



ORIGINAL ARTICLE

Can a Hebbian-like learning rule be avoiding the curse of dimensionality in sparse distributed data?

Maria Osório¹ · Luis Sa-Couto¹ · Andreas Wichert¹

Received: 11 June 2024 / Accepted: 20 August 2024 / Published online: 9 September 2024
© The Author(s) 2024

Abstract

It is generally assumed that the brain uses something akin to sparse distributed representations. These representations, however, are high-dimensional and consequently they affect classification performance of traditional Machine Learning models due to the “curse of dimensionality”. In tasks for which there is a vast amount of labeled data, Deep Networks seem to solve this issue with many layers and a non-Hebbian backpropagation algorithm. The brain, however, seems to be able to solve the problem with few layers. In this work, we hypothesize that this happens by using Hebbian learning. Actually, the Hebbian-like learning rule of Restricted Boltzmann Machines learns the input patterns asymmetrically. It exclusively learns the correlation between non-zero values and ignores the zeros, which represent the vast majority of the input dimensionality. By ignoring the zeros the “curse of dimensionality” problem can be avoided. To test our hypothesis, we generated several sparse datasets and compared the performance of a Restricted Boltzmann Machine classifier with some Backprop-trained networks. The experiments using these codes confirm our initial intuition as the Restricted Boltzmann Machine shows a good generalization performance, while the Neural Networks trained with the backpropagation algorithm overfit the training data.

Keywords Hebbian learning · Restricted Boltzmann machines · Sparse distributed representations · Curse of dimensionality

1 Introduction

Deep Learning enables high-level abstractions in data using architectures composed of multiple non-linear transformations. These models have significantly improved the state-of-the-art in different domains, such as speech recognition, visual object recognition, object detection and many others (LeCun and Bengio 1998; Huanget et al. 2019). Recent deep learning models have made significant strides in different learning paradigms such as supervised, unsupervised and self-supervised learning (Gui et al. 2024; Baeovski et al.

2023; Schmarje et al. 2021). Some examples of recent models that have shown great success include Vision Transformers (Vaswani et al. 2017), Generative Adversarial Networks (Goodfellow et al. 2014), Large Language Models (Brown et al. 2020; Thoppilan et al. 2022)

However, despite these advancements, the backpropagation learning rule and the deep architectures employed by these models represent a significant departure from the principles of biological neural networks (Serre 2019). The human brain seems to have better generalization performance with much fewer neurons and complexity, which greatly reduces energy consumption. This discrepancy highlights the need for exploring alternative approaches that more closely align with brain-related learning mechanisms.

The brain is the best natural example of a learning system that can perform extremely difficult tasks in a mostly unsupervised manner (Trappenberg 2009; Golomb et al. 1990). In order to represent information, the brain is thought to share 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 (Tyler et al. 2000; Ahmad and

Communicated by Taro Toyozumi.

✉ Maria Osório
maria.osorio@tecnico.ulisboa.pt

Luis Sa-Couto
luis.sa.couto@tecnico.ulisboa.pt

Andreas Wichert
andreas.wichert@tecnico.ulisboa.pt

¹ Department of Computer Science and Engineering,
INESC-ID & Instituto Superior Técnico - University of
Lisbon, Av. Prof. Dr. Aníbal Cavaco Silva, Porto Salvo
2744-016, Lisbon, Portugal

Hawkins 2015). When looking at any population of neurons in the neocortex their activity will be sparse, that is, a low percentage of neurons are highly active and the remaining neurons are inactive (Tyler et al. 2000; Hawkins et al. 2016).

Sparse Distributed Representations (SDRs) is the method used to implement computationally the way information is represented in the brain (Tyler et al. 2000; Tyler and Moss 2001; Quiroga 2012). An SDR is a binary vector composed of a large number of bits where each bit represents a neuron in the neocortex (Ouyang et al. 2020). Besides high-dimensional, these vectors are also sparse, which means there is a low percentage of informative (non-zero) bits (Hinton et al. 1997; Sa-Couto and Wichert 2023).

To recognize a particular activity pattern a neuron forms synapses to the active cells in that pattern of activity (Palm 1980). This way, a neuron only needs to form a small number of synapses, to accurately recognize a sparse pattern in many cells. (Hawkins et al. 2016; Milnor 1985; Hertz et al. 1991).

As discussed previously, SDRs are binary vectors composed of many bits, which means we are dealing with a high-dimensional input. These sparse representations are known to work well with associative memory models Palm (1982, 1980); Golomb et al. (1990); Sa-Couto and Wichert (2020), but when we try to classify them, we must deal with some problems.

Classic Machine Learning models, such as Feed-Forward Networks, struggle with high-dimensional sparse inputs. The dimensionality of the input data results in a vast number of parameters, making the models prone to overfitting Tan et al. (2010). Additionally, the sparseness of the data increases problem complexity, affecting classification performance due to the “curse of dimensionality”. This concept refers to the challenges that arise when dealing with data in high-dimensional spaces, often leading to increased complexity and decreased performance in machine learning models. As the number of dimensions in a dataset increases, the volume of the space grows exponentially, causing the data to become sparse. This sparsity can significantly affect the performance of machine learning models. In high-dimensional spaces, models can easily become overly complex, fitting the noise in the training data rather than capturing the underlying patterns. This results in overfitting, where models perform well on training data but poorly on unseen test data. The increased dimensionality introduces more parameters to estimate, thereby escalating the risk of overfitting Altman and Krzywinski (2018); Goodfellow et al. (2016).

1.1 Hypothesis

When dealing with sparse data there is a low percentage of informative bits in the vector. Thus, in order to learn a good and general classifier from these representations, it would be great if there was a model capable of ignoring the empty

Table 1 Train and test mean accuracies of LR when classifying a sample of the original binarized MNIST and a flipped version of the same sample

	Train accuracy(%)	Test accuracy(%)
Original version	99.6	85.2
Flipped version	99.7	84.5

dimensions and focusing on active units. Ignoring the zeros would ignore a lot of the dimensionality of the sparse vectors, thus solving many of the previously referred issues.

Classic Machine Learning models cannot ignore empty dimensions (zeros) in high-dimensional sparse data. Thus, these models have to learn all the dimensions of the sparse input.

A good way to prove this statement is to classify the same information encoded in two different ways. In Fig. 1, the ten top images represent the binarized version of the well-known MNIST¹ dataset, 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.

We started by performing the aforementioned experiment using a Logistic Regression (LR), which basically amounts to a softmax-based single layered Neural Network trained by backpropagation of errors and it is generally used as the output layer of a Deep Learning classifier.

By analysing the results on Table 1, one concludes that this model has similar performances when classifying the original and the flipped version of the MNIST dataset.

The similar accuracy results achieved by LR in both problems can be explained by the fact 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, all the dimensions are learnt.

Our hypothesis states that a biologically motivated Hebbian-like local learning rule can ignore the empty dimensions (zero values). In fact, the Restricted Boltzmann Machine (RBM) learning rule ignores the zeros present on the input data, which means it exclusively learns the non-zero dimensions. Considering that the empty dimensions represent the vast majority of the input dimensionality, by ignoring the zeros the “curse of dimensionality” problem can be avoided.

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

¹ <http://yann.lecun.com/exdb/mnist/>.

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

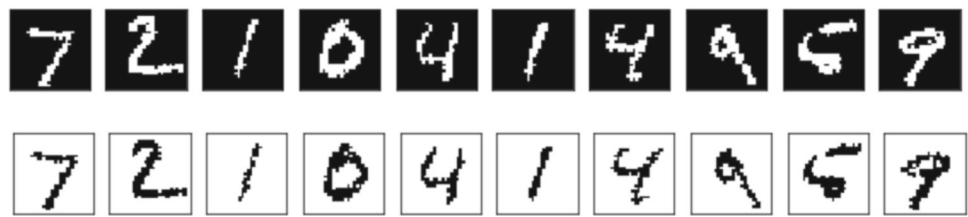


Table 2 Train and test mean accuracies of RBM when classifying a sample of the original binarized MNIST and a flipped version of the same sample

	Train accuracy(%)	Test accuracy(%)
Original version	91.6	86.2
Flipped version	15.5	13.8

1 s 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, Table 2 shows the results achieved by the RBM classifier.

By analysing the results achieved by the RBMs with 500 hidden units, one can conclude that with the original version of the binarized MNIST, the model seems to learn 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 may be justified by the fact that this model is unable to catch the correlations between all the active neurons that represent the background.

This small experiment validates the strong potential of RBMs to deal with high-dimensional sparse inputs. Our hypothesis is that RBMs ignore the zeros and exclusively capture correlations between active units. Could it be that by ignoring the zeros we can avoid the “curse of dimensionality” problem when classifying sparse data?

Actually, throughout this research paper we will answer this main question by:

1. Demonstrating that LR lacks generalization performance when dealing with high-dimensional sparse data as it learns all the dimensions of the sparse input.
2. Showing that the RBM has a good generalization capability as it exclusively learns the correlation between active neurons of the high-dimensional sparse data.
3. Comparing the RBM with a Multi-Layer Perceptron (MLP) to prove the generalization capability of the RBM is not a consequence of it being a non-linear classifier.

2 Background

2.1 Restricted Boltzmann machines

Restricted Boltzmann Machines (RBMs) were initially invented under the name Harmonium. They are a variant of Boltzmann machines, with the restriction that there is a single layer of m visible units $\mathbf{v} = (v_1, v_2, \dots, v_m)$ and a single layer of n hidden units $\mathbf{h} = (h_1, h_2, \dots, h_n)$ with no visible-visible or hidden-hidden connections Salakhutdinov et al. (2007).

The energy function of a Restricted Boltzmann Machine can be written as

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

For all $i \in 1, \dots, n$ and $j \in 1, \dots, m$, 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 Meyder et al. (2008).

2.1.1 Contrastive divergence algorithm

Markov Chain Monte Carlo (MCMC) methods generally require many sampling steps to obtain unbiased estimates of the log-likelihood gradient. However, research has shown that running the chain for only a few steps can provide estimates sufficient for model training Hinton (2002); Fischer and Igel (2014). Contrastive divergence (CD) speeds up the computing time of the negative learning phase by avoiding the need for Gibbs sampling to reach thermal equilibrium Fischer and Igel (2014). In this algorithm, the training phase begins by fixing the visible units at v^0 and then computing the hidden layer units h^0 as follows:

$$p(h_i = 1|\mathbf{v}) = \sigma \left(\sum_{j=1}^m w_{ij} \cdot v_j + c_i \right) \quad (2)$$

Equation (2) defines $\langle v_j h_i \rangle_{data}^0$. As previously discussed, there are no connections between visible-visible or hidden-hidden units. Consequently, each hidden unit h_i operates independently of the other hidden units. This independence allows for h^0 to be computed in parallel, as each hidden unit exclusively depends on the connected visible units Hinton et al. (2006).

The second step involves updating all the visible units simultaneously to obtain a “reconstruction” v^1 , which can be computed by:

$$p(v_j = 1|\mathbf{h}) = \sigma \left(\sum_{i=1}^n w_{ij} \cdot h_i + b_j \right) \quad (3)$$

Equation (3) defines $\langle v_j h_i \rangle_{recon}^1$. The visible units are now fixed with v^1 and the hidden layer units h^1 are computed in parallel using Eq. (2). This reconstruction algorithm can be repeated τ times or until convergence is reached. In some cases, CD may require many iterations ($1 \ll \tau$) to converge. When $\tau = 1$ we are performing a single-step reconstruction Hertz et al. (1991); Wichert (2020). The weights update computed for τ steps of the reconstruction algorithm is given by:

$$\Delta w_{ij} = \eta \cdot (\langle v_j h_i \rangle_{data}^0 - \langle v_j h_i \rangle_{recon}^\tau) \quad (4)$$

Hebbian learning is a type of activity-dependent synaptic plasticity in which the connection between two neurons is strengthened when the pre and postsynaptic neurons are activated simultaneously Jaeger and Jung (2014).

The learning rule of RBMs follows a form of Hebbian learning, as the weights are adjusted in proportion to the correlations between the states of nodes i and j . In this process, the visible vectors are fixed to a vector from the training data, and the hidden states are randomly set to 0 or 1 Goodfellow et al. (2016); Hinton (2002).

2.1.2 Persistent contrastive divergence algorithm

Initializing the Markov chains at each gradient step using their states from the previous step is an alternative approach to address many of the issues with Contrastive Divergence (CD). Initially discovered in the applied mathematics and statistics community as Stochastic Maximum Likelihood, this method was later rediscovered as Persistent Contrastive Divergence (PCD) Younes (1998).

The key idea behind this strategy is that if the stochastic gradient algorithm takes small steps, the model from the previous step will be similar to the current model. Therefore, samples from the previous model’s distribution will be nearly fair samples from the current model’s distribution.

By continuously updating each Markov chain throughout the learning process, rather than restarting them at each gradient step, the chains can thoroughly explore the model’s landscape and find all the minima. Consequently, Persistent Contrastive Divergence is significantly less likely to form models with spurious minima compared to the original Contrastive Divergence algorithm Mnih et al. (2011); Goodfellow et al. (2016).

2.1.3 Restricted Boltzmann machines for classification

In general, RBMs are described and thought of as generative models. However, one can look at the same architecture as Feed-Forward Network classifier with a different learning algorithm.

First we have the training phase, where the RBM learns to model the joint probability distribution of input data (explanatory variables) and the corresponding label (output variable), both represented by the visible units of the model as shown in the left network on Fig. 2. The RBM is trained with one of the previously described algorithm: either CD or PCD Fischer and Igel (2014); Hinton (2002).

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 joint 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 Fischer and Igel (2014).

2.2 Backpropagation neural networks

Deep Learning architectures are composed of multiple layers that perform non-linear transformations LeCun et al. (2015). The foundation of these complex multi-layer networks is the single-layered model known as Logistic Regression (LR), often represented as the final layer in these architectures. These networks are typically trained using Stochastic Gradient Descent. The backpropagation procedure can be used to compute gradients, as long as the neurons in each layer are smooth functions of their inputs and internal weights. This algorithm operates iteratively through two main phases. Initially, the forward phase propagates input data through the network to generate an output. Next, the backward phase computes the gradient of the objective with respect to the input of a layer by propagating backwards from the gradient with respect to the layer’s output. The backpropagation algo-

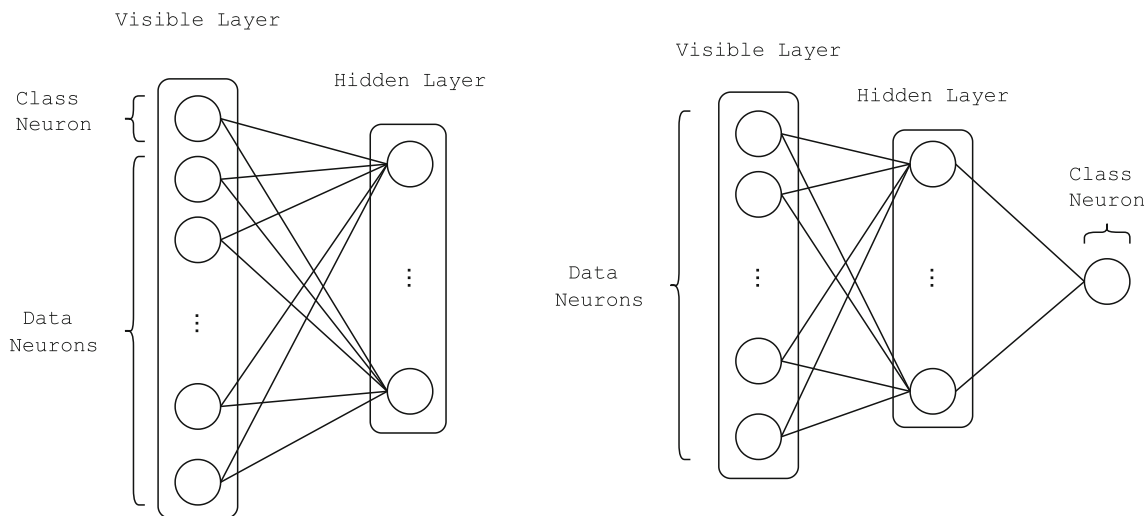


Fig. 2 RBM that models the joint probability distribution of input images and the corresponding labels (left) and the Feed-Forward perspective of the same network (right)

rithm allows for repeated propagation of gradients through all layers. It starts from the output layer, where predictions are generated, and propagates backward to the input layer, where data enters. This means error calculations begin at the output layer and are propagated backwards, giving rise to the concept of errors backpropagating through the network. LeCun et al. (2015); Goodfellow et al. (2016).

After computing these gradients, determining the gradients with respect to the weights between each layer k becomes straightforward by using the expression for the partial derivative of the error with respect to each weight:

$$\Delta w_{ij}^k = -\eta \cdot \frac{\partial E}{\partial w_{ij}^k} \quad (5)$$

Backpropagation learning rule (Eq. (5)), adjusts the weights to minimize the error between the actual output and the output predicted by the network. Actually, this is a non-Hebbian learning rule as it does not change weights based on local correlations between neurons.

Figure 3 illustrates two Neural Network architectures that can be trained using the backpropagation algorithm described earlier. The architecture on the left represents a Neural Network without any hidden layers, commonly known as Logistic Regression. The network on the right has the same input and output layers but includes one hidden layer in between. Although more layers could be added to these Neural Networks, the architectures in Fig. 3 are specifically chosen to be comparable to the Restricted Boltzmann Machine model, which also has a single hidden layer.

In fact, the only difference between this Neural Network with one hidden layer and the RBM is the learning algorithm. The Neural Networks in Fig. 3 are trained with backpropaga-

tion of error, which adjusts the weights to minimize the error between the actual output and the predicted output of the network. In contrast, the RBM uses the Contrastive Divergence algorithm, which follows a local Hebbian-like learning rule. Unlike in backpropagation, there is no error propagation in RBMs, instead it learns to model the joint probability distribution of input data and the corresponding label.

3 Experiments

The main hypothesis of our research work grounds on the advantage of the local Hebbian-like rule used by Restricted Boltzmann Machines (RBMs) to avoid the “curse of dimensionality” problem when dealing with sparse distributed data. With the desire to validate this hypothesis, we dedicate this section to analyze the performed experiments.

The first step required before diving into the experimental analysis, is the definition of the used sparse data.

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. This approach was driven by the need to create a sufficiently challenging dataset that could effectively demonstrate the problem.

Bearing in mind our hypothesis, we started by implementing a RBM. This implementation 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 Devia-

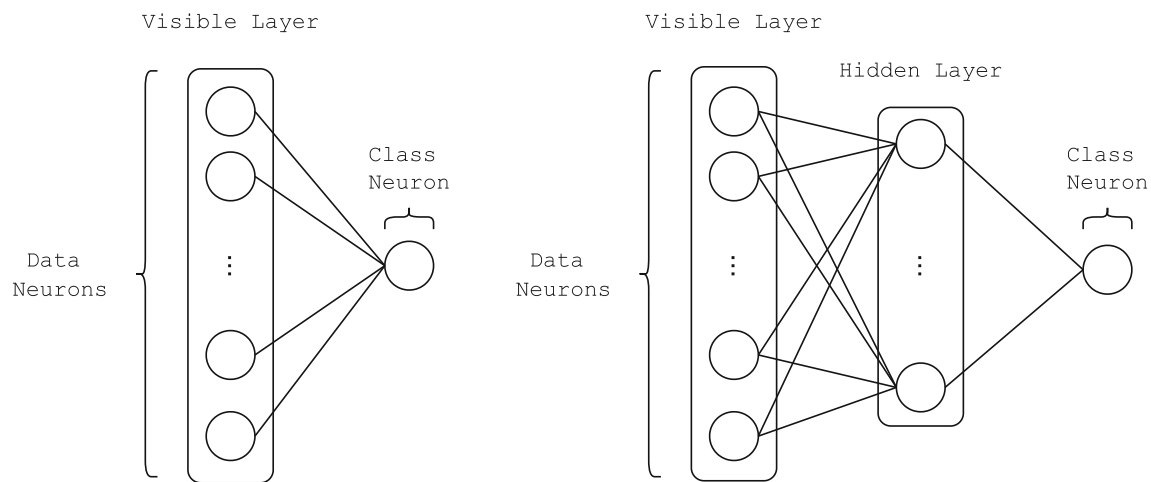


Fig. 3 Architecture of a single-layer Neural Network, which is also known as Logistic Regression (left) and a multi-layer Neural Network with one hidden layer (right)

tion parameter is a hard task, which can produce an unstable learning process (Hinton 2012).

In fact, the only difference between using Bernoulli or Gaussian visible units occurs when sampling the visible units of the negative phase of the learning 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.

To prove our hypothesis is accurate, a comparison was done between the implemented RBM and a Backprop-trained model. As previously stated the model used to be compared with the RBM is the Logistic Regression (LR). The main reason for the choice of this model is the fact that it represents a Backprop-trained single layered Neural Network and it is generally used as the output layer of a Deep Neural Network.

Previously to the experimental analysis results presentation and discussion, we describe in detail the dataset generation strategy and the pipeline followed in each experiment.

3.1 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 at the origin, the other half corresponds to class 1 and follows another Gaussian distribution with mean centered in a vector of fives. 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 problem complexity.

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

3.2 Pipeline

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. Centering 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, with the respective percentages of 80% and 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 the PCD algorithm.
8. Evaluating the RBM by performing Gibbs sampling to get the reconstruction of the class unit for both train and test sets. 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.

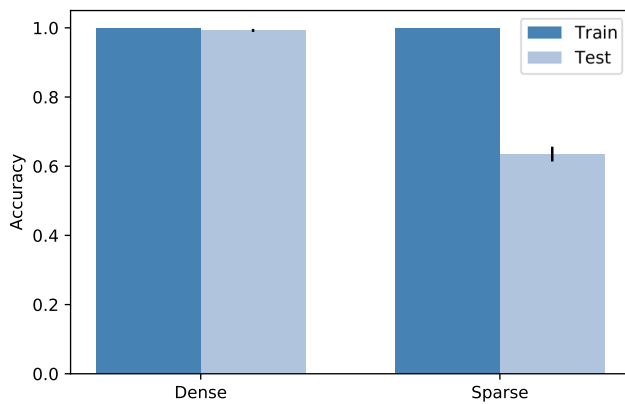


Fig. 4 Performance of the LR in a dense versus a high-dimensional sparse dataset

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.

3.3 Experimental analysis

We started by generating a dense and a sparse dataset. The dense dataset was generated as described in Sect. 3.1. 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 Sect. 3.1 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 Fig. 4, 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 the model registers tremendous overfitting, 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 intent 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 assess 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 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.

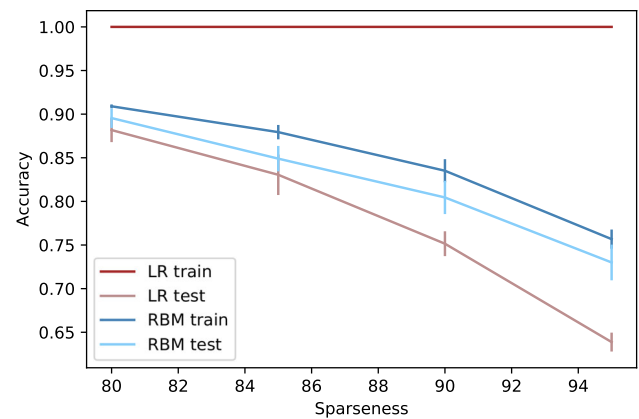


Fig. 5 Comparison between accuracies of LR and RBM classifiers with increasing percentage of sparseness

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 Sect. 3.2 was followed. To make an easier comparison between models, the sparsity percentage was set to each x-axis value and the mean accuracies of LR and RBM were plotted in Fig. 5.

By analysing the results plotted in Fig. 5, 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 that LR can represent the training set of sparse data perfectly, however, is unable to generalize, which results in a poor test set performance. On the contrary, the RBM classifier can generalize the learning problem.

To understand how LR and RBM behave with different dimensionalities and sparseness percentages, both models varying these two parameters were run. The 3-D surface plots in Fig. 6, show the difference between train and test accuracies of RBM and LR considering the number of features (dimensionality) and the sparseness percentage used with each different dimensionality dataset. In the 3-D surface plot on the right, one can observe that as the data gets sparser the difference between both surfaces, which represent the test accuracies, increases and the RBM shows to perform better than the LR. With lower sparseness the RBM and the LR have similar test results. The dimensionality behaves in the same way, which means that with higher dimensional data it is easier to notice the better results achieved by the RBM. The 3-D surface plot on the left shows the train accuracy difference of both models. The LR has a train accuracy of 100%, regardless of the sparsity or the dimensionality of the data. The surface plotted representing the RBM train accu-

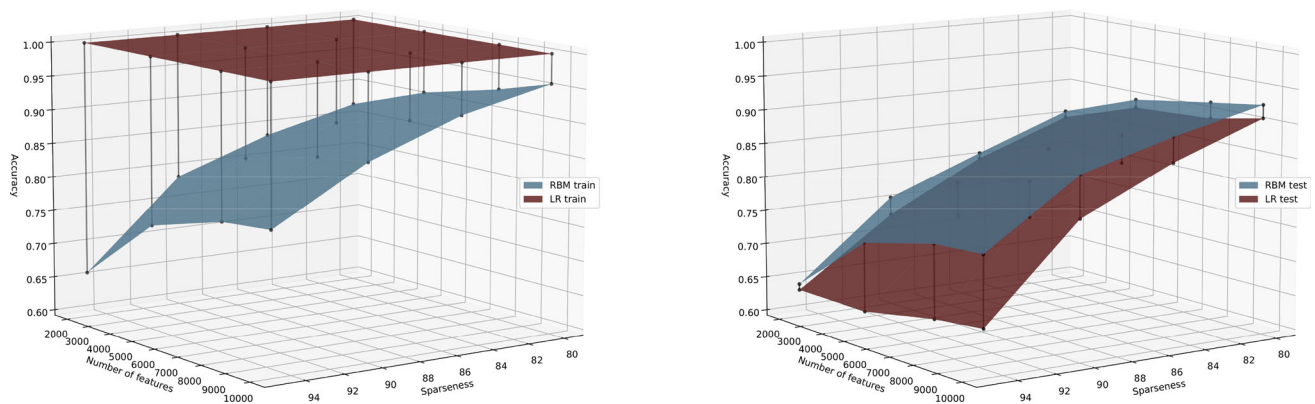


Fig. 6 The left plot shows the difference between train accuracies of RBM and LR considering increasing dimensionality and sparseness percentage. The right plot shows the difference between test accuracies of RBM and LR considering increasing dimensionality and sparseness percentage

racy is very similar to the RBM test surface, which shows a good generalization performance.

The good results and generalization performance achieved by the RBM classifier can be mainly justified by the fact that this model exclusively learns the correlations between active units and ignores the zeros, which represents most of the dimensionality of the input. Besides, the RBM 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 catches the relevant features present on the high-dimensional sparse vector.

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 Multi-Layer Perceptron (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 Sect. 3.2. 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 mean train and test accuracies in Table 3 were obtained.

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 it being a non-linear classifier. The insertion of a hidden layer does not changes the conclusions taken with the LR. Notably, the experiment with the MLP at 95% sparsity, which represents the harder

case, yielded similar results to those obtained with the LR. This consistency suggests that both LR and MLP exhibit similar behavior when dealing with high-dimensional sparse data. Thus, our initial intuition pointed in the right direction and one can conclude that the main reason for the good generalization performance of the RBM is justified by the Hebbian-like learning algorithm and not by the presence of hidden layers.

3.4 Discussion

Our work emphasizes the importance of sparse representations, which appear to be the way the brain encodes information. However, it is also crucial to consider that Hebbian Learning is recognized as the method by which the brain performs learning, where the activation of a neuron exclusively depends on the neurons connected to it. While modern neural networks utilize both active and inactive neurons to capture complex patterns, the Hebbian-like learning rule in RBMs focuses on learning the correlations among active neurons. The learning algorithms are distinct in their approaches: backpropagation updates weights globally, while Hebbian Learning performs updates based exclusively on local connections. This Hebbian Learning approach enhances generalization when learning Sparse Distributed Representations.

The experimental analysis performed on the previous section ground our initial hypothesis where we state that the local Hebbian-like learning rule used by RBMs represents the key factor to avoid the “curse of dimensionality” problem in sparse distributed data. By analysing the experimental results present on the previous section, one can conclude that the explored Backprop-trained networks, i.e. the LR and the MLP, show a lack of generalization performance on this kind of data. This is justified by the fact that they learn all the dimensions of the high-dimensional sparse representations. Contrarily, the RBM shows a good generalization capability

Table 3 Train and test mean accuracies of MLP and RBM given the generated data

	Train accuracy(%)	Test accuracy(%)
Multi-layer perceptron (MLP)	100	64.12
Restricted Boltzmann machine (RBM)	76.18	73.98

by exclusively learning the correlation between active neurons of the high-dimensional sparse representations.

In order to expand the performed analysis, we suggest the following directions of research: (1) Tackle a Real-World Sparse Data problem by using a RBM and compare it with Backprop-trained networks. (2) Assess the potential increase in performance by using more powerful networks. To that end, a deeper MLP can be used and compared with a Deep Belief Network (DBN), which is a deep architecture of multiple stacks of RBMs.

4 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 (Tyler and Moss 2001; Quiroga 2012; Hawkins et al. 2016). 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 is a hard task. Deep Networks with non-Hebbian learning progressively reduce the dimensionality of the SDR from layer to layer and have success in tasks in which there is a great amount of data with labels.

The present article explores the capabilities of a Hebbian-like learning rule to side step the limitations that classic Machine Learning models have, when classifying high dimensional sparse data.

Our hypothesis grounds on the capability of the Hebbian learning of RBMs to ignore the empty dimensions of the input data focusing on active units.

To analyze the validity of our hypothesis, we defined a dataset generation strategy in which each class followed a multivariate normal distribution. Several experiments were carried out using this high-dimensional sparse data, in which datasets with varying dimensionalities and sparseness were generated.

The good generalization capability achieved by the RBM shows that the local Hebbian-like rule used in the network learning algorithm represents the key factor to avoid the “curse of dimensionality” problem in sparse distributed data. This model by exclusively learning the correlation between non-zero neurons is able to map a high-dimensional sparse

vector into a hidden layer, which catches the relevant features present on the high-dimensional sparse vector.

On the other hand, both LR and MLP, by using the non-Hebbian backpropagation algorithm, have to learn all the dimensions of the sparse data and consequently become too adapted to the training set, leading to the overfitting problem.

Considering that we are dealing with biological-like representations, it is expected that a biologically plausible algorithm can better deal with these representations. We can conclude that using a Hebbian-like learning rule represents a clear advantage when dealing with high-dimensional sparse data.

Acknowledgements This work was funded by national funds from Fundação para a Ciência e Tecnologia (FCT) with reference UIDB/500 21/2020, through a doctoral grant SFRH/BD/04409/2023 awarded to the first author and a doctoral grant SFRH/BD/144560/2019 awarded to the second author. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Author Contributions A.W., L.S.C and M.O. conceived the presented idea and planned the experiments. M.O. carried out the experiments. M.O. wrote the main manuscript text. A.W., L.S.C and M.O. discussed the results and reviewed the final manuscript.

Funding Open access funding provided by FCTIFCCN (b-on).

Data availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors declare no competing interests.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Ahmad S, Hawkins J (2015) Properties of sparse distributed representations and their application to hierarchical temporal memory. CoRR, [arxiv:1503.07469](https://arxiv.org/abs/1503.07469)

- Altman N, Krzywinski M (2018) The curse (s) of dimensionality. *Nat Methods* 15(6):399–400
- Baevski A, Babu A, Hsu W-N, Auli M (2023) Efficient self-supervised learning with contextualized target representations for vision, speech and language. *International conference on machine learning* (pp.1416–1429)
- Brown, TB, Mann B, Ryder N, Subbiah M, Kaplan J, Dhariwal P, Amodei D (2020). Language models are few-shot learners. H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), *Advances in neural information processing systems 33: Annual conference on neural information processing systems 2020*, neurips 2020, december 6–12, 2020, virtual
- Fischer A, Igel C (2014) Training restricted Boltzmann machines: an introduction. *Pattern Recognit* 47(1):25–39. <https://doi.org/10.1016/j.patcog.2013.05.025>
- Golomb D, Sompolinsky Rubin NH (1990) Willshaw model: associative memory with sparse coding and low firing rates. *Phys Rev A* 41:1843–1854. <https://doi.org/10.1103/PhysRevA.41.1843>
- Goodfellow IJ, Bengio Y, Courville AC (2016). *Deep learning*. MIT Press. Retrieved from <http://www.deeplearningbook.org/>
- Goodfellow IJ, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Bengio Y (2014). Generative adversarial nets. Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, K.Q. Weinberger (Eds.), *Advances in neural information processing systems 27: Annual conference on neural information processing systems 2014*, december 8–13 2014, montreal, quebec, canada (pp. 2672–2680)
- Gui J, Chen T, Zhang J, Cao Q, Sun Z, Luo H, Tao D (2024). A survey on self-supervised learning: Algorithms, applications, and future trends. *IEEE Transactions on Pattern Analysis and Machine Intelligence*
- Hawkins J, Ahmad S, Purdy S, Lavin A (2016) Biological and machine intelligence (bami). Retrieved from <https://numenta.com/resources/biologicaland-machine-intelligence/> (Initial online release 0.4)
- Hertz JA, Krogh A, Palmer RG (1991) *Introduction to the theory of neural computation* (Vol. 1). Addison-Wesley. Retrieved from <https://www.worldcat.org/oclc/21522159>
- Hinton GE (2002) Training products of experts by minimizing contrastive divergence. *Neural Comput* 14(8):1771–1800. <https://doi.org/10.1162/089976602760128018>
- Hinton GE (2012). A practical guide to training restricted boltzmann machines. G. Montavon, G.B. Orr, K. Müller (Eds.), *Neural networks: Tricks of the trade - second edition* (Vol. 7700, pp. 599–619). Springer. https://doi.org/10.1007/978-3-642-35289-8_32
- Hinton GE, Ghahramani Z (1997) Generative models for discovering sparse distributed representations. *Philosophical transactions of the royal society of London. Series B: Biol Sci* 352:1177–1190
- Hinton GE, Osindero S, Teh YW (2006) A fast learning algorithm for deep belief nets. *Neural Comput* 18(7):1527–1554. <https://doi.org/10.1162/neco.2006.18.7.1527>
- Huang K, Hussain A, Wang Q-F, Zhang R (2019). *Deep learning: Fundamentals, theory and applications* (Vol. 2). Springer International Publishing
- Jaeger D, Jung R (eds) (2014) *Encyclopedia of computational neuroscience*. Springer, Retrieved from. <https://doi.org/10.1007/978-1-4614-7320-6>
- LeCun Y, Bengio Y (1998) *The handbook of brain theory and neural networks*. chapter convolutional networks for images, speech, and time series. MIT Press, Cambridge, MA, USA 3:255–258
- LeCun Y, Bengio Y, Hinton G (2015) Deep learning. *Nature* 521:436–444. <https://doi.org/10.1038/nature14539>
- Meyder A, Kiderlen C (2008) Fundamental properties of hopfield networks and boltzmann machines for associative memories. *Machine Learning*, vt
- Milnor J (1985) On the concept of attractor. *The theory of chaotic attractors* (pp. 243–264). Springer
- Mnih V, Larochelle H, Hinton GE (2011) Conditional restricted boltzmann machines for structured output prediction. F.G. Cozman & A. Pfeffer (Eds.), *UAI 2011, proceedings of the twenty-seventh conference on uncertainty in artificial intelligence*, barcelona, spain, july 14–17, 2011 (pp. 514–522). AUAI Press
- Ouyang B, Li Y, Song Y, Wu F, Yu H, Wang Y, Sant G, (2020). Learning from sparse datasets: Predicting concrete’s strength by machine learning. *CoRR*, [arXiv:2004.14407](https://arxiv.org/abs/2004.14407)
- Palm G (1980) On associative memory. *Bio-logical. Cybernetics* 36:19–31. <https://doi.org/10.1007/BF00337019>
- Palm G (1982) Chapter xii how useful are associative memories? *North-holland mathematics studies* (p. 145–153). Elsevier
- Quiroga RQ (2012) Concept cells: the building blocks of declarative memory functions. *Nat Rev Neurosci* 13(8):587–597
- Sa-Couto L, Wichert A (2020) Storing object-dependent sparse codes in a willshaw associative network. *Neural Comput* 32(1):136–152. https://doi.org/10.1162/neco_a_01243
- Sa-Couto L, Wichert A (2023) Competitive learning to generate sparse representations for associative memory. *Neural Netw* 168:32–43. <https://doi.org/10.1016/j.neunet.2023.09.005>
- Salakhutdinov R, Mnih A, Hinton GE (2007). Restricted boltzmann machines for collaborative filtering. Z. Ghahramani (Ed.), *Machine learning, proceedings of the twenty-fourth international conference (ICML 2007)*, corvallis, oregon, usa, june 20–24, 2007 (Vol. 227, pp. 791–798). ACM. Retrieved from <https://doi.org/10.1145/1273496.1273596>
- Schmarje L, Santarossa M, Schröder S-M, Koch R (2021) A survey on semi-self and unsupervised learning for image classification. *IEEE Access* 9:82146–82168
- Serre T (2019) Deep learning: the good, the bad, and the ugly. *Ann Rev Vision Sci* 5:399–426. <https://doi.org/10.1146/annurev-vision-091718-014951>
- Tan M, Wang L, Tsang IW (2010) Learning sparse SVM for feature selection on very high dimensional datasets. J. Fürnkranz T. Joachims (Eds.), *Proceedings of the 27th international conference on machine learning (icml-10)*, june 21–24, 2010, haifa, israel (pp. 1047–1054). Omnipress
- Thoppilan R, Freitas DD, Hall J., Shazeer, N., Kulshreshtha, A., Cheng, H., Le, Q. (2022). *Lamda: Language models for dialog applications*. *CoRR*, Retrieved from [arXiv:2201.08239](https://arxiv.org/abs/2201.08239)
- Trappenberg, T. (2009). *Fundamentals of computational neuroscience* (2. ed.). Oxford University Press
- Tyler LK, Moss HE (2001) Towards a distributed account of conceptual knowledge. *Trend Cognit Sci* 5(6):244–252
- Tyler LK, Moss HE, Durrant-Peatfield M, Levy J (2000) Conceptual structure and the structure of concepts: a distributed account of category-specific deficits. *Brain lang* 75(2):195–231
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Polosukhin, I. (2017). Attention is all you need. I. Guyon et al. (Eds.), *Advances in neural information processing systems 30: Annual conference on neural information processing systems 2017*, december 4–9, 2017, long beach, ca, USA (pp. 5998–6008)
- Wichert A (2020) *Principles of quantum artificial intelligence - quantum problem solving and machine learning*, second edition. Retrieved from, WorldScientific. <https://doi.org/10.1142/1193>
- Younes L (1998, 9). Stochastic gradient estimation strategies for markov random fields. A. Mohammad-Djafari (Ed.), *Bayesian inference for inverse problems* (p. 315–325)

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.