Applied Computing and Informatics 15 (2019) 81–93

Contents lists available at ScienceDirect

Applied Computing and Informatics

journal homepage: www.sciencedirect.com

# Deep belief networks and cortical algorithms: A comparative study for supervised classification

Yara Rizk, Nadine Hajj, Nicholas Mitri, Mariette Awad *

*Department of Electrical and Computer Engineering, American University of Beirut, Beirut, Lebanon*

ARTICLE INFO

*Article history:*
Received 23 March 2017
Revised 17 January 2018
Accepted 17 January 2018
Available online 3 March 2018

*Keywords:*
Deep learning
Deep belief networks
Cortical algorithms

ABSTRACT

The failure of shallow neural network architectures in replicating human intelligence led the machine learning community to focus on deep learning, to computationally match human intelligence. The wide availability of increasing computing power coupled with the development of more efficient training algorithms have allowed the implementation of deep learning principles in a manner and span that had not been previously possible. This has led to the inception of deep architectures that capitalize on recent advances in artificial intelligence and insights from cognitive neuroscience to provide better learning solutions. In this paper, we discuss two such algorithms that represent different approaches to deep learning with varied levels of maturity. The more mature but less biologically inspired Deep Belief Network (DBN) and the more biologically grounded Cortical Algorithms (CA) are first introduced to give readers a bird's eye view of the higher-level concepts that make up these algorithms, as well as some of their technical underpinnings and applications. Their theoretical computational complexity is then derived before comparing their empirical performance on some publicly available classification datasets. Multiple network architectures were compared and showed that CA outperformed DBN on most datasets, with the best network architecture consisting of six hidden layers.


**Contents**



* Corresponding author.
*E-mail addresses:* yar01@aub.edu.lb (Y. Rizk), njh05@aub.edu.lb (N. Hajj), ngm04@aub.edu.lb (N. Mitri), mariette.awad@aub.edu.lb (M. Awad).
Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.aci.2018.01.004

## 1. Introduction

In an endeavor to replicate human level intelligence, artificial intelligence (AI) research has fused insights from the fields of computer science, cognitive neuroscience, computational science, and a litany of others to produce algorithms that perform with increasing efficacy on what is arguably the core element of intelligence: learning.

Notable among the many learning algorithms in AI are artificial neural networks (ANN) and their many variants. ANN are collections of interconnected artificial neurons that incrementally learn from their environment and attempt to mimic some of the basic information processing processes in the brain. Their function is defined by the processing performed at the neuron level, the connection strengths between neurons (synaptic weights), and network structure (organization and linkage of neurons) [1]. It is the latter that resides at the core of the discussion presented herein.

Throughout their evolution, discussed in more details in the next section, shallow ANN still suffer from multiple issues in the context of complex applications requiring a higher level of abstraction. However, with the rapid increase in processing power, the opportunity to successfully implement the computationally demanding designs of deeper architectures has recently emerged. The development of efficient training algorithm such as Hinton et al.'s greedy algorithm [2] has also helped ANN's resurgence. Furthermore, findings in computational neuroscience have led to increased interest in deep, biologically inspired architectures [3–5] which adhere more faithfully to neuro-scientific theories of the human brain's topology.

In this paper, we limit the scope of our comparative study to two - nowadays popular - algorithms: Hinton et al.'s Deep Belief Networks (DBN) [2], and Cortical Algorithms (CA) [6]. While many other deep architectures have been developed, including long short-term memory for sequential data processing and convolutional neural networks for image processing, this comparative study compares feedforward architectures. Specifically, DBN, one of the more efficient deep architecture training algorithms is compared to CA, a feedforward architecture with more biologically faithful properties. Deep neural networks (DNN), specifically DBN, is presented as the state of the art of ANN in their traditional forms with network topologies built from layers of neuron models but with more advanced learning mechanics and deeper architecture, without modeling the detailed biological phenomena constituting human intelligence. Maintaining a high-level abstraction of the biological modeling, results in simpler mathematical models for DBN compared to CA. On the other hand, CA represents the shift towards incorporating more biologically inspired structures than DBN, like cortical columns and inhibiting and strengthening learning rules, as outlined by Edelman and Mountcastle's work [7].

The structure of the paper is such that Section 2 summarizes the history of ANN while Sections 3 and 4 review the fundamental concepts and learning schemes of DBN and CA, respectively. Section 5 derives both algorithms' theoretical computational complexity. Finally, Section 6 presents an empirical comparison on classification tasks before concluding with closing remarks in Section 7.

## 2. Artificial neural networks history

Before delving into the deeper network structures presented in this paper, we will go over the evolution of neural networks from their shallow beginnings to the complex structures that have recently become popular.

### 2.1. Concepts from neuroscience

Despite the advances in neuroscience and technology that have allowed for a detailed description of the structure of the brain, the learning process in the brain is yet to be completely understood. Biologically, the brain mainly consists of the cerebrum, the cerebellum, and the brain stem [8].

The cerebral cortex, biologically defined as the outer layer of tissue in the cerebrum and believed to be responsible for higher order functioning, is an association of an estimated 25 billion neurons interconnected through thousands of kilometers of axons propagating and spreading about $10^{14}$ synapses simultaneously [9], arranged in six layers and divided into regions, each performing a specific task [10].

Though it is not very clear how certain areas in the brain become specialized, it is known that multiple factors affect the functional specialization of the brain areas such as structure, connectivity, physiology, development and evolution [11]. Neurons, considered the basic element in the brain, have different shapes and sizes but are all variations of the same underlying scheme, i.e. they start the same general-purpose function but become specialized with training [12]. While dendrites are the site of reception of synaptic inputs, axons convey electrical signals over long distances. Inputs to neurons cause a slow potential change in the state of the neuron; its characteristics are determined by the membrane capacitance and resistance allowing temporal summation [13].

Studies showed that the organization of the cortex can be regarded as an association of columnar units [14,15], each column being a group of nodes sharing the same properties. Learning in the human brain is mainly performed using plastic connections, repeated exposures and firing and inhibition of neurons. In a simplified manner, information flowing in the cortex causes connections in the brain to become active, over time, with repeated exposures these connections are strengthened creating a representation of the information processed in the brain. Moreover, inhibition of neurons - physically defined as prohibiting neurons from firing - partly account for the forgetting process [16].

### 2.2. Shallow beginnings

At a nodal level, ANN started with the simplified McCulloch-Pitts neural model (1943) [17], which was composed of a basic summation unit with a deterministic binary activation function. Successors added complexity with every iteration. At the level of activation functions, linear, sigmoid, and Gaussian functions came into use. Outputs were no longer restricted to real values and extended to the complex domain. Deterministic models gave way to stochastic neurons and spiking neurons which simulated ionic exchanges. All these additions were made to achieve more sophisticated learning models.

At the network level, topologies started out with single layered architectures such as Rosenblatt's perceptron (1957) [18], Widrow and Hoff's ADALINE network (1960) [19] and Aizerman's kernel perceptron (1964) [20]. These architectures suffered from poor performance and could not learn the XOR problem, a simple but non-linear binary classification problem. This led to the introduction of more complex networks starting with the multilayer perceptron (Rumelhart, 1986) [21], self-recurrent Hopfield networks (1986) [22], self-organizing maps (SOM or Kohonen networks, 1986) [23], adaptive resonance theory (ART) networks (1980s) [24] and various others which are considered shallow architectures due to the small number of hidden layers.

Successive iterations incrementally improved on their predecessors' shortcomings and promised higher levels of intelligence, a claim that was made partially feasible due to the hardware's improved computational capabilities [25] and due to the development of faster and more efficient training and learning algorithms. Learning mechanics, whether supervised (back propagation) or unsupervised (feed forward algorithms), matured in parallel and allowed for better performance in a varied set of specific tasks. Nonetheless, the compound effect of the innovation targeting all aspects of these shallow networks was not enough to capture true human intelligence while large computational needs throttled the progress of deeper networks.

### 2.3. Shallow networks' limitations

Supervised learning presents many challenges including the curse of dimensionality [26] where the increase in the number of features and training samples makes learning more computationally demanding. Furthermore, non-linear data is more difficult to divide into classes due to the inherent feature overlap. Unable to position themselves as strong AI models - general intelligent acts as defined by Kurzweil - which can faithfully emulate human intelligence, ANN lagged Support Vector Machines (SVM) [27] in the 1990–2000s.

### 2.4. Deep architectures

The early 2000s saw a resurgence in ANN research due to increased processing power and the introduction of more efficient training algorithms which made training deep architectures feasible. Hinton et al.'s greedy training algorithm [2] simplified the training procedure of Boltzmann machines while deep stacking networks broke down training to the constituting blocks of the deep network to reduce the computational burden. Furthermore, Schmidhuber's long short-term memory architecture [28] allowed the training of deeper recurrent neural networks. While these architectures do not borrow biological properties from the brain beyond the neuron, deep architectures with neural network topologies that adhere more faithfully to neuro-scientific theories of the human brain's topology are gaining traction in the connectionist community due in part to the momentum achieved in computational neuroscience.

One of the major and most relevant contributions in that field was made by Edelman and Mountcastle [7]. Their findings lead to a shift from positioning simplified neuron models as fundamental functional units of an architecture to elevating that role to cortical columns, collections of cells characterized by common feed-forward connections and strong inhibitory inter connections. This provided a biologically feasible mechanism for learning and forming invariant representations of sensory patterns that earlier ANN did not.

Additionally, two supplementary discoveries were believed to be key in emulating human intelligence. The first was the suspected existence of a common computational algorithm in the neocortex [12]. This algorithm is pervasive throughout these regions irrespective of the underlying mental faculty. Whether the task is visual, auditory, olfactory, or other, the brain seems to deal with sensory information in very similar ways. The second was the hierarchical structure of the human neocortex [12]. The brain's regions are hierarchically connected so that the bidirectional flow of information merges into more complex representations with every layer, further abstracting the sensory stimuli.

The combination of these two findings forms potential grounds for building a framework that replicates human intelligence; a hierarchy of biologically inspired functional units that implement a common algorithm. These novel insights from neuroscience have been reflected in the machine learning (ML) and AI fields and have been implemented to varying layers in several algorithms.

While CA restructured the neurons and their connections as well as the learning algorithm [6] based on Edelman and Mountcastle's finding [7], other algorithms modeled other biological theories of the brain's workings. Symbolic architectures such as Adaptive Character of Thought (ACT-R) [29] modeled working memory coupled with centralized control that refers to long term memory when needed. Emergentist architectures such as Hierarchical Temporal Memory (HTM) [30] are based on globalist memory models and use reinforcement or competitive learning schemes to generate their models. Integrating both classes of architectures to form hybrid architectures also exist and include Learning Intelligent Distribution Agent (LIDA) [31].

## 3. Deep belief networks

### 3.1. Overview

DNN are deeper extensions of shallow ANN architectures that are composed of a simplified mathematical model of the biological neuron but do not aim to faithfully model the human brain as do CA or some other ML approaches. DNN are based on the Neocognitron, a biologically inspired image processing model [32], that attempt to realize strong AI models through hierarchical abstraction of knowledge. Information representation is learned as data propagates through the network, shallower layers learn low-level statistical features while deeper layers build on these features to learn more abstract and complex representations. Lacking clear skills for logical inferences, DNN need more morphing to be able to integrate abstract knowledge in a human manner. Recurrent and convolutional neural networks, first introduced in the 1980s, can be considered predecessors of DNN and were trained using back-propagation which has been available since 1974.

ANN were first trained using back-propagation, an algorithm that updates the network weights by propagating the output error backwards through the network [33]. However, the propagated error vanishes to zero as the network depth increases, preventing early-layer weights from updating and significantly reducing the performance of the network [34–37]. Thus, other training algorithms for DNN were investigated. In 1992, Schmidhuber proposed to train recurrent neural networks by pre-training layers in an unsupervised fashion then fine-tuning the network weights using back-propagation [28]. Momentum further picked up in 2006 when Hinton et al. proposed a greedy training algorithm for DBN specifically. In what follows, we restrict our discussion of DNN to DBN, a popular and widely used deep architecture, trained using Hinton et al.'s algorithm.

While DBN is a type of deep ANN, back-propagation fails to produce a suitable model that performs well on training and testing data due to DBN's architectural characteristics [2]. This has been attributed to the "explaining away" phenomenon. Explaining away, also known as Berkson's paradox or selection bias, renders the commonly held assumption of layer independence invalid and consequently adds complexity to the inference process. The hidden nodes become anti-correlated because their extremely low probabilities make the chances of both firing simultaneously impossible.

To remedy this issue, Hinton et al. proposed a training algorithm based on the observation that DBN can be broken down to sequentially stacked restricted Boltzmann machines (RBM), a two-layer network inter-layer neuron connections only. This novel approach rekindled the interest in these deep architectures and saw DBN applied to many problems from image processing [38,39], natural language processing [40–42], automatic speech recognition [43–46] and feature extraction and reduction [47–49], to name a few.

### 3.2. Network structure

#### 3.2.1. Restricted Boltzmann machines

RBM, first known as Harmonium by [50], are two-layer networks where only inter-layer neuron connections are allowed. They are a special case of Boltzmann machines (BM) which allow both inter and intra-layer connections. RBM's neurons form two disjoint sets (as indicated in Fig. 1 by the black boxes), satisfying the definition of bipartite graphs. Thus, training RBM is less complex and faster. The neuron connections in RBM may be directed or undirected; in the latter case, the network forms an auto-associative memory which is characterized by bi-directional information flow due to feedback connections [2].

#### 3.2.2. Deep belief networks

DBN are stacked directed RBMs, except for the first RBM which contains undirected connections, as shown in Fig. 1. This network architecture significantly reduces the training complexity and makes deep learning feasible. Focusing on two layers of the network, the weighted edges connecting the various neurons are annotated using the variable notation $\xi^{\ell}_{n,m}$ which implies that

**Table 1**
DBN nomenclature.

| | |
|---|---|
| $\xi^{\ell}_{n,m}$ | Weight of the edge connecting the $n$th neuron in the $\ell$th layer to the $m$th neuron in the $\ell$th $+1$ layer; $\ell$ is suppressed when there are only 2 layers in the network |
| $\xi^{r}_{n}$ | Vector of connection weights leaving the $n$th neuron in the $\ell$th layer |
| $\xi_{\ell}$ | Matrix of weights connecting the $\ell$th layer to the $\ell$th $+1$ layer |
| $\mu$ | Learning rate |
| $\kappa$ | Number of Gibbs sampling steps performed during CD |
| $N$ | Hidden-layer neuron cardinality |
| $M$ | Input-layer neuron cardinality |
| $L$ | Number of hidden layers |
| $t$ | Sampling step |
| $Q(.\|.)$ | Conditional probability distribution |
| $\mathbf{h}^{\ell}$ | Binary configuration of the $\ell$th layer |
| $p(\mathbf{h}^{\ell})$ | Prior probability of $h^{\ell}$ the current weight values |
| $x^0$ | Input layer data point |
| $x^{(t)}_m$ | Binary configuration of $m$th input-layer neuron at sampling step $t$ |
| $\mathcal{X}$ | Set of training points |
| $\mathcal{H}_n$ | Binary configuration variable of neuron $n$ in the hidden layer at sampling step $t$ |
| $h^{(t)}_n$ | Binary configuration value of neuron $n$ in the hidden layer at sampling step $t$ |
| $\beta_m$ | $m$th input-layer neuron bias |
| $\gamma_n$ | $n$th hidden-layer neuron bias |

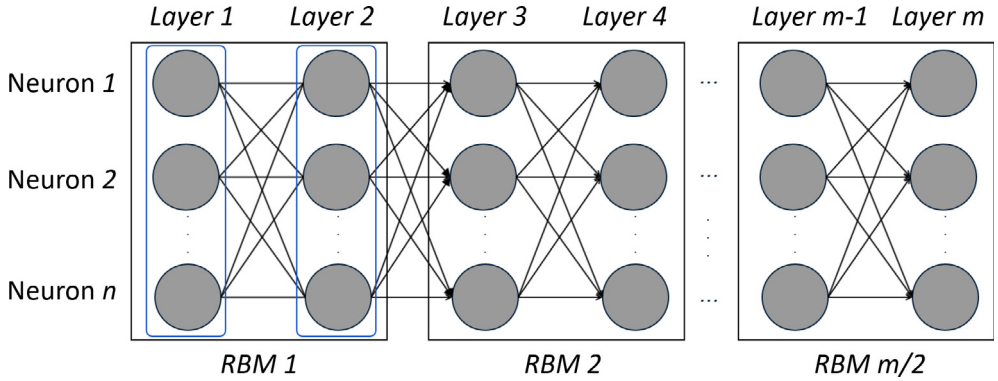


**Fig. 1.** DBN architecture [51].

neuron $n$ in layer $\ell$ is connected to neuron $m$ in layer $\ell+1$. An exhaustive list of the entire nomenclature adopted in this section is included in Table 1.

### 3.3. Training algorithm

#### 3.3.1. Restricted Boltzmann machines

Hinton et al. proposed the contrastive divergence (CD) algorithm to train RBM in both supervised and unsupervised scenarios. CD estimates the log-likelihood gradient using a $\kappa$ Gibbs sampling steps which is typically set to 1. The optimal weight vector is obtained by maximizing the objective function in (1) through gradient descent [52]. CD's pseudo-code is summarized in Table 2.

Gibbs sampling is a randomized MCMC algorithm that allows the sampling of approximate samples from a multivariate probability distribution [53]. The generated samples are correlated and form a Markov chain. Eq. (2) describes the energy function that represents the joint probability distribution, derived from Gibbs distribution and calculated using (3). $\xi_{n,m}$, $\beta_m$ and $\gamma_n$ are real valued weights; $h_n$ and $x_m$ can take values in the set $\{0,1\}$ [54].

$$max_{\xi}\Pi_{x\in X}P(x) \tag{1}$$

$$E(x,h) = -\sum_{n=1}^{N}\sum_{m=1}^{M}\xi_{n,m}h_n x_m - \sum_{m=1}^{M}\beta_m x_m - \sum_{n=1}^{N}\gamma_n h_n \tag{2}$$

$$p(x,h) = \frac{1}{\sum_x\sum_h e^{-E(x,h)}}e^{-E(x,h)} \tag{3}$$

#### 3.3.2. Deep belief networks

A simple and efficient layer-wise training algorithm was proposed for DBN by Hinton et al. in 2006 [2]. It trains the layers sequentially and greedily by tying the weights of unlearned layers, using CD to learn the weights of a single layer and iterating until all layers are trained. Tying the weights not only allows us to use RBM's training algorithm but also eliminates the "explaining away" phenomenon. Then, the network weights are fine-tuned using a two-pass "up-down" algorithm.

In general, deep networks are pre-trained using unsupervised learning before using labeled data to improve the model with supervised learning. This scheme almost always outperforms networks learned without pre-training [56] since this phase acts as a regularizer [57,58] and aid [59] for the supervised optimization problem.

The energy contained in the directed model can be calculated using (4) where the maximum energy is upper bounded by (5) and achieves equality when the network weights are tied. At equality, the derivative is equal to (6) and is used to solve the now simpler maximization problem.

$$E(x^0,h^0) = -(\log p(h^0) + \log p(x^0|h^0)) \tag{4}$$

$$\begin{aligned}\log p(x^0) \geqslant & \sum_{\forall h^0} Q(h^0|x^0)(\log p(h^0) + \log p(x^0|h^0)) \\ & -\sum_{\forall h^0} Q(h^0|x^0)\log Q(h^0|x^0)\end{aligned} \tag{5}$$

$$\frac{\partial \log p(x^0)}{\partial \xi_{n,m}} = \sum_{\forall h^0} Q(h^0|x^0)\log p(h^0) \tag{6}$$

After iteratively learning the weights of the network, the up-down algorithm [2] fine-tunes the network weights. This algorithm is a supervised variant of the wake-sleep algorithm that uses CD to modify the network weights. The wake-sleep algorithm [60] is an unsupervised algorithm used to train neural networks in two phases: the "wake" phase is applied on the feed-forward path to compute the weights and the "sleep" phase is applied on the feed-back path. The up-down algorithm, described in Table 3, is applied to network to reduce under-fitting which is commonly observed in greedily-trained networks.

Specifically, in the first phase (up-pass) of the algorithm, the weights on the directed connections, termed generative weights or parameters, are modified by calculating the wake-phase probabilities, sampling the states, and updating the weights using CD. On the other hand, the second phase (down-pass) stochastically activates earlier layers through the top-down links, termed inference weights or parameters. The sleep-phase probabilities are calculated, the states are sampled and the output is estimated.

## 4. Cortical algorithms

### 4.1. Overview

CA are a deep artificial neural network model, which borrows several concepts and aspects from the human brain. The main inspiration is drawn from the findings of Edelman and Mountcastle [15,7], which state that the brain is composed of cortical columns arranged in six layers. He also uses the concept of strengthening and inhibiting to build a computational training algorithm capable of extracting meaningful information from the sensory input and creating invariant representations of patterns. Further description of the CA model and its biologically plausible aspects can be found in [61,51,62].

### 4.2. Network structure

A typical CA network consists of an association of columns grouped in layers or levels. A column is a collection of neurons associated with the same stimulus, as shown in Fig. 2. Hence, CA can be considered as a three-level hierarchy structure. The neurons, like in other neural network architectures, use an activation

**Table 2**
Contrastive divergence workflow used in training RBM [55].

1. Set the weights to zero: $\xi_\ell = \mathbf{0}, \ell = 1\ldots L$
2. $\forall x \in \mathcal{X}$
   a. Propagate training instance through the network
   b. For $\kappa$ sampling steps
      i. Loop over $N$ hidden-layer neurons and sample $h_n^{(t)} \sim p(h_n|x^{(t)})$
      ii. Loop over $M$ input-layer neurons and sample $x_m^{(t)} \sim p(x_m|h^{(t)})$
   c. Loop over input and hidden-layer neurons and compute
      i. $\Delta\xi_{n,m} = \Delta\xi_{n,m} + p(\mathcal{H}_n = 1|x^{(0)})x_m^{(0)} - p(\mathcal{H}_n = 1|x^{(\kappa)})x_m^{(\kappa)}$
      ii. $\Delta\beta_m = \Delta\beta_m + x_m^{(0)} - x_m^{(\kappa)}$
      iii. $\Delta\gamma_n = \Delta\gamma_n + p(\mathcal{H}_n = 1|x^{(0)}) - p(\mathcal{H}_n = 1|x^{(\kappa)})$

**Table 3**
Up-Down training algorithm workflow [2].

1. Bottom-up phase
   a. Calculate wake-phase probabilities
   b. Sample network states
   c. Using wake-phase probabilities, calculate CD statistics
   d. Perform Gibbs sampling for $\kappa$ iterations
   e. Using (1.d), calculate sleep-phase CD statistics
2. Down-pass phase
   a. Compute sleep-phase probabilities through top-down pass
   b. Sample network states
   c. Estimate network output
3. Re-compute generative weights
4. Re-compute network's directed sub-graph weights
5. Re-compute inference weights

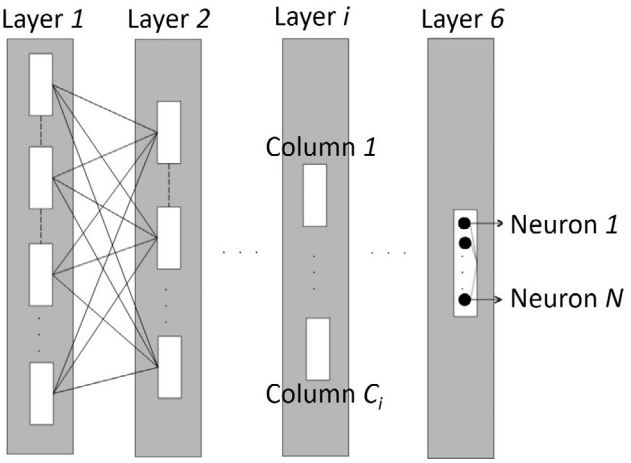


**Fig. 2.** Illustration of a cortical network.

function $f(.)$ to compute their output from their input. This activation function is common for all neurons in the architecture.

Columns in a layer connect to those in the subsequent level: these connections are referred to as vertical connections and exist only between consecutive levels. A synaptic input terminating at a column's input is shared within neurons constituting said column, however, the learned weights of the connections are different leading to distinct neuronal outputs. The latter are summed to form the column's output being forwarded to the next layer. Such configuration permits the column to act as a basic computational structure in CA as opposed to neurons in DBN and other neural network architectures.

Furthermore, lateral connections or intra-level connections, are "communication means" employed to deliver inhibiting signals that modify a column's weights based on the activity of other columns during training. Contrarily to their vertical counterpart, these connections do not transmit data and are hence are not explicitly shown in Fig. 2. Data can only flow in a bottom-up direction, from level to level, through vertical connections.

### 4.3. Mathematical model

The complete mathematical description based on the model of [6,63] can be found in [62,61], and is summarized below based on the adopted nomenclature in Table 4.

A column of $N$ neurons receives connections from the output of columns in the previous layer and hence is represented by a 2D matrix concatenating vectors of weights of incoming connections synapsing at each of its neurons as shown in Eq. (7). Epoch number is denoted by $e$, layer number by $l$, the receiving column index by $c$, the receiving neuron $n$ and the sending column by $s$.

Defining $\Upsilon^{l,e}$ as the output vector of level $l$ for epoch $e$ and $v_c^{l,e}$ the output of column $c$, within the same level; for the same training epoch, we can write (8). The output of a neuron, $z_{c,n}^{l,e}$ defined by (9) is the result of the nonlinear activation function $f(.)$ in (10) in response to the weighted sum of the input connections while the output of the column is the sum of the outputs of the column's neurons. $T$ is a constant (across layers) tolerance parameter empirically selected and the nonlinear activation function emulates the brain's observed nonlinear activity.

$$\underline{\Xi_c^{l,e}} = \begin{bmatrix} \Xi_{c,1}^{l,e} & \cdots & \Xi_{c,n}^{l,e} & \cdots & \Xi_{c,N}^{l,e} \end{bmatrix} = \begin{bmatrix} \xi_{c,1,1}^{l,e} & \cdots & \xi_{c,n,1}^{l,e} & \cdots & \xi_{c,N,1}^{l,e} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \xi_{c,1,s}^{l,e} & \cdots & \xi_{c,n,s}^{l,e} & \cdots & \xi_{c,N,s}^{l,e} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \xi_{c,1,C_{l-1}}^{l,e} & \cdots & \xi_{c,n,C_{l-1}}^{l,e} & \cdots & \xi_{c,N,C_{l-1}}^{l,e} \end{bmatrix} \tag{7}$$

$$\Upsilon^{l,e} = \begin{bmatrix} v_1^{l,e} & \cdots & v_c^{l,e} & \cdots & v_{C_l}^{l,e} \end{bmatrix} \tag{8}$$

$$v_c^{l,e} = \sum_{n=1}^{N} z_{c,n}^{l,e}$$
$$z_{c,n}^{l,e} = f\left( \sum_{c=1}^{C_{l-1}} \xi_{c,n,s}^{l,e} v_c^{l-1,e} \right) \tag{9}$$

$$f(\tau) = \frac{1}{1 + e^{\tau \cdot (\phi(\tau) - T)}}$$
$$\phi(\tau) = \begin{cases} -2, & \text{if } \tau = 1 \\ \tau, & \text{otherwise} \end{cases} \tag{10}$$

**Table 4**
CA nomenclature.

| | |
|---|---|
| $c$ | Destination column index |
| $n$ | Destination neuron index |
| $l$ | Destination level index |
| $s$ | Origin column index |
| $e$ | Training epoch |
| $N$ | Number of nodes in a column |
| $C_l$ | Number of columns in level $l$ |
| $T$ | Tolerance |
| $\xi_{c,n,s}^{l,e}$ | Weight of connection between neuron $n$, column $c$, level $l$, and column $s$ in previous level, during epoch $e$ |
| $\Xi_{c,n}^{l,e}$ | Vector of connection weights entering neuron $n$, of column $c$, level $l$ |
| $\underline{\Xi_c^{l,e}}$ | Matrix of weights entering column $c$, level $l$ |
| $\Upsilon^{l,e}$ | Output vector of level $l$ |
| $v_c^{l,e}$ | Output of column $c$ |
| $z_{c,n}^{l,e}$ | Output of neuron $n$, column $c$ of level $l$ |

### 4.4. Training algorithm

#### 4.4.1. Random initialization

The network is assumed to be initially fully connected with random weak weights (with absolute values less than 0.1). This is a common "blank slate" approach to ensure that the network is not initially biased to any specific pattern. As learning proceeds, these weights are incrementally updated so that the connectivity of the network is modified. All weights that fall to zero represent disabled connections. Therefore, an initially fully connected network is not necessarily preserved.

#### 4.4.2. Unsupervised feed-forward learning

The first stage in training a cortical network aims at creating input-specific representations via random firing and repeated exposure. An input propagating through the levels of the network causes certain columns to fire (i.e. generate a threshold crossing response) based on initially random weights. This activation is then reinforced by strengthening the active columns' weights and inhibiting neighboring ones. Repeated exposure or batch learning trains columns to identify (via activation) particular aspects or patterns of the training data, extracting discriminatory features with increasing complexity through levels (lower levels recognize basic elements, higher levels in the hierarchy learn higher order concepts and reasoning). The strengthening process increases a column's weights rendering it more receptive to stimulation, while inhibition weakens the weights diminishing activity of a certain column. Connections formed or weakened

are plastic, i.e. can be altered during the feedback learning phase and vice versa. Strengthening and inhibition rules are shown in Eqs. (11)–(13).

$$\xi_{c,n,s}^{l,e+1} = \upsilon_c^{l,e} \cdot (\xi_{c,n,s}^{l,e} - \Omega(\Xi_c^{l,e})) \tag{11}$$

$$\xi_{c,n,s}^{l,e+1} = \upsilon_c^{l,e} \cdot \left( \xi_{c,n,s}^{l,e} + \alpha_{c,n,s}^{l,e} + \rho \cdot \frac{1}{1 + e^{\frac{\xi_{c,n,s}^{l,e} - T}{\Omega(\Xi_c^{l,e})}}} \right) \tag{12}$$

$$\begin{aligned} \Omega(\Xi_c^{l,e}) &= \sum_{n=1}^{N} \sum_{c=1}^{C_{l-1}} \alpha_{c,n,s}^{l,e} \xi_{c,n,s}^{l,e} \\ \alpha_{c,n,s}^{l,e} &= \begin{cases} 1 & \text{if } \xi_{c,n,s}^{l,e} > \epsilon \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{13}$$

*4.4.3. Supervised feedback learning*

The feed-forward learning phase relies only on aspects of the given data to train columns that identify significant features with no error propagation or label information. The supervised feedforward stage aims at correcting misclassifications occurring when the network is exposed to variations of the same patterns (more accurately training class) which may result due to the absence of label information in the previous phase.

Following the unsupervised phase, a "unique representation" based on the average firing scheme observed for a particular pattern is stored, the feedforward learning fine-tunes the network's weights to achieve this scheme (within certain bounds to avoid overfitting) for all instances of a known class. A misclassification hence triggers an error signal at the top-most or output layer (where the final output is produced) correcting misfiring columns (through inhibition) and strengthening weakened columns firing for the original pattern forcing the column firing for the original pattern (strengthening and inhibition are the only weight update rules adopted in opposition to gradient descent employed in the backpropagation learning of traditional artificial neural networks). After multiple exposures, the top level attains a stable state also known as a "stable activation" in which columns are able to correctly generate the desired firing scheme. The term "firing scheme" refers to the firing pattern of neurons for a given input.

After stabilizing the top layer, the error signal is propagated back to the previous level which in turn executes a series of inhibition and strengthening to achieve the desired firing scheme. The same process in repeated for each of the layers until a convergence condition (expressed as a discrepancy between the desired and actual activation) is met. The training exits when all layers are stable and the network can recognize all variations of the same pattern in the training data (within a certain tolerance).

A pseudo-code showing the implementation of the training algorithm is shown in Table 5. An illustration of the feedback training process is shown in Fig. 3. The top row shows the firing scheme (blue squares represent active columns) obtained after the feedforward propagation of two variations of the same pattern as well as the desired firing scheme (average activation obtained based on all variations of this pattern in the training set). The middle and bottom rows show a succession of training epochs (for pattern 1 and 2 respectively) in which the error signal is generated at the top level and stable activations are formed from top to bottom. One can see that at convergence both instances are represented with the same firing scheme in the network.

## 5. Theoretical computational complexity

In this section, we derive the theoretical computational complexity of each algorithm to assess the required resources when deploying such a network for real world problems. We investigate

**Table 5**
CA generic training workflow.

1. Random initialization
   for $l = 1 : 6$
      for $c = 1 : C_l$
         $\underline{\Xi_c^l} = randn(C_{l-1}, N)$
2. Feedforward phase
   a. Processing of training data
      for all training instances
         for $l = 1 : 6$
            compute $\Upsilon^{l,e}$
         for $l = 1 : 6$
            for $c = 1 : C_l$
               if $\upsilon_c^{l,e} > \epsilon$
                  for $n = 1 : N$ apply (12)
               else
                  for $n = 1 : N$ apply (11)
   b. Store average representations of patterns
   for all classes
      for $l = 1 : 6$
         for $c = 1 : C_l$
            $\upsilon_c^{l,avg} = \overline{\upsilon_c^l}$
3. Feedback phase
   while $MSE > \theta$
      for all training instances
         for $l = 1 : 6$
            while $\left| \overline{\Upsilon^{l,e}} - \Upsilon^{l,e} \right| > \theta$
               for $c = 1 : C_l$
                  if $\left| \upsilon_c^{l,avg} - \overline{\upsilon_c^l} \right| > \theta$
                     for $n = 1 : N$ apply (11)
                  else
                     for $n = 1 : N$ apply (12)

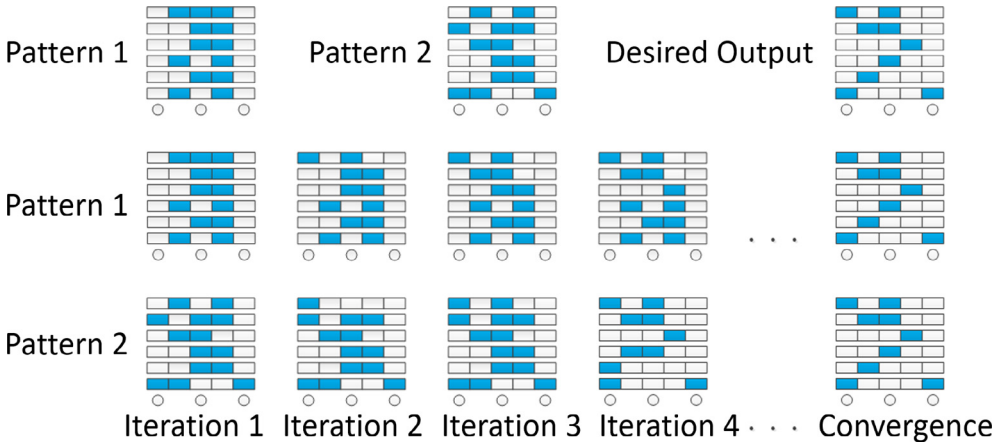


**Fig. 3.** An example of the feedback process.

two aspects of the computational complexity: memory and computations. While the required memory storage depends on the number of non-zero weights in the network, the number of computations depends on the non-zero weights and the adopted activation function. Comparing CA to DBN, the more computationally demanding network is data specific since each problem would result in a different number of non-zero weights. We empirically compare the network sizes in the next section.

### 5.1. Number of non-zero weights

#### 5.1.1. DBN

DBN is formed of $R$ layers with $M_r$ neurons in layer $r$. During training, the network starts out fully connected leading to a total number of weights equal to $N_w = \sum_{r=1}^{R} M_r \cdot M_{r-1}$. Assuming not all weights are non-zero when training is complete, with $\gamma \in [0, 1]$ is the fraction of non-zero weights or firing rate, the number of non-zero weights is equal to $N_{NZW} = \sum_{r=1}^{R} \gamma \cdot M_r \cdot M_{r-1}$. Empirical results in Section 6 reveal that $\gamma$ is usually greater than 90%. Knowing that the weights are double precision floating point numbers which require 8 bytes of storage, an upper bound on the memory requirements during training are $\sum_{r=1}^{R} 8 \cdot M_r \cdot M_{r-1}$. During testing, the number of bytes is equal to $\sum_{r=1}^{R} 8 \cdot \gamma \cdot M_r \cdot M_{r-1}$.

#### 5.1.2. CA

CA is formed of $R = 6$ levels; the first level contains $L_1 = I$ columns with M neurons per column. We assume, without loss of generality, the number of columns is cut in half in each subsequent level, as used in [6,63,64] i.e. level $r$ contains $L_r = \frac{I}{2^{r-1}}$. Each level has a weight matrix with dimensions $L_r \times M$. Therefore, the total number of weights is equal to $N_w = \sum_{r=1}^{R} L_r \cdot M \cdot L_{r-1}$. However, some of these weights could be equal to zero. Therefore, we denote by $\gamma \in [0, 1]$ the firing rate of the network, the fraction of neurons that are non-zero, which leads to the number of firing neurons equal to $N_{FC} = \gamma \sum_{r=1}^{R} L_r$. To simplify the computations, a uniform distribution of firing columns across levels will be assumed. Therefore, the number of firing columns per level is $N_{FC/L} = \frac{N_{FC}}{R}$. Finally, the total number of non-zero weights can be approximated by $N_{NZW} = \sum_{r=1}^{R} M \cdot L_r \cdot N_{FC/L}$. Empirical results in Section 6 reveal that $\gamma$ does not usually exceed 50%. Knowing that the weights are double precision floating point numbers which require 8 bytes of storage, an upper bound on the memory requirements during training are $\sum_{r=1}^{R} 8 \cdot L_r \cdot M \cdot L_{r-1}$. During testing, the number of bytes is equal to $\sum_{r=1}^{R} 8 \cdot \gamma \cdot M \cdot L_r \cdot \sum_{r=1}^{R} \frac{L_r}{R}$.

### 5.2. Number of operations per neuron

Since a neuron is composed of a summation and an activation function, the number of operations performed to obtain the output of each neuron can be divided into the operations to compute the sum and the activation function computational complexity.

#### 5.2.1. Summation

The number of floating point operations required to compute the summation depends on the number of input connections of a neuron. Assuming there are $m$ connections, $m$ multiplications and $m$ additions (including the bias term) are required, which is of the order of $O(m^2)$. For an input layer neuron, $m$ is equal to the number of features in the input vector. For a hidden layer neuron, $m$ is at most equal to the number of neurons in the previous layer.

#### 5.2.2. Activation function

Depending on the activation function, the computation of each neuron output will require a certain number of floating point and comparison operations. Some of popular activation functions and their computational complexity are summarized in Table 6.

The hard limit activation function, described by (14), requires one comparison. The linear function in (15) requires one multiplication and one addition whereas the piece-wise linear function in (16) requires two comparisons and at most one multiplication and addition operation.

$$\phi(x_i) = \begin{cases} 1 & \text{if } x_i \geqslant 0 \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

$$\phi(x_i) = \alpha x_i + \beta \tag{15}$$

$$\phi(x_i) = \begin{cases} \beta & \text{if } x_i \geqslant \beta \\ \alpha x_i + \beta & \text{if } -\beta \leqslant x_i < \beta \\ -\beta & \text{if } x_i < -\beta \end{cases} \tag{16}$$

**Table 6**
Computational complexity of some activation functions

| Activation function | Equation | Computations (per neuron) |
|---|---|---|
| Hard limit | (14) | $O(1)$ |
| Linear | (15) | $O(2)$ |
| Piecewise linear | (16) | $O(4)$ |
| Gaussian | (17) | $O(m^3)$ |
| Sigmoid, Tangent | (18) | $O(165)$ |
| Sigmoid, Logarithm | (19) | $O(83)$ |
| Softmax | (20) | $O(83M_R)$ |

Computing the Gaussian activation function, described by (17), requires $m$ multiplications and $m(m-1)$ additions to compute the magnitude term, where $m$ represents the dimensionality of the vector $x_i$ and is equivalent to the number of input connections of a neuron. Dividing by the standard deviation requires 3 additional multiplication operations. In addition, the calculation of an exponential, estimated using the Taylor series expansion with approximately 10 terms, requires approximately 81 operations. In total, $m^2(m-1)+81$ operations are required.

$$\phi(x_i) = \exp\left(\frac{-\|x_i-\mu_i\|^2}{2\sigma^2}\right) \tag{17}$$

The tangent sigmoid function in (18) has two exponential terms, each requiring approximately 81 operations, in addition to two additions and one multiplication operations which leads to a total of approximately 165 operations. On the other hand, the logarithmic sigmoid in (19) has one exponential term and one addition and multiplication for a total of 83 operations.

$$\phi(x_i) = \tanh(x_i) = \frac{1-e^{-2x_i}}{1+e^{-2x_i}} \tag{18}$$

$$\phi(x_i) = \frac{1}{1+e^{-x_i}} \tag{19}$$

Each exponential term in the softmax activation function, described by (20), requires approximately 81 operations. The numerator has one exponential term while the denominator has $M_R$ terms. In total, $81(M_R+1)+1$ operations are required.

$$\phi(x_i) = \frac{e^{x_i}}{\sum_{j\in M_R} e^{x_j}} \tag{20}$$

### 5.3. Pruning

The term synaptic pruning refers to the procedure by which synaptic connections are terminated during the early ages of mammals' lives [65]. Starting with approximately $86\pm8$ billion neurons at birth, the human brain quintuples its size until adolescence after which the volume of synaptic connection decreases again [66]. This process of pruning is mainly thought of as the byproduct of learning. While the total number of neurons remains roughly unaltered, the distribution and number of connections are tailored by learning [67,68]: the structure of the brain moves from a stage where its primary function falls under perception–action due to disconnected cortical hubs [69,70] to distributed networks spanning the brain performing a variety of complex cognitive tasks [71].

The training of a cortical network bears several resemblances to the synaptic pruning procedure: starting with a fully connected structure, the network goes through an unsupervised phase where connections are pruned through inhibition leaving only "significant" ones. In more accurate terms, the strengthening of firing columns ensures the establishment of selective connections while inhibition prunes irrelevant connections. This process plays a crucial role not only in extracting meaningful characteristics of the input stimuli but also in avoiding overfitting due to the especially complex structure of a cortical network. This is further demonstrated in the number of non-zero weights in different network architectures as shown in our theoretical and empirical analysis: the number of "alive" connections decreases with the increase of parameters, hence a minimal effect on performance as shown in our experiments. This property of CA's structure and training is not shared with DBN.

### 5.4. Overall computational complexity

In summary, the overall computational cost of DBN and CA depends on the architecture of the network: the number of layers and the number of neurons per layer which affect the number of connections. The activation function can be fixed for both DBN and CA. Sigmoid is a common activation function used in both algorithms. After training, some of these connections will have a weight of zero. Based on the previous sections, we notice that for a fixed network architecture, CA will have less non-zero weights due to the pruning algorithm. However, the depth of the best networks for each algorithm vary based on the data. six-layer architectures are commonly used in the literature for CA while DBN depth's varied from 3 to 5 hidden layers.

## 6. Empirical comparison

In this section, we report on the experimental results of DBN and CA on multiple databases from the UCI ML repository [72]. We compare the classification accuracy, network complexity and computational complexity of both algorithms.

### 6.1. Experimental setup

CA and DBN were empirically compared on classification problems using datasets from the UCI machine learning repository, described in Table 7. The datasets contained real-valued features. A 4-fold cross validation was adopted, i.e. each database is randomly divided into 4 sets where 3 sets (or 75% of the data samples) are used in training and 1 set is used in testing. The results are averaged over four runs where each set is used for testing once and the remaining for training. The CA library is a set of Matlab functions obtained from [73,74]; it was run on a Windows 7 machine with Intel Core i5. The DBN library is a set of Matlab functions modified from [75]; it was run on an Intel Core i7 processor machine. Multiple network architectures, summarized in Table 8, were tested on the datasets. The number of neurons for the hidden layers are displayed only. The input layer's neurons are equal to the number of features and the output layer's neurons are equal to the number of classes. The chosen architectures can be grouped into

**Table 7**
UCI dataset characteristics.

| Dataset | Number of instances | Number of features | Number of classes |
|---|---|---|---|
| Planning relax | 182 | 12 | 2 |
| Breast cancer diagnostic Wisconsin | 569 | 32 | 2 |
| Tic Tac Toe | 958 | 9 | 2 |
| Spambase | 4601 | 57 | 2 |
| Wilt | 4889 | 5 | 2 |
| White wine | 4898 | 11 | 2 |
| MNIST | 70,000 | 784 | 10 |
| Skin segmentation | 245,057 | 3 | 2 |

**Table 8**
Nomenclature adopted for network architectures.

| Network name | Network architecture (hidden layers) |
|---|---|
| N1 | [5, 5] |
| N2 | [50, 50] |
| N3 | [50, 50, 50] |
| N4 | [500, 500, 500] |
| N5 | [500, 500, 2000] |
| N6 | [1000, 1000, 2000] |
| N7 | [2000, 1000, 500, 250, 125, 62] |

one of three sets based on the pattern of hidden layer neurons: networks with an equal number of neurons in all hidden layers, networks with an increasing (doubling) number of neurons as the layer depth increases and networks with a decreasing (halving) number of neurons as the layer depth increases. Furthermore, the number of hidden layers is increased from 2 to 6 hidden layers. For the results reported in Table 9, DBN's unsupervised training and fine-tuning algorithms were each run for 50 epochs and the batch size was set to 10. The second column indicates the number of columns per layer for CA (there were 20 neurons per column) and the number of neurons per column for DBN. The connectivity of a network is computed by taking the ratio of weights greater than 5% of the average value of weights to the total number of weights in the network. The threshold is not set to a fixed value since the weights range of weights varies based on the input data, i.e. for some datasets all the weights might be less than this set threshold even though this threshold might be very small. Furthermore, the threshold is not set to zero since some weights will not exactly zero but significantly smaller than the other weights in the network and their contribution is insignificant. The classification

**Table 9**
Classification results.

| | | DBN | | CA | |
|---|---|---|---|---|---|
| Dataset | Net. size | Acc. (%) | Connec. (%) | Acc. (%) | Connec. (%) |
| Planning relax | N1 | 69.4 | 91.1 | 71.4 | 86.4 |
| | N2 | 68.8 | 49.6 | 65.2 | 72.5 |
| | N3 | 69.4 | 55.0 | 70.8 | 68.7 |
| | N4 | **70.6** | 9.1 | 75.6 | 55.1 |
| | N5 | 68.8 | 3.8 | 80.8 | 43.4 |
| | N6 | 68.1 | 3.2 | 82.5 | 30.9 |
| | N7 | 66.3 | 6.0 | **86.1** | 29.1 |
| Breast cancer diagnostic Wisconsin | N1 | 62.1 | 99.1 | 97.0 | 95.8 |
| | N2 | **88.4** | 98.7 | 97.1 | 90.7 |
| | N3 | 88.4 | 98.5 | 96.8 | 86.4 |
| | N4 | 87.5 | 97.3 | 97.5 | 77.1 |
| | N5 | 87.5 | 97.3 | 98.2 | 68.3 |
| | N6 | 87.0 | 97.2 | 98.2 | 56.5 |
| | N7 | 67.9 | 96.9 | **99.0** | 35.6 |
| Tic tac toe | N1 | 65.1 | 100.0 | 67.0 | 86.3 |
| | N2 | 65.1 | 94.9 | 79.3 | 83.5 |
| | N3 | 65.1 | 97.0 | 80.1 | 79.8 |
| | N4 | 65.1 | 99.4 | 89.4 | 68.1 |
| | N5 | 65.1 | 99.4 | 90.5 | 56.4 |
| | N6 | 65.1 | 98.9 | 92.7 | 45.9 |
| | N7 | 65.1 | 99.1 | **93.1** | 34.7 |
| Spambase | N1 | 89.9 | 98.1 | 90.5 | 89.7 |
| | N2 | 90.5 | 98.7 | 91.8 | 87.6 |
| | N3 | 91.4 | 98.9 | 92.8 | 79.9 |
| | N4 | **92.2** | 96.7 | 93.7 | 66.4 |
| | N5 | 91.3 | 96.1 | 94.3 | 58.1 |
| | N6 | 91.8 | 95.3 | 95.6 | 46.8 |
| | N7 | 88.5 | 93.5 | **97.5** | 36.3 |
| Wilt | N1 | 94.6 | 100.0 | 94.6 | 88.4 |
| | N2 | 94.6 | 99.9 | 95.5 | 80.7 |
| | N3 | 94.6 | 99.9 | 96.5 | 77.5 |
| | N4 | 94.6 | 99.9 | 97.8 | 66.9 |
| | N5 | 94.6 | 99.4 | **98.2** | 55.3 |
| | N6 | 94.6 | 99.2 | 98.0 | 45.5 |
| | N7 | 94.6 | 97.8 | **98.2** | 35.8 |
| White wine | N1 | **96.4** | 94.1 | 95.1 | 81.7 |
| | N2 | 96.4 | 99.7 | 96.0 | 80.3 |
| | N3 | 96.4 | 98.6 | 97.5 | 73.2 |
| | N4 | 96.4 | 70.1 | 98.1 | 67.9 |
| | N5 | 96.1 | 73.0 | 98.9 | 47.3 |
| | N6 | 96.3 | 60.0 | 99.5 | 36.8 |
| | N7 | 96.4 | 62.8 | **99.7** | 26.1 |
| MNIST | N1 | 38.8 | 88.1 | 96.2 | 89.8 |
| | N2 | 84.1 | 91.8 | 97.1 | 85.2 |
| | N3 | 84.2 | 98.8 | 98.5 | 79.1 |
| | N4 | 89.0 | 47.2 | 98.5 | 58.7 |
| | N5 | 88.3 | 54.9 | 99.2 | 43.5 |
| | N6 | **89.4** | 53.9 | 99.8 | 37.6 |
| | N7 | 86.5 | 40.8 | **99.8** | 28.5 |
| Skin segmentation | N1 | 79.6 | 80.6 | 94.5 | 83.2 |
| | N2 | 79.6 | 94.4 | 94.8 | 79.5 |
| | N3 | 79.6 | 95.8 | 94.8 | 75.3 |
| | N4 | 79.6 | 97.2 | 95.7 | 64.3 |
| | N5 | 79.6 | 97.1 | 97.8 | 53.8 |
| | N6 | 79.6 | 97.1 | 97.8 | 50.1 |
| | N7 | 79.6 | 97.3 | **99.9** | 33.5 |

The bold values refer to the best accuracy obtained per dataset.

accuracy simply reports the percentage of correctly classified instances in the test set. Running time is not compared since each algorithm is written and run in a different environment.

### 6.2. Classification results

First, we compare the performance of DBN when varying the network architecture. For some databases, such as White wine, the performance does not significantly vary as the network architecture varies, achieving approximately 96.4% accuracy. On the other hand, performance varied significantly across architectures for MNIST, ranging from 38.8% to 89.47%. Hence, DBN is either able to achieve good accuracy or bad accuracy on a specific database. This is mainly due to the lack of enough training points to allow DBN to learn a good model of the data. However, for other datasets such as Breast cancer diagnostic Wisconsin, the classification accuracy varied significantly as the network architecture varied. Next, considering CA's performance on the various datasets when varying the network size, we notice that the six-layered network always outperforms other network sizes on all datasets, although it occasionally marginally outperforms some networks. For example, CA achieved 99.9% accuracy for N7 on skin segmentation compared to 94.5% for N1 (2 layers). Comparing CA to DBN, we notice that CA outperforms DBN on almost all datasets. In addition, a six-layered architecture seemed to always perform better than other architectures, reducing the burden of searching for the optimal network architecture by training multiple networks.

### 6.3. Network connectivity

DBN exhibit high connectivity for most datasets on various network sizes. Therefore, Hinton et al.'s greedy training algorithm does not eliminate a large number of connections resulting in an almost fully connected graph. This is evident in the experimental results reported in Table 9 which reports the percentage of non-zero weights in a DBN network to be around 90%. Unlike DBN, CA generally results in a sparsely connected network, evident by the low percentage of non-zero weights obtained after training; CA networks had less than 50% of their connections in tact compared to almost 90% for DBN. Furthermore, we notice that CA's connectivity tends to decrease as the network size increases which implies that if certain data does not require a large network, CA's training algorithm will reduce the number of connections to produce a sparsely connected network. For example, connectivity percentage decreases from 98.7% to 40.8% for MNIST when increasing the network architecture from a two-hidden layer network to a six-layer architecture.

### 6.4. Effect of batch size

The DBN code randomly reorders and divides the data into mini batches. While training, each mini batch is loaded into memory and used to update the weights of the network connections. Increasing the size of a batch meant loading a larger chunk of data into memory. This renders training slower if the chunk was too large to fit into memory. However, decreasing the batch size meant more memory transfers per epoch and therefore, increasing training time. On the other hand, CA does not randomly reorder and divide the data into mini batches. Therefore, the network is exposed to the data in the order it was presented. Theoretically, reordering or dividing the data has no effect on the performance of the algorithm as all patterns are employed and must achieve a stable activation.

### 6.5. Statistical analysis

Next, we perform the pairwise *t*-test [76] and Friedman test [77] to gain further insight into the differences between CA and DBN. The N7 architecture is trained on the various databases. Table 10 summarizes the p-values of both tests and show that

**Table 10**
Statistical significance tests.

| | Pairwise *t*-test | | Friedman test | | |
|---|---|---|---|---|---|
| Database | p-value | Statistically significant | p-value | Statistically significant | Nemenyi critical distance |
| Planning relax | 2.88E−4 | Y | 3.64E−02 | Y | 0.1 |
| Breast cancer diagnostic Wisconsin | 0 | Y | 9.26E−126 | Y | 0.1 |
| Tic tac toe | 5.28E−10 | Y | 1.92E−09 | Y | 0.1 |
| Spambase | 0 | Y | 0 | Y | 0.0 |
| Wilt | 4.65E−79 | Y | 4.74E−75 | Y | 0.0 |
| White wine | 0 | Y | 0 | Y | 0.0 |
| MNIST | 9.11E−84 | Y | 6.09E−61 | Y | 0.0 |
| Skin segmentation | 0 | Y | 0 | Y | 0.0 |

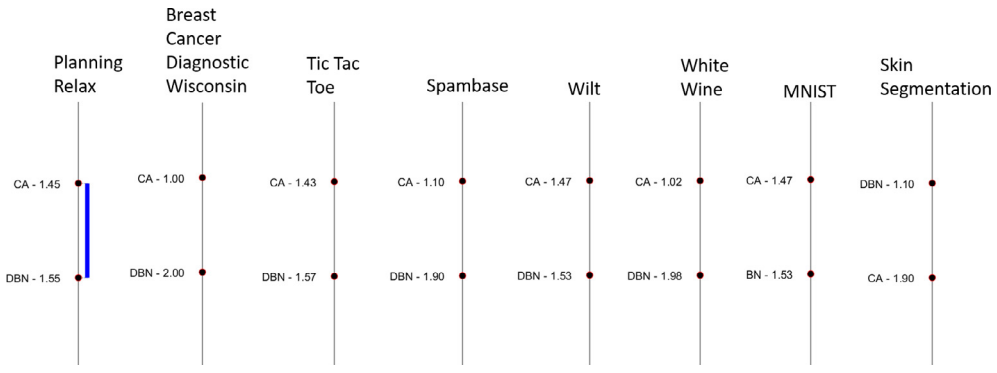


**Fig. 4.** CA vs. DBN ranking based on Nemenyi post hoc test.

there is a statistical significance between CA and DBN on all databases; the p-value was less than 5%. Furthermore, the Nemenyi post hoc [78] was performed once a significant difference was observed, to rank the algorithms. The rankings revealed that CA outperformed DBN on most databases except for Skin Segmentation, as shown in Fig. 4. The critical distance is summarized in Table 10.

## 7. Conclusion

In this work, we compared two DNN architectures on supervised classification problems. While DBN can be easily seen as an established technique developed from within a traditional AI perspective, CA are more biologically inspired and can be classified as theories in the making, solidly rooted in principles inherited from neuroscience research. A theoretical computational complexity analysis for both algorithms was presented before empirically comparing CA and DBN. Experiments were run on eight publicly available classification databases. Multiple CA and DBN network architectures were compared based on their classification accuracy and resulting network connectivity. CA achieved the best performance on most databases using a six-layer architecture with decreasing number of hidden neurons for deeper layers. Results showed that deeper CA networks had lower connectivity than shallower CA networks. Furthermore, DBN did not prune as many connections as CA's training algorithm. On the tested databases, CA generally had a higher classification accuracy than DBN. In the span of this work, we attempted to provide the reader with enough background and technical details for each of the algorithms while understanding that the breadth of the topic necessitates the inclusion of more involved insights. We therefore urge the interested reader to use this paper as a foundation for further exploration of the rapidly expanding sub-field of deep learning.

## Acknowledgment

This work has been partly supported by the National Center for Scientific Research in Lebanon and the University Research Board at the American University of Beirut.

## References

[1] S. Samarasinghe, Neural Networks for Applied Sciences and Engineering: From Fundamentals to Complex Pattern Recognition, CRC Press, 2006.
[2] G. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural Comput. 18 (7) (2006) 1527–1554.
[3] E.M. Izhikevich, Which model to use for cortical spiking neurons?, IEEE Trans Neural Netw. 15 (5) (2004) 1063–1070.
[4] H. De Garis, C. Shuo, B. Goertzel, L. Ruiting, A world survey of artificial brain projects, part i: large-scale brain simulations, Neurocomputing 74 (1) (2010) 3–29.
[5] B. Goertzel, R. Lian, I. Arel, H. De Garis, S. Chen, A world survey of artificial brain projects, part ii: biologically inspired cognitive architectures, Neurocomputing 74 (1) (2010) 30–49.
[6] A.G. Hashmi, M.H. Lipasti, Cortical columns: building blocks for intelligent systems, in: IEEE Symposium on Computational Intelligence for Multimedia Signal and Vision Processing, IEEE, 2009, pp. 21–28.
[7] G.M. Edelman, V.B. Mountcastle, in: The Mindful Brain: Cortical Organization and the Group-Selective Theory of Higher Brain Function, Masachusetts Inst of Technology Pr, 1978.
[8] R.J. Baron, The Cerebral Computer: An Introduction to the Computational Structure of the Human Brain, Psychology Press, 2013.
[9] J. Nolte, The Human Brain: An Introduction to its Functional Anatomy.
[10] J.M. DeSesso, Functional anatomy of the brain, in: Metabolic Encephalopathy, Springer, 2009, pp. 1–14.
[11] N. Geschwind, Specializations of the human brain, Scientific American 241 (3) (1979) 180–201.
[12] R.C. O'Reilly, Y. Munakata, Computational Explorations in Cognitive Neuroscience: Understanding the Mind by Simulating the Brain, MIT Press, 2000.
[13] M. Catani, D.K. Jones, R. Donato, et al., Occipito-temporal connections in the human brain, Brain 126 (9) (2003) 2093–2107.
[14] J. Szentagothai, The ferrier lecture, 1977: the neuron network of the cerebral cortex: a functional interpretation, Proc. R. Soc. Lond. Ser. B. Biol. Sci. 201 (1144) (1978) 219–248.
[15] V.B. Mountcastle, The columnar organization of the neocortex, Brain 120 (4) (1997) 701–722.
[16] A.S. Benjamin, J.S. de Belle, B. Etnyre, T.A. Polk, The role of inhibition in learning, Human Learn.: Biol., Brain, Neurosci.: Biol., Brain, Neurosci. 139 (2008) 7.
[17] W.S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bull. Math. Biophys. 5 (4) (1943) 115–133.
[18] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, Psychol. Rev. 65 (6) (1958) 386.
[19] B. Widrow, et al., Adaptive adaline neuron using chemical memistors, 1960.
[20] A. Aizerman, E.M. Braverman, L. Rozoner, Theoretical foundations of the potential function method in pattern recognition learning, Autom. Rem. Control 25 (1964) 821–837.
[21] J.L. McClelland, D.E. Rumelhart, P.R. Group, et al., Parallel distributed processing, Explorations in the microstructure of cognition 2 (1986) 184.
[22] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, Proc. Nat. Acad. Sci. 79 (8) (1982) 2554–2558.
[23] T. Kohonen, Self-organized formation of topologically correct feature maps, Biol. Cybernet. 43 (1) (1982) 59–69.
[24] S. Grossberg, Competitive learning: from interactive activation to adaptive resonance, Cognit. Sci. 11 (1) (1987) 23–63.
[25] J. Misra, I. Saha, Artificial neural networks in hardware: a survey of two decades of progress, Neurocomputing 74 (1) (2010) 239–255.
[26] L. Arnold, S. Rebecchi, S. Chevallier, H. Paugam-Moisy, An introduction to deep learning, in: ESANN, 2011.
[27] V. Vapnik, The Nature of Statistical Learning Theory, Springer Science & Business Media, 2000.
[28] J. Schmidhuber, Learning complex, extended sequences using the principle of history compression, Neural Comput. 4 (2) (1992) 234–242.
[29] J.R. Anderson, Act: a simple theory of complex cognition, Am. Psychol. 51 (4) (1996) 355.
[30] J. Hawkins, S. Blakeslee, On Intelligence, MacMillan, 2007.
[31] S. Franklin, F. Patterson Jr., The lida architecture: adding new modes of learning to an intelligent, autonomous, software agent, pat 703 (2006) 764–1004.
[32] K. Fukushima, Neocognitron: a hierarchical neural network capable of visual pattern recognition, Neural Netw. 1 (2) (1988) 119–130.
[33] P.J. Werbos, Backpropagation through time: what it does and how to do it, Proc. IEEE 78 (10) (1990) 1550–1560.
[34] S. Hochreiter, Untersuchungen zu dynamischen neuronalen netzen, Master's thesis, Institut fur Informatik, Technische Universitat, Munchen.
[35] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.
[36] G.E. Hinton, To recognize shapes, first learn to generate images, Prog. Brain Res. 165 (2007) 535–547.
[37] Y. Bengio, J. Louradour, R. Collobert, J. Weston, Curriculum learning, in: Proceedings of the 26th Annual International Conference on Machine Learning, ACM, 2009, pp. 41–48.
[38] V. Nair, G.E. Hinton, 3d object recognition with deep belief nets, in: Advances in Neural Information Processing Systems, 2009, pp. 1339–1347.
[39] Y. LeCun, K. Kavukcuoglu, C. Farabet, Convolutional networks and applications in vision, in: Proceedings of the IEEE International Symposium on Circuits and Systems, IEEE, 2010, pp. 253–256.
[40] R. Collobert, J. Weston, A unified architecture for natural language processing: Deep neural networks with multitask learning, in: Proceedings of the 25th Internatuional Conference on Machine Learning, ACM, 2008, pp. 160–167.
[41] S. Zhou, Q. Chen, X. Wang, Active deep networks for semi-supervised sentiment classification, in: Proceedings of the 23rd International Conference on Computational Linguistics: Posters, Association for Computational Linguistics, 2010, pp. 1515–1523.
[42] X. Glorot, A. Bordes, Y. Bengio, Domain adaptation for large-scale sentiment classification: a deep learning approach, in: Proceedings of the 28th International Conference on Machine Learning, 2011, pp. 513–520.
[43] G.E. Dahl, D. Yu, L. Deng, A. Acero, Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition, IEEE Trans. Audio, Speech, Lang. Process. 20 (1) (2012) 30–42.
[44] T.N. Sainath, B. Kingsbury, B. Ramabhadran, P. Fousek, P. Novak, A.-R. Mohamed, Making deep belief networks effective for large vocabulary continuous speech recognition, in: IEEE Workshop on Automatic Speech Recognition and Understanding, IEEE, 2011, pp. 30–35.
[45] A.-R. Mohamed, D. Yu, L. Deng, Investigation of full-sequence training of deep belief networks for speech recognition, in: INTERSPEECH, 2010, pp. 2846–2849.
[46] A.-r. Mohamed, T.N. Sainath, G. Dahl, B. Ramabhadran, G.E. Hinton, M. Picheny, et al., Deep belief networks using discriminative features for phone recognition, in: International Conference on Acoustics, Speech and Signal Processing, IEEE, 2011, pp. 5060–5063.
[47] A.-R. Mohamed, G.E. Dahl, G. Hinton, Acoustic modeling using deep belief networks, IEEE Trans. Audio, Speech, Lang. Process. 20 (1) (2012) 14–22.
[48] P. Hamel, D. Eck, Learning features from music audio with deep belief networks, in: ISMIR, Utrecht, The Netherlands, 2010, pp. 339–344.
[49] G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.

[50] P. Smolensky, Information processing in dynamical systems: Foundations of harmony theory, Department of Computer Science, University of Colorado, Boulder, 1986.
[51] R. Khanna, M. Awad, Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers, Apress, 2015.
[52] G. Hinton, A practical guide to training restricted boltzmann machines, Momentum 9 (1) (2010) 926.
[53] S. Geman, D. Geman, Stochastic relaxation, gibbs distributions, and the bayesian restoration of images, IEEE Trans. Pattern Anal. Mach. Intell. (6) (1984) 721–741.
[54] B. Aleksandrovsky, J. Whitson, G. Andes, G. Lynch, R. Granger, Novel speech processing mechanism derived from auditory neocortical circuit analysis, Proceedings of the 4th International Conference on Spoken Language, vol. 1, IEEE, 1996, pp. 558–561.
[55] A. Fischer, C. Igel, An introduction to restricted boltzmann machines, in: Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, Springer, 2012, pp. 14–36.
[56] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, S. Bengio, Why does unsupervised pre-training help deep learning?, J Mach. Learn. Res. 11 (2010) 625–660.
[57] Y. Bengio, Learning deep architectures for AI, Found. Trends® Mach. Learn. 2 (1) (2009) 1–127.
[58] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, P. Vincent, The difficulty of training deep architectures and the effect of unsupervised pre-training, in: Internartional Conference on Artificial Intelligence and Statistics, 2009, pp. 153–160.
[59] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, et al., Greedy layer-wise training of deep networks, Adv. Neural Inf. Process. Syst. 19 (2007) 153.
[60] G.E. Hinton, How neural networks learn from experience, Scient. Am. 267 (3) (1992) 145–151.
[61] N. Hajj, Y. Rizk, M. Awad, A mapreduce cortical algorithms implementation for unsupervised learning of big data, Proc. Comp. Sci. 53 (2015) 327–334.
[62] N. Hajj, M. Awad, Weighted entropy cortical algorithms for isolated arabic speech recognition, in: International Joint Conference on Neural Networks, IEEE, 2013, pp. 1–7.
[63] A. Hashmi, M.H. Lipasti, Discovering cortical algorithms, in: IJCCI (ICFC-ICNC), 2010, pp. 196–204.
[64] A. Hashmi, A. Nere, J.J. Thomas, M. Lipasti, A case for neuromorphic isas, ACM SIGARCH Computer Architecture News, vol. 39, ACM, 2011, pp. 145–158.
[65] G. Chechik, I. Meilijson, E. Ruppin, Synaptic pruning in development: a computational account, Neural Comput. 10 (7) (1998) 1759–1777.
[66] F.I. Craik, E. Bialystok, Cognition through the lifespan: mechanisms of change, Trends Cog. Sci. 10 (3) (2006) 131–138.
[67] L. Steinberg, Cognitive and affective development in adolescence, Trends Cog. Sci. 9 (2) (2005) 69–74.
[68] E. D'Angelo, A. Antonietti, S. Casali, C. Casellato, J.A. Garrido, N.R. Luque, L. Mapelli, S. Masoli, A. Pedrocchi, F. Prestori, et al., Modeling the cerebellar microcircuit: new strategies for a long-standing issue, Front. Cell. Neurosci. 10 (2016) 176.
[69] G.M. Shepherd, The Synaptic Organization of the Brain, Oxford University Press, 2003.
[70] P. Fransson, U. Åden, M. Blennow, H. Lagercrantz, The functional architecture of the infant brain as revealed by resting-state FMRI, Cereb. Cort. 21 (1) (2011) 145–154.
[71] N. Gogtay, J.N. Giedd, L. Lusk, K.M. Hayashi, D. Greenstein, A.C. Vaituzis, T.F. Nugent, D.H. Herman, L.S. Clasen, A.W. Toga, et al., Dynamic mapping of human cortical development during childhood through early adulthood, Proc. Nat. Acad. Sci. USA 101 (21) (2004) 8174–8179.
[72] M. Lichman, UCI Machine Learning Repository, 2013 <http://archive.ics.uci.edu/ml>.
[73] J. Mutch, U. Knoblich, T. Poggio, CNS: A GPU-Based Framework for Simulating Cortically-Organized Networks, Massachusetts Institute of Technology, Cambridge, MA, Tech. Rep. MIT-CSAIL-TR-2010-013/CBCL-286.
[74] J. Mutch, Cns: Cortical Network Simulator, 2017 <http://cbcl.mit.edu/jmutch/cns/>.
[75] R. Salakhutdinov, G. Hinton, Deep Belief Networks, 2015 <http://www.cs.toronto.edu/hinton/MatlabForSciencePaper.html>.
[76] C. Nadeau, Y. Bengio, Inference for the generalization error, in: Advances in Neural Information Processing Systems, 2000, pp. 307–313.
[77] M. Friedman, The use of ranks to avoid the assumption of normality implicit in the analysis of variance, J. Am. Statist. Assoc. 32 (200) (1937) 675–701.
[78] P. Nemenyi, Distribution-Free Multiple Comparisons, Ph.D. thesis, Princeton University, NJ, 1963.