Stochastic Models for Sparse Codes

Maria Osório
Instituto Superior Técnico
Lisboa, Portugal
maria.osorio@tecnico.ulisboa.pt

***Abstract*—There is a consensus about information in the brain being represented by using Sparse Distributed Representations. These representations, however, are high-dimensional and consequently they affect classification performance due to the notoriously complex problem known as "the curse of dimensionality". In tasks for which there is a vast amount of labeled data, Deep Learning seems to solve this issue with many layers and a non-biologically plausible Backpropagation algorithm. The purpose of this research is to find a way to learn from high-dimensional sparse data side stepping these limitations and adopting a more biologically plausible approach. Actually, hidden units in Stochastic Models, represent hidden correlations between present dimensions of sparse vectors. These models can map a high-dimensional sparse vector into a hidden layer with few hidden units, while capturing the relevant features. Motivated by this reasons, we implement several classifiers inspired by Stochastic Models. In order to test them on a high-dimensional sparse data, we start by using the sparse codes generation mechanism structured in [1]. The implemented Stochastic Models are tested on these codes and their performance is compared with a simple Logistic Regression. Both the Stochastic Models and the Logistic Regression achieve good results. However, these good results archived by the Logistic Regression classifier led us to believe that the generated codes lie on a low-dimensional manifold embedded in a higher-dimensional space, which suggests that the real dimensionality of the data is highly inferior to the number of features. Afterwards, we propose a different way to generate sparse data, where each class follows a multivariate normal distribution and the sparseness is controlled by randomly deleting values in each sample. The experiments using these codes confirm our initial intuition as the Restricted Boltzmann Machine shows a good generalization performance, while the Logistic Regression overfits the training data.**

***Index Terms*—Stochastic Models, Restricted Boltzmann Machines, Sparse Distributed Representations, Learning**

I. Introduction

Traditional computer data structures cannot represent efficiently all concepts, the relationships between them, and the exceptions that each concept definition may hold.

The human brain does not have this problem: in order to represent information, it shares neurons between concepts, which means that a single neuron can be part of the representation of many different concepts. Furthermore, empirical evidence demonstrates that every region of the neocortex represents information by using sparse activity patterns [2]. When looking at any population of neurons in the neocortex their activity will be sparse, whenever a low percentage of neurons are highly active and the remaining neurons are inactive.

Sparse Distributed Representations (SDRs) is the method used to implement computationally the way information is represented in the brain [3]. An SDR is a binary vector composed of a large number of bits where each bit represents a neuron in the neocortex.

Consider that one wants to recognize a particular activity pattern in a neuron. Then, one says that a neuron forms synapses to the active cells in that pattern of activity [4]. This way, a neuron only needs to form a small number of synapses, to accurately recognize a sparse pattern in many cells. The formation of new synapses is the pillar of all memory in the brain [5] [6].

Memory in the brain is called Associative Memory, in which different SDRs (input patterns) become associated with one another depending on the similarity between them [7]. This means, SDRs are recalled through "association" with other SDRs. In Associative Memories, there is no centralized memory and no random access [8]. Every neuron present in the brain is an integral part of each SDR, which means each neuron participates in forming both SDRs and in learning the associations between them [5] [9].

A. Problem

As discussed previously, SDRs are binary vectors composed of many bits [10], which means we are dealing with a high dimensional input. These sparse representations are known to work well with associative memories but when we try to classify them, we must deal with some problems.

The main problem that brought us here is known as "the curse of dimensionality" caused by the high-dimensionality of SDRs. Classic Machine Learning models, for example, Feed-Forward Networks, are not good at dealing with high-dimensional sparse inputs given the vast number of parameters. [11], [12].

Many machine learning problems become exceedingly difficult when the number of dimensions in the data is high. As discussed in [13] the number of possible distinct configurations of a set of variables increases exponentially with the number of variables.

There is no universal answer for how data sparsity would affect learning convergence behaviour in Machine Learning models [14]. Though, [12] discusses that most features in high-dimensional vectors are usually non-informative or noisy and may decrease the model's generalization performance.

Identify applicable funding agency here. If none, delete this.

In a Neural Network, the input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes. Therefore, considering an SDR as an input (high dimensionality vector), the number of weights from the input layer to the first hidden layer will be the number of entries of the binary input vector multiplied by the number of hidden units in the first hidden layer [13]. This way, the model will have a really large number of parameters making it prone to overfitting.

The phenomenon of overfitting occurs when a network fits the training data more than it should. When overfit happens, it captures noise from the training data. This leads to a model with weak generalization capability, increasing the error when classifying new instances from the test set [15].

II. Motivation

As argued in [11] and discussed in the former section, a well-known problem in Machine Learning is sparse data, which alters the performance of Machine Learning algorithms and their ability to calculate accurate predictions. A high-dimensional sparse input leads to the well-known problem denominated "the curse of dimensionality".

The purpose of this thesis is to show that it is possible to learn good and general classifiers from high-dimensional sparse representations generated by biologically plausible models while, unlike the deep learning approach, staying under the biological constraints.

With this idea in mind, we chose to investigate Stochastic Models, i.e., Restricted Boltzmann Machine (RBM) and Deep Belief Network (DBN), to classify high-dimensional sparse data motivated on the following arguments:

- As proved in [1], Willshaw's model of associative memory has been showed to work well with high-dimensional sparse codes. Therefore, given the inspiration of Stochastic Models on Associative Memories, they seem to be a great candidate to perform well with those codes.
- Boltzmann Machines have local learning rules (Hebbian rule), which are biologically plausible [16].
- Hidden units in Stochastic Models represent hidden correlations between present dimensions of sparse vectors. As these models only learn the correlation between active units, then stochastic models can map a high-dimensional sparse vector into a hidden layer with few hidden units, while capturing the relevant features.

The later argument represents the core motivation of using Stochastic Models to classify high-dimensional sparse data. These models learn the correlations between active neurons, which means the hidden units will exclusively change their state based on the input units that are different from zero. This allow Stochastic Models to have a compact hidden layer that captures the information present on the high-dimensional sparse data without falling into overfitting.

To deeply ground this motivation a trivial experiment was performed in which, a RBM was used to classify a sample of the original binarized MNIST dataset and a flipped version of that same sample. In Figure 1, the ten top images represent the binarized version of the original MNIST, in which the bits representing each digit are set to 1 and the background information to 0. The ten bottom images represent a flipped sample of the original binarized MNIST, where the bits representing each digit are set to 0 and the background information to 1.

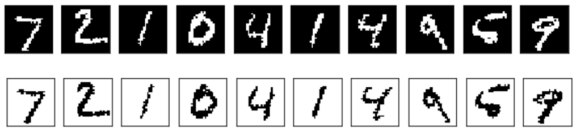

Fig. 1. Ten image sample of MNIST test set. The ten top images represents a binarized version of an original MNIST sample, while the bottom images represent the same sample flipping the bits.

We considered a RBM with the same architecture to classify both versions of the binarized MNIST, the original and the flipped one. If our intuition points in the right direction, the model should be able to accurately classify the original version in which the digits information are represented by 1s and fail on the flipped version in which the digits are represented by 0s.

With a sample of 5000 training examples and 1000 test examples of both datasets, in the experiment with the original dataset an accuracy of 91.6% was achieved in the train and 86.2% in the test set. In the flipped version of the data the train and test accuracies were, respectively, 15.5% and 13.8%.

By analysing the results achieved by a RBM with 500 hidden units, one can conclude that in the original version of the binarized MNIST, the model learns the correlations between active neurons, which represent the digits. As the active bits represent a relatively small percentage of each sample, the model is able to capture the correlations between these active features and have a good generalization performance.

In the flipped version of the binarized MNIST, the RBM fails completely. This is justified by the fact that this model exclusively learns correlations between present dimensions and not between 0s. With a hidden layer of 500 units, the RBM is unable to catch the correlations between all the active neurons that represent the background, and consequently fails when classifying the flipped version of the MNIST.

This small experiment validates the strong potential of Stochastic Models to deal with high-dimensional sparse inputs, as they can capture the correlation between present dimensions of the input data.

By performing the same experiment with the Logistic Regression (LR) classifier, we achieve similar results when classifying the original and the flipped version of the MNIST sample.

The similar accuracy results achieved by LR in both problems suggests that this model learns the information given by 1s in the same way it does with 0s. This implies that, when LR is dealing with high-dimensional sparse inputs, it learns all the dimensions of the sparse vector. Consequently, this model is prone to fall into overfitting.

On the other hand, the RBM classifier learns exclusively the correlations between active neurons. Given that high-dimensional sparse inputs have a low percentage of 1s, this model can map the high-dimensional sparse vector into a hidden layer with few hidden units, while capturing the relevant features.

But does this really indicates that the RBM avoids the overfitting problem when classifying high-dimensional sparse data? The purpose of this research work is to answer this question by investigating the potentiality of the Stochastic Models to deal with high-dimensional sparse data.

III. Stochastic Models

Stochastic Models show a great potential to solve the presented problem, before diving into the experiments we need a theoretical overview on them.

A. Restricted Boltzmann Machine

Restricted Boltzmann Machines were initially invented under the name Harmonium [17]. They are a variant of Boltzmann machines, with the restriction that there is a single layer of m visible units $\mathbf{v} = (v_1, v_2, ..., v_m)$ and a single layer of n hidden units $\mathbf{h} = (h_1, h_2, ..., h_n)$ with no visible-visible or hidden-hidden connections.

The energy function of a restricted Boltzmann machine can be written as

$$H(v,h) = -\sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij}h_i v_j - \sum_{j=1}^{m} b_j v_j - \sum_{i=1}^{n} c_i h_i. \quad (1)$$

For all $i \in 1, ..., n$ and $j \in 1, ..., m$, w_{ij} is a real valued weight associated with the edge between the units v_j and h_i, and b_j and c_i are real valued bias terms associated with unit j of the visible layer and unit i of the hidden layer, respectively.

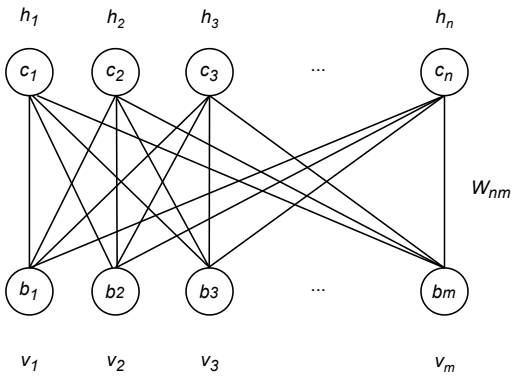

Fig. 2. Network graph of an RBM with n hidden units and m visible units

The graph of an RBM has no connections between two variables of the same layer, as we can observe in Figure 2. In terms of probability, this means that the visible variables are independent given the state of the hidden variables and vice versa:

$$p(h|v) = \prod_{i=1}^{n} p(h_i|v), \quad (2)$$

and

$$p(v|h) = \prod_{j=1}^{m} p(v_j|h). \quad (3)$$

The conditional independence between the variables in the same layer makes Gibbs sampling an easy task. Instead of sampling new values for all variables subsequently, the states of all variables in each layer can be sampled jointly. Thus, Gibbs sampling can be performed sampling a new state h for the hidden neurons based on $p(h|v)$ and sampling a state v for the visible layer based on $p(v|h)$. This process is also referred to as block Gibbs sampling [18] and it represents the main advantage in using Restricted Boltzmann Machines instead of Boltzmann Machines as the Negative phase of the learning process (unlearning phase) becomes considerably simplified [19].

The weights update of the RBM is computed using a type of learning rule similar to the one used in Boltzmann machines. In particular, it is possible to create an efficient algorithm based on mini-batches. The weights w_{ij} are initialized to small values, and for the current set of weights w_{ij}, they are updated as follows:

1) Positive phase: Visible units are clamped and the hidden units are randomly chosen (0 or 1). The algorithm uses a mini-batch of training instances, and computes the probability of the state of each hidden unit in exactly one step. Then a single sample of the state of each hidden unit is generated from this probability. This process is repeated for each element in a mini-batch of training instances. The correlation between these different training instances of v_i and generated instances of h_j is computed; it is denoted by $\langle v_i h_j \rangle_{data}$. This correlation is essentially the average product between each such pair of visible and hidden units [20].
2) Negative phase: Visible and hidden units are chosen randomly (0 or 1). The algorithm starts with a mini-batch of training instances and then for each training instance, it goes through a phase of Gibbs sampling after starting with randomly initialized states. This is achieved by using Equations (2) and (3) to compute the probabilities of the visible and hidden units, and using these probabilities to draw samples. The values of v_i and h_j at thermal equilibrium are used to compute $\langle v_i h_j \rangle_{model}$ in the same way as the positive phase [20].

We can write our update rule as in Boltzmann Machines:

$$\Delta w_{ij} = \eta \cdot (\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}). \quad (4)$$

After training the model we clamp the visible units with some configuration s_{query}. The network will converge to an attractor (stored pattern) after performing several steps using the update rule [21].

1) Contrastive divergence: Obtaining unbiased estimates of the log-likelihood gradient using Markov Chain Monte Carlo (MCMC) methods typically requires many sampling steps. However, it has been shown that estimates obtained after running the chain for just a few steps can be sufficient for model training [18] [22].

Contrastive divergence (CD) speeds up the computing time of $\langle v_i h_j \rangle_{model}$ as it does not use Gibbs sampling to reach thermal equilibrium. In this algorithm, the training phase starts by clamping the visible units with v^0 and the hidden layer units h^0 can be computed by

$$p(h_i = 1|v) = \sigma\left(\sum_{j=1}^{n} w_{ij} \cdot v_j + c_i\right), \tag{5}$$

that define

$$\langle v_i h_j \rangle_{data}^0 . \tag{6}$$

As we saw previously there are no visible-visible or hidden-hidden connections. For that reason each unit h_i is independent of the other hidden units. Therefore, h^0 can be computed in parallel as each hidden unit only depends on the visible units connected to it [23].

The second step consists in updating all the visible units in parallel to get a "reconstruction" v^1, which can be computed by

$$p(v_i = 1|h) = \sigma\left(\sum_{j=1}^{n} w_{ij} \cdot h_j + b_i\right), \tag{7}$$

that define

$$\langle v_i h_j \rangle_{recon}^1 . \tag{8}$$

The visible units are now clamped with v^1 and the hidden layer units h^1 are computed in parallel using Equation (5).

The reconstruction algorithm can be computed τ times or until convergence is reached. Sometimes CD may take many iterations $(1 \ll \tau)$ to converge. When $\tau = 1$ we are computing a single-step reconstruction [24] [7].

The weights update computed for τ steps of the reconstruction algorithm is given by

$$\Delta w_{ij} = \eta \cdot (\langle v_i h_j \rangle_{data}^0 - \langle v_i h_j \rangle_{recon}^\tau). \tag{9}$$

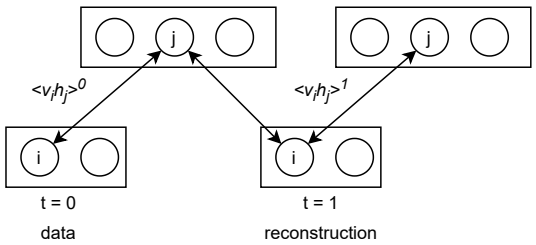

Fig. 3. Contrastive divergence with single-step reconstruction

2) Persistent Contrastive divergence: A different strategy that resolves many of the problems with CD is to initialize the Markov chains at each gradient step with their states from the previous gradient step. This approach was first discovered under the name stochastic maximum likelihood (SML) in the applied mathematics and statistics community and later independently rediscovered under the name persistent contrastive divergence (PCD) [25].

The idea behind this approach is that, as long as the steps taken by the stochastic gradient algorithm are small, the model from the previous step will be similar to the current model. It follows that the samples from the previous model's distribution will be very close to being fair samples from the current model's distribution.

As each Markov chain is continually updated throughout the learning process, rather than restarted at each gradient step, the chains are free to wander far enough to find all the model's minima. PCD is thus considerably more resistant to forming models with spurious minima than the original CD algorithm is [26].

B. RBM for classification

Until now we have described RBM as a generative model, where given a corrupted pattern we can reconstruct the original pattern. Actually, this section describes how RBM model can also be used as a classifier.

First we have the training phase, where the RBM learns to model the joint probability distribution of input data (explanatory variables) and the corresponding labels (output variables), both represented by the visible units of the model as shown in Figure 4. The RBM is trained with one of the previously described algorithm: either CD (described in III-A1) or PCD (described in III-A2).

Following the training phase, we have the sampling where the label corresponding to an input example can be obtained by fixing the visible variables that correspond to the data and then sampling the remaining visible variables allocated to the labels from the joined probability distribution of data and labels modeled by the RBM. Hence, a new input example can be clamped to the corresponding visible neurons and the label can be predicted by sampling [18] [27].

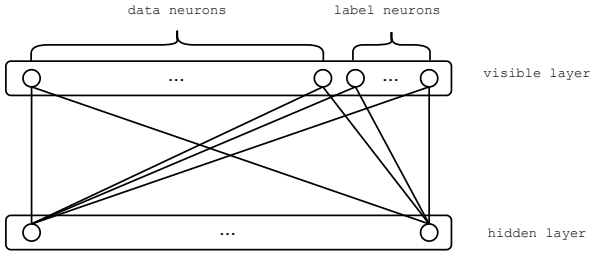

Fig. 4. Restricted Boltzmann Machine (RBM) that models the joint probability distribution of input images and the corresponding labels.

C. Deep Belief Nets

A Deep Belief Network (DBN) is a powerful generative model that uses a deep architecture of multiple stacks of RBM .

DBNs were one of the first non-convolutional models to successfully admit training of deep architectures [28], [29]. Deep belief networks are generative models composed of several layers of latent variables (hidden units) with connections between adjacent layers but not between units within each layer [30], [13]. Deep belief networks capture the higher-level representations of input features [24].

There is an efficient, layer-by-layer procedure for learning the top-down, generative weights that determine how the variables in one layer depend on the variables in the layer above, and is composed by the following steps:

1) An RBM, as discussed in the section above, is trained directly on the input data, therefore the stochastic units in the hidden layer of the RBM are able to capture the important features that characterize the input data.
2) The activations of the trained features are then treated as input data which are used to train a second RBM. In effect, we can learn the features of features in a second hidden layer.
3) The process of learning the features of features is continued until the number n of hidden layers is reached, which is the same of saying until n RBMs have been trained.

The main idea is to apply DBNs in order to transform a Sparse Distributed Representation into a compact representation given the dimensionality reduction from layer to layer. The compact representation generated by this model will then be used for classification.

IV. Experiments

This section starts describing the first used datasets and briefly introduce the generation process of the sparse data.

Next, the first set of performed experiments are described in order to understand the potential of the previously described Stochastic Models. By baring in mind that a reliable model is crucial, some baseline experiments were carried out and their complexity was incrementally increased until reaching a set of suitable models, inspired in Stochastic Models and capable of performing classification.

To understand whether or not stochastic models have a better generalization performance than classic Machine Learning models when dealing with high-dimensional sparse data, a comparison was made between the implemented models and a LR. The good test accuracy achieved by the LR when classifying the generated high-dimensional sparse data, led to the suspicion that these generated datasets were "living" in a lower dimensional space.

Subsequently, a different way to generate high-dimensional sparse data was proposed, where each class follows a multivariate normal distribution and the sparseness of the data is controlled by deleting the values of random features in each sample. By investigating the RBM and the LR performance with this sparse data, we were able to conclude that the RBM shows a good generalization performance, while LR falls into overfitting.

A. Datasets description

Before diving into the experiments it is important to describe the datasets used in the research process. The first dataset we briefly describe is the MNIST, which was the starting point for our experimental analysis. Then, the encoding process that provided the MNIST sparse representations was described and used for the core experiments.

1) MNIST: The MNIST dataset[1], created by Yann Le Cun, contains 60,000 digits in the range 0 to 9 for training image recognition models, and another 10,000 digits as test data. Each digit is normalized and centered in a gray-level image with size 28×28, or with 784 pixel in total.

2) Sparse MNIST generation: The strategy used to generate the sparse codes is structured in [1]. In this paper, Sa-Couto & Wichert propose an encoding function that maps visual patterns into informative binary sparse vectors. This encoding requires the following two steps:

1) The Retinotopic Step: This first layer performs a local feature extraction which is organized in K planes of $I \times J$ feature extraction units. Each image is parsed with K sliding windows, with size $f \times f$, to extract the K most relevant visual features of the images. The occurrence of the extracted features is then signalled at the middle layer. These features are determined beforehand using the unsupervised *K-means* algorithm. This layer performs information compression by establishing a many-to-one relationship between groups of pixels and receptive units. Thereby, this step transforms the dense representation presented in the input layer into a sparse representation.
2) The Object-Dependent Step: The fixed coordinate system where the features occurrences are signalled is turned into an object-dependent, radius one polar coordinate system. In this step, the mapping of the previously extracted features for each plane K to the new coordinate system is performed by a $Q \times Q$ units plane. This operation provides invariance to size and position, and makes the resulting representations well-distributed since the features detected in the previous step will be mapped quasi-uniformly into all the dimensions of the object space.

The encoding function proposed in [1] requires as input the following parameters: K specifies the number of features we want to be extracted, $f \times f$ specifies the window size that will be parsed trough the image to extract the K most relevant features, Q represents the size of the new coordinate system plane and, finally, T_what that controls the percentage of similarity needed to recognize a feature in an image.

[1] http://yann.lecun.com/exdb/mnist/

Thus, in order to generate the sparse representations of the MNIST dataset we provide the encoder with the train and test of the MNIST handwritten digits as well as the defined parameters, and the encoder function returns a train and a test sets with 3 dimensions. These returned sets represent a sparse binary MNIST encoding with the following shape: *(N, QxQ, K)*, where *N* is the number of samples we encoded, *Q* and *K* are the parameters described above.

B. Classification using Stochastic Models

In this section, experiments with classifiers inspired in stochastic models were performed, given that the main goal of this thesis is to explore the potentiality of stochastic models to deal with high dimensional sparse inputs.

1) Restricted Boltzmann Machine: In order to use the RBM model to perform classification, we followed the architecture described in section III-B. In this set of experiments we started by training the model on labeled data, MNIST images combined with ten binary indicator variables, one of which is set to 1 indicating that the image shows a particular digit while the others are set to 0, this is also known as one hot encoding representation. In order to take the most of our model capacities we are required to perform parameter tuning during training, i.e. adjusting the learning rate, momentum and weight-decay during learning. Choosing the best parameters combination is not an easy task, so guided by [31], a practical guide to train Restricted Boltzmann Machines, and with some experimental analysis we managed to reduce significantly the reconstruction error.

In the prediction phase two different approaches were implemented and tested. In the first approach, an image was given to the model and the label corresponding to that input image could be obtained by fixing the image neurons and performing *N* steps of Gibbs Sampling until a reconstruction of the ten visible units corresponding to the class is obtained. The alternative approach consists of calculating the probability of activation for the ten visible units corresponding to the class. To calculate this probability, one needs to multiply the hidden units' probabilities by the weights entries that correspond to the label units and sum the bias of the visible units corresponding to the label. We apply the softmax function to this vector and the label corresponding to that input image is the index of the maximum probability value.

If the aim is to show that it is possible to produce a good and general classifier from Stochastic Models, particularly RBM and DBN, first one needs to understand how these models deal with high-dimensional sparse data. Thus, before exploring deeper models we performed an experiment where we compared the performance of a LR and a RBM. The main purpose of this set of experiments was to analyse the behavior of these two models given sparse MNIST datasets generated with increasing sparseness.

To generate the sparse codes for these experiments, we used the strategy described in section IV-A2. The encoder parameters were defined as *K = 7*, which means we consider the seven most significant features, each defined as a *5x5* window. Parameter *Q* was defined as *Q = 12*, which means that for each *K* there is a new coordinate system plane with size *Q x Q*. Additionally, given the high computational time of these experiments, we started by using a sample of the sparse datasets, 5000 training samples and 1000 test samples. By considering the above information, one has a training set with shape *(5000, 144, 7)* and a test set with shape *(1000, 144, 7)*. The generated data was reshaped to a two-dimensional array before serving as input to the RBM model, so the final dimensionality of the data was given by *Q x Q x K*, which in this case was 1018. Thereby, the dimensionality of the data indicates the number of visible units of the model.

As discussed above, in section IV-A2, parameter T_what defines the level of similarity that is needed for a feature in the image to be considered, which means that a higher T_what requires a higher similarity for the feature to be recognized and, consequently, the sparseness of the generated codes increases. Therefore, for these set of experiments we have fixed the remaining parameters and increased T_what, which indirectly means that the sparseness of the data was increased.

In Figure 5 the T_what parameter was set to each x-axis value, so that the behaviour of the RBM and LR can be compared when increasing the sparsity of the data, i.e. increasing the level of similarity needed to consider a feature in the image.

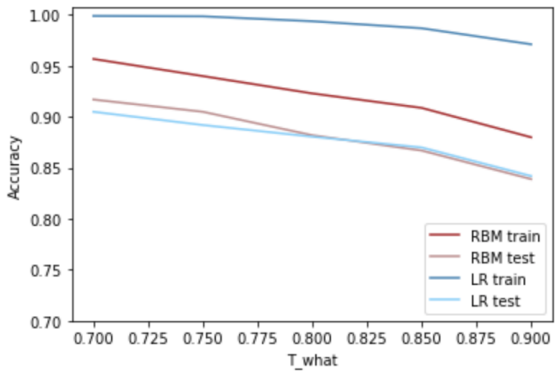

Fig. 5. Train and test accuracy of LR and RBM given sparse MNIST datasets with increasing sparseness

By analysing the plot in Figure 5 one can observe that the RBM shows a smaller gap between train and test accuracies, so it seems to be generalizing the training set better than the LR.

Moreover, as mentioned above, the experiments plotted in Figure 5 consider just a sample of the generated sparse datasets. Thus, to understand the behavior of the RBM model and its potentiality to learn a good and general classifier, the dataset with $T\_what = 0.85$ was chosen and the same experiment with all the dataset performed. In Table I, in the columns referring to Sparse dataset 1, one has the best accuracy of the RBM and LR, both with this dataset.

In fact, the sparse codes we generated for the experiments plotted in Figure 5 had all the same dimensionality. To

conclude the experiments that compare these two classifiers, a much higher dimensional and sparser dataset was generated and the same experiments repeated. The encoder parameters were defined as $K = 30$, which means that the thirty most significant features were considered and $Q = 18$, which means that for each K there is a new coordinate system plane with size $Q \times Q$. By considering these parameters, the dimensionality of the generated sparse dataset is 9720 and the best accuracy results are in Table I, in the columns referring to Sparse dataset 2.

Once again the best results of the RBM model were with 500 hidden units for both datasets. Concerning the remaining parameters, the learning rate was initialized to 0.01 and decreased during training until reaching 0.001, the momentum used was 0.5 to start the learning process and then increased to 0.9. In these experiments we did not need to use any regularization (Weight-Decay).

By analysing the results in Table I, one can conclude that LR has a slightly higher accuracy than the RBM. By observing the difference between train and test accuracies in Sparse dataset 2, the RBM seems to be better generalizing the rules learned during training than LR. Guided by this conclusion, the Sparse dataset 2 was selected to continue our research described in the following sections.

In what follows, we will introduce the DBN architecture used for classification with the intent to explore whether this model is able to overtake the accuracy reached in Sparse dataset 2 by the models described above.

2) Deep Belief Network: To implement the DBN, the architecture presented in Figure 6 inspired by the paper "Learning multiple layers of representation" [28] was used. Instead of having one RBM, this model consists of two stacked RBMs, which is called a DBN. The first RBM will be trained just on the image neurons which are the high dimensional part of our dataset. Then, the activation of the trained features in the first RBM combined with ten binary indicator variables which represent the class, are treated as input data to train the second RBM. In effect, the features of features can be learned in the second hidden layer.

Afterwards, like in the RBM model, one has the prediction phase where the label corresponding to an input image is obtained by the index of the maximum probability value.

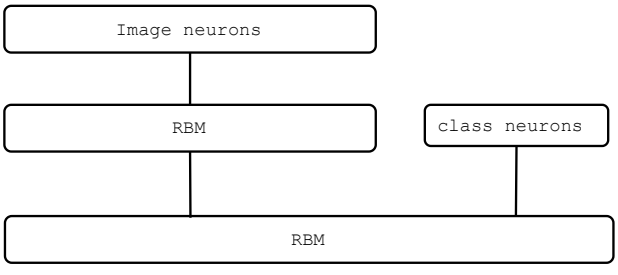

Fig. 6. DBN with 2 layers that models the joint probability distribution of hidden activations given the input images and the corresponding labels.

The number of hidden units in each layer corresponds to the features of the input images stored in the model. So, it is crucial to find the adequate number of hidden units as well as hidden layers, since models with too few or too many hidden units can result in slow learning and poor performance.

The DBN layers were trained with an initial learning rate of 0.01 and an initial momentum of 0.5, which were decreased and increased, respectively, during the learning process.

By considering the training parameterization described above to train the Sparse dataset 2, a comparison between a 2 layered DBN and a 3 layered DBN was made. Both models start with a first layer of 500 hidden units, then the DBN with 3 layers has another 500 hidden units layer preceding the last layer. Increasing number of hidden units were tested in the last layer of both models, considering this is the layer that receives the label neurons as input in both DBNs.

The maximum accuracy with this architecture was achieved by the DBN with 2 layers, with 2000 hidden units in the second layer. In fact, this model surpasses the test accuracy achieved by the LR with a train and test accuracy of 98.09% and 97.94%, respectively. Besides having a better accuracy, we continue to have close train and test accuracies which indicates that it has a high generalization power, which means that the rules learned during training are equally valid to the test set.

3) Restricted Boltzmann Machine followed by Logistic Regression: In a second stage of experiments, we tried a different approach. Instead of using a RBM or a DBN to model the joint probability distribution of input images and the corresponding labels, we used them to model the input images into activation of the hidden units. In this new approach the classification is not performed by the Stochastic Models. They have the role of providing the classifier, which in this case is a LR, with a compact representation of the input.

The main idea behind the approach is to give the images as input to the RBM. Then, after the model is trained, it computes the activations of the hidden units, which give a compact representation of the input. That compact generated representation is then used as input to the LR. In this way, instead of giving a high-dimensional sparse vector to the LR, a compact representation generated by the hidden units of the RBM is provided.

Regarding the experiments performed with the sparse MNIST dataset, the difference between the train and test accuracies slightly decreases compared with the LR applied directly on the sparse dataset. The accuracy with a 5000 hidden unit RBM followed by a LR is 100% on the training set and 97.98% on the test set.

In fact, one can conclude that the LR classifier does not produce significantly better results when receiving a hidden representation of the RBM instead of the sparse codes. In addition, the large number of hidden units extremely increases the time spent in the learning process and it does not reduce significantly the difference between train and test accuracies.

The stochastic models that were used before as classifiers, now have the role of reducing the input's dimensionality. Consequently, the LR, instead of receiving a high-dimensional sparse input, it receives a compact representation given by the

TABLE I
TRAIN AND TEST ACCURACY OF RBM AND LR GIVEN TWO DIFFERENT GENERATED SPARSE DATASETS

	Sparse dataset 1		Sparse dataset 2	
	Train accuracy	Test accuracy	Train accuracy	Test accuracy
Restricted Boltzmann Machine	92.35%	92.02%	97.35%	96.98%
Logistic Regression	93.62%	92.77%	100%	97.64%

Stochastic Model's hidden units. In this section, we concluded that the RBM does not provide the gradual dimensionality reduction we needed to decrease the difference between train and test accuracies. Thus, in the following section, the use of the DBN model to obtain a gradual dimensionality reduction was explored.

4) Deep Belief Network followed by Logistic Regression: Instead of having just one hidden layer to get the compact hidden representation, one can have a gradual dimensionality reduction. This is possible by stacking more than one RBM, which is also known as a DBN.

Various experiments were carried out with this architecture to understand if the dimensionality reduction from layer to layer could increases even more the accuracy when classifying the sparse codes. In fact, after analysing all the results obtained, the best performance was achieved with a DBN with 3 layers, in which the first layer has 2000 hidden units, the second 1000 and the last 500. The train and test accuracies with this DBN architecture were 98.32% and 97.36%, respectively.

Despite the dimensionality reduction given by the decreasing number of hidden units from one layer to the next, this model was not the one that resulted in a better accuracy.

In fact, it is noteworthy that the difference between train and test accuracies was reduced. By providing a compact representation as input to the LR, the generalization capability of the LR increased. However, the test accuracy slightly decreased which means that the hidden compact representation of the input does not perfectly represent the high-dimensional sparse input.

C. Models comparison

Throughout the former sections, the implementation of several Stochastic Models was described as well as the results derived from the study of their behaviour when dealing with the high-dimensional sparse dataset generated from MNIST. To make the final remarks about the performance of these models given the proposed dataset, in Table II the best performance results of each studied model are presented.

All the models presented in the table show to perform well given a high-dimensional sparse dataset generated from MNIST. The model which showed a better test performance was the RBM followed by a LR, however, the DBN with 2 hidden layers achieved almost the same accuracy.

One of the advantages of using the RBM and DBN is the fact that besides classifiers, they are generative models. Consequently, additionally to predicting the labels of the dataset, they can perform image reconstruction, whereas, in the other models the classification is done by a LR, which is exclusively a classifier. Besides, these classifiers show a smaller difference between train and test accuracies, which means that they have a greater generalization performance when classifying the high-dimensional sparse dataset generated from the original MNIST.

Actually, the LR scores 100% on the training set, which suggests that this model is highly adapted to the training data and consequently more prone to overfitting. However, the good results archived by the LR classifier in the test set leave us wondering whether or not these generated codes are high-dimensional and sparse. It is plausible to think that the generated sparse data lies on a low-dimensional manifold embedded in a higher-dimensional space. As a result, the learning problem becomes too easy and the LR can accurately classify the test set. To have a well-grounded research one cannot be restricted to the generated MNIST sparse codes. In the next section, a deeper research with a different way to generate sparse data is presented.

D. Learn from a Sparse Normal Distributed Dataset

Revisiting the problem raised in the former section, it is plausible to consider that the generated sparse MNIST data lies on a low-dimensional manifold embedded in a higher-dimensional space, which means that, although the data has many features, it only has a few degrees of freedom.

With this idea in mind, and with the desire to understand if the RBM model accurately classifies high-dimensional sparse data while LR falls into overfitting, several experiments were carried out in which both a LR and a RBM had to perform the same classification task.

In fact, generating binary sparse data is not a trivial task as it is hard to find a complexity balance in the learning problem. Thus, instead of generating binary data, we decided to generate a dataset where each class follows a multivariate normal distribution.

The implementation of the RBM we used in the experiments is modelled with Bernoulli visible and hidden units, which means that this RBM is prepared to receive input data in the range [0,1]. By considering that the generated data is real-valued, some exploratory experiments with a Gaussian-Bernoulli RBM were performed, but tuning the value of the Standard Deviation parameter is a hard task, which can produce an unstable learning process [31].

Since the expected results using Gaussian visible units were not achieved, the Bernoulli-Bernoulli RBM was explored to address this problem. In fact, the only difference between using Bernoulli or Gaussian visible units occurs when sampling the data visible units of the negative phase of the learning

TABLE II
TRAIN AND TEST ACCURACY OF ALL THE MODELS IMPLEMENTED CONSIDERING THE SAME SPARSE DATASET

	Train accuracy	Test accuracy
Logistic Regression	100%	97.64%
Restricted Boltzmann Machine (RBM)	97.35%	96.98%
Deep Belief Network (DBN)	98.09%	97.94%
RBM followed by Logistic Regression	100%	97.98%
DBN followed by Logistic Regression	98.32%	97.36%

algorithm. Constraining the values of the visible units to be between 0 and 1 imparts a kind of regularization to the learning process. In the sampling phase, the use of Bernoulli visible units is necessary as the sampled label is binary.

In the next steps, the dataset generation pipeline and the experiments performed are explained, with the aim of reaching meaningful conclusions on the behaviour of LR with sparse data, compared with the Stochastic Models; in this case, only the RBM model was addressed.

1) Dataset generation: To reach the desired results, we start by explaining the dataset generation. Each dataset is generated with two classes, in which each one follows a Gaussian distribution. The first half of the samples belong to class 0 and follow a Gaussian distribution with mean centered in the origin, the other half corresponds to class 1 and follows another Gaussian distribution with mean centered in five. The covariance matrix for each class is defined as a diagonal matrix, in which the diagonal values are set to the norm of the difference between the mean vector of each class multiplied by a small number as to reach a good balance in the problem complexity.

Moreover, the number of features will be further defined for each experiment. In the case the dataset to be is intended to be dense, the number of features is set to a low value, whereas in the case the dataset is intended to be high-dimensional, the number of features is fixed to a high value.

2) Pipeline: With the objective of structuring the steps performed by the experiments, a simple pipeline is described. Thus, for each experiment, we started by setting the parameters and then running 10 times the following pipeline:

1) Populating the dataset by sampling from the two multivariate normal distribution with the previously defined parameters and associate each multivariate normal distribution to a class, either 0 or 1.
2) Centring the data, which consists in subtracting the mean of each feature to every value of that feature.
3) Transforming the dataset into sparse data, which means choosing a few random features to keep in each sample and set the remaining features to zero.
4) Dividing the samples of the dataset into train and test. For this step we use the function $train\_test\_split()$ from the sklearn library was used, in which the input parameter $test\_size$ was set to 20%.
5) Training a LR model with the generated train set.
6) Evaluating a LR model by computing and storing the train and test accuracies.
7) Training the RBM with the generated train set using PCD algorithm.
8) Evaluating the RBM by performing the Gibbs sampling to get the reconstruction of the class unit for both train and test sets (explained with more detail in section III-B). After having all the reconstructions, the model's train and test accuracies can be computed and stored.

After running the described pipeline, four lists with 10 train and test accuracy values for both models were obtained. Subsequently, the mean and the standard deviation for each list was calculated. In the end, a single train accuracy for both models and a single test accuracy for both models were stored, as well as the respective standard deviations.

3) Experimental Analysis: This research was instigated by the desire to understand if Stochastic Models perform better than classic Machine Learning models, like LR, when classifying high-dimensional sparse data. For that reason, and with the aim of having a baseline experiment which guided the next steps, a comparison was made with the LR performance classifying a non-sparse dense dataset and a high-dimensional sparse dataset.

Starting by the dense dataset, it was generated as described in section IV-D1. For the dataset to be dense, the number of features was fixed to 500 and all the values of the data were kept. When generating the high-dimensional sparse dataset, the methodology described in the section IV-D1 was also used. In this case, the number of features was set to 5000. Furthermore, the sparsity was fixed to 95%, which means that for each sample 5% of the features were kept and the remaining values set to 0.

In Figure 7, the results show that LR performs well when the dataset is dense, with a mean train accuracy of 100% and a mean test accuracy of 99.25%. However, when analysing its performance on the high-dimensional sparse dataset a huge overfitting is observed, with a mean train accuracy of 100% and a mean test accuracy of 63.5%.

This first experiment provides a baseline to guide the next steps. In what follows, the intend was to show that the RBM performs accurately in a classification task with high-dimensional sparse data. Before diving into the experiments, the parameters must be defined. With these experiments, the aim was to access the behaviour of a LR and a RBM with increasing sparseness of the dataset. For this reason, the remaining parameters of both models were fixed to the same

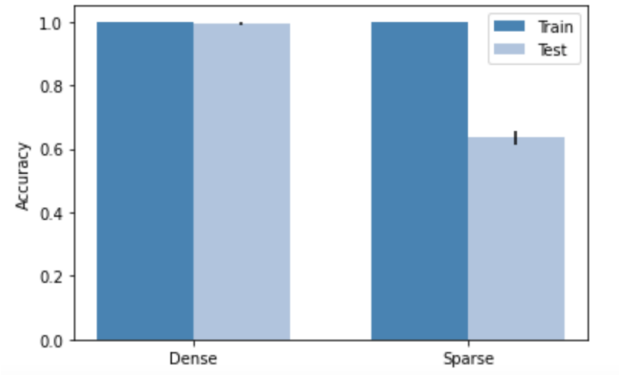

Fig. 7. Performance of the LR in a dense versus a high-dimensional sparse dataset.

values, as to achieve a trustful comparison between models.

The number of samples was fixed to 2000 and the dimensionality of the input to 5000. As far as the parameters of the RBM architecture were concerned, the number of hidden units was set to 500. Additionally, a batch size of 50 and a learning rate of 0.1 was used.

With the final objective of taking meaningful conclusions about both models when the dataset sparsity increases, i.e., the number of zero values increases, the pipeline described in section IV-D2 was followed. To make an easier comparison between models, the sparsity value was set to each x-axis value and the mean accuracies of LR and RBM were plotted in Figure 8.

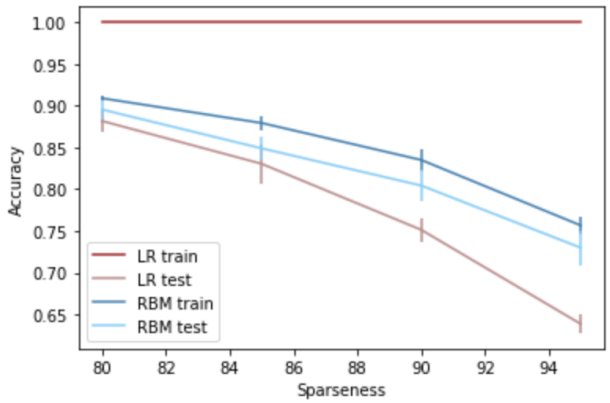

Fig. 8. Comparison between LR and RBM classifiers with increasing percentage of sparseness.

By analysing the results plotted in Figure 8, one can observe that the LR classifier has an accuracy of 100% on the training set, though it is not able to perform accurately on the test set, which suggests that this model is learning the noise in the training data. As the data gets sparser the learning problem becomes harder and test set accuracy decreases. This means LR can represent the training set of sparse data perfectly but unable to generalize, which result in a poor performance in the test set.

On the contrary, the RBM classifier can generalize the learning problem. Although, with the increasing sparseness the model's performance decreases, it never falls into overfitting as LR does. When the data is generated with 95% sparseness, the LR has a mean test accuracy of 63.5%, while the RBM shows a mean train and test accuracies of 75.67% and 73.15%, respectively. So, comparing the test set performance, the RBM is nearly 10% more accurate than LR.

The good results and generalization performance achieved by the RBM classifier can be mainly justified by the fact that it has a hidden layer that represents hidden correlations between active features of sparse vectors. Therefore, this model can map a high-dimensional sparse vector into a lower dimensional hidden layer, which would catch the relevant features present on the high-dimensional sparse vector.

Besides, the LR learns the conditional probability of the class given the features, while the RBM learns the joint probability of the features and class. The difference between learning the conditional or the joint probability may be a factor influencing the performance of each model. The LR is learning a simpler problem than the RBM. Thus, given its huge capacity, instead of learning just the training patterns, the model is also learning the noise. Consequently, the LR becomes too adapted to the training set, which leads to the overfitting problem.

This justification seems to be well grounded, although one may still wonder: Can the good performance of the RBM be justified by the presence of hidden units, which makes it a non-linear classifier?

To answer this question, a comparison was made between the performances of the RBM and a MLP. To derive meaningful conclusions, a RBM and a MLP with the same number of hidden units were defined. Additionally, the MLP activation function for the hidden layer units used was the logistic sigmoid function.

More experiments were carried out, in which normally distributed datasets were created following the pipeline described in section IV-D2. However, instead of comparing the RBM to a LR, the comparison was made with a MLP. In this experiment, the number of samples was fixed to 2000, the dimensionality of the input to 5000 and the number of hidden units of the RBM and the MLP is fixed to 500. With these parameters and a sparsity of 95%, the results in Table III were obtained.

By analysing the results, the MLP classifier has an accuracy of 100% on the training set, although it is not able to perform accurately on the test set. This means that the model is overfitting the training data, and so, it lacks generalization capability.

As a matter of fact, the overfitting undergone by the MLP model led to the conclusion that, the generalization capability of the RBM is not a consequence of being a non-linear classifier.

Consequently, the good generalization performance of the RBM is justified by the hidden neurons that represent correlations exclusively between active features of the sparse vectors. The model receives a high-dimensional sparse vector and by

TABLE III
TRAIN AND TEST MEAN ACCURACIES OF MLP AND RBM GIVEN THE GENERATED DATA.

	Mean train accuracy	Mean test accuracy
Multi-Layer Perceptron (MLP)	100%	64.12%
Restricted Boltzmann Machine (RBM)	76.18%	73.98%

capturing the relevant features in a much lower dimensional hidden layer can perform classification without falling into overfitting.

After the research performed throughout this chapter, the question raised in section II can, finally, be answered. Actually, our intuition pointed in the right direction and, thus, one can conclude that the main reason for the good generalization performance of the RBM is the fact that hidden neurons learn correlations between active features of the high-dimensional sparse vectors.

V. CONCLUSION

SDRs are the fundamental form of representing information in the brain. The activity of any population of neurons in the neocortex is sparse, where a low percentage of neurons are highly active, and the remaining neurons are inactive [5]. Previous research explored these representations with biologically plausible models to perform associative memory tasks. To learn a good and general classifier without running into the "curse of dimensionality" problem, however, is a hard task. Deep learning models progressively reduce the dimensionality of the SDR from layer to layer and have some success in tasks in which there is a great amount of data with labels, although they use a non-biologically plausible algorithm.

The present thesis explores the capabilities of classifiers inspired in Stochastic Models to side step the limitations that classic Machine Leaning models have, when classifying high dimensional sparse data.

The main evidence that motivated the use of Stochastic Models is the fact that hidden units in these models represent hidden correlations between active neurons of sparse vectors. As sparse vectors have few active neurons, then stochastic models can map a high-dimensional sparse vector into a hidden layer with few hidden units, which represents the visible correlations between present dimensions of the high-dimensional sparse vector [27].

In order to solve the problem that instigated this research work, in section III we described in detail the main characteristics of the Stochastic Methods. This chapter gathers all the theoretical knowledge needed to understand each stochastic model implemented to perform the experiments described in section IV.

During the experimental analysis, we started by using the strategy structured in [1] to generate the sparse codes. With these codes, we tested the implemented classifiers inspired in Stochastic Models and compared their accuracy results with a simple Logistic Regression. Both the Stochastic Models and the LR achieved good results when classifying these codes. In fact, these good results archived by the LR classifier left us wondering if the generated codes were effectively high-dimensional and sparse. Actually, it is plausible to think that the generated MNIST sparse data reside on a low-dimensional manifold embedded in a higher-dimensional space. This suggests that the real dimensionality of the data is highly inferior to the defined number of features of the dataset, which can justify the good performance of LR.

With the desire to study the performance of a RBM and compare it with a LR in high-dimensional sparse data, we defined a different dataset generation strategy, in which each class followed a multivariate normal distribution. To control the sparseness of the data a pre-defined number of features were randomly deleted from each sample. We performed several experiments using this high-dimensional sparse data, in which datasets with varying dimensionalities and sparseness were generated. The good generalization capability achieved by RBM showed that this model can map a high-dimensional sparse vector into a hidden layer, which catches the relevant features present on the high-dimensional sparse vector, while the LR, with the increasing dimensionality and sparseness, becomes too adapted to the training set, which leads to the overfitting problem.

To understand if the good generalization performance of the RBM is really justified by its capability to map a high-dimensional sparse vector into few hidden units, which capture the relevant features present in the data, or if it results from being a non-linear classifier, a comparison was made between the RBM and the MLP with the same number of hidden units. Considering the poor generalization performance obtained by the MLP when classifying the high-dimensional sparse data, one can conclude that the main reason for the good performance of the RBM is the ability to capture correlations between active features of the high-dimensional sparse vectors.

Thus, one can conclude that the motivation for this research work pointed in the right direction. The results achieved by the implemented Stochastic Models demonstrate that by learning the correlations between active features of the sparse input data, good results can be achieved by side stepping the overfitting problem, which affects classic Machine Learning models.

REFERENCES

[1] L. Sa-Couto and A. Wichert, "Storing object-dependent sparse codes in a willshaw associative network," *Neural Computation*, vol. 32, no. 1, pp. 136–152, 2020.

[2] S. Ahmad and J. Hawkins, "Properties of sparse distributed representations and their application to hierarchical temporal memory," *arXiv preprint arXiv:1503.07469*, 2015.

[3] G. E. Hinton, "Distributed representations," 1984.

[4] G. Palm, "On associative memory," *Biological cybernetics*, vol. 36, no. 1, pp. 19–31, 1980.

[5] J. Hawkins, S. Ahmad, S. Purdy, and A. Lavin, "Biological and machine intelligence (bami)," 2016, initial online release 0.4.

[6] J. Milnor, "On the concept of attractor," in *The theory of chaotic attractors*. Springer, 1985, pp. 243–264.

[7] J. A. Hertz, *Introduction to the theory of neural computation*. CRC Press, 2018.

[8] G. Palm, "Chapter xii how useful are associative memories?" in *North-Holland Mathematics Studies*. Elsevier, 1982, vol. 58, pp. 145–153.

[9] ——, "Neural associative memories and sparse coding," *Neural Networks*, vol. 37, pp. 165–171, 2013.

[10] B. Ouyang, Y. Li, Y. Song, F. Wu, H. Yu, Y. Wang, M. Bauchy, and G. Sant, "Learning from sparse datasets: Predicting concrete's strength by machine learning," *arXiv preprint arXiv:2004.14407*, 2020.

[11] J. Bissmark and O. Wärnling, "The sparse data problem within classification algorithms: The effect of sparse data on the naïve bayes algorithm," 2017.

[12] M. Tan, L. Wang, and I. W. Tsang, "Learning sparse svm for feature selection on very high dimensional datasets," in *ICML*, 2010.

[13] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1, no. 2.

[14] X. Li, "Classification with large sparse datasets: Convergence analysis and scalable algorithms," 2017.

[15] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning from data*. AMLBook New York, NY, USA:, 2012, vol. 4.

[16] A. Meyder and C. Kiderlen, "Fundamental properties of hopfield networks and boltzmann machines for associative memories," *Machine Learning, vt*, 2008.

[17] G. E. Hinton, "Boltzmann machine," *Scholarpedia*, vol. 2, no. 5, p. 1668, 2007.

[18] A. Fischer and C. Igel, "Training restricted boltzmann machines: An introduction," *Pattern Recognition*, vol. 47, no. 1, pp. 25–39, 2014.

[19] I. Stoianov, M. Zorzi, S. Becker, and C. Umilta, "Associative arithmetic with boltzmann machines: The role of number representations," in *International Conference on Artificial Neural Networks*. Springer, 2002, pp. 277–283.

[20] C. C. Aggarwal *et al.*, *Neural networks and deep learning*. Springer, 2018.

[21] G. E. Hinton *et al.*, "Boltzmann machines." 2017.

[22] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[23] ——, "Deep belief nets." 2010.

[24] A. M. Wichert, *Principles Of Quantum Artificial Intelligence: Quantum Problem Solving And Machine Learning*. World Scientific, 2020.

[25] L. Younes, "Stochastic gradient estimation strategies for markov random fields," in *Bayesian inference for inverse problems*, vol. 3459. International Society for Optics and Photonics, 1998, pp. 315–325.

[26] V. Mnih, H. Larochelle, and G. E. Hinton, "Conditional restricted boltzmann machines for structured output prediction," *arXiv preprint arXiv:1202.3748*, 2012.

[27] K. Cho, A. Ilin, and T. Raiko, "Tikhonov-type regularization for restricted boltzmann machines," in *International Conference on Artificial Neural Networks*. Springer, 2012, pp. 81–88.

[28] G. E. Hinton, "Learning multiple layers of representation," *Trends in cognitive sciences*, vol. 11, no. 10, pp. 428–434, 2007.

[29] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[30] A.-r. Mohamed, G. Dahl, G. Hinton *et al.*, "Deep belief networks for phone recognition," in *Nips workshop on deep learning for speech recognition and related applications*, vol. 1, no. 9. Vancouver, Canada, 2009, p. 39.

[31] G. E. Hinton, "A practical guide to training restricted boltzmann machines," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 599–619.