Exploration of Boltzmann Machines via Bars and Stripes

Joan Ariño Bernad and Adam Teixidó Bonfill

Abstract: In this article the use of Restricted Boltzmann Machines, a machine learning technique, will be discussed. It will be applied to a toy example, Bars and Stripes. The algorithm will be presented theoretically and practically implemented. Special attention will be paid to execution time under different conditions, hyper-parameter tuning for improving the algorithm and its capabilities on detecting patterns.

I. THEORETICAL INTRODUCTION

A. Machine Learning and Restricted Boltzmann Machines

Tasks which are mundane to us, such as understanding words we hear or read, recognizing faces, driving a car, translating, or even distinguishing between a cat and a chair, are not easily done by a computer. Machine learning (ML) has solved these problems in recent years [1].

The existing big interest in ML has made that us, as students, had the desire of having a first contact with it. For doing so we have chosen, under the tutelage of our supervisors, a type of ML, Boltzmann Machines, and a toy example, Bars & stripes, which we will explore.

Boltzman machines (BM) is a type of ML connected to statistical physics. A BM is based on units which can be on (1) or off (0). Those units are classified on whether they are observed or hidden, forming two layers. Their state can be represented by two vectors of ones and zeros, called $\boldsymbol{x}$ and $\boldsymbol{h}$ respectively. The whole BM assigns a probability to each state of the units. To do so first the BM assigns an energy to the system, $\mathrm{Energy}(\boldsymbol{x}, \boldsymbol{h})$. Then the energy determines the probability that such $\boldsymbol{x}$ and $\boldsymbol{h}$ occur as in a a Boltzmann distribution [3]:

$$p(\boldsymbol{x}, \boldsymbol{h}) \propto e^{-\mathrm{Energy}(\boldsymbol{x}, \boldsymbol{h})}$$

In short, a high energy state is more improbable.

The units are interconnected with weights and when a pair of units are activated they change the energy of the system by the weight of their connection. BMs functioning require heavy computations and we will instead use a Restricted Boltzman Machine (RBM), in which there are only connections between observed and hidden units, forming a bipartite graph. This makes each observed unit independent of other observed units, allowing efficient computation of the learning algorithm. In a RBM the energy function is:

$$\mathrm{Energy}(\boldsymbol{x}, \boldsymbol{h}) = -\boldsymbol{b}^\intercal \boldsymbol{x} - \boldsymbol{c}^\intercal \boldsymbol{h} - \boldsymbol{x}^\intercal W \boldsymbol{h}$$

B. Use of a RBM

RBMs will be used to create an energy function such that assigns a high probability to a set of vectors of observed states which will be called valid states. The probability of a observed state will be defined as $p(\boldsymbol{x}) = \sum_{\boldsymbol{h}} p(\boldsymbol{x}, \boldsymbol{h})$. To define the energy function the weights $\{\boldsymbol{b}, \boldsymbol{c}, W\}$, will be obtained.

The RBM will be trained with the set of valid states and an iterative algorithm will be applied to progressively modify the weights to decrease the energy of the valid states while increasing the energy of other states. To do so we will maximize the log-likelihood of the valid states. It will be done with an approximate form of gradient ascent.

C. Contrastive divergence and Gibbs Sampling

We like to perform gradient ascent, however the gradient takes the following form [2] :

$$\frac{\partial \log P(\boldsymbol{x})}{\partial \theta} = -\sum_{\boldsymbol{h}} P(\boldsymbol{h}|\boldsymbol{x}) \frac{\partial \,\mathrm{Energy}(\boldsymbol{x}, \boldsymbol{h})}{\partial \theta} ...$$

$$+ \sum_{\tilde{\boldsymbol{x}}, \boldsymbol{h}} P(\tilde{\boldsymbol{x}}, \boldsymbol{h}) \frac{\partial \,\mathrm{Energy}(\tilde{\boldsymbol{x}}, \boldsymbol{h})}{\partial \theta}$$

The exact computation directly stumps into the biggest problem faced in RBM, which is that the number of possible states is $2^{\#\mathrm{units}}$, that easily becomes immense. The gradient ascent requires to sum over all the possible states ($\sum_{\tilde{\boldsymbol{x}}, \boldsymbol{h}}$), which is unfeasible. With the restriction to RBMs, the log-likelihood gradient can be rewritten in a form that only requires to sum over all possible observed states ($\sum_{\tilde{\boldsymbol{x}}}$). Such simplification is the purpose of RBM, however that sum remains unfeasible for most purposes.

The Contrastive Divergence (CD) algorithm substitutes the sum by a sampling:

$$\sum_{\boldsymbol{x}} p(\boldsymbol{x}) f(\boldsymbol{x}) \xrightarrow{\text{substitute by}} \text{sample } \boldsymbol{x};\ \text{return } f(\boldsymbol{x});$$

The RBM probability distribution is too large to be exactly sampled, and CD-k uses instead the faster Gibbs sampling with k iterations, which provides a sampling similar to the given by the distribution [5]. CD works and allows to train the network for much bigger number of units, improvement that we will explore in section III.

D. Bars & Stripes

The problem we seek to solve is to recognize the Bars & Stripes data-set, which means to distinguish if a $n \times m$ matrix of X and 0 contains either exclusively bars, or exclusively stripes, where X represents a unit on and 0 a unit off. See FIG. 1 for some examples. In this particular problem the total number of cases grows as $2^{n \cdot m}$, while the number of valid cases grows only as $2^n + 2^m - 2$, so the provability of randomly guessing a valid example decreases very quickly.

```
X0XX0      00000      X0XXX      XX000
X0XX0      XXXXX      X0000      XXXXX
X0XX0      XXXXX      XX000      XXXXX
X0XX0      00000      X0000      XX000
```

(a) Valid (b) Valid (c) Not valid (d) Not valid

FIG. 1: Examples of matrices validity or non validity

II. CODE AND PRACTICAL ANALYSIS

Code for this project was generated by the authors using Python 3. All the relevant functions were coded in an online Jupyter notebook environment, Google Colab, and are available here. Published code also includes a simple demo to see how in the 3×3 case the system learns to repeat the instructed cases.

In the following sections we perform a practical analysis of our RBM implementation. To conduct it three points have been taken into account:

As the algorithms are non-deterministic both in its initialization and its execution, there is some variation in the results. To take this into account each data was collected in triplicate. Raw data used to plot the figures in the article can be obtained here.

The analysis of the algorithm has been performed with the following nominal set of parameters:
3×3 *matrix, 12 hidden units, learning rate 0.5, CD-1*
The nominal set of parameters has been chosen to lead to reasonable results in a relatively short amount of time.

As we want to evaluate performance in solving the Bars & Stripes task, our figure of merit is the failure rate in producing Bars & Stripes elements when the RBM is sampled by Gibbs sampling.

Aside from these points, we will use epochs as a standard measure of the number of iterations of our algorithms. An epoch passes each time the entire valid set is provided to the iterative algorithm.

III. EXECUTION TIME

Computation time is a limiting resource. If we only dispose of 20 minutes of computation, what size of Bars & Stripes can our RBM learn? We will measure time in Google Colab environment, which is approximately as powerful as a personal computer.

We compared gradient ascent and CD-1 performance at different $n \times m$ cases, in FIG. 2. We executed both algorithms for 2000 epochs or 20 minutes at most. Execution parameters were the nominal, except for the number of hidden units which were 1.5 times the number of observed units.

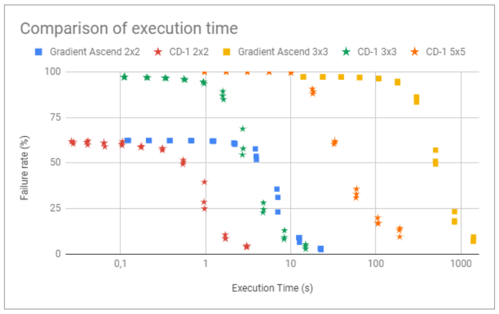

FIG. 2: Execution time needed to learn

Note that in FIG. 2, cases with a larger $n \times m$ start with higher failure rate. This happens because the valid set has a smaller relative size.

CD-1 clearly outperforms gradient ascent (GA) in lowering the failure rate: when CD-1 has solved the 3×3 case, the GA for 2×2 has not yet started. GA does not allow to use more than 9 observable (3×3) units, while CD-1 allows to train an RBM of up to $5 \times 5 = 25$ in under 20 minutes of Google Colab computation. Knowing that GA running time doubles with each new observed unit, we can predict that it would last around 2.5 years ($\approx 2^{16} \cdot 20min$) to solve 5×5. Therefore using CD-1 for practical application is usually necessary.

Now we continue the comparison taking into account the quality of the solution. Theoretically the algorithm tends to maximize the sum of log-likelihoods over the valid set. If the sum of log-likelihoods was a maximum the distribution would select, uniformly, only elements from the valid set.

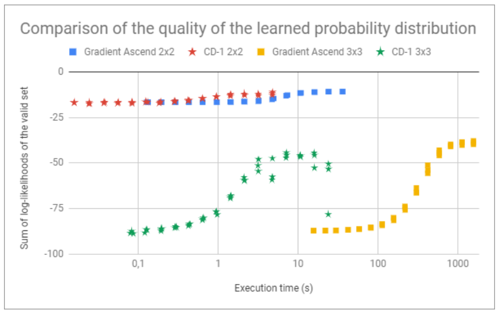

FIG. 3: Evolution of the sum of log-likelihoods

In FIG. 3 it is shown the evolution of the sum of log-likelihoods, for 3000 epochs, comparing the different methods.

It is clear that CD1 does not robustly increase the sum of log-likelihoods. Although it generally increases, significant fluctuations are very common. Conversely, GA reliably increases the sum of log-likelihoods, and converges to values higher than those reached by CD1.

The log-likelihood is not a figure of merit that evolves consistently with CD-1. Taking this into account, in following analysis, the failure rate will be used instead. It allows us to monitor the evolution of the performance with less noise.

The behaviours found in this section suggests using the CD-1 algorithm to find weights approximately correct, and then performing the slower exact gradient ascent to refine them. It remains as a future possibility.

IV. PARAMETER ANALYSIS

The described learning process for training a Boltzmann Machine requires to set many parameters such as the learning rate used in gradient ascent, the number of hidden neurons or the k of CD-k used. Those parameters have a very important effect on both the performance of the algorithm and its time consumption. Here we perform an analysis of those parameters.

For each parameter being studied all others will remain constant, taking the nominal value defined in section II.

A. Learning rate

When using gradient ascent, once the gradient is (exactly or approximately) obtained, the algorithm will advance in that direction, so:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \varepsilon \cdot \nabla f(\boldsymbol{x}_n)$$

Clearly the size of ε, the learning rate, has a great importance. A big learning rate could cause the algorithm to do not converge but at the same time an unnecessary small learning rate would cause the algorithm to be too slow.

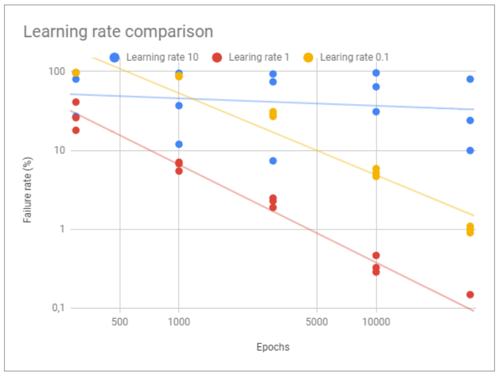

FIG. 4: Comparison of the learning rate

In the results of FIG. 4 it can be seen how for a learning rate of 10 the algorithm clearly does not converge and simply oscillates. On the other hand for learning rates of 1 and 0.1 the RBM converges. When comparing the learning rates of 1 and 0.1 the learning rate of 0.1 has more constant results, and it seems likely that with a very high number of epochs could perform better. However, selecting a learning rate of 1 instead of 0.1 makes the algorithm 10 times faster, specially at the start. This is computationally very relevant, as when the code was executed on the cloud, with the services provided by Google Colab, 30,000 epochs required of 15 minutes, a significant amount of time.

B. CD-k

As discussed in the introduction, CD uses Gibbs sampling to approximately sample a variable from a probability distribution that would be too expensive to compute. Gibbs sampling would provide a sample exactly according to the probability distribution if given an infinite number of iterations. Again, this is impossible in a practical implementation and a limited number of iterations are performed. When k iterations of Gibbs sampling are used, the resulting method is called CD-k.

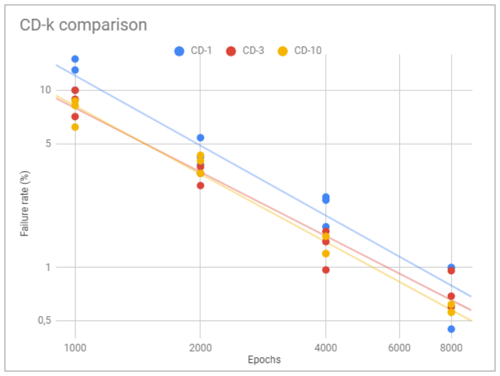

FIG. 5: Comparison on CD-1, CD-3 and CD-10

In the results of FIG. 5 it is appreciated how increasing the number of iterations above 1 results in a faster convergence. However no clear differences exists between CD-3 and CD-10. In addition, when the number of epochs increases the difference between the methods blurs.

The cost of running the algorithm is linear with the k of CD-k. Taking this into account, it is reasonable to limit the usage to CD-1, the form in which CD is commonly used [2].

C. Hidden variables

The number of hidden units also has an important impact on the algorithm. Very few hidden units will not provide enough flexibility to the algorithm. On the other hand, the algorithm time execution has a cost roughly linear with the number of hidden units and too many will make the algorithm too slow.

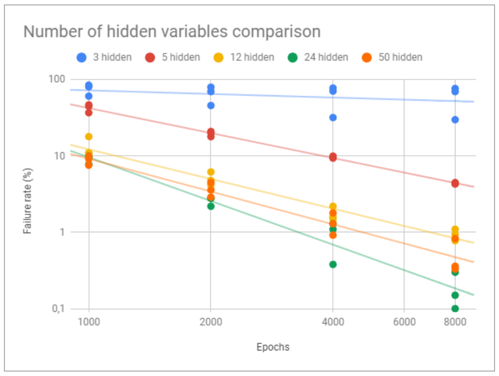

FIG. 6: Comparison on the number hidden units

From the results in FIG. 6 it can be concluded that 3 or less hidden units make it impossible for the algorithm to learn. Surprisingly, 5 hidden units are enough to train the RBM although with a reduced speed. 12 hidden units are enough for the algorithm to learn at good speed, and the capabilities of the training do not increase beyond 24 hidden units. Those results are consistent with the bibliography, which establishes that the number of hidden units should be set to be between 1 and 3 times the number of observed units [4].

V. BEYOND REPETITION

When using a ML method we expect the algorithm to really be able to learn. The precise meaning of learning highly depends on the context but in general we expect the algorithm to be able to deal with situations to which it has not been exposed before. Up to this point we have analyzed the capability of our algorithm to repeat a set of values that we have exposed to it. However it is also highly interesting to monitor what the algorithm is able to learn beyond its capability of repeating values. In particular it is interesting which kind of patterns the algorithm creates when it does not produce bars or stripes.

For the following experiment (with nominal parameters) we have focused on the *almost valid* matrices, matrices that differ at most from one X or 0 from a valid matrix, and *simple superposition* matrices, that have exactly one bar and one stripe. For examples see FIG. 7.

It is worth mentioning that the space of *almost valid* matrices and the space of *simple superposition* matrices represent respectively the 25% and the 1.8% of the total space. Any variation of these numbers would not be naively expected.

Of the results in FIG. 8 the most relevant conclusion is that by performing iterations, the RBM does not only start to repeat cases to which it has been exposed, but also gains some insight about those cases. For *almost valid* matrices the probability of them being selected quickly raises and after only 1,000 iterations the RBM starts to predict them most often. This might be not surprising as *almost valid* matrices only differ on one unit from a valid matrix, so they are expected to have a similar, low energy.

More strikingly, matrices of the form *simple superposition* differ from a valid matrix from 2 units, as most of the 3×3 matrices, but yet their resemblance with the train set cause this matrices to progressively gain importance much beyond their initial 1.8%.

X0X	XXX	0X0	00X
X00	X0X	XXX	00X
X00	000	0X0	XXX
(a) Almost valid	(b) *Almost valid*	(c) *Simple superposition*	(d) *Simple superposition*

FIG. 7: Examples of *almost valid* matrices and *simple superposition* matrices

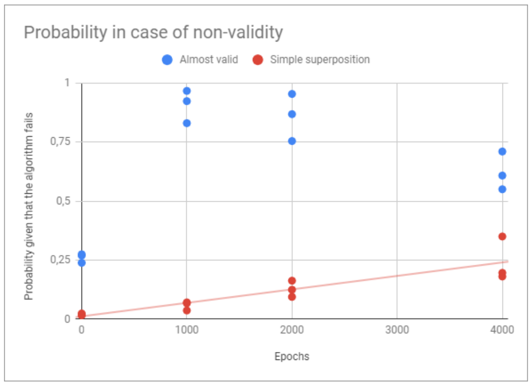

FIG. 8: Analysis of non valid results

VI. CONCLUSIONS

RBM was successfully applied to resolve Bars & Stripes for 3x3 and even 5x5 matrices. CD-1 approximation was necessary to solve the problem in a reasonable amount om time, but did not offer a robust increase of the log-likely-hood. Trained RBM learned to reproduce the train set beyond repetition.

[1] Sonka, Milan, Vaclav Hlavac, and Roger Boyle. *Image processing, and machine vision* Cengage Learning, (2014)

[2] Bengio, Yoshua. *Learning deep architectures for AI* Foundations and trends in Machine Learning 2.1 (2009)

[3] Rowlinson, J. S. *The Maxwell–Boltzmann distribution.* Molecular Physics 103.21-23 (2005)

[4] Heaton, Jeff. *Introduction to neural networks with Java.* Heaton Research, Inc., (2008).

[5] Geman, S.; Geman, D. *Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images.* . IEEE Transactions on Pattern Analysis and Machine Intelligence. (1984)