Deep Learning with Random Neural Networks

Erol Gelenbe and Yonghua Yin
Intelligent Systems and Networks Group
Electrical & Electronic Engineering Department
Imperial College, London SW7 2AZ, UK

***Abstract*—This paper introduces techniques for Deep Learning in conjunction with spiked random neural networks that closely resemble the stochastic behaviour of biological neurons in mammalian brains. The paper introduces clusters of such random neural networks and obtains the characteristics of their collective behaviour. Combining this model with previous work on extreme learning machines, we develop multilayer architectures which structure Deep Learning Architectures a a "front end" of one or two layers of random neural networks, followed by an extreme learning machine. The approach is evaluated on a standard – and large – visual character recognition database, showing that the proposed approach can attain and exceed the performance of techniques that were previously reported in the literature.**

I. Introduction

In recent years, deep learning with regular and tightly-coupled arrays of sigmoidal neurons has come to the forefront as a possible way to overcome the limitations of neural networks when they are applied to real-world challenges [1], [2]., while many engineering applications require significant advances in learning from big data [3], [4], [5], [6].

Tightly coupled clusters of *natural neuronal cells* communicate with each other in multiple ways, both through spiking [7], via soma-type interactions with other cells [8], through neuromodulators [9], and with the help of important structures such as glial cells [10] which are known to exercise complex functions in the hippocampus and cerebellum that contribute to synaptic transmission and modulate synaptic function. This complexity of natural neural information processing and learning [11] goes well beyond the models traditionally exploited in machine learning [12], and goes significantly beyond the capabilities of sigmoid-based neural models.

The Random Neural Network (RNN) [13] is a spiking "integrate and fire" model where an arbitrarily large set of cells interact with each other via excitatory and inhibitory spikes which modify each cell's action potential in continuous time, and mathematically described by a system of differential equations known as the Chapman-Kolmogorov equations [14]. It was originally developed to mimic the behaviour of biological neurons [15]. However subsequently it was exploited for numerous applications that exploit the *recurrent structure* of the network and of its learning algorithm [16], including combinatorial optimisation [17], several examples of image and video processing [18], [19], [20], [21], [22], [23], and routing [24], [25], [26] in cyber-physical systems and computer networks. All these applications exploit the *recurrent structure* of the network.

The computational power of the RNN originates in the fact that in steady-state the network is characterised by the joint probability distribution of the activation state of each cell, which is equal to the product of the marginal probabilities of the activation states. This property, known as "product form" in the probability literature [14] makes the RNN particularly amenable to exact yet simple, fast (and easily parallisable) computational algorithms.

II. The Mathematical Model

We consider the Random Neural Network Model developed in [27], [28], composed of M neurons or cells, each of which receives excitatory (positive) and inhibitory (negative) spike trains from external sources which may be sensory sources or cells. These arrivals occur according to independent Poisson processes of rates λ_m^+ for the excitatory spike train, and λ_m^- for the inhibitory spike train, respectively, to cell $m \in \{1, \, .. \, , M\}$.

In this model, ach neuron is represented at time $t \geq 0$ by its internal state $k_m(t)$ which is a non-negative integer. If $k_m(t) > 0$, then the arrival of a negative spike to neuron m at time t results in the reduction of the internal state by one unit: $k_m(t^+) = k_m(t) - 1$. The arrival of a negative spike to a cell has no effect if $k_m(t) = 0$. On the other hand, the arrival of an excitatory spike always increases the neuron's internal state by $+1$.

If $k_m(t) > 0$, then the neuron m is said to be "excited", and it may "fire" a spike with probability $r_m \Delta t$ in the interval $[t, t + \Delta t[$, where $r_m > 0$ is its "firing rate", so that r_m^{-1} may be viewed as the average firing delay of the excited $m - th$ neuron.

Neurons in this model can interact in the following manner at time $t \geq 0$. If neuron i is excited, i.e. $k_i(t) > 0$, then when i fires its internal state drops by 1 and we have $k_i(t^+) = k_i(t) - 1$, and:

- It can either send a positive or excitatory spike to neuron j with probability $p^+(i, j)$ resulting in $k_i(t^+) = k_i(t) - 1$ and $k_j(t^+) = k_j(t) + 1$,
- Or it may send a negative or inhibitory spike to neuron j with probability $p^-(i, j)$ so that $k_i(t^+) = k_i(t) + 1$ and $k_j(t^+) = k_j(t) - 1$ if $k_j(t) > 0$, else $k_j(t^+) = 0$ if $k_j(t) = 0$,
- Or neuron i can "trigger" neuron j with probability $p(i, j)$ so that $k_i(t^+) = k_i(t) - 1$ and $k_j(t^+) = k_j(t) - 1$ if $k_j(t) > 0$.
- When neuron i triggers neuron j, both $k_i(t^+) = k_i(t) - 1$ and $k_j(t^+) = k_j(t) - 1$, and one of two things may

happen. Either:

- (A) With probability $Q(j,m)$ we have $k_m(t^+) = k_m(t)+1$ so that i and j together have incremented the state of m. Thus we see that a trigger allows two neurons i and j to increase the excitation level of a third neuron m by $+1$, while i and j are both depleted by -1.
- (B) Or with probability $\pi(j,m)$ the trigger moves on to the neuron m and then with probability $Q(m,l)$ the sequence (A) or (B) is repeated.

- Note that $\sum_{j=1}^{M}[p(i,j)+p^-(i,j)+p^+(i,j)] = 1-d_i$ where d_i is the probability that when the neuron i fires, the corresponding spike or trigger is lost or it leaves the network. Also, $1 = \sum_{m=1}^{M}[Q(j,m)+\pi(j,m)]$.

Since cells in different layers of mammalian brain also communicate through simultaneous firing patterns of densely packet somas, the RNN was extended in [29], [28] using a branch of the theory of stochastic networks called G-Networks [30]. In the sequel we will exploit this structure for deep learning.

III. Modelling Soma to Soma Interactions

Now let $z(m) = (i_1, \ ... \ , i_l)$ be any ordered sequence of *distinct* numbers $i_j \in S; i_j \neq m$; obviously $1 \leq l \leq M-1$. Let us denote by $q_m = \lim_{tarrow\infty} Prob[k_m(t) > 0]$ the probability that the neuron m is excited. It is given by the following expression [27], [30]:

$$q_m = \frac{\Lambda_m^+}{r_m + \Lambda_m^-}, \tag{1}$$

where the variables in (1) are of the form:

$$\Lambda_m^+ = \lambda_m^+ + \sum_{j=1, j\neq m}^{M} r_j q_j p^+(j,m) + \tag{2}$$

$$+ \sum_{all \ z(m)} r_{i_1} \prod_{j=1, \ ... \ ,l-1} q_{i_j} p(i_j, i_{j+1}) Q(i_{j+1}, m), \tag{3}$$

$$\Lambda_m^- = \lambda_m^- + \sum_{j=1, j\neq m}^{M} r_j q_j p^-(j,m) \tag{4}$$

$$+ \sum_{all \ z(m)} r_{i_1} \prod_{j=1, \ ... \ ,l-1} q_{i_j} p(i_j, i_{j+1}) p(i_{j+1}, m). \tag{5}$$

In the sequel, to simplify the notations we will write $w_{ji}^+ = r_r p^+(j,i)$ and $w_{ji}^- = r_r p^-(j,i)$.

A. Clusters of Identical and Densely Connected Cells

Let us now consider the construction of special clusters of densely interconnected cells. We first consider a special network, call it $\mathbf{M}(n)$, that contains n identically connected cells, each of which has firing rate r and external inhibitory and excitatory arrivals of spikes denoted by λ^- and λ^+, respectively. The state of each cell is denoted by q, and it receives an *inhibitory input* from the state of some cell u which does not belong to $\mathbf{M}(n)$. Thus for any cell $i \in M(n)$ we have an inhibitory weight $w_u^- \equiv w_{u,i}^- > 0$ from u to i.

For any $i,j \in \mathbf{M}(n)$ we have $w_{i,j}^+ = w_{i,j}^- = 0$, but all whenever one of the cells fires, it triggers the firing of the other cells with $p(i,j) = \frac{p}{n}$ and $Q(i,j) = \frac{(1-p)}{n}$. As a result, we have:

$$q = \frac{\lambda^+ + rq(n-1)\sum_{l=0}^{\infty}[\frac{qp(n-1)}{n}]^l \frac{1-p}{n}}{r + \lambda^- + q_u w_u^- rq(n-1)\sum_{l=0}^{\infty}[\frac{qp(n-1)}{n}]^l \frac{p}{n}} \tag{6}$$

which reduces to:

$$q = \frac{\lambda^+ + \frac{rq(n-1)(1-p)}{n-qp(n-1)}}{r + \lambda^- + q_u w_u^- + \frac{rqp(n-1)}{n-qp(n-1)}}, \tag{7}$$

where (7) is a second degree polynomial in q:

$$\begin{aligned} 0 \ &= \ q^2 p(n-1)[\lambda^- + q_u w_u^-] - q(n-1)[r(1-p) - \lambda^+ p] \\ &\quad + n[\lambda^+ - r - \lambda^- - q_u w_u^-]. \end{aligned} \tag{8}$$

Hence it can be easily solved for its *positive root(s)* which are *less than one*, which are the only ones of interest since q is a probability.

B. A RNN with Multiple Clusters of a $\mathbf{M}(n)$ Architectures

In this section we build a Deep Learning Architecture (DLA) based on multiple clusters, each of which is made up of a $\mathbf{M}(n)$ cluster. This DLA is shown schematically in Figure 1. The DLA is composed of C clusters $\mathbf{M}(n)$ each with n hidden neurons. For the c-th such cluster, $c = 1, \ \cdots \ , C$, the state of each of its identical cells is denoted by q_c. In addition, as shown in Figure 1, there are U input cells which do not belong to these C clusters, and the state of the u-th cell $u = 1, \ \cdots \ , U$ is denoted by $\bar{q}_u$.

Each hidden cell in the clusters c, with $c \in \{1, \ ... \ , C\}$ receives an inhibitory input from each of the U input cells. Thus, for each cell in the c-th cluster, we have inhibitory weights $w_{u,c}^- > 0$ from the u-th input cell to each cell in the c-th cluster. Thus the u-th input cell will have a total inhibitory "exit" weight, or total inhibitory firing rate r_u^- to all of the clusters which is of value:

$$r_u^- = n\sum_{c=1}^{C} w_{u,c}^-. \tag{9}$$

Then, from (7) and (8), we have

$$q_c = \frac{\lambda_c^+ + \frac{r_c q_c (n-1)(1-p_c)}{n - q_c p_c (n-1)}}{r_c + \lambda_c^- + \sum_{u=1}^{U} \bar{q}_u w_{u,c}^- + \frac{r_c q_c p_c (n-1)}{n - q_c p_c (n-1)}} \tag{10}$$

yielding a second degree polynomial for each of the q_c:

$$q_c^2 p_c (n-1)[\lambda_c^- + \sum_{u=1}^{U} \bar{q}_u w_{u,c}^-] \tag{11}$$

$$-q_c(n-1)[r_c(1-p_c) - \lambda_c^+ p_c] \tag{12}$$

$$+n[\lambda_c^+ - r_c - \lambda_c^- - \sum_{u=1}^{U} \bar{q}_u w_{u,c}^-] = 0.$$

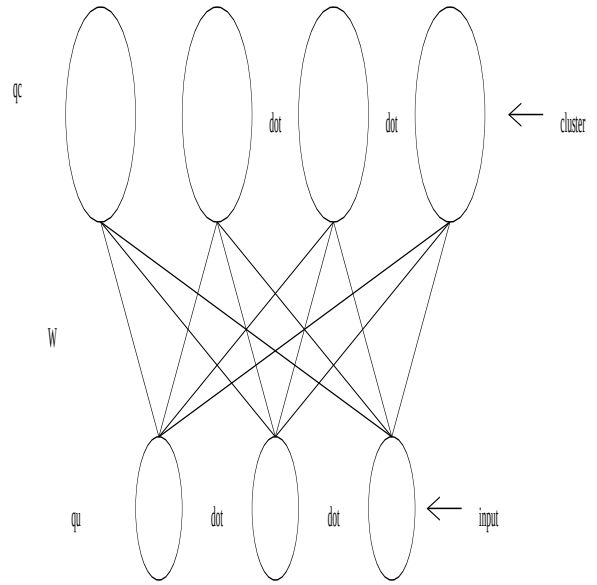

Fig. 1. Schematic representation of the Deep Learning network architecture

Its positive root is then:

$$q_c = \frac{-b_c + \sqrt{b_c^2 - 4a_c d_c}}{2a_c}, \tag{13}$$

where $a_c = p_c(n-1)[\lambda_c^- + \sum_{u=1}^{U} \bar{q}_u w_{u,c}^-]$, $b_c = -(n-1)[r_c(1-p_c) - \lambda_c^+ p_c]$ and $d_c = n[\lambda_c^+ - r_c - \lambda_c^- - \sum_{u=1}^{U} \bar{q}_u w_{u,c}^-]$. Let us now define the activation function of the cth cluster as:

$$\zeta_c(x) = \frac{-b_c + \sqrt{b_c^2 - 4p_c(n-1)[\lambda_c^- + x]n[\lambda_c^+ - r_c - \lambda_c^- - x]]}}{2p_c(n-1)[\lambda_c^- + x]}$$

where:

$$x = \sum_{u=1}^{U} w_{u,c}^- \bar{q}_u. \tag{14}$$

When all the parameters $b_c = b$, $p_c = p$, n, $\lambda_c^+ = \lambda$, $\lambda_c^- = \lambda$ with $c = 1, \cdots, C$ are the same for all of the clusters, we will have:

$$x = \frac{-b + \sqrt{b^2 - 4p(n-1)[\lambda^- + x]n[\lambda^+ - r - \lambda^- - x]}}{2p(n-1)[\lambda^- + x]}. \tag{15}$$

IV. Application to the Design of an Auto-Encoder

In this section we will construct an auto-encoder based on two instances of the network shown in Figure 1. For the network shown, we shall call these two network instances Network-1 and Network-2.

Network-1 has U input cells and C clusters (as shown in the figure). On the other hand, Network-2 has C input cells and U clusters. Suppose now that there is a dataset X that is represented by a U-vector $X \in [0,1]^U$.

We first construct the Network-1 such that the U-vector of input cells is: $\bar{q}^{(1)}$, and we construct the $U \times C$ matrix of weights from the input cells to the cells in each of the C clusters as

$$W^1 = [w_{u,c}^-]. \tag{16}$$

Denoting by Q the C-vector of cells whose state is q_c for cluster c, and for an n-vector y denoting by:

$$\zeta(y) = (\zeta(y_1), \ \cdots, \zeta(y_n)), \tag{17}$$

we have:

$$Q^{(1)} = \zeta(\bar{q}^{(1)} W^{(1)}). \tag{18}$$

On the other hand, Network-2 is a pseudo-inverse of Network-1, with C input cells and U clusters, and the $C \times U$ weight matrix between its input cells and the cells of each of the clusters will be denoted by $W^(2)$. we will then have:

$$\bar{q}^{(2)} = \zeta(W^{(2)} \zeta(W^{(1)} \bar{q}^{(1)}). \tag{19}$$

Problem 1 The learning problem is then to adjust $W^{(1)}$ and $W^{(2)}$ so that $\bar{q}^{(2)}$ becomes as close to $\bar{q}^{(1)}$ as possible. When we have a set of data $\mathbf{X}$ which has the form of D rows of U-vectors $x \in [0,1]^U$, the problem we address can be described as:

$$\min_{W^{(1)}, W^{(2)}} ||X - \zeta(\zeta(XW^{(1)})W^{(2)})||^2,$$
$$\text{s.t. } W^{(1)}, W^{(2)} \geq 0,$$

where the matrices $W^{(1)}$ and $W^{(2)}$ each have D blocks of $U \times C$ and $C \times U$ (respectively) matrices, and the function $\zeta(.)$ is understood to be extended to the matrix case.

V. A First Approach

We may generalise the approach developed by Liu [31] to solve Problem 1. To this effect, let us define a cost function

$$L(W^{(1)}, W^{(2)}) = ||X - \zeta(\zeta(XW^{(1)})W^{(2)})||^2. \tag{20}$$

First calculate:

$$\eta(x) = \frac{\partial \zeta(x)}{\partial x} = \frac{b}{([\lambda^- + x])^2} - \frac{\sqrt{b^2 - 4p(n-1)[\lambda^- + x]n[\lambda^+ - r - \lambda^- - x]}}{[\lambda^- + x]^2} + \frac{-n[\lambda^+ - r - \lambda^- - x] + n[\lambda^- + x]}{[\lambda^- + x]\sqrt{b^2 - 4p(n-1)[\lambda^- + x]n[\lambda^+ - r - \lambda^- - x]}}.$$

We also define another element-wise operation $\eta(H) \in R^{D \times C}$ with $H \in R^{D \times C}$, where the element in the ith row and jth column of $\eta(H)$ is calculated by $\eta(H_{i,j})$ with $i = 1, \cdots, D$ and $j = 1, \cdots, C$. Then we can derive

$$\frac{\partial L}{\partial W^{(1)}} = -X^{\mathrm{T}}(\eta(XW^{(1)}) * (((X - \zeta(\zeta(XW^{(1)})W^{(2)})) * \eta(\zeta(XW^{(1)})W^{(2)}))(W^{(2)})^{\mathrm{T}})).$$

Note that, the operation $*$ is defined as an element-wise multiplication operation. For example, if $H = H^{(1)} * H^{(2)}$, then $H \in R^{D \times C}$, $H_1 \in R^{D \times C}$ and $H_2 \in R^{D \times C}$. Besides,

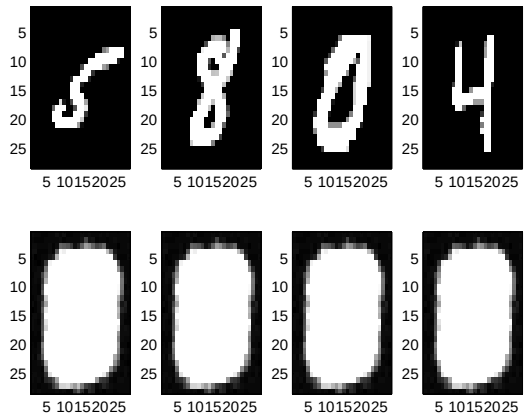

Fig. 2. Results of the autoencoder with only spiking models.

the element in the ith row and jth column of H, which is $H_{i,j}$, is calculated by $H_{i,j} = H^{(1)}_{i,j} H^{(2)}_{i,j}$, where $i = 1, \cdots, D$ and $j = 1, \cdots, C$.

To make it clearer, let $\varphi_1 = \eta(XW^{(1)})$, $\varphi_2 = \zeta(\zeta(XW^{(1)})W^{(2)})$, $\varphi_3 = \eta(\zeta(XW^{(1)})W^{(2)})$ and $\varphi_4 = \zeta(XW^{(1)})$. Then,

$$\begin{aligned}\frac{\partial L}{\partial W^{(1)}} &= -X^{\mathrm{T}}(\varphi_1 * (((X - \varphi_2) * \varphi_3)(W^{(2)})^{\mathrm{T}})) \\ &= -X^{\mathrm{T}}(\varphi_1 * ((X * \varphi_3)(W^{(2)})^{\mathrm{T}})) \\ &+X^{\mathrm{T}}(\varphi_1 * ((\varphi_2 * \varphi_3)(W^{(2)})^{\mathrm{T}})),\end{aligned}$$

and

$$\frac{\partial L}{\partial W^{(2)}} = -\varphi_4^{\mathrm{T}}((X - \varphi_2) * \varphi_3) \tag{21}$$

$$= -\varphi_4^{\mathrm{T}}(X * \varphi_3 - \varphi_2 * \varphi_3) \tag{22}$$

$$= -\varphi_4^{\mathrm{T}}(X * \varphi_3) + \varphi_4^{\mathrm{T}}(\varphi_2 * \varphi_3).$$

The updates rules for $W^{(1)}$ and $W^{(2)}$ then become

$$W^{(1)}_{i,j} = W^{(1)}_{i,j} \frac{(X^{\mathrm{T}}(\varphi_1 * ((X * \varphi_3)(W^{(2)})^{\mathrm{T}})))_{i,j}}{(X^{\mathrm{T}}(\varphi_1 * ((\varphi_2 * \varphi_3)(W^{(2)})^{\mathrm{T}})))_{i,j}} \tag{23}$$

and

$$W^{(2)}_{i,j} = W^{(2)}_{i,j} \frac{(\varphi_4^{\mathrm{T}}(X * \varphi_3))_{i,j}}{(\varphi_4^{\mathrm{T}}(\varphi_2 * \varphi_3))_{i,j}}, \tag{24}$$

where the symbol $(H)_{i,j}$ denotes the element in the ith row and jth column of H.

To be more specific, in the right-hand side of (23) and (24), we use the original values of $W^{(1)}$ and $W^{(2)}$ in the lth iteration. Then, the left-had side of (23) and (24) would be the updated values of $W^{(1)}$ and $W^{(2)}$ in the lth iteration.

VI. An Autoencoder combining the RNN and the extreme learning machine

Hoping for achieve better performance, we modify the learning problem as follows:

Problem 2 Find $W^{(1)}$ such that

$$\min_{W^{(1)}} ||X - W^{(2)}\zeta(W^{(1)}X)||^2, \text{ s.t. } W^{(1)} \geq 0. \tag{25}$$

Thus specifically, we will use Network-1 for encoding, but we use an idea from the "extreme learning machine (ELM)" [32] for the decoding part of the probelm.

Generally, when we use an autoencoder for classification, we connect only the encoding part to the classifier. Thus while $W^{(1)} \geq 0$, we will *remove* the constraint that $W^{(2)} \geq 0$. We will call this autoencoder the Random Neural Network Extreme Learning Machine (RNN-ELM).

Adapting the idea in [32] we use the following operation to determine $W^{(2)}$:

$$W^{(2)} = \text{pinv}(\varphi_4)X, \tag{26}$$

where

$$\text{pinv}(x) = (x^{\mathrm{T}}x)^{-1}x^{\mathrm{T}}, \tag{27}$$

which is the variant presented in [32].

Let us define $\bar{W}^{(2)} = \max(W^{(2)}, 0)$. Let $\varphi_5 = \zeta(XW^{(1)})\bar{W}^{(2)} = \varphi_4\bar{W}^{(2)}$. Then, the update rule for $W^{(1)}$ will be

$$W^{(1)}_{i,j} = W^{(1)}_{i,j} \frac{(X^{\mathrm{T}}(\varphi_1 * (X(\bar{W}^{(2)})^{\mathrm{T}})))_{i,j}}{(X^{\mathrm{T}}(\varphi_1 * (\varphi_5(\bar{W}^{(2)})^{\mathrm{T}})))_{i,j}}, \tag{28}$$

which guarantees that $W^{(1)} \geq 0$.

VII. Testing the RNN-ELM

To test the RNN-ELM, we use the MNIST dataset of handwritten digits [33] which contains $60,000$ images in the training dataset and $10,000$ images in the test dataset, we conduct numerical experiments on the autoencoder with two different structures: one is a $784 \to 50$ structure with 50 intermediate or hidden units, while the second one is a $784 \to 500$ structure with 500 hidden units. In both cases we exploit small clusters with $n = 2$. Exhaustive tests were carried out as follows:

- We first randomly generated the elements of $W^{(1)}$ in the range of $[0, 1]$.
- Then, we used (26) to determine $W^{(2)}$.

Examples of the results obtained with this approach are shown in Figure 3.

In a second approach, we use (28) to update $W^{(1)}$, and then use (26) again to update $W^{(2)}$. The results obtained are shown in Figures 4 and 5.

It is evident that the results in the second approach Figures 4 and 5 are far better that those in Figure 3. This illustrates that both (26) and (28) are important for adjusting the parameters of the autoencoder.

VIII. Stacking the Classifiers

Following Tang's work [34], we could stack multi-autoencoders together and connect them to an ELM to construct a multi-layer classifier. First, let us consider a different approach from the one in Section VI, using the advice from [34] regarding the use the $L-1$ norm to generate more sparse

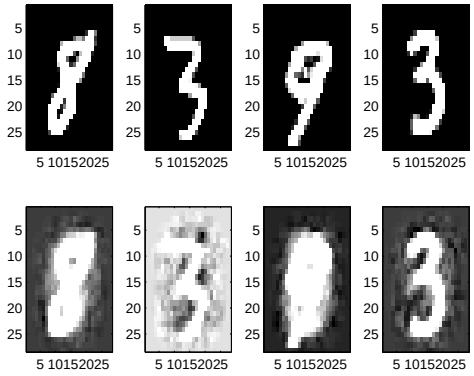

Fig. 3. Results of the autoencoder with structure 784-500 via only (26): up for original figures and down for reconstructions.

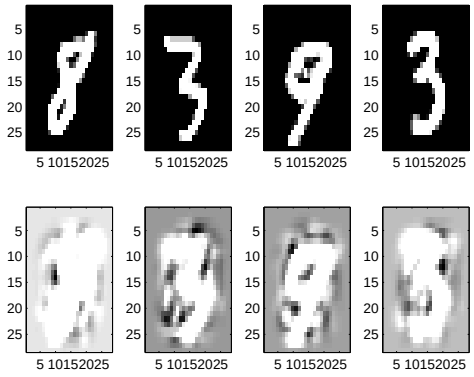

Fig. 4. Results of the autoencoder with structure 784-50 via both (26) and (28): up for original figures and down for reconstructions.

and compact features. Then, the problem to be addressed can be described as

$$\min_{W^{(2)}} ||X - XW^{(1)}W^{(2)}||^2 + ||W^{(2)}||_{\ell_1}, \quad (29)$$

$$\text{s.t. } W^{(2)} \geq 0, \quad (30)$$

showing that we only need to adjust $W^{(2)}$. I Indeed, based on [32], [34], a randomly-generated $W^{(1)}$ could be sufficient to obtain effective learning with reduced computational complexity. Note that the constraint of $W^{(2)} \geq 0$ is the characteristic that allows us to use the $W^{(2)}$ in the RNN.

We can then use the fast iterative shrinkage-thresholding algorithm (FISTA) in [35] to solve problem (29), with the modification that we set the negative elements in the solution to zero in each iteration.

Once (29) is solved, $W^{(2)}$ is obtained and let $\tilde{W}^{(1)} = W^{(2)}$. Then, we import $\tilde{W}^{(1)}$ to the RNN with input X as input, and output $X^{(2)} = \zeta(X(\tilde{W^{(1)}}^{\mathrm{T}}$. By using $X^{(2)}$ as the input to the next autoencoder, we then seek the weights $\tilde{W}^{(2)}$ for the next layer of the multi-layer classifier. Note that the last layer of the multi-layer classifier is an ELM with activation function $\zeta(x)$.

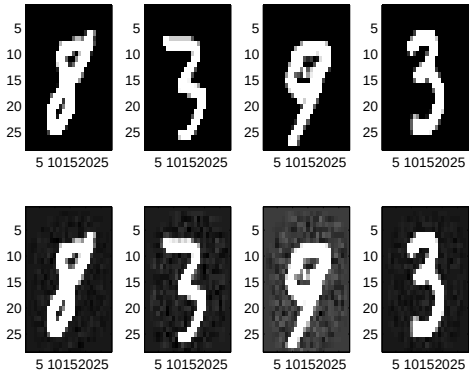

Fig. 5. Results of the autoencoder with structure 784-500 via both (26) and (28): up for original figures and down for reconstructions.

A. Test Results

Consider a multi-layer classifier with a structure denoted by $784 - 700 - 700 - 5000 - 10$. This is a multi-level structure where the weights between successive layers are denoted by $\tilde{W}^{(i)}$ with $i = 1, \cdots, 4$.

The details of this structure are given by the nature of the layers themselves, and the interconnections of each of the successive feedforward layers. The RNN layers use instead of single nodes, new spiking clusters that we have defined, where we have used clusters of two cells, rather than singe cells. The specific experiments we have run deal with architectures with characteristics such as:

- The first two layers $784 \rightarrow 700$ and $700 \rightarrow 700$ are RNNs where the $\tilde{W}^{(1)}$ and $\tilde{W}^{(2)}$ are determined by the autoencoder.
- The third layer $700 \rightarrow 5000$ is also an RNN, where $\tilde{W}^{(3)}$ is et by a random generation of each entry in $[0, 1]$.
- Finally, the last layer $5000 \rightarrow 10$ is an ELM.

With this architectural structure, but different numbers of intermediate interconnected nodes as shown below, we have run exhaustive and thorough tests using the MNIST dataset [33] with $60,000$ images in the training dataset and 10,000 images in the testing dataset. The following results were obtained:

- For the structure of $784 - 700 - 700 - 5000 - 10$, we obtained 96.25% accuracy with the test set.
- For the structure $784-500-500-8000-10$, we achieved 95.79% testing accuracy.
- For the structure of $784 - 500 - 8000 - 10$, we attained 98.64% testing accuracy.

In addition, we reverted to the pure RNN-ELM architecture without the he autoencoder. The structure is of the form $784-8000-10$, and we observed 97.51% accuracy at testing.

IX. Conclusions

This paper has developed a RNN-ELM deep learning architecture that combines spiking random neurla networks and

extreme learning machines. We have considered very large networks with hundreds of cells in each layer, and have exploited clustered neurons in the RNN layers.

Our main experimental results show that:

- On a standard and significant problem of visual character recognition on very large data sets, the RNN-ELM provides better recognition performance than the extreme learning machines on their own, reaching recognition ratios that exceed 98.5%.
- In all cases, the best results are achieved with large networks that exceed thousands of cells.
- The quality of the results observed seem to improve with the size of the network.

In future work we plan to examine the value of recurrent networks, and we will address more particularly the types of recurrent networks that may be used and also we will exploit the asymptotic properties of RNN clusters to render the learning process more efficient..

ACKNOWLEDGEMENT

We gratefully acknowledge the support of the EC 7th Framework Program PANACEA Project, Grant Agreement No. 610764, to Imperial College London.

REFERENCES

[1] R. Raina, A. Madhavan, and A. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proc. 26th Int. Conf. on Machine Learning*. ACM, 2009, pp. 873–880.

[2] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep big simple neural nets for handwritten digit recognition," *Neural Computation*, vol. 22, pp. 3207–3220, 2010.

[3] J. Doncel, U. Ayesta, O. Brun, and B. Prabhu, "A resource-sharing game with relative priorities," *Performance Evaluation*, vol. 79, p. 287305, September 2014.

[4] G. Gorbil, D. Perez, and E. H. Cuesta, "Principles of pervasive monitoring," in *ISCIS 2014*, T. Czachorski and E. Gelenbe, Eds. Springer Verlag, 2014, pp. 117–124.

[5] P. D. Sanzo, A. Pellegrini, and D. R. Avresky, "Machine learning for achieving self-* properties and seamless execution of applications in the cloud," in *NCCA 2015, Munich, Germany*. IEEEXplore, 2015, pp. 51–58.

[6] L. Wang and E. Gelenbe, "Adaptive dispatching of tasks in the cloud," *Cloud Computing, IEEE Transactions on*, vol. PP, 2015.

[7] W. Gerstner, "Spiking neurons," in *Pulsed Neural Networks*, W. Maass and C. M. Bishop, Eds. MIT Press, 2001, p. 377.

[8] Y. Gu, Y. Chen, X. Zhang, G. W. Li, C. Y. Wang, and L.-Y. M. Huang, "Neuronal soma-satellite glial cell interactions in sensory ganglia and the participation of purinergic receptors," *Neuron Glia Biology*, vol. 6, no. 1, p. 5362, February 2010.

[9] E. S. Krames, P. H. Peckham, and A. R. Rezai, Eds., *Neuromodulation*. Academic Press, 2009, vol. 1-2.

[10] E. A. Newman, "New roles for astrocytes: Regulation of synaptic transmission," *TRENDS in Neurosciences*, vol. 10, no. 26, p. 536542, 2001.

[11] M. Arbib, Ed., *The Handbook of Brain Theory and Neural Networks*. MIT Press, 2003.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[13] E. Gelenbe, "Random neural networks with negative and positive signals and product form solution," *Neural computation*, vol. 1, no. 4, pp. 502–510, 1989.

[14] J. Medhi, *Stochastic Processes*. New Age International, 1994.

[15] E. Gelenbe and C. Cramer, "Oscillatory corticothalamic response to somatosensory input," *Biosystems*, vol. 48, no. 1, pp. 67–75, 1998.

[16] E. Gelenbe, "Learning in the recurrent random neural network," *Neural Computation*, vol. 5, pp. 154–164, 1993.

[17] E. Gelenbe, V. Koubi, and F. Pekergin, "Dynamical random neural network approach to the traveling salesman problem," in *Proceedings IEEE Symp. Systems, Man and Cybernetics*. IEEE, 1993, p. 630635.

[18] E. Gelenbe and T. Koçak, "Area-based results for mine detection," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 38, no. 1, pp. 12–24, 2000.

[19] C. Cramer, E. Gelenbe, and H. Bakircloglu, "Low bit-rate video compression with neural networks and temporal subsampling," *Proceedings of the IEEE*, vol. 84, no. 10, pp. 1529–1543, 1996.

[20] C. E. Cramer and E. Gelenbe, "Video quality and traffic qos in learning-based subsampled and receiver-interpolated video sequences," *Selected Areas in Communications, IEEE Journal on*, vol. 18, no. 2, pp. 150–167, 2000.

[21] H. M. Abdelbaki, K. Hussain, and E. Gelenbe, "A laser intensity image based automatic vehicle classification system," in *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*. IEEE, 2001, pp. 460–465.

[22] E. Gelenbe and K. F. Hussain, "Learning in the multiple class random neural network," *Neural Networks, IEEE Transactions on*, vol. 13, no. 6, pp. 1257–1267, 2002.

[23] E. Gelenbe, K. Hussain, and V. Kaptan, "Simulating autonomous agents in augmented reality," *Journal of Systems and Software*, vol. 74, no. 3, pp. 255–268, 2005.

[24] A. Filippoupolitis, L. Hey, G. Loukas, E. Gelenbe, and S. Timotheou, "Emergency response simulation using wireless sensor networks," in *Proceedings of the 1st international conference on Ambient media and systems*, 2008, p. 21.

[25] E. Gelenbe, "Steps toward self-aware networks," *Communications of the ACM*, vol. 52, no. 7, pp. 66–75, 2009.

[26] E. Gelenbe and F.-J. Wu, "Large scale simulation for human evacuation and rescue," *Computers & Mathematics with Applications*, vol. 64, no. 12, pp. 3869–3880, 2012.

[27] E. Gelenbe, "G-networks with triggered customer movement," *Journal of Applied Probability*, pp. 742–748, 1993.

[28] E. Gelenbe and S. Timotheou, "Random neural networks with synchronized interactions," *Neural Computation*, vol. 20, no. 9, pp. 2308–2324, 2008.

[29] E. Gelenbe and J.-M. Fourneau, "Random neural networks with multiple classes of signals," *Neural computation*, vol. 11, no. 4, pp. 953–963, 1999.

[30] E. Gelenbe, "The first decade of g-networks," *European Journal of Operational Research*, vol. 126, no. 2, pp. 231–232, 2000.

[31] X. Liu, S. Yan, and H. Jin, "Projective nonnegative graph embedding," *Image Processing, IEEE Transactions on*, vol. 19, no. 5, pp. 1126–1137, 2010.

[32] L. L. C. Kasun, H. Zhou, and G.-B. Huang, "Representational learning with extreme learning machine for big data," *IEEE Intelligent Systems*, vol. 28, no. 6, pp. 31–34, 2013.

[33] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[34] J. Tang, C. Deng, and G.-B. Huang, "Extreme learning machine for multilayer perceptron," *to appear in IEEE Transactions on Neurla Networks and Learning Systems*, May 2015.

[35] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM journal on imaging sciences*, vol. 2, no. 1, pp. 183–202, 2009.