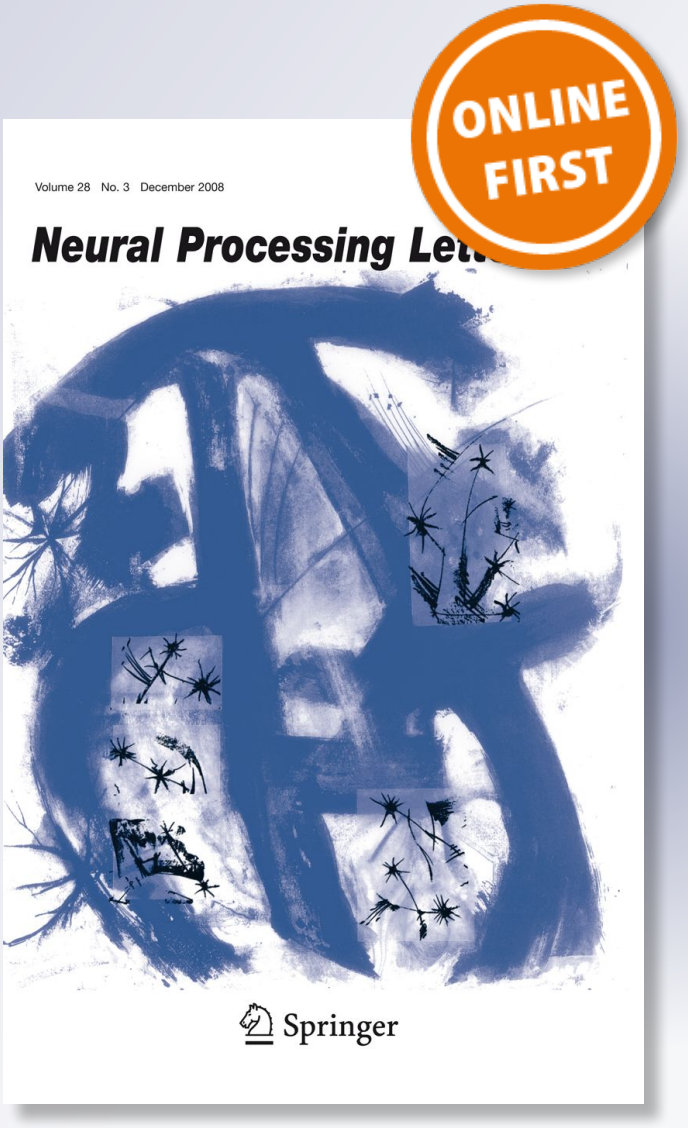
Restricted Convolutional Neural Networks

Mehran Mirkhan & Mohammad Reza Meybodi

Neural Processing Letters

ISSN 1370-4621

Neural Process Lett
DOI 10.1007/s11063-018-9954-x



Springer

Neural Processing Letters
https://doi.org/10.1007/s11063-018-9954-x

Restricted Convolutional Neural Networks

Mehran Mirkhan[1] · **Mohammad Reza Meybodi**[1]

Abstract
In this paper, a new type of convolutional neural network is proposed which is inspired by cellular automata research. This model is referred to as "restricted convolutional neural network" and its characteristic is that the feature maps are not fully connected, i.e. each feature map is only connected to a small neighborhood of previous feature maps. First this model is formally defined. Then it is used for image classification. Two layerwise pretraining methods have been proposed, and some structural variations have been analyzed. The model is tested on both MNIST and CIFAR-10 datasets. Results suggest that this model in some cases can outperform a convolutional neural network with similar architecture.

Keywords Restricted convolutional neural network · Cellular automata · Image classification · Self-organizing map · Boltzmann machine

1 Introduction

The dream of creating intelligent systems is very dependent on developing systems that are capable of visual perception. By considering the history of evolution, one can see that the evolution of brain happened after the evolution of eyes; because, as long as there is nothing to be processed, there is no need for a processor. There are some Jellyfishes that have eyes but do not have any brains, and the visual signals are directly connected to muscles [6].

Visual perception has been studied in many scientific fields. Here, we are interested in deep learning approach. In deep learning, a hierarchy of abstract features are automatically discovered and images are described according to them; for example, in an image of a face, one can define eye, ear and nose as high level features and describe the image in a more abstract way.

Convolutional neural networks (CNNs) are state-of-the-art deep learning models [19]. These models have shown very good performance on many machine learning tasks, especially image processing. The unique feature of CNNs is their semi-local architecture (Neurons have

✉ Mehran Mirkhan
mmirkhan@aut.ac.ir

Mohammad Reza Meybodi
mmeybodi@aut.ac.ir

[1] Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran

Published online: 13 November 2018

restricted connections on spatial dimensions but are fully connected on channel dimension). This paper proposes a new convolutional network model in which all the connections are local. This idea has been motivated by cellular automata (CA) research. In what follows, first this motivation is explained and then our contributions are introduced.

1.1 Motivation

CA[1] are distributed computational models; that is, they are composed of simple computational elements which are connected together and interact with each other. Their interaction is used to perform more complex computations. These elements are arranged in a cellular structure and each cell only interacts with its neighbors.

Neural networks are also distributed models but their emphasis is on the behavior of each element rather than how elements are connected to each other. Therefore, CA and NNs can be combined and form a model in which the connections are local and elements behave like neurons.

It has been proved that fully connected feedforward neural networks are universal approximators [14]; yet, convolutional neural networks (which are special forms of feedforward networks) can easily outperform them. CNNs have special architecture which enables them to better model the real world data. This architecture acts like an infinitely strong prior over the parameters [7]. This shows that the role of architecture in deep learning models can be significant; and since CA have strong emphasis on architecture, it would be interesting to examine their architecture in deep learning.

Another thing that makes such a research interesting is that brain can also be regarded as a CA, since in a brain, each cell is only connected to a small local neighborhood; Moreover, these cells are arranged layerwise and in fact the deep learning research has been motivated by the layerwise behavior of the brain. In other words, brain is both a CA and a deep learning model; Therefore, it is a proof by example that a combination of these two models can show strong performance.

1.2 Contributions

In Sect. 2 a brief history about deep learning and cellular automata research is provided.

CA by themselves are not very useful for deep learning. Here, we first propose a special CA which we refer to as "Dense cellular automaton". This model has irregular structure. Then a special case of it is introduced and referred to as "Deep cellular automaton (DCA)". This model is a layerwise neural network. Finally, A four dimensional DCA is proposed which can be used for image classification and is very similar to CNNs. This model is referred to as "Restricted convolutional neural network (RCNN)". These models are introduced in Sect. 3 and are experimentally evaluated in Sect. 4.

Layerwise pretraining of these cellular models is very challenging, because a local correlation between neighbor hidden variables is required, otherwise the hidden variables on the next layer (which are only connected to a small neighborhood of previous layer) cannot learn good concepts. In Sect. 5, two layerwise pretraining methods have been proposed and their performances are evaluated.

[1] We refer to both cellular automaton and cellular automata (plural) as CA. The distinction would be clear from the context.

The cellular structure of RCNN, enables one to define other structural variations. Section 6 provides three such variations.

Finally a conclusion about this research, its implications and possible continuations are provided in Sect. 7.

2 Background

In this section, a brief background about deep learning and cellular automata research is provided. Also, a review about related works to this research is included.

2.1 Deep Learning

Even though the term "deep learning" is used in recent years, but its approach can be traced back to half a century ago. Deep learning research can be divided into three eras [7]:

1. Research under the name "cybernetics", between 1940 and 1960
2. Research under the name "connectionism", between 1980 and 1990
3. Research under the name "deep learning", from 2006 until today

The initial aim of this research has been to mimic the learning mechanism in species. That is the reason the models used in this research have been called "artificial neural networks". Although in today's deep learning research, "learning multiple levels of composition" is considered to be the definition of this research [7].

The early research on deep learning (referred to as cybernetics) was focused on simple units that behave like neurons. It was assumed that the unit has a vector of weights $\mathbf{w}$ and takes an input vector $\mathbf{x}$. Then the output of the unit is defined as

$$y = f\left(\sum_i \mathbf{x}_i \mathbf{w}_i\right) \tag{1}$$

where f is called the activation function. This model is a linear classifier. By properly setting the weights, this model can separate data points that can linearly be separated. Models such as "McCulloch–Pitts neuron" [22], "ADALINE" [33] and "Perceptron" [26] were the early models in this regard.

The second wave of deep learning began around 1980 called "connectionism". The idea in this research was that by combining simple computational units, complex behaviors would emerge. Terms like "distributed representation" and "representation learning" were coined in this era. One of the major achievements of this era was the "back-propagation" algorithm [32]. This algorithm was based on chain rule of derivatives and enabled researchers to train multilayer models automatically. This technique is still until today, the dominant approach to training deep models. Many deep learning models (such as convolutional neural networks (CNNs) [19]) emerged in this era; but due to difficulty in training and lack of large datasets, they were not fully investigated.

The main characteristic of a CNN is that spatial connections are restricted and weight sharing is used across spatial dimensions; therefore, it is as if some convolutional kernels are being applied to a layer.

The third wave of deep learning began in 2006 when an efficient method to training deep belief networks was proposed [12]. Deep belief networks are generative models and this technique is an unsupervised learning method. Moreover, a fast method for training

restricted Boltzmann machines (RBMs) was proposed [13], and it showed that deep models could be trained greedy and layerwise, and the term "deep learning" was coined.

In the following years, CNN-based deep learning models outperformed other algorithms in many areas and showed the promise of deep learning. For example models like GoogLeNet [28] and ResNet [10] achieved state of the art performance on image classification.

2.2 Cellular Automata

CA are distributed computational models in which the computational units are arranged in a grid structure and are referred to as cells. Each cell has a state and interacts with its neighbor cells and based on a local rule, changes its state through time.

CA were first introduced by John Von Neumann in 1966 [30]. He intended to build a model which is capable of self-reproduction.

Since then, CA have been used in many broad applications. The research on CA can be divided into four categories (inspired by [27]):

1. Classical
2. Games
3. Experimental
4. Learning

These categories are based on the nature of research but they can also be considered temporal.

The classical research on CA was very much focused on computational properties and capabilities of this model. The followings are some of the subjects studied in this category:

- Self-reproducing automata: the aim is to define a CA in which the model is capable of copying a pattern infinitely (although not a trivial pattern).
- Computation universality: it has been showed that CA is capable of universal computation. In fact, the CA proposed by Von Neumann was universal.
- Biological studies: in this research, the structure of CA changes through time and a cell decomposes into more cells. Similar to biological cells.
- Language and pattern recognition: CA can be used as language acceptors. The initial state is the input and the acceptance is determined based on the state of a particular cell.
- Invertibility: a CA with local rule δ is invertible if there exists another local rule (δ^{-1}) which would run the CA backwards. Some parts of this research has been focused on cryptography. The problem of "Garden of Eden" (whether there is a global state which the CA can never reach) is also studied in this research.

Another trend in CA research has been its use as a model of competition (i.e. game). "Firing squad problem", "σ-game" and "Game of Life" are some examples. Game of life is much more famous [5]. It has been shown that it is capable of universal computation and is predicted that in very large scales can lead to self-replicating patterns.

Experimental research on CA has been started around mid-1980s and is largely due to the works of Wolfram [34,35]. In previous research phases, first a problem would be proposed and then by the use of CA, researchers would try to solve it. But Wolfram made a paradigm shift. He would first construct a CA and then study its behavior and properties. One of his achievements was to propose a classification for CA based on their global behavior. Also he discovered the most simple Turing machine on elementary CA. In these years, many algorithms based on CA were proposed for applications in physics [29] and chemistry [4].

The fourth phase of CA research was started around 2000s and was focused on combining learning with CA. This idea would enable a researcher to describe behavior of the cells in a

more abstract way instead of precise rules and the CA would learn to converge to the desired behavior. "Cellular learning automata (CLA)" [2] is such a model. CLA is a CA in which each cell is a learning automaton and performs reinforcement learning. Each cell has a probability distribution and it is updated trough time. This has been used in many machine learning applications such as image segmentation, edge detection, noise reduction and evolutionary computation. our research is in this category.

2.3 Related Works

The idea that some parameters in neural networks are unnecessary and redundant has been studied before. One of the approaches proposed for eliminating such parameters is "network pruning" [8,9,23]. In this approach, first the model is trained; then in fine tuning stage of training, the parameters that have the least contribution in the model are eliminated. This technique is different than ours in two aspects: (1) In pruning the architecture is modified after the training, while in our approach the architecture is established in the beginning. This means that in our approach, it is the architecture that guides the learning process not the other way around. (2) In our approach, the architecture is symmetric and regular, while in network pruning, the final architecture will be highly irregular. Therefore in network pruning, no memory optimization is achieved and all the parameters are stored and consume memory.

The idea of combining CA with neural networks has also been studied before [3,31,36]. However in these studies, the proposed model is a recurrent neural network while in our research, the model is feed-forward; therefore the model has different applications. Moreover, all of these studies have used regular CA, while here we have proposed and used "Dense cellular automata" which enables the model to have an arbitrary number of hidden variables.

3 The Proposed Model

In this research, the CA approach to model definition is chosen. The reason is that CA put strong emphasis on formal structure definition. This emphasis can make structure specification more clear. With the help of such formalizations, a wide range of structures can be easily and unambiguously defined. This benefit will be more clear in Sect. 6.

One of the earliest formalizations of CA has been proposed in 1974 by Jump and Kirtane [16]. A CA is defined by 4-tuple (C, N; S, δ) where:

- C is a countable set of cells,
- $N : C \rightarrow C^k$ is the neighborhood function which given a cell, returns its neighbors,
- S is a finite set of states of each cell,
- $\delta : S^k \rightarrow S$ is the local rule of the cells.

In each time step, each cell changes its state according to the states of the neighborhood cells (the cell itself included) and according to a local rule.

This definition is quite old and it belongs to the first era of CA research. It has two restrictions that are discarded in todays research. First restriction is that it has assumed that the set of states of each cell is finite but in many CA models like "probabilistic CA" [15] and CLA, the cells can take continuous values as their states. The second restriction is the assumption that all cells are identical and have similar neighborhoods. This assumption has also been loosened with models like "hyperbolic CA" [21] and "irregular CA" [25]. We adopt this definition without the above two restrictions.

In this section we first propose an irregular CA which we refer to as "Dense cellular automaton". Then a special form of this model (that activates layerwise) is introduced and referred to as "Deep cellular automaton (DCA)". Finally a special four dimensional form of DCA is used as image classifier and is referred to as "Restricted convolutional neural network (RCNN)".

Even though this research is focused on deep learning and image classification, but the models proposed are more general and they can possibly be used for other applications too.

3.1 Dense Cellular Automaton

Dense CA is a CA in which different parts of the network have different cell density. Since in general, specifying the neighborhood function for such a model is very difficult, we assume that the density of cells changes along only one dimension (layer dimension). Figure 1 shows such a CA. As can be seen, the cells are countable and can be coordinated with integer vectors.

Lets assume the first dimension is the layer dimension. Since the density changes over this dimension, the neighborhood function should be broken into one function for each layer:

$$N(\mathbf{u}) = \bigcup_{i \in \mathbb{Z}} N_i(\mathbf{u}), \tag{2}$$

where i refers to the relative position of a layer (for example $N_{-1}(\mathbf{u})$ refers to the neighbors of $\mathbf{u}$ in the previous layer). Then each neighborhood function can be defined as

$$N_i(\mathbf{u}) = v_1 \times v_2 \times \cdots \times v_d \tag{3}$$

$$v_1 = \{\mathbf{u}_1 + i\} \tag{4}$$

$$v_{j|_{j=2,\ldots,d}} = \left\{ z \in \mathbb{Z} \mid \left\lceil \lfloor \left(\mathbf{u}_j + \frac{1}{2} \right) c_i \rfloor - \frac{r_i^{(\mathbf{u}_1)}}{2} \right\rceil \leq z < \left\lceil \lfloor \left(\mathbf{u}_j + \frac{1}{2} \right) c_i \rfloor + \frac{r_i^{(\mathbf{u}_1)}}{2} \right\rceil \right\} \tag{5}$$

where

$$c_i = \begin{cases} \left(\prod_{k=\mathbf{u}_1+i+1}^{\mathbf{u}_1} \mathcal{D}^{(k)} \right)^{-1} & i < 0 \\ 1 & i = 0 \\ \prod_{k=\mathbf{u}_1+1}^{\mathbf{u}_1+i} \mathcal{D}^{(k)} & i > 0 \end{cases} \tag{6}$$

$r_y^{(x)}$ refers to the number of neighbors of cells in layer x in the relative layer y and is set a priori by the user (for example $r_{-1}^{(4)} = 3$ means each cell in layer 4 is connected to 3 neighbor cells of previous layer). $\mathcal{D}^{(k)}$ determines the density factor of layer k with respect to its previous layer. For example for CA in Fig. 1 we have:

$$\mathcal{D}^{(0)} = 1, \quad \mathcal{D}^{(1)} = 4, \mathcal{D}^{(2)} = \frac{1}{3}, \quad \mathcal{D}^{(3)} = \frac{1}{2}, \quad \mathcal{D}^{(4)} = 3$$

A special property of dense CA is that some groups of cells in a layer can have similar neighborhoods in their previous layer. We refer to such groups as "homogeneous cells".

3.2 Deep Cellular Automaton

Dense CA is only a structure definition; In other words, it has no emphasis on the behavior of each cell. In order to use dense CA for deep learning, some specifications about the behavior

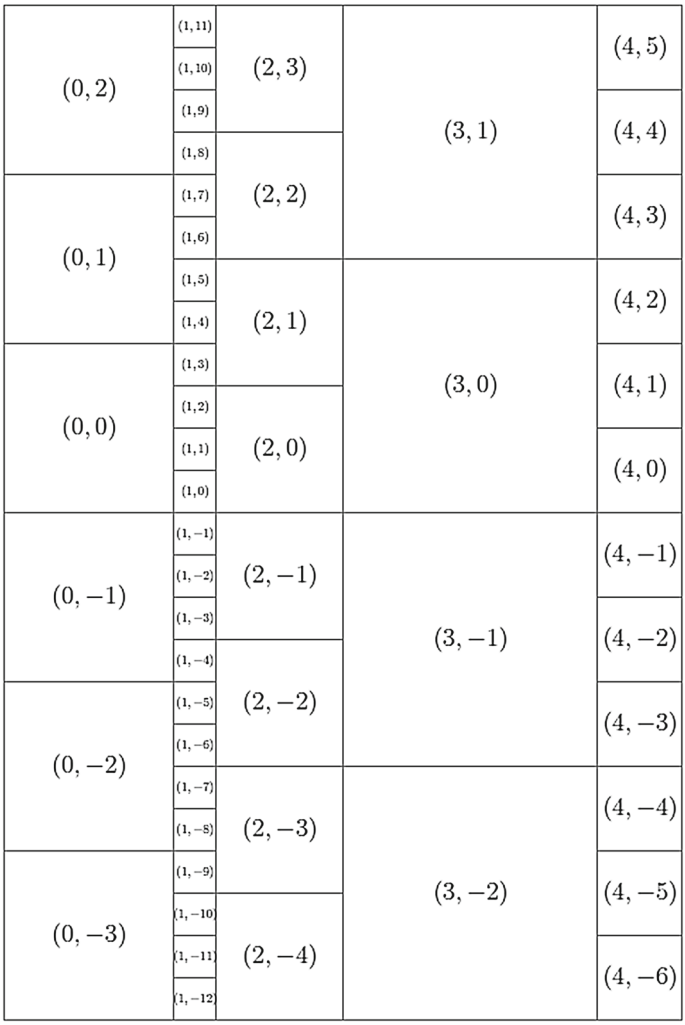


Fig. 1 An example of dense cellular automaton. Different parts of the network have different densities. Density changes only along one dimension

of cells should be considered. We refer to a dense CA with such considerations as deep CA (DCA).

3.2.1 Layerwise Activation

Deep learning models perform in a directed structure in which each layer is activated only when its previous layer has performed its computation. Such scheme can be implemented in CA with the use of an external signal.

Another approach is to extend the set of states. For example a CA with state set S can be modified like $S' = (S, \rho)$ where $\rho \in \{0, 1\}$. Initially the ρ for all cells would be set to zero except first layer. Then the local rule would be defined in a way that the cells with $\rho = 1$

update their states and cells that all their neighbors of the previous layer have $\rho = 1$ would be activated in the next step. With this method, the cells will be activated layerwise.

3.2.2 Local Rule

An artificial neuron can be considered an automaton since at each time step it has a state (namely its weights and its output) and has a rule by which this state changes. Therefore the set of states for perceptron like cells can be defined as

$$S = \left\{ (\mathbf{w}, b, v) | \mathbf{w} \in \mathbb{R}^k, b \in \mathbb{R}, v \in \mathbb{R} \right\}, \tag{7}$$

where k is the number of neighbors, $\mathbf{w}$ is the weights of perceptron, b is the bias and v is the output. Then the local rule would be

$$v_{\mathbf{u}} \leftarrow \sigma \left(\mathbf{w}^T \mathbf{v}_{N(\mathbf{u})} + b \right), \tag{8}$$

where σ is the activation function and $\mathbf{v}_{N(\mathbf{u})}$ refers to state v of the neighbors of $\mathbf{u}$ arranged in a vector. Similar states and local rules can be defined for subsampling units like max-pooling.

3.2.3 Back-Propagation

The back-propagation algorithm is a local algorithm in which each cell needs local information in order to update its parameters. This local information and backward activation can also be implemented with state extension technique explained above.

Since in this approach, cells are activated in both directions, it is enough to only define N_{-i} for $i > 0$, representing the neighbor cells in previous layers. $N_i(\mathbf{u})$ would be defined as the set of cells that $\mathbf{u}$ is in their N_{-i} neighborhood. This approach guarantees that the neighborhood relation is reflective.

3.3 Restricted Convolutional Neural Network

DCA is a general deep learning model and can be used for different applications. The main subject of this paper is image classification. For this purpose, we define a four dimensional DCA in which the first dimension is the layer dimension and the other three dimensions are height, width and channel. Similar to CNNs, we use weight sharing for cells that belong to the same layer and channel. This model is similar to CNNs except that the connection between channels is restricted. In [7] it is briefly mentioned that these connections can be restricted but the writers have not referenced to any research in this regard. As far as we know, the restriction proposed in this paper has not been proposed before.

We refer to this model as "Restricted convolutional neural network (RCNN)". Figure 2 shows the structure of this model and Fig. 3 presents a taxonomy of the models proposed in this section.

The justification for RCNN model is that CNNs assume a concept in a layer (a.k.a. a feature map) is dependent to all the concepts in previous layer; however, this assumption does not seem to be always the case. For example, the concept of tree depends on concepts like branch and leaf while the concept of cat is dependent on concepts like eye and ear. This means that many connections in CNNs are unnecessary. Therefore RCNN is making an infinitely strong assumption over such connections.

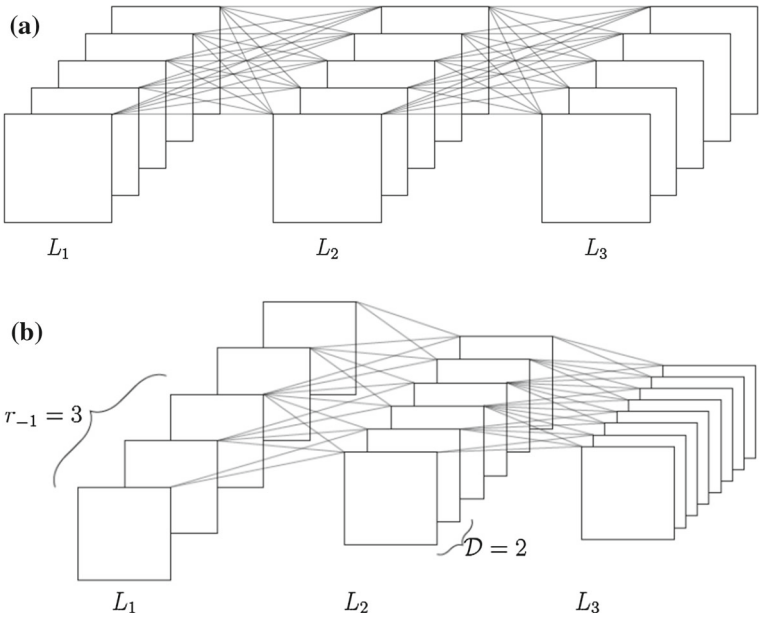


Fig. 2 A comparison between CNN and RCNN models. Each rectangle represents a feature map and lines represent dependence between feature maps. **a** CNN. **b** RCNN

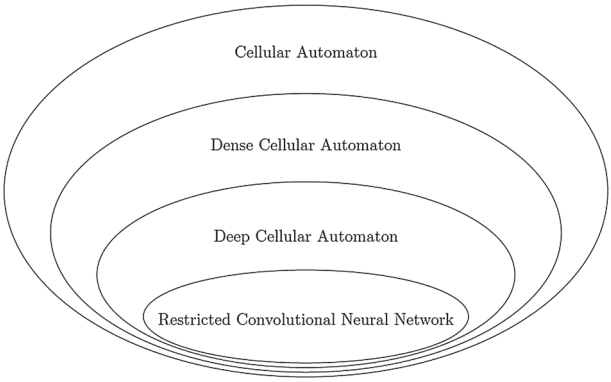


Fig. 3 Classification of the proposed models

4 Experimental Results

In this section, the proposed models will be evaluated. First DCA is used for classifying extended XOR problem. Then RCNN is used for classifying MNIST and CIFAR-10 datasets.

It should be noted that we did not have GPUs and all the experiments in this research are performed on a single CPU. To keep training time short, regularizers are not used. Therefore our results are not comparable to state of the art results on these datasets; however, the situation is similar for both RCNN and CNN and therefore it seems it would be a fair comparison between these two models. Also, this is the reason why training times are not

Table 1 Extended XOR problem

x_1	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
x_2	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
y_1	0	1	0	1	1	0	1	0	0	1	0	1	1	0	1	0
y_2	1	0	1	0	0	1	0	1	1	0	1	0	0	1	0	1

Table 2 DCA model for classifying extended XOR problem

Layer	Cell type	r_{-1}	$\mathcal{D}$	Layer size	Activation
1	Perceptron	(2)	(3)	(6)	Sigmoid
2	Perceptron	(3)	(2)	(12)	Sigmoid
3	Perceptron (fully connected)	–	–	(2)	Sigmoid

Each row is a layer of the model

included since training on CPU does not properly represent how fast a parallel model can be. RCNN with proper implementation on GPUs would be most likely faster.

4.1 Simple 2D Classification

The extended XOR problem is defined in Table 1. To classify this dataset, a DCA specified in Table 2 is defined.

In this experiment, we are interested to see what each cell learns in a DCA model. For this purpose, a cell's response to every point in the input space is regarded as the concept learned by that cell. This model is trained using back-propagation and the concept learned by each cell is represented in Fig. 4.

An interesting observation is that neighbor cells have learned similar concepts, and it appears as if these concepts have been sorted. This observation makes sense, since each cell in layer l is only connected to a small neighborhood of cells in layer $l-1$ and if these cells are not meaningfully relevant to each other, cells in layer l cannot construct useful concepts. Note that the cells in layer L_2 are not sorted because there is no cellular layer after to enforce local correlation.

4.2 MNIST

MNIST is an image dataset of hand written digits (0–9) [20]. There are 60,000 train and 10,000 test images. Each image is $28 \times 28 \times 1$. We use RCNN specified in Table 3 to classify this dataset. The model is trained with back-propagation on 20,000 batches each containing 128 images. In deep learning models, it is useful to decrease the learning rate over time; Even though this would increase the training time, but it results in a more precise model. Therefore the learning rate was set to 0.1 for the first 10,000 batches and was decreased to 0.02 for the next 10,000 batches. The learning process was repeated 20 times. The average accuracy obtained is $97.8535 \pm 0.385\%$.

To compare this result with CNN model, a CNN was constructed with exactly the same architecture but fully connected channels (Table 4 shows the architecture of this CNN). With the same training process, CNN achieves $98.428 \pm 0.211\%$ which is higher than our model. T test on these results gives a p value of 0.00016 which means the difference is meaningful.

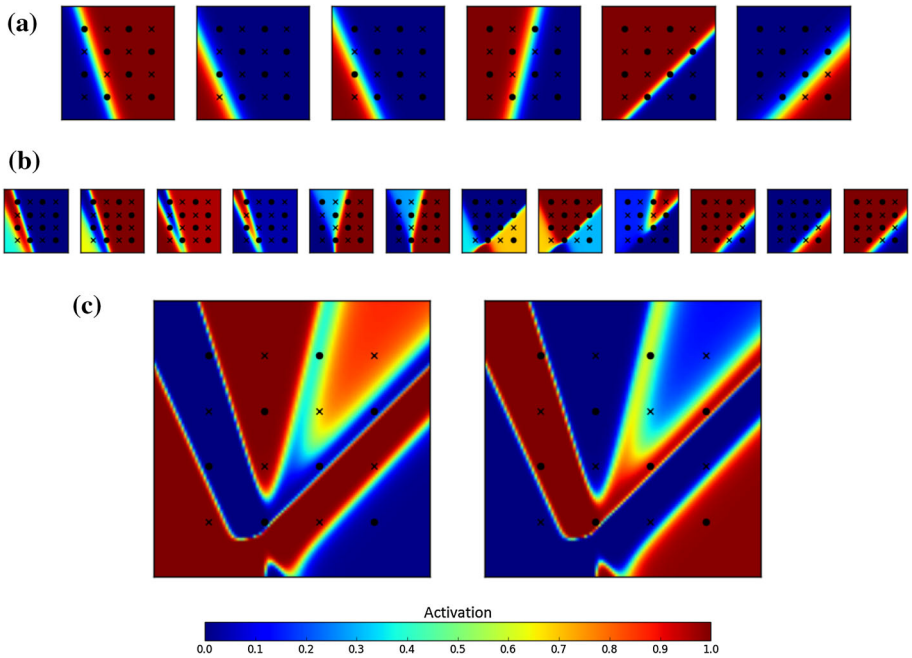


Fig. 4 Concepts learned by each cell on extended XOR problem by DCA. **a** L_1. **b** L_2. **c** L_3

Table 3 RCNN model for classifying MNIST dataset

Layer	Cell type	r_{-1}	$\mathcal{D}$	Layer size	Activation
1	Perceptron	(5, 5, 1)	(1, 1, 5)	(24, 24, 5)	Relu
2	Perceptron	(5, 5, 4)	(1, 1, 5)	(20, 20, 10)	–
3	Max pooling	(2, 2, 1)	$\left(\frac{1}{2}, \frac{1}{2}, 1\right)$	(10, 10, 10)	Relu
4	Perceptron	(6, 6, 6)	(1, 1, 4)	(5, 5, 20)	Relu
5	Perceptron	(5, 5, 6)	(1, 1, 3)	(1, 1, 45)	Relu
6	Perceptron (fully connected)	–	–	(64)	Relu
7	Perceptron (fully connected)	–	–	(32)	Relu
8	Perceptron (fully connected)	–	–	(10)	–

Layer size represents the dimensions of each layer: (height, width, channel)

Table 4 CNN model for classifying MNIST dataset

Layer	Layer type	Kernel size	Number of kernels	Layer size	Activation
1	Convolutional	5×5	5	(24, 24, 5)	Relu
2	Convolutional	5×5	10	(20, 20, 10)	–
3	Max pooling	2×2	–	(10, 10, 10)	Relu
4	Convolutional	6×6	20	(5, 5, 20)	Relu
5	Convolutional	5×5	45	(1, 1, 45)	Relu
6	Fully connected	–	–	(64)	Relu
7	Fully connected	–	–	(32)	Relu
8	Fully connected	–	–	(10)	–

Even though our model has shown a bit lower accuracy, it has much fewer number of parameters. CNN has 31,155 parameters (without considering fully connected layers) while our model has only 12,275 (less than half). This reduction in the number of parameters, leads to a reduction in the number of multiplications and consequently a reduction in computational cost.

4.3 CIFAR-10

CIFAR-10 is a dataset of natural images [18]. each image belongs to one of 10 categories (bird, cat, truck, …). There are 50,000 train and 10,000 test images. Each image is $32 \times 32 \times 3$. For this dataset, RCNN specified in Table 5 was used.

The model was trained with back-propagation on 50,000 image batches each containing 128 images with learning rate set to 0.1 (to avoid long training time, no learning rate reduction was used). The process was repeated 10 times. The average accuracy was $65.237 \pm 0.452\%$.

Then a CNN with the same architecture (with fully connected channels) was constructed (Table 6) and trained similarly. The CNN reaches $63.844 \pm 1.241\%$ accuracy. Preforming T test on these accuracies gives a p value of 0.01252; which means our model has performed meaningfully better.

This result is very interesting. The CNN model has 30,232 parameters (without fully connected layers) while the RCNN model has only 6232 parameters (one-fifth) and yet the RCNN has performed better. This means that 80% of convolutional parameters in CNN have been redundant!

It seems the reason RCNN has performed better on CIFAR-10 rather than MNIST is that the classes in CIFAR-10 are more different from each other. Earlier we argued that a concept does not depend on all the lower level concepts (we made a comparison between tree and

Table 5 RCNN model for classifying CIFAR-10 dataset

Layer	Cell type	r_{-1}	$\mathcal{D}$	Layer size	Activation
1	Perceptron	(5, 5, 3)	(1, 1, 18)	(28, 28, 18)	–
2	Max pooling	(2, 2, 1)	$\left(\frac{1}{2}, \frac{1}{2}, 1\right)$	(14, 14, 18)	Relu
3	Perceptron	(5, 5, 3)	(1, 1, 4)	(10, 10, 64)	Relu
4	Perceptron (fully connected)	–	–	(384)	Relu
5	Perceptron (fully connected)	–	–	(192)	Relu
6	Perceptron (fully connected)	–	–	(10)	–

Table 6 CNN model for classifying CIFAR-10 dataset

Layer	Layer type	Kernel size	Number of kernels	Layer size	Activation
1	Convolutional	5×5	18	(28, 28, 18)	–
2	Max pooling	2×2	–	(14, 14, 18)	Relu
3	Convolutional	5×5	64	(10, 10, 64)	Relu
4	Fully connected	–	–	(384)	Relu
5	Fully connected	–	–	(192)	Relu
6	Fully connected	–	–	(10)	–

Table 7 The performance of CNN and RCNN on MNIST and CIFAR-10 datasets

Dataset	RCNN accuracy (%)	CNN accuracy (%)
MNIST	97.8535 ± 0.385	98.428 ± 0.211
CIFAR-10	65.237 ± 0.452	63.844 ± 1.241

Table 8 Model for sensitivity analysis of hyperparameters on MNIST

Layer	Cell type	r_{-1}	$\mathcal{D}$	Layer size	Activation
1	Perceptron	(5, 5, 1)	(1, 1, 7)	(24, 24, 7)	–
2	Max pooling	(2, 2, 1)	$\left(\frac{1}{2}, \frac{1}{2}, 1\right)$	(12, 12, 7)	Relu
3	Perceptron	(5, 5, r)	(1, 1, D)	(8, 8, 60)	Relu
4	Perceptron (fully connected)	–	–	(100)	Relu
5	Perceptron (fully connected)	–	–	(10)	–

Table 9 Model for sensitivity analysis of hyperparameters on CIFAR-10

Layer	Cell type	r_{-1}	$\mathcal{D}$	Layer size	Activation
1	Perceptron	(5, 5, 3)	(1, 1, 20)	(28, 28, 20)	–
2	Max pooling	(2, 2, 1)	$\left(\frac{1}{2}, \frac{1}{2}, 1\right)$	(14, 14, 20)	Relu
3	Perceptron	(5, 5, r)	(1, 1, D)	(10, 10, 60)	Relu
4	Perceptron (fully connected)	–	–	(400)	Relu
5	Perceptron (fully connected)	–	–	(200)	Relu
6	Perceptron (fully connected)	–	–	(10)	–

Table 10 Different values for r and D for sensitivity analysis of hyperparameters on MNIST

r	2	3	4	5	6	7
D	10	12	15	20	30	60

cat); it seems that this argument is more meaningful in datasets in which classes are more diverse. In MNIST, all digits are made of very similar parts and therefore a less local model like CNN can perform better.

Table 7 presents the accuracies obtained in these experiments.

4.4 Sensitivity Analysis on Hyperparameters

The RCNN model proposed here has two more hyperparameters than original CNN model, namely r and D. Here we provide an analysis on the effect of different values of these hyperparameters on the performance of the model. To make results comparable, the number of hidden units are kept constant so that the amount of information passed trough layers remains constant. This means that by increasing r, D should also be increased.

The models used for MNIST and CIFAR-10 datasets are presented in Tables 8 and 9 respectively. Parameters are changed according to Tables 10 and 11 for MNIST and CIFAR-10 respectively (The last column represents a CNN since all feature maps are connected).

Table 11 Different values for r and D for sensitivity analysis of hyperparameters on CIFAR-10

r	6	11	16	18	20
D	4	6	12	20	60

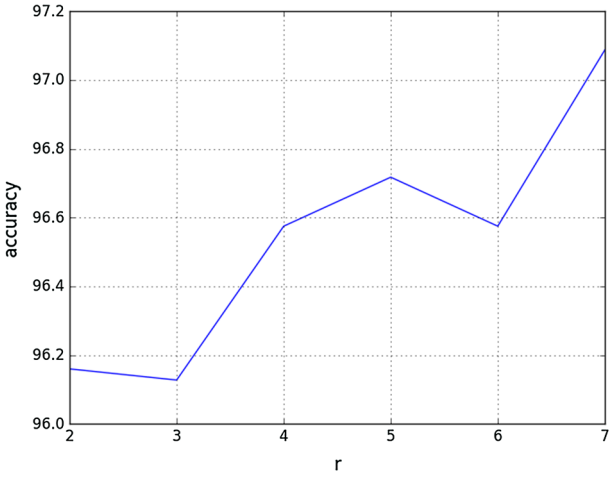


Fig. 5 Sensitivity on r on MNIST

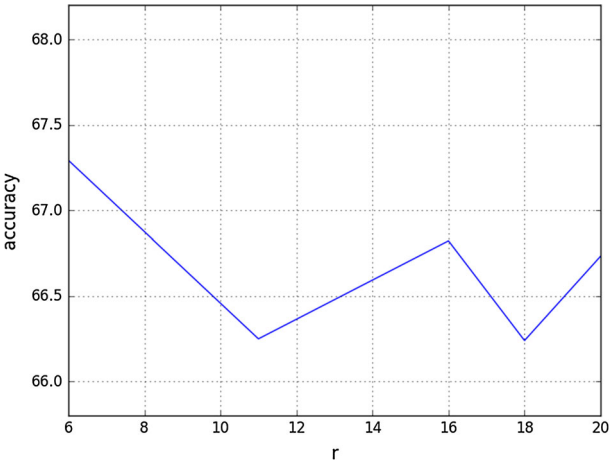


Fig. 6 Sensitivity on r on CIFAR-10

Figure 5 shows that in MNIST dataset, by increasing r the accuracy also increases. This is similar to the result obtained earlier. However, Fig. 6 shows that in CIFAR-10 dataset, increasing r does not increase accuracy. This means that on a dataset with more diverse classes, full dependence of concepts is not a necessary assumption and a restricted model would be at least as good as a fully connected representation.

5 Performance Improvement with Pretraining

In CNN, concepts are fully dependent; therefore neurons are only concerned with finding useful concepts. This means each neuron can converge to any concept so long as it is a useful concept. However, this is not the case in RCNN. In RCNN, feature maps have a specific meaningful order. That is, each cell is only allowed to converge to concepts that are highly correlated with concepts discovered by neighborhood cells. This can potentially increase the training time and also can result in a worse local minimum.

Our idea is to use a layerwise pretraining method that enforces local correlation on parameters. Such a method can enable the cells to start from better initial conditions.

A pretraining method is an unsupervised learning algorithm that can be used to obtain better initial conditions for deep learning models. There has been many pretraining methods introduced so far for deep learning models (e.g. autoencoders, RBMs and etc.). Autoencoders [1] are models in which the model outputs the input itself. Autoencoders first learn a more abstract representation of input and then try to construct the input from those abstractions. Autoencoders are not layerwise pretraining methods. Therefore it can be costly to use them as a pretraining method for deep and complex models. RBMs, are layerwise pretraining methods but it does not seem to be very useful for DCA; since it does not enforce local correlation on concepts. Although one can use deep RBMs to enforce this condition (like autoencoders), but such methods are very costly to perform.

In this section, two layerwise pretraining methods are proposed. First approach is to use self-organizing maps (SOM) with some modifications (which is referred to as discriminative self-organizing map (DSOM)). This algorithm is only useful in datasets that can be expressed with binary features (like MNIST) but it results in very good correlation between neighborhood cells. The next algorithm is called "cellular semi-restricted Boltzmann machine" (CSRBM) which is a modified RBM and can discover locally correlated concepts. These two algorithms have downsides but they can be used as starting point for developing better approaches. In what follows, these two methods are introduced and are evaluated.

5.1 Discriminative Self-Organizing Map

SOM introduced by Kohonen in 1982 [17] is a clustering algorithm defined on a cellular structure similar to a CA. Let $\mathbf{x} \in \mathbb{R}^D$ denote a data instance. In the start of the SOM algorithm, each cell is initialized with a random state $\Phi \in \mathbb{R}^D$. For each data instance $\mathbf{x}$, its Euclidean distance to each cell state is calculated. The cell that has minimum distance is called the winner. Let $\mathbf{v}$ denote the index of the winning cell. At time t, the cell with index $\mathbf{u}$ is updated as Eq. (9):

$$\Phi_{\mathbf{u}}^{(t+1)} \leftarrow \Phi_{\mathbf{u}}^{(t)} + \Lambda_{\mathbf{v},\mathbf{u}}(t) \cdot \alpha \cdot \left(\mathbf{x} - \Phi_{\mathbf{u}}^{(t)}\right) \tag{9}$$

where α is the learning rate and $\Lambda_{\mathbf{v},\mathbf{u}}(t)$ is a Gaussian function defined over the cellular structure that represents the closeness of cell $\mathbf{u}$ to the winning cell and is defined as

$$\Lambda_{\mathbf{v},\mathbf{u}}(t) = e^{-\frac{||\mathbf{u}-\mathbf{v}||_2^2}{\sigma^2(t)}} \tag{10}$$

where $\sigma^2(t)$ is a function that determines the variance of this Gaussian function at time t. It is defined in a way that generates big variance in the beginning and is slowly decreased throughout the algorithm. This function is the key to preserve the topological structure of cells while converging to clusters.

5.1.1 Algorithm

SOM algorithm by itself is not good enough as a pretraining method; because we are interested in an algorithm that results in parameters that can be directly used as perceptron's parameters. In other words, parameters need to be discriminative.

Let $\mathbf{w}$ be the parameters learned by SOM. To make these parameters discriminative, we propose a second phase based on gradient decent. Let $\mathbf{v}$ denote the winning cell with respect to $\mathbf{x}$. This means for every input vector $\mathbf{x}$, one can define a target vector $\mathbf{t_v}$ which is a one hot vector (one at $\mathbf{v}$ and zero everywhere else) or Gaussian vector (around index $\mathbf{v}$). Then gradient descent can be used to minimize

$$e = \frac{1}{2}(\mathbf{t_v} - \mathbf{o})^T(\mathbf{t_v} - \mathbf{o}) \tag{11}$$

where $\mathbf{o}$ denotes the output of perceptrons. The second phase would change the parameters in a way that the output of perceptrons (inner product with the input vector) would show the winning cell, and also would preserve the correlation between neighborhood cells.

There are some points to consider for DSOM to work properly. During the second phase, the parameters obtained in the first phase should remain intact and be used to find the winner cells ($\mathbf{v}$). Therefore a copy of the parameters are needed. Before the beginning of the second phase, it is better to divide $\mathbf{w}$ by $n > 1$ so that $\mathbf{w}$ is close to zero, otherwise gradient descent cannot work properly. To use DSOM on DCA, winning in both phases should be determined among homogeneous cells (explained earlier). This means that there would be a winner in each homogeneous group. Algorithm 1 shows the DSOM pretraining method.

5.1.2 Results

First to show how DSOM works in general, consider a simple two dimensional dataset with three classes. Classes are three Gaussian functions with means (0, 1), (1, 0) and (1, 1). The result of first phase (SOM) and second phase are presented in Fig. 7a, b respectively. As can be seen, the first phase discovers clusters and updates $\mathbf{w}$. The second phase updates both $\mathbf{w}$ and b of perceptrons, and separates each cluster from other clusters. Now, for each data point, the inner product would determine the winner. It should be noted that DSOM assumes that important concepts are in the corners of input space (as can be seen in this experiment).

In the next experiment, DSOM is used on a 1×16 cellular structure on MNIST dataset. It is assumed that each cell is connected to the whole input image (model is not a DCA). The results are presented in Fig. 8.

The purpose of DSOM algorithm was to make cells learn concepts that are meaningfully related to the concepts learned by their neighbors. To see if DSOM fulfills this requirement, the correlation matrix between cells is calculated and presented in Fig. 9. As can be seen, neighbor cells are correlated. It should be noted that this method might not be really useful in fully connected models.

Next we use DSOM for layerwise pretraining of the model defined in Table 3. Parameters of the first layer learned by DSOM are presented in Fig. 10. Since each cell only observes a limited part of the image, it is possible that the patch would be fully white or black. Such patches do not contain useful information; therefore, the average of the values of pixels in each patch is used to determine the importance of the patch. If this value is close to 0 or 1, the learning rate is decreased for that patch.

After all the layers are pretrained, back-propagation is used to train the model. Final accuracy (after 20 times repetition) is $98.651 \pm 0.172\%$. This result is higher than what was

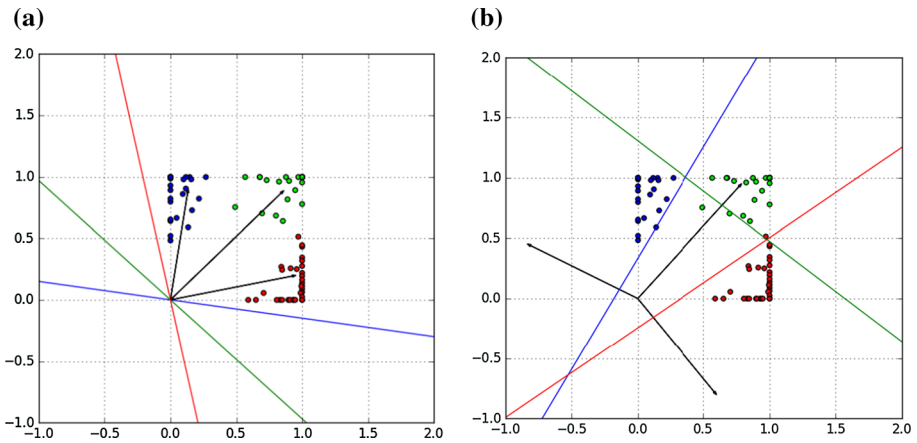


Fig. 7 Result of DSOM on a 1×3 cellular structure. Each vector represents the parameter $\mathbf{w}$ of a perceptron, and each line is how the perceptron divides the space. **a** First phase (SOM). **b** Second phase (GD)

Fig. 8 Pretraining a fully connected 1×16 cellular structure on MNIST with DSOM. **a** First phase (SOM). **b** Second phase (GD)

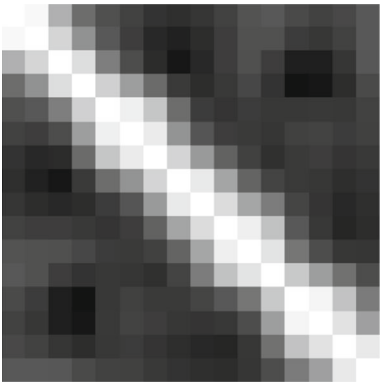


Fig. 9 Correlation between hidden variables after DSOM. Neighbor hidden variables are correlated

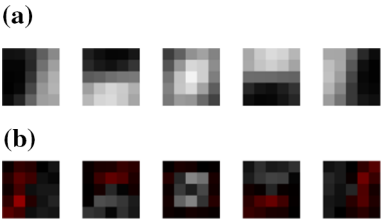


Fig. 10 First layer of the model defined in Table 3 pretrained with DSOM. Parameters with negative value are presented with red. **a** First phase (SOM). **b** Second phase (GD). (Color figure online)

Algorithm 1 Pretraining DCA with DSOM algorithm

Set l, the index of layer to be trained
Set $\alpha_1 \in [0, 1]$, the learning rate for SOM
Set $\alpha_2 \in [0, 1]$, the learning rate for gradient descent
Set $\mathbf{w_u}$, the state of cell $\mathbf{u}$ to a small random value
Set ρ, the decrement factor to a small number (like 0.1)
while not converged (SOM loop) **do**
    Sample a batch of examples from training data (X)
    Set σ, the variance of Gaussian function to a large number
    Set $\Delta_\mathbf{u}$, the update for cell $\mathbf{u}$ to $\mathbf{0}$
    for $m = 1$ to $|X|$ **do**
        Calculate $\mathbf{o}^{(l-1)}$, the output of layer $l-1$ based on $\mathbf{x}^{(m)}$
        for each cell $\mathbf{u}$ **do**
            Let $\mathbf{o}_\mathbf{u}^{(l-1)}$ denote the part of $\mathbf{o}^{(l-1)}$ that cell $\mathbf{u}$ observes.
            Let $\mathbf{v}_\mathbf{u}^{(k)}$ denote the closest cell of kth pack of homogeneous cells to $\mathbf{o}_\mathbf{u}^{(l-1)}$ that is homogeneous with $\mathbf{u}$.
            $\Delta_\mathbf{u} \leftarrow \Delta_\mathbf{u} + \left(\sum_k \Lambda(\mathbf{u}, \mathbf{v}_\mathbf{u}^{(k)})\right)(\mathbf{o}_\mathbf{u}^{(l-1)} - \mathbf{w_u})$
        end for
    end for
    $\Delta_\mathbf{u} \leftarrow \Delta_\mathbf{u} \times \frac{1}{|X|}$
    $\mathbf{w_u} \leftarrow \mathbf{w_u} + \alpha_1 \Delta_\mathbf{u}$
    Decrease σ toward zero
end while
for each cell $\mathbf{u}$ **do**
    $\mathbf{w}'_\mathbf{u} \leftarrow \mathbf{w_u}$
    $\mathbf{w_u} \leftarrow \mathbf{w_u} \times \rho$
end for
while not converged (gradient descent loop) **do**
    Sample a batch of examples from training data (X)
    Set $\Delta_\mathbf{u}$, the update for cell $\mathbf{u}$ to 0
    for $m = 1$ to $|X|$ **do**
        Calculate $\mathbf{o}^{(l-1)}$, the output of layer $l-1$ based on $\mathbf{x}^{(m)}$
        Let $\mathbf{o}_\mathbf{u}^{(l-1)}$ denote the part of $\mathbf{o}^{(l-1)}$ that cell $\mathbf{u}$ observes.
        Let $\mathbf{v}_\mathbf{u}^{(k)}$ denote the closest cell to $\mathbf{o}_\mathbf{u}^{(l-1)}$ according to $\mathbf{w}'$ that is homogeneous with $\mathbf{u}$.
        Set $\mathbf{t}$, the ideal output of cells (e.g. a one-hot vector indexed on $\mathbf{v}_\mathbf{u}^{(k)}$)
        Set $\mathbf{y}$, the output of cells
        Set error: $e \leftarrow \frac{1}{2}(\mathbf{t} - \mathbf{y})^T(\mathbf{t} - \mathbf{y})$
        $\Delta_\mathbf{u} \leftarrow \Delta_\mathbf{u} + \frac{\partial e}{\partial \mathbf{w_u}}$
    end for
    $\Delta_\mathbf{u} \leftarrow \Delta_\mathbf{u} \times \frac{1}{|X|}$
    $\mathbf{w_u} \leftarrow \mathbf{w_u} + \alpha_2 \Delta_\mathbf{u}$
end while

achieved with CNN and RCNN without pretraining (p value of T test with respect to these two cases is 0.00356 and 0.00009 respectively). This shows that pretraining can increase that performance of the model since parameters start from better initial conditions.

Finally the same process was repeated for the model defined in Table 5. Accuracy after 10 repetitions is $65.141 \pm 0.356\%$ which is lower than the case without pretraining. This means DSOM has a negative effect on CIFAR-10 dataset. This result was expected since in color images, corners of input space do not necessarily represent proper concepts, and therefore DSOM fails to discover useful features.

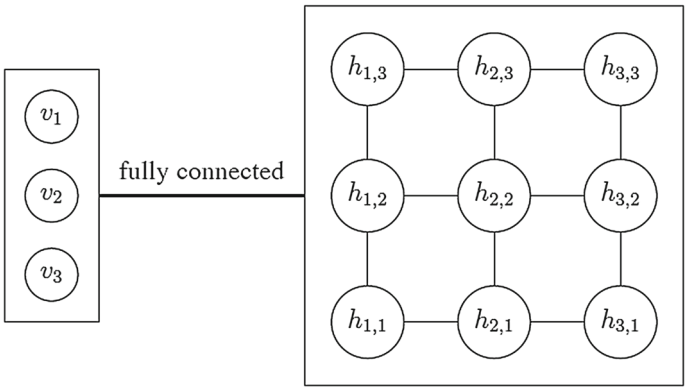


Fig. 11 Cellular semi-restricted Boltzmann machine (CSRBM)

Table 12 Factor between neighbor hidden variables ($\beta > 0$)

	$h_j = 0$	$h_j = 1$
$h_k = 0$	e^0	$e^{-\beta}$
$h_k = 1$	$e^{-\beta}$	e^0

5.2 Cellular Semi-Restricted Boltzmann Machine

DSOM can result in a good correlation between parameters learned by neighbor cells, but the downside is that it can only be used for datasets that can be expressed with binary features (like MNIST). Here we propose another approach based on restricted Boltzmann machines (RBMs) [11] which can be used for color images as well.

RBMs are generative models that can discover useful concepts from unlabeled data instances and they can be used as a layerwise pretraining method for deep models. However, RBM by itself is not useful for DCA because it doesn't force correlation on neighbor hidden variables.

The term semi-restricted Boltzmann machine has been used before [24] but it is used to indicate visible variables are connected together not the hidden variables. Here, we propose cellular semi-restricted Boltzmann machine (CSRBM). In CSRBM, visible units have no connection to each other; instead, hidden variables are connected to each other. Hidden variables are placed in a cellular structure and each hidden variable is connected to its neighbors (Fig. 11).

5.2.1 Algorithm

The purpose of this algorithm is to enforce a correlation between neighbor hidden variables. Therefore a factor should be defined between every two adjacent hidden variables. These factors play a regularization role and hence are not required to be parametric. Table 12 defines these factors. These factors are defined in a way that assign high probability to the cases where neighborhood hidden variables behave similarly, and low probability to the cases where they behave differently. After the learning process, these factors would be removed.

In CSRBM, the energy function would be

$$E(\mathbf{v},\mathbf{h}) = -\mathbf{v}^T\mathbf{W}\mathbf{h} - \mathbf{b}^T\mathbf{v} - \mathbf{c}^T\mathbf{h} + \sum_{N(h_j,h_k)} \beta 1_{h_j \neq h_k} \tag{12}$$

where $\mathbf{v}$ and $\mathbf{h}$ denote visible and hidden variables respectively. $\mathbf{W}$, $\mathbf{b}$ and $\mathbf{c}$ are parameters to be learned. $N(h_j, h_k)$ is the set of adjacent hidden variables. $1_{h_j \neq h_k}$ returns one if h_j and h_k are not equal and zero otherwise.

RBMs are easy to train because hidden variables are independent from each other conditioned on visible variables. Here, hidden variables are connected to each other and conditional probabilities are not easy to define. For this purpose, we divide hidden variables into two groups: even and odd variables (similar to black and white squares in a chess board). This trick makes variables in one group independent from each other conditioned on the variables of the other group. This concept is applicable to higher dimensions as well.

Let $h_j^{(o)}$ ($h_j^{(e)}$) denote that jth hidden unit is in odd (even) group. For the case where hidden variables are placed in a one dimensional structure, we will have

$$P\left(h_j^{(o)} = 1|\mathbf{h}^{(e)},\mathbf{v}\right) = \frac{1}{Z} exp\left(c_j h_j^{(o)} + \mathbf{v}^T\mathbf{W}_{:,j}h_j^{(o)} - \beta\left(1 - h_{j-1}^{(e)}\right) - \beta\left(1 - h_{j+1}^{(e)}\right)\right) \tag{13}$$

$$P\left(h_j^{(o)} = 0|\mathbf{h}^{(e)},\mathbf{v}\right) = \frac{1}{Z} exp\left(c_j h_j^{(o)} + \mathbf{v}^T\mathbf{W}_{:,j}h_j^{(o)} - \beta h_{j-1}^{(e)} - \beta h_{j+1}^{(e)}\right) \tag{14}$$

where Z is the partition function. If a hidden unit is on the edge, one of the local factors would be removed. Similar equation can be written for higher dimensions.

Let

$$u_j = c_j + \mathbf{v}^T\mathbf{W}_{:,j}, \tag{15}$$

Then we can write:

$$P\left(h_j^{(o)} = 1|\mathbf{h}^{(e)},\mathbf{v}\right) = \frac{P\left(h_j^{(o)} = 1|\mathbf{h}^{(e)},\mathbf{v}\right)}{P\left(h_j^{(o)} = 1|\mathbf{h}^{(e)},\mathbf{v}\right) + P\left(h_j^{(o)} = 0|\mathbf{h}^{(e)},\mathbf{v}\right)} \tag{16}$$

$$= \frac{exp\left(u_j\right) exp\left(-\beta\left(1 - h_{j-1}^{(e)}\right) - \beta\left(1 - h_{j+1}^{(e)}\right)\right)}{exp\left(u_j\right) exp\left(-\beta\left(1 - h_{j-1}^{(e)}\right) - \beta\left(1 - h_{j+1}^{(e)}\right)\right) + exp\left(-\beta h_{j-1}^{(e)} - \beta h_{j+1}^{(e)}\right)} \tag{17}$$

$$= \frac{exp\left(u_j\right)}{exp\left(u_j\right) + exp\left(-\beta\left(1 - 2h_{j-1}^{(e)}\right) - \beta\left(1 - 2h_{j+1}^{(e)}\right)\right)} \tag{18}$$

This probability is similar to that of RBM except an extra term is emerged. One can check that by setting $\beta = 0$ (no local factor), the RBM probability would be obtained.

Another problem that should be addressed is to generate positive samples. Here, we use "mean field approximation" which is used in deep Boltzmann machines (DBMs). The idea is to approximate probability $P(\mathbf{x})$ with another probability $Q(\mathbf{x})$ that assumes independence between variables and makes the calculations simpler:

$$Q\left(\mathbf{h}^{(o)},\mathbf{h}^{(e)}|\mathbf{v}\right) = \prod_j Q\left(\mathbf{h}_j^{(o)}|\mathbf{v}\right) \prod_k Q\left(\mathbf{h}_k^{(e)}|\mathbf{v}\right) \tag{19}$$

$$= \prod_j Q\left(\mathbf{h}_j|\mathbf{v}\right) \tag{20}$$

The learning process is as follows: first visible variables are initialized with a sample from dataset. Then, hidden variables are sampled from conditional probability without considering local factors. Now we begin mean field approximation phase. First the even hidden variables are sampled conditioned on all other variables, then odd hidden variables are sampled. This process is repeated until convergence. Now the value of variables can be used as positive samples. In the next phase, Gibbs sampling is performed (visible, even and odd variables are updated respectively). Here we initiate Gibbs sampling with the values obtained from mean field approximation phase. Algorithm 2 shows this procedure.

Algorithm 2 CSRBM training procedure

Set $\epsilon \in [0, 1]$, the learning rate
Set k, the number of Gibbs steps
while not converged (learning loop) **do**
  Sample a batch of m examples from training data and arrange them as a matrix $\mathbf{V}$
  Initialize $\mathbf{H}$ with $P(\mathbf{H}|\mathbf{V})$ (without considering cellular factors)
  while not converged (mean field inference loop) **do**
    Sample $\mathbf{H}^{(e)}$ from $P(\mathbf{H}^{(e)}|\mathbf{H}^{(o)}, \mathbf{V})$
    Sample $\mathbf{H}^{(o)}$ from $P(\mathbf{H}^{(o)}|\mathbf{H}^{(e)}, \mathbf{V})$
  end while
  Store $\mathbf{H}$ and $\mathbf{V}$ as positive statistics
  for $l = 1$ to k (Gibbs sampling) **do**
    Gibbs block 1:
    Sample $\mathbf{V}$ from $P(\mathbf{V}|\mathbf{H})$
    Gibbs block 2:
    Sample $\mathbf{H}^{(e)}$ from $P(\mathbf{H}^{(e)}|\mathbf{H}^{(o)}, \mathbf{V})$
    Gibbs block 3:
    Sample $\mathbf{H}^{(o)}$ from $P(\mathbf{H}^{(o)}|\mathbf{H}^{(e)}, \mathbf{V})$
  end for
  Store $\mathbf{H}$ and $\mathbf{V}$ as negative statistics
  Update parameters according to positive and negative samples with learning rate ϵ
end while

5.2.2 Results

First this technique is used on MNIST dataset with a fully connected model (1×16 cellular structure). Training is performed on 2000 batches each with 20 images. Learning rate and β were set to 0.05 and 3 respectively. β was decreased exponentially toward zero through the process. Learned parameters and correlation between hidden variables are shown in Figs. 12 and 13a respectively. The correlation for the model trained with RBM is presented in Fig. 13b. As can be seen, CSRBM results in a better local correlation. The reconstruction capability of the model is presented in Fig. 14.

In the next experiment, CSRBM is used to pretrain the RCNN model specified in Table 13. The reason that the earlier model cannot be used is that CSRBM requires sigmoid as activation function. This function hugely reduces the performance of the model; therefore a model with less layers is more preferable.

The first layer pretrained with CSRBM is presented in Fig. 15. This figure shows the parameters through learning procedure. As can be seen, neighbor concepts are similar. Then model is trained using back-propagation and finally the accuracy $98.244 \pm 0.0677\%$ is obtained. This accuracy is less than previous accuracies and the main reason is sigmoid function. This

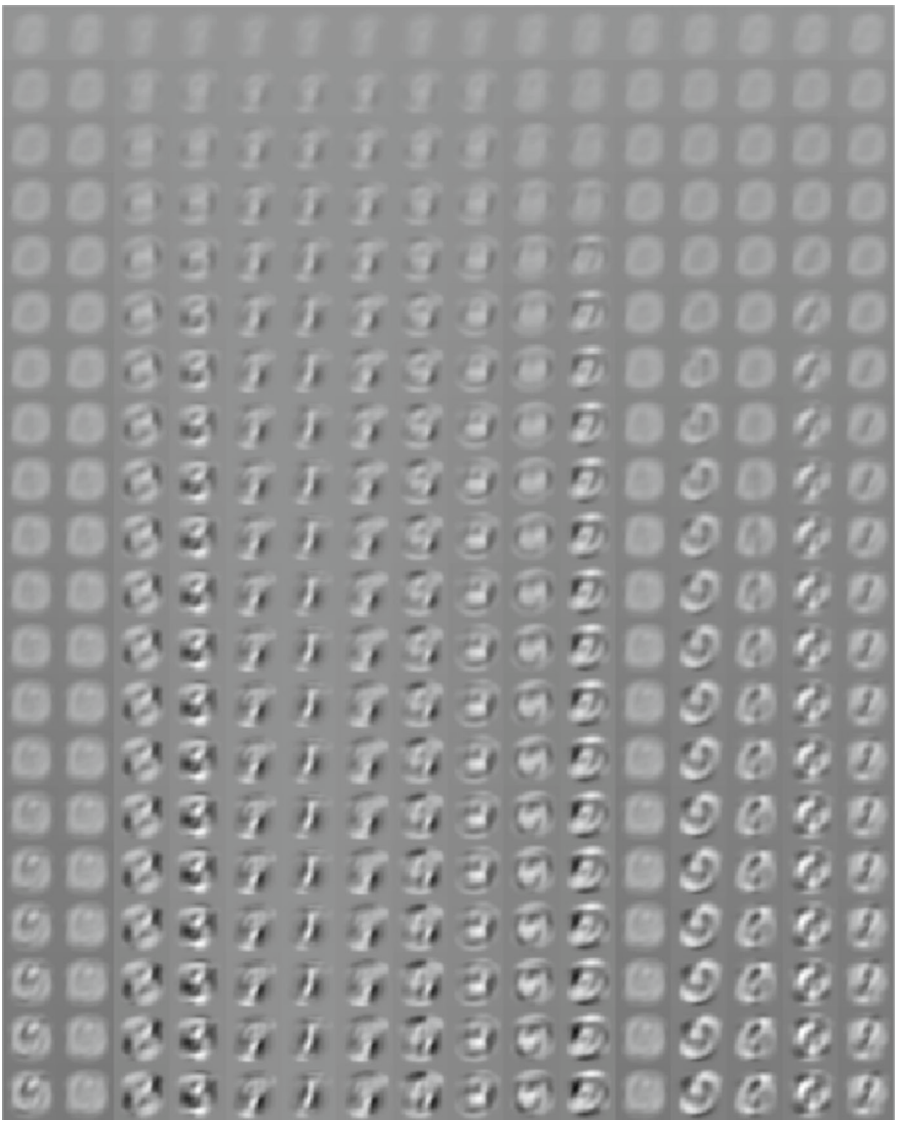


Fig. 12 Pretraining a fully connected 1×16 cellular structure on MNIST with CSRBM. Each column is a hidden variable and each row is a time step

function has zero slope in $+\infty$ and $-\infty$ and this prevents gradient to properly be passed to previous layers.

To see whether local correlation actually is beneficial for DCA, we performed the same process with RBM. Model finally reaches an accuracy of $98.225 \pm 0.074\%$ which is a bit lower than CSRBM (p value obtained by T test is 0.5759). The difference is not big since as argued earlier, in MNIST, different digits have similar parts and each concept in layer can be useful for most concepts in the next layer.

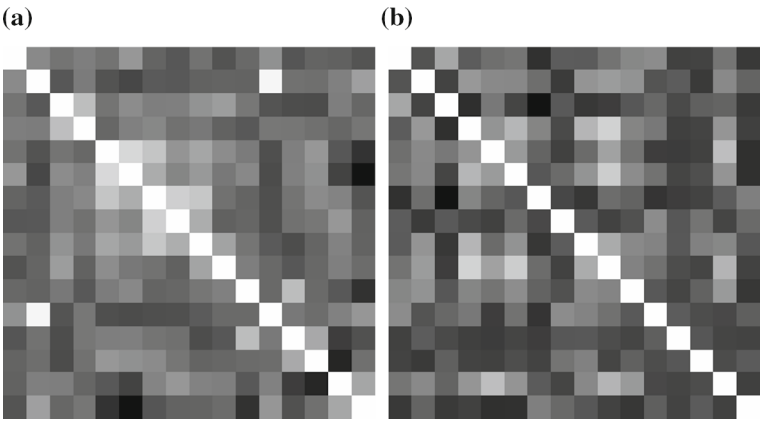


Fig. 13 A comparison between correlation of hidden variables trained with CSRBM and RBM. The results suggest that CSRBM has obtained better local correlation. **a** CSRBM. **b** RBM

Fig. 14 Reconstruction capability of CSRBM. Errors are due to the low number of hidden variables. RBM reaches similar results. **a** Samples from MNIST dataset. **b** Reconstruction with CSRBM

Finally, CSRBM is used on CIFAR-10 dataset with RCNN model defined in Table 5 (relu functions in cellular layers are replaced with sigmoid functions). Figure 16 shows the parameters learned in the first layer. After pretraining, model is trained with back-propagation and reaches $58.964 \pm 0.275\%$ accuracy. Then, RBM is used to pretrain this model and in this case the final accuracy obtained is $54.806 \pm 0.625\%$. CSRBM has achieved 4 percent higher accuracy than RBM. Again it can be concluded that in more complicated datasets, local correlation is more important and more beneficial.

It should be noted that the results obtained with CSRBM are no comparable with results of experiments without pretraining; since CSRBM uses sigmoid activation function and this

Table 13 RCNN model to classify MNIST and pretrain with CSRBM

Layer	Cell type	r_{-1}	$\mathcal{D}$	Layer size	Activation
1	Perceptron	(5, 5, 1)	(1, 1, 16)	(24, 24, 16)	–
2	Max pooling	(2, 2, 1)	$\left(\frac{1}{2}, \frac{1}{2}, 1\right)$	(12, 12, 16)	Sigmoid
3	Perceptron	(5, 5, 7)	(1, 1, 6)	(8, 8, 60)	Sigmoid
4	Perceptron (fully connected)	–	–	(256)	Relu
5	Perceptron (fully connected)	–	–	(128)	Relu
6	Perceptron (fully connected)	–	–	(10)	Relu

Fig. 15 Parameters of the first layer pretrained with CSRBM on MNIST. Local correlation has been achieved

Fig. 16 Parameters of the first layer pretrained with CSRBM on CIFAR-10. Some parameters are negative, therefore colors are not accurately represented. (Color figure online)

is a serious drawback. However it might be possible to change CSRBM in a way to use ReLU activation function. Such method can even outperform DSOM. Therefore, CSRBM can be a starting point for further research in this regard.

6 Structural Variations

The formal definition proposed for DCA enables one to define very different models with different structures. The benefit of such a formal definition is that specifying different structures can become simple and straight forward. In this section some structural variations are introduced and evaluated.

6.1 Multidimensional Channels

The RCNN model enforces a local connection between feature maps. This means that feature maps are ordered. This has an interesting implication, that is a layer can be represented with multi-dimensional channels (Fig. 17).

Table 14 defines a model with two dimensional channels for classifying MNIST. This model is first pretrained with two dimensional DSOM. Then it is trained with back-propagation. Model finally achieves 99.05% accuracy.

6.2 Deep Neighborhood

The neighborhood function defined for DCA model is not restricted to only the previous layer. One can easily extend the neighborhood to contain multiple layers. Here we define an

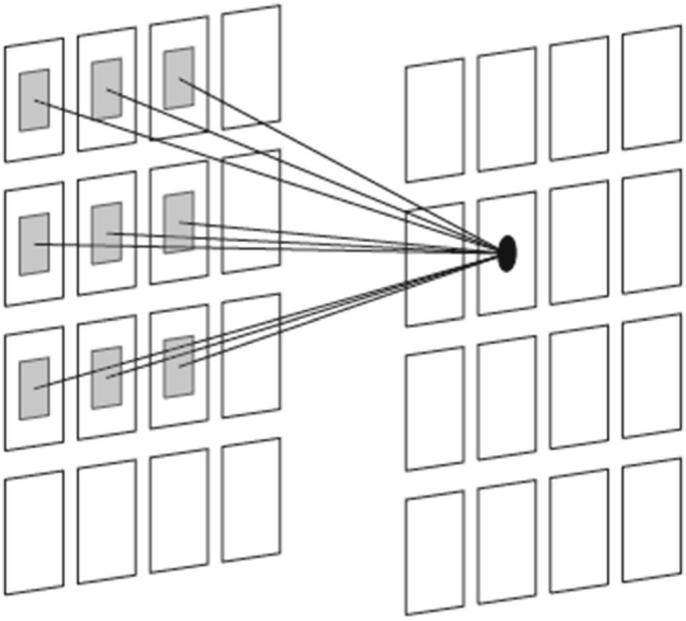


Fig. 17 RCNN with two dimensional channels. Each rectangle represents a feature map. RCNN allows feature maps to be arranged in multiple dimensions

Table 14 RCNN with two dimensional channels to classify MNIST dataset

Layer	Cell type	r_{-1}	$\mathcal{D}$	Layer size	Activation
1	Perceptron	(5, 5, 1, 1)	(1, 1, 3, 3)	(24, 24, 3, 3)	–
2	Max pooling	(2, 2, 1, 1)	$\left(\frac{1}{2}, \frac{1}{2}, 1, 1\right)$	(12, 12, 3, 3)	Relu
3	Perceptron	(5, 5, 2, 2)	(1, 1, 3, 3)	(8, 8, 6, 6)	Relu
4	Perceptron (fully connected)	–	–	(128)	Relu
5	Perceptron (fully connected)	–	–	(64)	Relu
6	Perceptron (fully connected)	–	–	(10)	Relu

RCNN model with deeper neighborhood (Table 15). This model is used to classify MNIST dataset. By training with back-propagation, 98.42% accuracy is obtained.

6.3 Periodic Boundary Conditions

A downside of all the proposed models so far is that the contribution of cells is not uniform; The central cells participate in more connections contrary to the boundary cells. In CA, boundary conditions can be defined periodic; In this condition, the most right cells are neighbors of the most left cells and vice versa. This trick will make all the connections uniform. Figure 18 shows a RCNN with periodic boundary conditions.

The models for classifying MNIST and CIFAR-10 datasets are presented in Tables 16 and 17 respectively. An accuracy of 99.24% on MNIST and 65.90% on CIFAR-10 was obtained. Both results are better than non-periodic RCNNs.

Table 15 RCNN with deep neighborhood

Layer	Cell type	r_{-1}	r_{-2}	$\mathcal{D}$	Layer size	Activation
1	Perceptron	(5, 5, 1)	–	(1, 1, 5)	(24, 24, 5)	Relu
2	Perceptron	(5, 5, 3)	(9, 9, 1)	(1, 1, 3)	(20, 20, 9)	–
3	Max pooling	(2, 2, 1)	–	$\left(\frac{1}{2}, \frac{1}{2}, 1\right)$	(10, 10, 9)	Relu
4	Perceptron	(5, 5, 5)	–	(1, 1, 4)	(6, 6, 20)	Relu
5	Perceptron	(5, 5, 11)	(9, 9, 6)	(1, 1, 3)	(2, 2, 30)	Relu
6	Perceptron (fully connected)	–	–	–	(64)	Relu
7	Perceptron (fully connected)	–	–	–	(10)	–

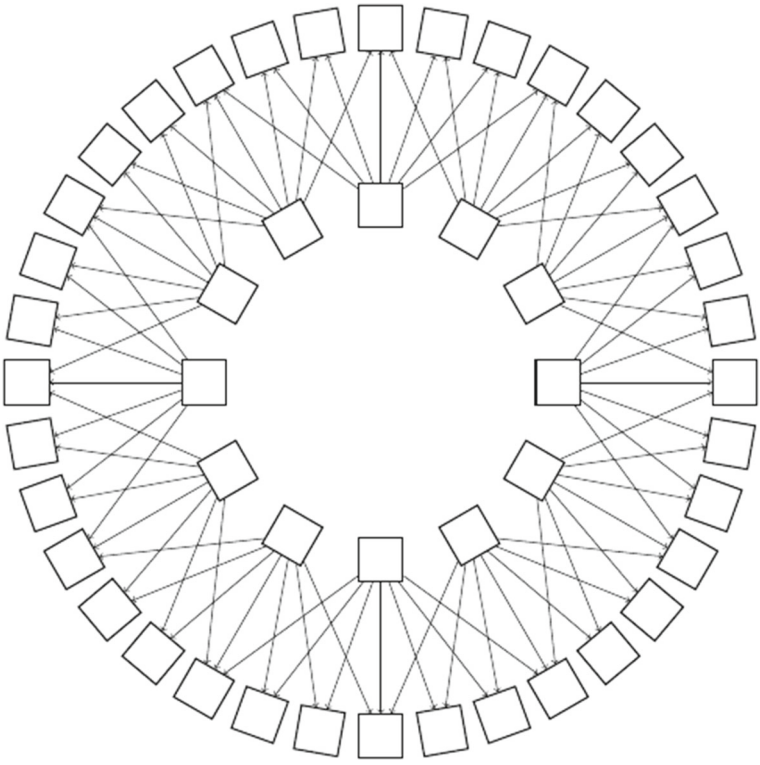


Fig. 18 RCNN with periodic boundary conditions. Each rectangle represents a feature map

7 Conclusions

In this paper, a cellular automata approach to deep learning was introduced. First a special CA called "dense cellular automaton" was proposed and based on that, deep cellular automaton (DCA) and restricted convolutional neural network (RCNN) were suggested. RCNN was experimentally compared with CNN on MNIST and CIFAR-10 datasets and it was observed that RCNN can perform better on CIFAR-10. Next, two pretraining methods were proposed referred to as "discriminative self-organizing map" (DSOM) and "cellular semi-restricted

Table 16 RCNN model with periodic boundary conditions for classifying MNIST

Layer	Cell type	r_{-1}	$\mathcal{D}$	Layer size	Activation
1	Perceptron	(5, 5, 1)	(1, 1, 16)	(24, 24, 16)	–
2	Max pooling	(2, 2, 1)	$\left(\frac{1}{2}, \frac{1}{2}, 1\right)$	(12, 12, 16)	Relu
3	Perceptron (periodic)	(5, 5, 9)	(1, 1, 4)	(8, 8, 64)	Relu
4	Max pooling	(2, 2, 1)	$\left(\frac{1}{2}, \frac{1}{2}, 1\right)$	(4, 4, 64)	Relu
5	Perceptron (fully connected)	–	–	(200)	Relu
6	Perceptron (fully connected)	–	–	(100)	Relu
7	Perceptron (fully connected)	–	–	(10)	–

Table 17 RCNN model with periodic boundary conditions for classifying CIFAR-10

Layer	Cell type	r_{-1}	$\mathcal{D}$	Layer size	Activation
1	Perceptron	(5, 5, 3)	(1, 1, 24)	(28, 28, 24)	–
2	Max pooling	(2, 2, 1)	$\left(\frac{1}{2}, \frac{1}{2}, 1\right)$	(14, 14, 24)	Relu
3	Perceptron (periodic)	(5, 5, 18)	(1, 1, 3)	(10, 10, 72)	Relu
4	Max pooling	(2, 2, 1)	$\left(\frac{1}{2}, \frac{1}{2}, 1\right)$	(5, 5, 72)	Relu
5	Perceptron (fully connected)	–	–	(384)	Relu
6	Perceptron (fully connected)	–	–	(192)	Relu
7	Perceptron (fully connected)	–	–	(10)	–

Boltzmann machine" (CSRBM). It was showed that these approaches can be used to increase the performance of the model. Finally, three structural variations were introduced: RCNN with two dimensional channels, RCNN with deep neighborhood and RCNN with periodic boundary conditions. These variations obtained good results.

It is important to understand that RCNN is not just a "learning" technique for CNN (like dropout or batch-normalization); Rather, it proposes a better "representation" for image datasets by enforcing infinitely strong priors on CNN. This is similar to CNNs compared to multi-layer perceptrons (MLPs). CNNs provide better representation for images compared to MLPs. The real advantage of our model is not that it achieves higher accuracy; but that it shows that many parameters in CNNs are redundant and unnecessary (i.e. by eliminating them, the performance does not decrease). This is really important since over time, CNNs are becoming larger and deeper, and hence more costly to train and more sensitive to overfitting. Eliminating unnecessary parameters, can become a necessity in this process.

Manifold hypothesis (MH) is an important subject in machine learning in general and deep learning in particular. This hypothesis states that natural data instances reside on high dimensional manifolds. Although this hypothesis has not been proven but some arguments can be made to justify it. For example, the brightness of an image can be changed continuously and by doing so, new valid data instances can be generated. Other transformations like rotation and scaling have similar effects.

It seems that another form of manifold hypothesis can be suggested, stating that "features" of natural data reside on manifolds. Let us refer to this hypothesis as extended manifold hypothesis (EMH). EMH is evident in for example images, since by continuously sliding

a window on an image, the concept would also change continuously. Models like CNNs recognize this kind of manifold and by structurally imposing an infinitely strong prior over the parameters, they can learn very good concepts and achieve very good results.

The question that arises is that whether EMH is correct on dimensions other than spatial dimensions. Can features extracted from an image be ordered in a way that would form a manifold? That is, if a window is continuously moved in channel dimension (in a higher layer), its meaning would also change continuously. This idea cannot be immediately rejected, because for example Gabor filters which usually arise in the first layer of CNNs, can form a manifold; Or higher concepts like 'dog' and 'wolf', reside on a conceptual spectrum. The RCNN model proposed here is intended to expose this kind of manifold. If EMH is true in all dimensions, then RCNNs would be preferable to CNNs.

To continue this research, one can use DCA on larger datasets with deeper layers and on applications like speech recognition and natural language processing.

References

1. Baldi P (2012) Autoencoders, unsupervised learning, and deep architectures. In: Proceedings of ICML workshop on unsupervised and transfer learning, pp 37–49
2. Beigy H, Meybodi MR (2004) A mathematical framework for cellular learning automata. Adv Complex Syst 7(03n04):295–319
3. Chua LO, Yang L (1988) Cellular neural networks: applications. IEEE Trans Circuits Syst 35(10):1273–1290
4. de Korte A, Brouwers H (2013) A cellular automata approach to chemical reactions: 1 reaction controlled systems. Chem Eng J 228:172–178. https://doi.org/10.1016/j.cej.2013.04.084
5. Gardner M (1970) Mathematical games: the fantastic combinations of john conways new solitaire game life. Sci Am 223(4):120–123
6. Gehring WJ (2005) New perspectives on eye development and the evolution of eyes and photoreceptors. J Hered 96(3):171–184
7. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge
8. Guo Y, Yao A, Chen Y (2016) Dynamic network surgery for efficient DNNs. In: Lee DD, Sugiyama M, Luxburg UV, Guyon I, Garnett R (eds) Advances in neural information processing systems 29. Curran Associates, Inc., pp 1379–1387
9. Han S, Pool J, Tran J, Dally W (2015) Learning both weights and connections for efficient neural network. In: Advances in neural information processing systems, pp 1135–1143
10. He K, Zhang X, Ren S, Sun J (2015) Deep residual learning for image recognition. CoRR. arXiv:1512.03385
11. Hinton G (2010) A practical guide to training restricted boltzmann machines. Momentum 9(1):926
12. Hinton GE, Osindero S, Teh YW (2006) A fast learning algorithm for deep belief nets. Neural Comput 18(7):1527–1554
13. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–507
14. Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Netw 2(5):359–366. https://doi.org/10.1016/0893-6080(89)90020-8
15. Huberman B (1985) Probabilistic cellular automata. In: Nonlinear phenomena in physics. Springer, pp 129–137
16. Jump JR, Kirtane JS (1974) On the interconnection structure of cellular networks. Inf Control 24(1):74–91
17. Kohonen T (1982) Self-organized formation of topologically correct feature maps. Biol Cybern 43(1):59–69
18. Krizhevsky A, Hinton G (2009) Learning multiple layers of features from tiny images. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf; https://www.cs.toronto.edu/~kriz/cifar.html
19. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
20. LeCun Y, Cortes C, Burges CJ (2010) Mnist handwritten digit database. AT&T Labs (Online). http://yann.lecun.com/exdb/mnist
21. Margenstern M (2007) Cellular automata in hyperbolic spaces: theory, vol 1. Archives contemporaines

22. McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys 5(4):115–133
23. Molchanov P, Tyree S, Karras T, Aila T, Kautz J (2016) Pruning convolutional neural networks for resource efficient transfer learning. CoRR. arXiv:1611.06440
24. Osindero S, Hinton GE (2008) Modeling image patches with a directed hierarchy of Markov random fields. In: Platt JC, Koller D, Singer Y, Roweis ST (eds) Advances in neural information processing systems 20. Curran Associates, Inc., pp 1121–1128
25. O'Sullivan D (2001) Exploring spatial process dynamics using irregular cellular automaton models. Geogr Anal 33(1):1–18
26. Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65(6):386
27. Sarkar P (2000) A brief history of cellular automata. ACM Comput Surv (CSUR) 32(1):80–107
28. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1–9
29. Toffoli T, Margolus N (1987) Cellular automata machines: a new environment for modeling. MIT Press, Cambridge
30. Von Neumann J, Burks AW et al (1966) Theory of self-reproducing automata. IEEE Trans Neural Netw 5(1):3–14
31. Wang L, Zhang J, Shao H (2014) Existence and global stability of a periodic solution for a cellular neural network. Commun Nonlinear Sci Numer Simul 19(9):2983–2992
32. Werbos PJ (1974) Beyond regression: new tools for prediction and analysis in the behavioral sciences. Doctoral dissertation, Applied Mathematics, Harvard University, MA
33. Widrow B, Hoff ME (1960) Adaptive switching circuits. Technical report, Stanford University, Stanford Electronics Labs
34. Wolfram S (1994) Cellular automata and complexity: collected papers, vol 1. Addison-Wesley, Reading
35. Wolfram S (2002) A new kind of science, vol 5. Wolfram Media, Champaign
36. Yang T, Yang LB, Wu CW, Chua LO (1996) Fuzzy cellular neural networks: theory. In: 1996 4th IEEE international workshop on cellular neural networks and their applications, 1996. CNNA-96. Proceedings. IEEE, pp 181–186