this condition. The chosen model is called Binary Non-Negative Matrix Factorization (BNMF), a simplification of the standard Non-negative Matrix Factorization (NMF) [1] that can however be explained in an easier way as a modification of Binary Sparse Coding (BSC) [2].

Standard-NMF is an algorithm in which a matrix V is factorized into two matrices W and H, having all the three matrices no negative elements. The elements of a column of H (encoding) are the coefficients by which a column of V (data point) is represented as a linear combination of the columns of W (basis functions). In other words, the coefficients of H express the degree of activation, and therefore of contribution, of a hidden unit (called latent variable) in the generation of a data point. That contribution is made through its related basis function and there is a one-to-one correspondence between a column of H and a column of V.

For my model, I consider a set of N independents data points $\{\vec{y}^{(n)}\}_{1,\ldots,N}$ where $\vec{y}^{(n)} \in \mathbb{R}^D$, i.e. D is the number of observed variables. These data points are generated according to the following generative model:

$$p(\vec{s}|\Theta) = \prod_{h=1}^{H} \left(\pi^{s_h}(1-\pi)^{1-s_h}\right) \tag{2.1}$$

$$p(\vec{y}|\vec{s},\Theta) = \prod_{d=1}^{D} Pois(y_d; \sum_h W_{dh}s_h) = \prod_{d=1}^{D} \left(\frac{a^{y_d}}{y_d!}e^{-a}\right) \tag{2.2}$$

$$with \qquad a = \sum_h W_{dh}s_h$$

where $W \in \mathbb{R}^{DxH}$, $W_{d,h} \geq 0 \ \forall d,h$, H is the number of hidden units s_h and π is the sparsity parameter.

The hidden latent values are generated according a Bernoulli distribution, which creates a bit vector $\vec{s} \in \{0,1\}^H$, that shows which units are active ($s_h = 1$) and inactive ($s_h = 0$).

The activation of these hidden units is supposed to be sparse, meaning that only few of them compared to the total number will be active at the same time. This sparsity assumption is application dependent, since some real hidden units exhibit this typical sparse behaviour, as it can be case of how neurons activate or the way speech is produced. Many data bases where sparse assumption is taken could then fit well to the model, for example.

Each observed variable y_d is drawn from a Poisson distribution with mean given by a linear combination of the basis functions (dictionary elements) $\vec{W}_h = (W_{1h}, \ldots W_{Dh})^T$ with the latent values, which means that only the basis functions of the active hidden units contribute to generate the observed variable (data point). This data point is then a linear combination of the basis functions plus some Poisson noise. Because of the Poisson distribution properties, all the observed variables will be non-negative integers.

Compared to other algorithms, the difference with standard NMF arises in the values of the encoding matrix, which in the case of BNMF are set to be unary values, either one or zero, while in NMF they only have a more general non-negativity constraint. That means, in BNMF the hidden units can only be active or inactive, as it happens in the BSC, but it is not possible to have an intermediate activation value.
In comparison with BSC, the main difference lies in the non-negativity of the W matrix (values can be negative in BSC) and the Poisson noise instead of the Gaussian one.

However, both of can be understood in a similar way, as a linear combination of active or inactive basis functions plus noise. The binary approach causes a loss of generality, but still represents a good approximation with a much simpler, easier to derive and implement method.

Chapter 3

Maximum Likelihood/ Expectation Maximization

3.1 EM Algorithm

The aim of a learning algorithm is to optimize the parameters $\Theta = (\pi, W)$ that maximize the data likelihood $L(\Theta) = \prod_n \big(p(\vec{y_n}|\Theta)\big)$. The likelihood gives us an estimation on how good the generative model, taking in account the parameters, fits our data. Thus, maximizing the likelihood will enable us to recover the parameters that better fit the generated data. There are several mathematical methods to maximize the likelihood, the one used in this thesis is called Expectation-Maximization (EM) Algorithm and a detailed explanation about it is now given.

First, for easiness, the log-likelihood will be used instead of the likelihood. We are enabled to use this simplification because the logarithm is a monotone function (i.e. the maximum of the logarithm of a function corresponds the maximum of the function itself).

$$\mathcal{L}(\Theta) = log\big[\prod_n \big(p(\vec{y_n}|\Theta)\big)\big] = \sum_n \log\big(p(\vec{y_n}|\Theta)\big) = \sum_n \log\Big[\sum_{\vec{s}} p(\vec{y_n}, \vec{s}\,|\Theta)\big)\Big] \tag{3.1}$$

where the sum over n is the sum over all the data points and the sum over $\vec{s}$ is the sum over all the 2^H possible vectors $\vec{s}$ for the hidden units.

It can be shown, using the Jensen's Inequality in the sum over the $\vec{s}$ states, that $\mathcal{L}(\Theta) - \mathcal{F}(q, \Theta) \geq 0$ where $\mathcal{F}(q, \Theta)$ is a function called Free Energy equal to:

$$\mathcal{F}(\Theta, q(\Theta^{old})) = \sum_n \sum_{\vec{s}} q_n\big(\vec{s}, \Theta^{old}\big) \Big[\log\big(p(\vec{y}_n|\vec{s}, \Theta)\big) + \log\big(p(\vec{s}|\Theta)\big)\Big] + \sum_n H\Big[q_n\big(\vec{s}, \Theta^{old}\big)\Big] \tag{3.2}$$

where $H\big[q_n\big(\vec{s}, \Theta^{old}\big)\big]$ is the Shannon entropy (only dependent on the old parameters) and $q_n\big(\vec{s}, \Theta\big)$ is an approximation to the exact posterior probability. Following some derivations, it can also be shown that this quantity is exactly equal to:

$$\mathcal{L}(\Theta) - \mathcal{F}(\Theta) = \sum_n D_{KL}\Big(q_n\big(\vec{s}, \Theta^{old}\big), p(\vec{s}|\vec{y}_n, \Theta)\Big) \geq 0 \tag{3.3}$$

where $p\big(\vec{s}|\vec{y}_n, \Theta)\big)$ is the exact posterior probability and

$$D_{KL}\Big(q_n\big(\vec{s}, \Theta^{old}\big), p(\vec{s}|\vec{y}_n, \Theta)\Big) = -\sum_{\vec{s}} q_n\big(\vec{s}, \Theta^{old}\big) \log\left(\frac{p(\vec{s}|\vec{y}_n, \Theta)}{q_n\big(\vec{s}, \Theta^{old}\big)}\right)$$

is the Kullback-Leibler Divergence with properties:

$$D_{KL}(q, p) \geq 0 \qquad D_{KL}(q, p) = 0 \iff q = p \tag{3.4}$$

The algorithm goal is to maximize the log-likelihood. This would be done by setting the derivatives of the log-likelihood w.r.t the parameters equal to 0, and deriving then the update rule for the parameters from these equations. However, as these derivatives are usually difficult to compute, $\mathcal{L}(\Theta) = \mathcal{E}(\Theta)$ is set and the derivatives of the free energy w.r.t the parameters are computed instead, since they are analytically easier to derive. The previous equality implies, by means of 3.3 and 3.4, that $q_n\big(\vec{s}, \Theta^{old}\big) := p(\vec{s}|\vec{y}_n, \Theta)$. Once the free energy has the same value as the log-likelihood, we can proceed to compute its derivatives and maximize it with respect to the parameters. There will be an increase (or kept constant) in the free energy value, and because of 3.3, an increase (or kept constant) in the log-likelihood as well, provided that we compute its new value with the new parameters. The whole process can be described as follows:

$$\mathcal{L}(\Theta^{old}) = \mathcal{F}\big(\Theta^{old}, q\big(\vec{s}, \Theta^{old}\big) = p(\vec{s}|\vec{y}, \Theta)\big) \leq \mathcal{F}\big(\Theta^{new}, q\big(\vec{s}, \Theta^{old}\big)\big) \leq \mathcal{L}(\Theta^{new})$$

The free energy is maximized in a two-step process: in the first step, called the E-Step, there is a maximization with respect to the q_n (while Θ remains fixed), and in the second step, called the M-Step, there is a maximization with respect to the parameters

$\Theta = (\pi, W)$ (while q_n remains fixed now). The iterative repetition of this optimization process leads to an always increasing log-likelihood value until we end in a (local or global) maximum. It can be summarized in the following pseudocode:

```
init $\Theta^{old}$
    $q_n\big(\vec{s}, \Theta^{old}\big) = p(\vec{s}\,|\vec{y}_n, \Theta^{old})$        $E - Step$
    $\Theta^{new} = \underset{x}{\text{argmax}}\{\mathcal{F}\big(\Theta, q(\Theta^{old})\big)\}$        $M - Step$
    $\Theta^{old} = \Theta^{new}$
until convergence
```

3.1.1 E-Step

In the EM algorithm, the posterior probabilities are computed, and next the parameters are updated according to the new posterior probabilities following the update rules. The value of the posterior probability can be derived as follows, after making use of Bayes' Theorem.

$$p\big(\vec{s}\,|\vec{y}_n, \Theta)\big) = \frac{\big(p(\vec{y}_n|\vec{s}, \Theta)\big)\big(p(\vec{s}\,|\Theta)\big)}{\sum_{\vec{s}\,'}\big(p(\vec{y}_n|\vec{s}\,', \Theta)\big)\big(p(\vec{s}\,'|\Theta)\big)} \tag{3.5}$$

However, as it will be shown in 3.2.2 the explicit computation of the posterior probability is not necessary.

3.1.2 M-Step. Derivation of the update rules

The parameter update rules are derived by setting the derivatives of 3.2 with respect to the different parameters to zero. As the Shannon entropy (second term of the sum 3.2) does not depend on the new parameters, these derivatives are equal to the derivatives of the first term of the sum in 3.2, called $\mathcal{Q}(\Theta)$.

$$\mathcal{Q}(q_n, \Theta) = \sum_n \sum_{\vec{s}} q_n\big(\vec{s}, \Theta^{old}\big)\Big[\log\big(p(\vec{y}_n|\vec{s}, \Theta)\big) + \log\big(p(\vec{s}\,|\Theta)\big)\Big]$$

Using 2.1 and 2.2, the previous equation can be simplified to:

$$\mathcal{Q}(q_n,\Theta) = \sum_n \sum_{\vec{s}} q_n\left(\vec{s},\Theta^{old}\right)\Bigg[\sum_d \left[y_d^{(n)}\log(a) - a - \log\left(y_d^{(n)}!\right)\right] + \sum_{h=1}^{H}\left(s_h\log(\pi) + (1-s_h)\log(1-\pi)\right)\Bigg] \quad (3.6)$$

We first derive the update rule for π :

$$\frac{\partial}{\partial\pi}\mathcal{Q}(q_n,\Theta) \stackrel{!}{=} 0$$

$$\Rightarrow \pi^{new} = \frac{1}{N}\sum_n\sum_{\vec{s}} q_n\left(\vec{s},\Theta\right)\frac{|\vec{s}\,|}{H} = \frac{1}{HN}\sum_n \left\langle|\vec{s}\,|\right\rangle_{q_n(\vec{s},\Theta)} \quad (3.7)$$

where

$$\left\langle g(\vec{s})\right\rangle_{q_n(\vec{s},\Theta)} = \sum_{\vec{s}}\left(p(\vec{s}\,|\,\vec{y}_n,\Theta)\right)g(\vec{s})$$

The whole derivation is shown in the Appendix A.

We derive next the update rule for W in the same way.:

$$\frac{\partial}{\partial W_{d'h'}}\mathcal{Q}(q_n,\Theta) \stackrel{!}{=} 0$$

$$\Rightarrow W^{new} = \left(\sum_n \vec{y}^{\,(n)}\left\langle\vec{s}^{\,T}\right\rangle_{q_n(\vec{s},\Theta)}\right)\left(\sum_n\left\langle\vec{s}\,\vec{s}^{\,T}\right\rangle_{q_n(\vec{s},\Theta)}\right)^{-1} \quad (3.8)$$

The whole derivation is shown in the Appendix A.

Note: it is important to remark, that, since there is a matrix inversion involved in the computation of the update rule for W (See 3.8), negative values are possible to appear which would contradict the non-negativity constraint of the data. To avoid such problems, a simple solution is implemented: after each iteration, all negative values of W are set to 0.

3.1.3 Log-Likelihood Computation

After the E-Step and the M-Step, a computation of the likelihood will be done in the algorithm. Setting the generative model equations 2.1 and 2.2 in the equation 3.1 and following some derivations we obtain the log-likelihood formula for this particular generative model:

$$\mathcal{L}(\Theta) = \sum_{n} \log \left[\sum_{\vec{s}} \prod_{d=1}^{D} \left(\frac{a^{y_d}}{y_d!} e^{-a} \right) \cdot \prod_{h=1}^{H} \left(\pi^{s_h} (1-\pi)^{1-s_h} \right) \right] \tag{3.9}$$

3.2 Specifics of the algorithm

In this section, some of the procedures used in the implementation for a better performance of the algorithm are explained.

3.2.1 Exp and Log simplification

One simple implementation is set to avoid numerical problems in the running of the algorithm. Taking a look in the already simplified (elimination of factorials) expression of the posterior probability given by:

$$p\big(\vec{s}\,|\vec{y}_n, \Theta)\big) = \frac{\big(p(\vec{y}_n|\vec{s}, \Theta)\big)\big(p(\vec{s}\,|\Theta)\big)}{\sum_{\vec{s}\,'} \big(p(\vec{y}_n|\vec{s}\,', \Theta)\big)\big(p(\vec{s}\,'|\Theta)\big)} = ... = \frac{\prod_{d=1}^{D} \left(a^{y_d} e^{-a}\right) \prod_{h=1}^{H} \left(\pi^{s_h} (1-\pi)^{1-s_h}\right)}{\sum_{\vec{s}\,'} \left[\prod_{d=1}^{D} \left(a^{y_d} e^{-a}\right) \prod_{h=1}^{H} \left(\pi^{s'_h} (1-\pi)^{1-s'_h}\right) \right]}$$

it can be noticed that a product over many terms appear in both numerator and denominator. These different terms may have values differing in many orders of magnitude, so the multiplication between them could lead to round-off errors because of the different precisions and cause numerical problems. The solution to avoid that big differences in values is to simplify the previous expression by computing its logarithm, which will reduce the gap in the magnitude order, and subsequently, by computing the exponential of the result, so both functions cancel mutually, with no effect in the final result. Written in equations:

$$p(\vec{s}\,|\vec{y}_n,\Theta)) = \frac{exp\left[\sum\limits_{d=1}^{D}\log\left(a^{y_d}e^{-a}\right) + \sum\limits_{h=1}^{H}\log\left(\pi^{s_h}(1-\pi)^{1-s_h}\right)\right]}{\sum\limits_{\vec{s}\,'} exp\left[\sum\limits_{d=1}^{D}\log\left(a^{y_d}e^{-a}\right) + \sum\limits_{h=1}^{H}\log\left(\pi^{s_h^{'}}(1-\pi)^{1-s_h^{'}}\right)\right]}$$

$$= \frac{exp\left[\sum\limits_{d=1}^{D} y_d\log(a) - a + \sum\limits_{h=1}^{H} s_h\log\pi + (1-s_h)\log(1-\pi) + B\right]}{\sum\limits_{\vec{s}\,'} exp\left[\sum\limits_{d=1}^{D} y_d\log(a) - a + \sum\limits_{h=1}^{H} s_h^{'}\log\pi + (1-s_h^{'})\log(1-\pi) + B\right]}$$

The appearance of the B as a numerical stabilization constant is due to the fact that this simplification paradoxically could lead to another type of numerical problems, because now the terms inside the exponential could have a large negative value ($\approx -10^3$) and when computing the exponential of them, the result would be directly zero as they are out of machine precision. To avoid that, this factor B is added in both numerator and denominator, so it cancels itself with no effect in the final result, but allowing us to compute these quantities.

Note: this procedure is also used in the computation of the likelihood (Eq. 3.9 will be updated)

3.2.2 Computation of Expectation Values

It can be seen in the update rules 3.7 and 3.8 that only the expectations values with respect to $\vec{s}^T$ and with respect to $\vec{s}\,\vec{s}^T$ are needed, not the posterior probabilities themselves. These two expectation values are called sufficient statistics, since their computation is sufficient to compute the update rules and therefore the explicit computation of the posterior probabilities can be avoided saving computing time. The expectation value of a function $g(\vec{s})$ is defined as follows:

$$\langle g(\vec{s})\rangle_{q_n(\vec{s},\Theta)} = \sum_{\vec{s}} \left(p(\vec{s}|\vec{y}_n,\Theta)\right) g(\vec{s}) \tag{3.10}$$

Making use of 3.5, equation 3.10 can be rewritten as:

$$\langle g(\vec{s})\rangle_{q_n(\vec{s},\Theta)} = \frac{\sum\limits_{\vec{s}} \big(p(\vec{y}_n|\vec{s},\Theta)\big)\big(p(\vec{s}\,|\Theta)\big)g(\vec{s})}{\sum\limits_{\vec{s}\,'} \big(p(\vec{y}_n|\vec{s}\,',\Theta)\big)\big(p(\vec{s}\,'|\Theta)\big)} \tag{3.11}$$

where $g(\vec{s}) = \vec{s}^{\,T}$ or $g(\vec{s}) = \vec{s}\,\vec{s}^{\,T}$.

3.2.3 Approximation schemes: Expectation truncation

It can be noticed that the computation of the exact posterior probability requires a sum over 2^H terms (See eq. 3.5 and 3.11). That means, this computation becomes rapidly intractable for an increasing number of latent variables H, as the numbers of terms in the sum increases exponentially with the number of hidden units. To look for tractable approximations, several approaches can be tried, including MAP, sampling, factored variational EM or expectation truncation among others.

Although it would not be needed given the scale of my project to use these approximations, to implement one is a good way to get a small insight on how advanced researchers address these kind of problems and it will also optimize my algorithm by making it faster with little loss of precision. In our case, expectation truncation [3] is the approximation that will be applied to get a computational tractable algorithm. It is explained as follows. The expectation value (eq. 3.11) can be written without loss of generality like:

$$\langle g(\vec{s})\rangle_{q_n(\vec{s},\Theta)} = \frac{\sum\limits_{\vec{s}=0}\big(p(\vec{y}_n,\vec{s}|\Theta)\big)g(\vec{s}) + \sum\limits_{\substack{\vec{s}\\|\vec{s}|=1}}\big(p(\vec{y}_n,\vec{s}|\Theta)\big)g(\vec{s}) + \sum\limits_{\substack{\vec{s}\\|\vec{s}|=2}}\big(p(\vec{y}_n,\vec{s}|\Theta)\big)g(\vec{s}) + ...}{\sum\limits_{\vec{s}=0}\big(p(\vec{y}_n,\vec{s}|\Theta)\big) + \sum\limits_{\substack{\vec{s}\\|\vec{s}|=1}}\big(p(\vec{y}_n,\vec{s}|\Theta)\big) + \sum\limits_{\substack{\vec{s}\\|\vec{s}|=2}}\big(p(\vec{y}_n,\vec{s}|\Theta)\big) + \sum\limits_{\substack{\vec{s}\\|\vec{s}|=3}}\big(p(\vec{y}_n,\vec{s}|\Theta)\big) + ...}$$

The sum over all the possible bit vectors is rewritten in terms dependent on its "module", i.e., the number of active hidden units. If we now assume sparsity on the model (small π), we can see that the average number of active hidden units is πH, which means that the data is less likely to come from the bit vectors with most of the hidden units active, as long as π remains low. Therefore, the values of $\big(p(\vec{y}_n,\vec{s}|\Theta)\big)$ in those terms are really small in comparison to the other terms, meaning that we can remove them from the sum over all the possible states without a big change in the result.
Only vectors with $|\vec{s}| \leq \gamma$ are computed in the sum, where γ (truncation parameter) is set to be equal to πH. The sum in the denominator is extended to vectors with $|\vec{s}| \leq \gamma + 1$ due to numerical reasons.
This makes the computation more tractable, as much of the terms of the sum are removed

and don't need to be computed, but it still remains polynomial, which means that there will still be computational problems if H is really large, where a sampling approach could be used instead.

However, this approximation is valid for the W update because it is a parameter of the noise model, but not for the π update, since it is a special case due to the generative model where π is a parameter of the prior distribution.
To solve this problem, a correction to the update rule of π can be derived [2], but it is beyond the scope of this thesis. To avoid this problem, one possible solution could be to set the π parameter to the ground truth value, and learn only the W parameter. However, this implies a previous knowledge of the ground truth parameters to be able to run the algorithm, which is not what an unsupervised learning algorithm is supposed to do. The easiest way to solve the problem is to raise the truncation limit to a higher number of active units for the last iterations, once the algorithm has already converged to the maximum, allowing us to learn the π parameter then with little effect in the learning of W.

Chapter 4

Numerical Experiments

The previous equations constitute a learning algorithm that optimizes the parameters of the generative model. In order to test the performance of the algorithm, we run several tests on artificial and realistic data.

4.1 Linear bars test

The algorithm is applied to artificial bars data as shown in the Figure 4.1. A relative grey scale is used to represent the data where black represents the minimum value and white the maximum one. To generate this data, $H = 8$ linear horizontal and vertical bars are taken as the basis functions $\vec{W}_h$ of the matrix W. Each bar occupies 4 pixels on a grid made by D = 4x4 pixels. The values of the pixels are set to be $W_{d,h} = \{0.0, 10.0\}$. The number of data points N is equal to 1000. These bars can be seen in the Figure 4.2. The π value is set to be 0.3, so that on average $H\pi = 2.4$ hidden units are active.

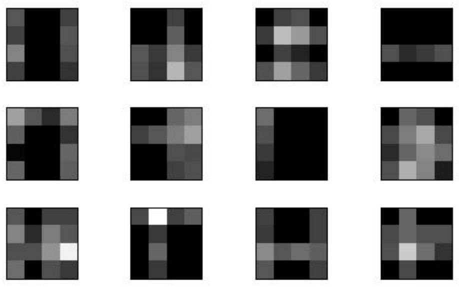


FIGURE 4.1: Random sample of 12 data points.

I run the algorithm for 60 iterations, and for both cases, with and without the expectation truncation approximation to check the differences. The initialization of the W

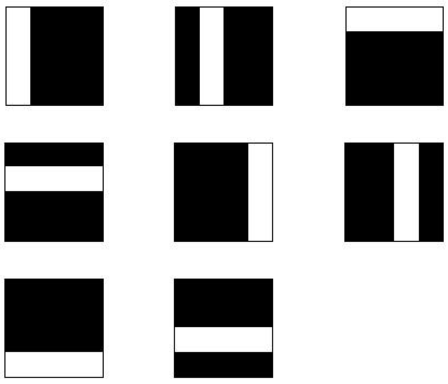
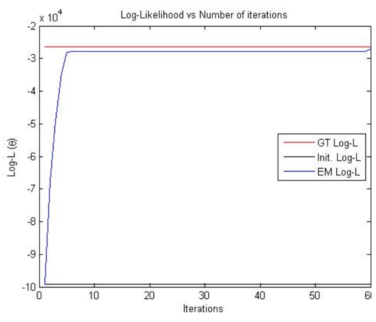


FIGURE 4.2: The 8 linear bars used for data generation. D = 4 x 4

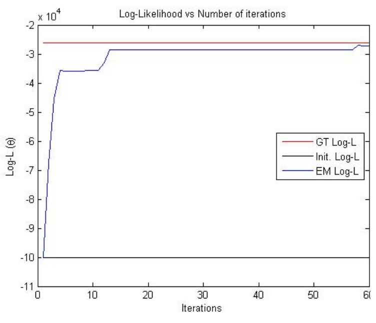
values is uniform random between 0 and 1, and the π value is also set to be uniform random between 0 and 1. After each iteration some small random positive Poisson noise is added to the W parameter to avoid local optima where the algorithm can get stuck.

To test if the EM-algorithm is working properly, the log-likelihood is computed to check if the log-likelihood is increasing (or remaining the same) on each iteration, as it is supposed to do. Several typical evolutions of the log-likelihood function are plotted in Figure 4.3. In all of them a continuous increase of the log-likelihood can be observed, which gives us a positive feedback on the operation of the algorithm.
In the Figure 4.3a what we can observe is a fast convergence to somewhere close to the global maximum having learned all the bars, being the log-likelihood of the learned parameters after convergence only slightly lower as the one of the generative (ground truth) parameters. In the figure 4.3b , convergence after learning all the bars can also be observed, but in this case, it can be noticed how the system gets stuck in a local optima (flatness of the function) during some iterations before getting out of it to reach the final point. In figure 4.3c however, it can be noticed that all the bars are not properly learned, remaining the system in the local optima after the 60 iterations.
The small increment of the function that can be noticed at the end of the process is due to the change in the expectation truncation condition in the last iterations in order to learn the π parameter, which leads the algorithm to a new maximum.

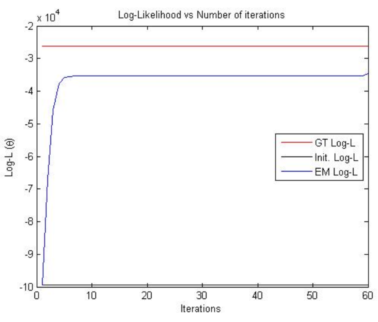
It needs to be remarked that there is a situation where a decrease in the log-likelihood occurs, although I believe it is due to numerical problems and not to errors in the algorithm, because finally the parameters are learned correctly in most of the cases and besides, there is an explanation for it. This situation takes place when the parameters are initialized as the ground truth parameters. Because there is a matrix inversion and the values are the same, that could lead sometimes to numerical errors, but the decrease



(A) Global maximum



(B) Global maximum after local optima



(C) Local optima

FIGURE 4.3: Likelihood plots for proper functioning

in the likelihood is so small (see Fig. 4.4) that I believe the algorithm is running properly. Another possible reason for this decrease can be the approximation done in the W update rule (See Appendix A.2). Since it is an approximated update rule, we have no more an exact EM-Algorithm, and therefore, the non-decreasing likelihood outcome could not happen. It is important to notice that this concrete initialization is almost impossible to happen randomly, but it has to be set artificially.

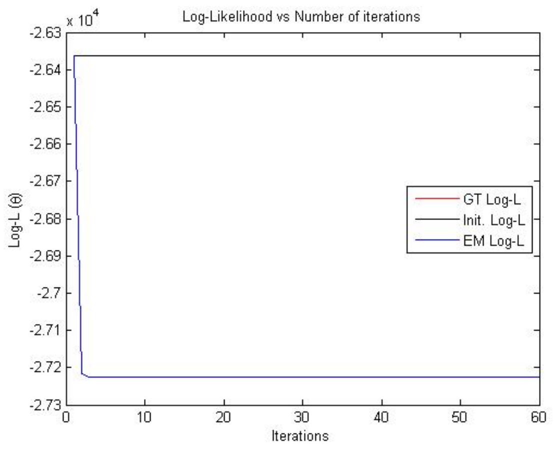
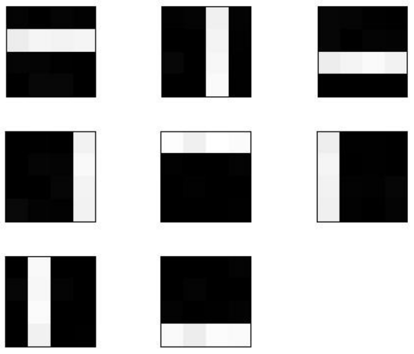


FIGURE 4.4: Likelihood decrease due to exceptional initialization

In general, the algorithm is able to learn all the bars, which is the desired output, in $91 \pm 5\%$ of the runs when no ET is applied and in $84 \pm 5\%$ when ET is applied for 50 runs, seeming not to be a big difference in the success rate when the approximation is applied.

To show the final result of the learning process, two W learned by the algorithm in two different runs are shown in Figure 4.5: the Figure 4.5a shows the learned W when the algorithm converges after learning all the bars, which is almost indistinguishable to the ground truth W value shown in Fig. 4.2, although the relative grey scale has to be reminded. On the other hand, Figure 4.5b shows the final W learned in a run where the algorithm does not converge to the desired result but to a local optima. The usual case happens by a superposition of some bars while the others are learned correctly.


(A) W of global maximum

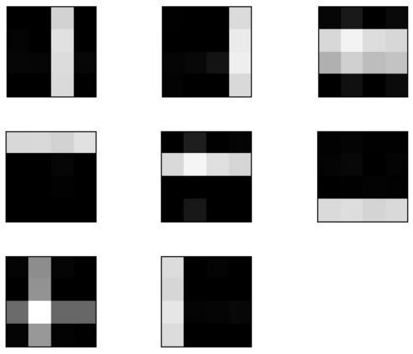

(B) W of local optima

FIGURE 4.5: Final results of the learned W

Finally, the evolution of the parameters W and π with the number of iterations is shown in the figures 4.6 and 4.7, respectively. The evolution chosen to be shown is the one in which the algorithm converges after a small number of iterations, as it is, by far, the most common case and also the most representative one.

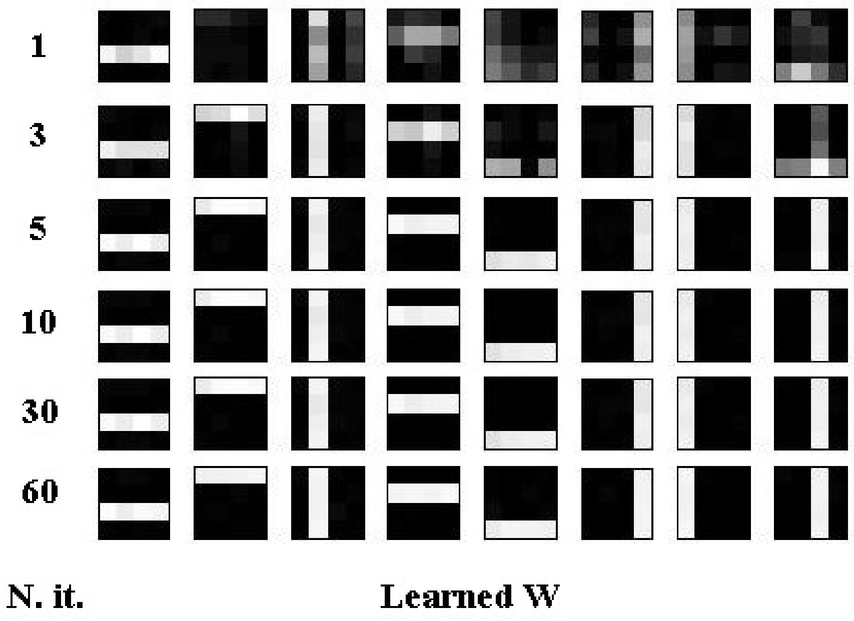
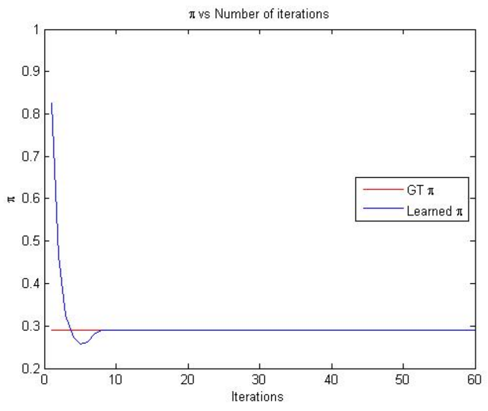
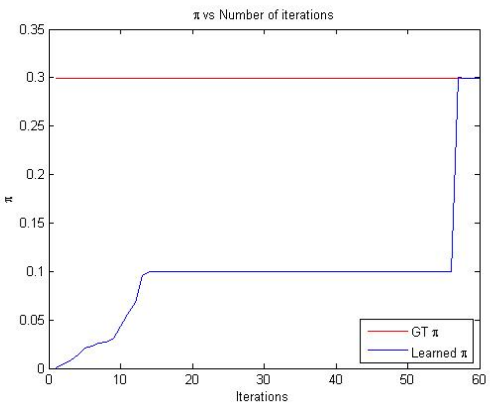


FIGURE 4.6: Evolution of W with the number of iterations

It needs to be reminded that, while the evolution of the W is practically independent of the application of expectation truncation, π's evolution is not. Therefore, two figures are shown, one for a process in which expectation truncation is not applied (Figure 4.7a),



(A) Evolution of π vs. N. of It. (No ET)



(B) Evolution of π vs. N. of It. (ET)

FIGURE 4.7: Final results of the learned W

where the convergence takes place at the early iterations, and the other one for a process in which expectation truncation is applied (Figure 4.7b), having a convergence at the last iterations due to the condition implemented for the learning of π (See Subsection 3.2.3).

4.2 Handwritten digits

Although it was not requested in the thesis goal, I decided to run the algorithm with some realistic data as input to check its performance. However, The real data is taken from the MNIST database of handwritten digits, composed of 60000 preprocessed images of 28x28 pixels. In my thesis, for simplicity only two digits are used; 1 and 7, chosen because of their similarity. Because of computation capacity , the number of data points is reduced to a total amount of 6500 data points with a normalized maximum pixel value of 10, as in the previous eight test bar. In figure 4.8 some random data points are shown.

The number of hidden units is chosen to be $H = 16$, the initialization of W is again uniform random between 0 and 1 and the algorithm is run over 60 iterations with the expectation truncation approximation. The W learned is shown in the figure 4.9:

In the figure it can be noticed that the algorithm learns holistic representations (whole digits), instead of the expected parts-learning of NMF, which is of course not the desired output but still a good result. This typical NMF parts learning may be be seen in a couple of basis functions which are made of only one stroke of the digits, but they are still too dim to be considered a good learning. Nevertheless, the objective of this numerical experiment is not to fully learn the basis functions but to check whether the algorithm could perform some correct learning with real data, which it indeed does.

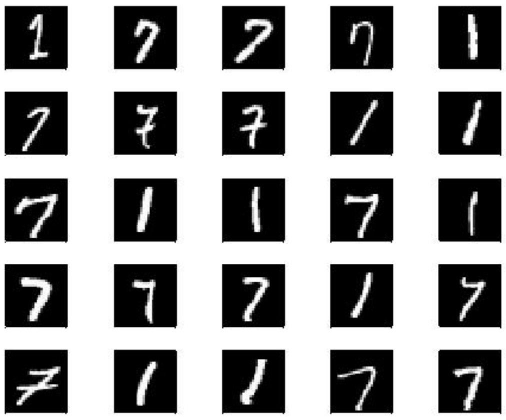


Figure 4.8: Random sample of data points (digits)

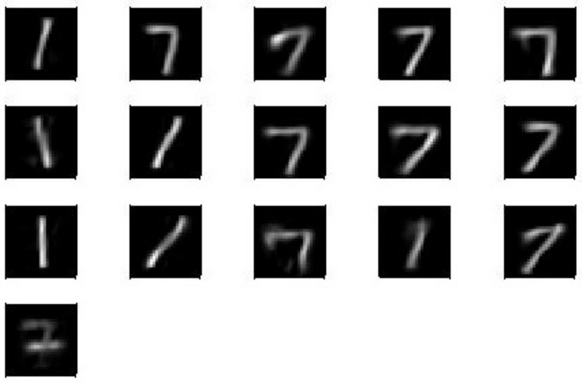


Figure 4.9: Learned W for handwritten digits

Chapter 5

Conclusions

In this bachelor thesis I have investigated a probabilistic generative model for unsupervised learning for non-negative data, that can be considered as a simplification of the Standard NMF. We use a Bernoulli probability distribution for the hidden units, yielding a simplified probabilistic model.
An EM Algorithm is used to learn the parameters of the model by maximizing the likelihood of the data. I have derived the update rules for these parameters according to this EM Algorithm and implemented them in a computer algorithm.
To implement them, first an exact approach and later a simple approximation one (to make possible tractable algorithms) have been taken, emphasising the advantages and disadvantages of both methods.
The learning algorithm has later been tested on artificial data (eight bars test) with good results, resulting in an operating algorithm that learns correctly the parameters in a high percentage of the cases, although not optimal, since no more advanced implementations have been tried taking account the scope of a bachelor thesis.
Finally, I run the algorithm on handwritten digits just to check how it works with real data with no special implementation for it. The expected behaviour in NMF (parts learning) is not observed but rather holistic representation learning occurs, being anyway a good sign because it shows the algorithm is applicable to real data.

Appendix A

Complete Derivation of the Update Rules

In this Appendix the update rules for the parameters in the M-Step will be derived completely.
The derivation of the update rule for W was given to me by my supervisor, since this derivation was considered to be out of the scope of this bachelor thesis.

A.1 Derivation for π

Derivation of the update rule for π is made by setting the derivative of $\mathcal{Q}(\Theta)$ w.r.t. π equal 0.

$$\frac{\partial}{\partial \pi} \mathcal{Q}(\Theta) \overset{!}{=} 0$$

Substituting the value of $\mathcal{Q}(\Theta)$, we obtain:

$$\frac{\partial}{\partial \pi}\left[\sum_{n}\sum_{\vec{s}} q_n\left(\vec{s}, \Theta^{old}\right)\left[\sum_{d}\left[y_d^{(n)}\log(a) - a - \log\left(y_d^{(n)}!\right)\right] + \sum_{h=1}^{H}\left(s_h\log(\pi) + (1-s_h)\log(1-\pi)\right)\right]\right] \overset{!}{=} 0$$

$$\sum_n \sum_{\vec{s}} q_n(\vec{s},\Theta) \sum_{h=1}^{H} \left[\frac{\partial}{\partial \pi} s_h \log(\pi) + (1-s_h)\log(1-\pi))\right] \stackrel{!}{=} 0$$

$$\sum_n \sum_{\vec{s}} q_n(\vec{s},\Theta) \sum_{h=1}^{H} \left[s_h \frac{1}{\pi} + (1-s_h)\frac{1}{1-\pi}(-1)\right] \stackrel{!}{=} 0$$

$$\sum_n \sum_{\vec{s}} q_n(\vec{s},\Theta) \sum_{h=1}^{H} \left[s_h(1-s_h) - \pi(1-s_h)\right] \stackrel{!}{=} 0$$

$$\sum_n \sum_{\vec{s}} q_n(\vec{s},\Theta) \sum_{h=1}^{H} (s_h - \pi) \stackrel{!}{=} 0$$

$$\Rightarrow \sum_n \sum_{\vec{s}} q_n(\vec{s},\Theta)|\vec{s}\,| = \sum_n \sum_{\vec{s}} q_n(\vec{s},\Theta)\pi H$$

where

$$|\vec{s}\,| := \sum_{h=1}^{H} s_h$$

Therefore, the update rule for π as follows :

$$\pi^{new} = \frac{1}{N} \sum_n \sum_{\vec{s}} q_n(\vec{s},\Theta) \frac{|\vec{s}\,|}{H}$$

A.2 Derivation for W

Derivation of the update rule for W is made by setting the derivative of $\mathcal{Q}(\Theta)$ w.r.t. W equal 0.

$$\frac{\partial}{\partial W_{d'h'}} \mathcal{Q}(\Theta) \stackrel{!}{=} 0$$

Substituting the value of $\mathcal{Q}(\Theta)$, we obtain:

$$\frac{\partial}{\partial W_{d'h'}} \left[\sum_n \sum_{\vec{s}} q_n(\vec{s},\Theta^{old}) \left[\sum_d \left[y_d^{(n)} \log(a) - a - \log\left(y_d^{(n)}!\right)\right] + \sum_{h=1}^{H} \left(s_h \log(\pi) + (1-s_h)\log(1-\pi)\right) \right] \right] \stackrel{!}{=} 0$$

$$\sum_n \sum_{\vec{s}} q_n(\vec{s},\Theta) \frac{\partial}{\partial W_{d'h'}} \sum_d \left[y_d^{(n)} \log(a) - a - \log\left(y_d^{(n)}!\right)\right] \overset{!}{=} 0$$

$$\sum_n \sum_{\vec{s}} q_n(\vec{s},\Theta) \frac{\partial}{\partial W_{d'h'}} \left(\sum_d y_d^{(n)} \log\Big(\sum_h W_{dh} s_h\Big) - \sum_d \sum_h W_{dh} s_h \right) \overset{!}{=} 0$$

$$\sum_n \sum_{\vec{s}} q_n(\vec{s},\Theta) \left[\sum_d \Big(y_d^{(n)} \cdot \frac{1}{\sum\limits_h W_{dh} s_h} \cdot \frac{\partial}{\partial W_{d'h'}} \Big(\sum_h W_{dh} s_h \Big)\Big) - \frac{\partial}{\partial W_{d'h'}} \Big(\sum_d \sum_h W_{dh} s_h \Big) \right] \overset{!}{=} 0$$

$$\sum_n \sum_{\vec{s}} q_n(\vec{s},\Theta) \left[\sum_d y_d^{(n)} \cdot \frac{1}{\sum\limits_h W_{dh} s_h} \sum_h s_h \delta_{h,h'} \delta_{d,d'} - \sum_d s_{h'} \delta_{h,h'} \delta_{d,d'} \right] \overset{!}{=} 0$$

$$\sum_n \sum_{\vec{s}} q_n(\vec{s},\Theta) \left[y_{d'}^{(n)} \cdot \frac{s_{h'}}{\sum\limits_h W_{d'h} s_h} - s_{h'} \right] \overset{!}{=} 0$$

$$\sum_n \sum_{\vec{s}} q_n(\vec{s},\Theta) \cdot s_{h'} \cdot \left[\frac{y_{d'}^{(n)} - \sum\limits_h W_{d'h} s_h}{\sum\limits_h W_{d'h} s_h} \right] \overset{!}{=} 0$$

An approximation of W update rule can be derived by neglecting the denominator in the previous equation:

$$\sum_n \sum_{\vec{s}} q_n(\vec{s},\Theta) \cdot y_{d'}^{(n)} \cdot s_{h'} \overset{!}{=} \sum_h W_{d'h} \sum_n \sum_{\vec{s}} q_n(\vec{s},\Theta) \cdot s_{h'} \cdot s_h$$

$$\sum_n y_{d'}^{(n)} \langle s_{h'} \rangle_{q_n(\vec{s},\Theta)} \overset{!}{=} \sum_h W_{d'h} \sum_n \langle s_{h'}\, s_h \rangle_{q_n(\vec{s},\Theta)}$$

Changing to matrix notation:

$$\sum_n \vec{y}^{(n)} \langle \vec{s}^T \rangle_{q_n(\vec{s},\Theta)} \overset{!}{=} W \sum_n \langle \vec{s}\, \vec{s}^T \rangle_{q_n(\vec{s},\Theta)}$$

The update rule for W is derived as follows:

$$W^{new} = \Big(\sum_n \vec{y}^{(n)} \, \langle \vec{s}^T \rangle_{q_n(\vec{s},\Theta)} \Big) \Big(\sum_n \langle \vec{s}\, \vec{s}^T \rangle_{q_n(\vec{s},\Theta)} \Big)^{-1}$$

Bibliography

[1] Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999. URL http://www.columbia.edu/~jwp2128/Teaching/W4721/papers/nmf_nature.pdf.

[2] Marc Henniges, Gervasio Puertas, Jörg Bornschein, Julian Eggert, and Jörg Lücke. Binary sparse coding. In *Latent Variable Analysis and Signal Separation*, pages 450–457. Springer, 2010. URL http://fias.uni-frankfurt.de/~bornschein/papers/HennigesEtAl_lva2010.pdf.

[3] Jörg Lücke and Julian Eggert. Expectation truncation and the benefits of preselection in training generative models. *The Journal of Machine Learning Research*, 11:2855–2900, 2010. URL http://www.jmlr.org/papers/volume11/lucke10a/lucke10a.pdf.