Pattern Recognition 47 (2014) 114–127

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

# Towards adaptive learning with improved convergence of deep belief networks on graphics processing units

Noel Lopes [a,b,*], Bernardete Ribeiro [a,c]

[a] CISUC – Center for Informatics and Systems of University of Coimbra, Portugal
[b] UDI/IPG – Research Unit, Polytechnic of Guarda, Portugal
[c] Department of Informatics Engineering, University of Coimbra, Portugal

ARTICLE INFO

Available online 4 July 2013

*Keywords:*
Deep learning
Deep belief networks
Restricted Boltzmann machines
Contrastive divergence
Adaptive step size
GPU computing

ABSTRACT

In this paper we focus on two complementary approaches to significantly decrease pre-training time of a deep belief network (DBN). First, we propose an adaptive step size technique to enhance the convergence of the contrastive divergence (CD) algorithm, thereby reducing the number of epochs to train the restricted Boltzmann machine (RBM) that supports the DBN infrastructure. Second, we present a highly scalable graphics processing unit (GPU) parallel implementation of the CD-*k* algorithm, which boosts notably the training speed. Additionally, extensive experiments are conducted on the MNIST and the HHreco databases. The results suggest that the maximum useful depth of a DBN is related to the number and quality of the training samples. Moreover, it was found that the lower-level layer plays a fundamental role for building successful DBN models. Furthermore, the results contradict the pre-conceived idea that all the layers should be pre-trained. Finally, it is shown that by incorporating multiple back-propagation (MBP) layers, the DBNs generalization capability is remarkably improved.



## 1. Introduction

Recent advances in deep learning methods have led to a widespread enthusiasm among pattern recognition and machine learning (ML) researchers [1,2]. Inspired by the depth structure of the brain, deep learning architectures encompass the promise of revolutionizing and widening the range of tasks performed by computers [1]. In recent months deep learning applications have been growing both in number and accuracy [1]. Moreover, just a few months ago, a team of graduate students of Geoffrey E. Hinton won the top prize in a contest aiming at finding molecules that might lead to new drugs. This was a particularly impressive achievement because never before a deep learning architecture based-system had won a similar competition and the software was designed with no prior knowledge on how the molecules bind to their targets, using only a relatively small dataset [1].

Deep models reflect many levels of composition of non-linear operations in their outputs [2–4]. The idea is to have feature detector units at each layer (level) that gradually extract more sophisticated and invariant features from the original raw input signals. Lower layers aim at extracting simple features that are then clamped into higher layers, which in turn detect more complex features [5]. In contrast, shallow models (e.g. two-layers neural network (NNs), support vector machine (SVMs)) present very few layers that map the original input features into a problem-specific feature space [2,6].

Deep architectures can be exponentially more efficient than shallow ones [7]. The latter may require a huge number of elements to represent highly varying functions [2–4]. On the other hand deep architectures can represent these functions efficiently, in particular when their Kolmogorov complexity is small [2]. Since each element of the architecture is learned using examples, the number of computational elements one can afford is limited by the number of training samples [4]. Thus, the depth of architecture can be very important from the point of view of statistical efficiency. Hence, using shallow architectures may result in poor generalization models [4]. As a result, deep models tend to outperform shallow models such as SVMs [2]. Moreover, theoretical results suggest that deep architectures are fundamental to learn the kind of complex functions that can represent high-level abstractions (e.g. vision, language) [4], characterized by many factors of variation that interact in non-linear ways, making the learning process difficult [2].

However, the challenge of training deep NNs remained elusive for a long time [4], until the development of DBNs [8] which were successfully applied to several domains including classification, regression, dimensionality reduction, object segmentation, information retrieval, language processing, robotics, speech, audio, and collaborative filtering [2–4,9,6] thus demonstrating its ability to often outperform state of the art algorithms in these areas [4].

Nevertheless, training a DBN is a computationally expensive task that involves training independently several RBMs and

* Corresponding author at: UDI/IPG – Research Unit, Polytechnic of Guarda, Portugal. Tel.: +351 271222690; fax: +351 271220100.
*E-mail addresses:* noel@ipg.pt (N. Lopes), bribeiro@dei.uc.pt (B. Ribeiro).

0031-3203/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.patcog.2013.06.029

requires a considerable amount of time and effort [10,11]. Moreover, the proper choice of the learning parameters is a fundamental aspect that affects considerably the networks convergence [10]. Recently, there has been a renewed interest in accelerating the training of NNs [12]. In particular, concerning the RBMs, several approaches relying on customized hardware (Field-Programmable Gate Array (FPGAs)) [13,12] and GPU [14,15] have been proposed. In our view, the GPU represents the most compelling option, since dedicated hardware fails to meet the expectations, as it is typically expensive, unreliable, poorly documented, with reduced flexibility, and obsolete within a few years [16]. Additionally, the FPGA implementations cannot be shared and validated by other researchers who probably do not have access to the hardware. GPUs on the other hand are widely available and relatively inexpensive [16–18].

In this paper we present two complementary approaches to speedup the training of RBMs and DBNs. First, an adaptive step size technique that solves the difficulty of choosing an adequate learning rate and momentum terms, while enhancing the training convergence, is presented. Second, we rely on a multi-core GPU parallel implementation of the CD algorithm to speedup the training process. The resulting implementation is unique in that it incorporates the proposed adaptive step size technique. Moreover, unlike other implementations, we have made our code open-source so that others can readily use and improve it. Finally, we use the resulting tool to analyze the effects of varying the number of layers and neurons of a DBN.

This paper is structured as follows. Section 2 details both the DBN and RBM networks. Section 3 presents the proposed adaptive step size technique. Section 4 describes the GPU parallel implementation. Section 5 asserts the validity of both approaches on speeding up the training process and analyzes the effects of varying the number of layers and neurons in a DBN. Finally Section 6 draws the conclusions and points out future work.

## 2. Deep belief network

A DBN is composed of several RBM layers. Each RBM receives the inputs of the previous layer and feeds the RBM in the next layer. Hence, training a DBN consists of independently training each one of the RBMs, starting by the lower-level RBM and progressively moving up in the hierarchy.

### 2.1. Restricted Boltzmann machine

An RBM is an energy-based generative model that consists of a layer of $I$ binary visible units (observed variables), $\mathbf{v}\in\{0,1\}^I$, and a layer of $J$ binary hidden units (explanatory factors), $\mathbf{h}\in\{0,1\}^J$, with bidirectional weighted connections [19], as depicted in Fig. 1. RBMs follow the encoder–decoder paradigm [20] where both the encoded representation and the (decoded) reconstruction are stochastic by nature. The encoder–decoder architecture is useful because: (i) after training, the feature vector can be computed in a very fast way and (ii) by reconstructing the input we can assess how well the model was able to capture the relevant information from the data [20].

Given an observed state, the energy of the joint configuration of the visible and hidden units $(\mathbf{v},\mathbf{h})$ is given by (1)

$$E(\mathbf{v},\mathbf{h}) = -\mathbf{c}\mathbf{v}^\top - \mathbf{b}\mathbf{h}^\top - \mathbf{v}^\top\mathbf{W}\mathbf{h} = -\sum_{i=1}^{I} c_i v_i - \sum_{j=1}^{J} b_j h_j - \sum_{j=1}^{J}\sum_{i=1}^{I} W_{ji} v_i h_j, \tag{1}$$

where $\mathbf{c}\in\mathbb{R}^I$ represents the bias of the visible units, $\mathbf{b}\in\mathbb{R}^J$ the bias of the hidden units and $\mathbf{W}\in\mathbb{R}^{J\times I}$ a matrix containing the RBM connection weights. In order to break symmetry, typically the weights are initialized with small random values (e.g. between $-0.01$ and 0.01) [19]. The hidden bias, $b_j$, can be initialized with a large negative value (e.g. $-4$) in order to encourage sparsity and the visible units bias, $c_i$, to $\log(\hat{p}_i/(1-\hat{p}_i))$, where $\hat{p}_i$ is the proportion of training vectors in which $v_i=1$ [19]. Fig. 2 shows the advantages of initializing $c_i$ in this manner.

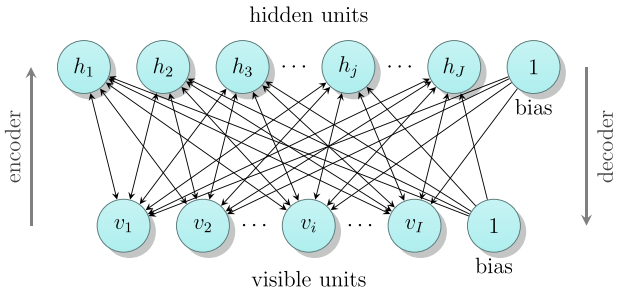


**Fig. 1.** Schematic representation of a restricted Boltzmann machine (RBM).

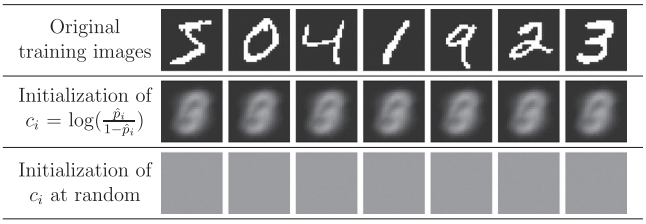


**Fig. 2.** Reconstruction of the MNIST digits made by a newly initialized restricted Boltzmann machine (RBM) ($\hat{p}_i$ is the proportion of vectors in which the pixel $i$ is on).

The RBM assigns a probability for each configuration $(\mathbf{v},\mathbf{h})$, using (2)

$$p(\mathbf{v},\mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z}, \tag{2}$$

where $Z$ is a normalization constant called *partition function* by analogy with physical systems, given by the sum of all energy configurations [4,19,21]

$$Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}. \tag{3}$$

Since there are no connections between any two units within the same layer, given a particular random input configuration, $\mathbf{v}$, all the hidden units are independent of each other and the probability of $\mathbf{h}$ given $\mathbf{v}$ becomes

$$p(\mathbf{h}|\mathbf{v}) = \prod_j p(h_j = 1|\mathbf{v}), \tag{4}$$

where

$$p(h_j = 1|\mathbf{v}) = \sigma\left(b_j + \sum_{i=1}^{I} v_i W_{ji}\right), \tag{5}$$

and $\sigma(x)$ is the sigmoid function $1/(1+e^{-x})$. For implementation purposes, $h_j$ is set to 1 when $p(h_j=1|\mathbf{v})$ is greater than a given random number (uniformly distributed between 0 and 1) and 0 otherwise. Similarly given a specific hidden state, $\mathbf{h}$, the probability of $\mathbf{v}$ given $\mathbf{h}$ is given by (6)

$$p(\mathbf{v}|\mathbf{h}) = \prod_i p(v_i = 1|\mathbf{h}), \tag{6}$$

where

$$p(v_i = 1|\mathbf{h}) = \sigma\left(c_i + \sum_{j=1}^{J} h_j W_{ji}\right). \tag{7}$$