Probability in the Engineering and Informational Sciences, **35**, 2021, 6–36.
doi:10.1017/S026996481800058X

RANDOM NEURAL NETWORK METHODS AND DEEP LEARNING

YONGHUA YIN
Intelligent Systems and Networks Group Department of Electrical and Electronic Engineering Imperial College, London SW7 2BT, UK
E-mail: y.yin14@imperial.ac.uk

The random neural network (RNN) is a mathematical model for an "integrate and fire" spiking network that closely resembles the stochastic behavior of neurons in mammalian brains. Since its proposal in 1989, there have been numerous investigations into the RNN's applications and learning algorithms. Deep learning (DL) has achieved great success in machine learning. Recently, the properties of the RNN for DL have been investigated, in order to combine their power. Recent results demonstrate that the gap between RNNs and DL can be bridged and the DL tools based on the RNN are faster and can potentially be used with less energy expenditure than existing methods.

Keywords: computational probability, queueing theory, stochastic modeling

1. INTRODUCTION

Originally developed to mimic the behavior of the brain's biological neurons, the random neural network (RNN) proposed by E. Gelenbe, functions as a stochastic integer state "integrate-and-fire" system [43,44], and was detailed in early papers [42,45,49]. In an RNN, an arbitrarily large set of neurons interact with each other via excitatory and inhibitory spikes that modify each neuron's action potential in continuous time. It has been proved that, in a steady state, the stochastic spiking behaviors of the RNN have a remarkable property called "product form", with the state probability distribution given by an easily-solvable system of non-linear equations. Compared with many other neural network models, e.g., the multi-layer perceptron (MLP) [131], the radial basis function (RBF) neural network [119], extreme learning machines (ELM) [21], and orthogonal polynomial networks (OPN) [163], the RNN has the following advantages:

1) The RNN model is closer to the biological neuronal network and can better represent how signals are transmitted through the human brain, whereby the complex stochastic behaviors among neurons can be modeled and analyzed more easily in a mathematical form.
2) The RNN model generally has a better predictive capability because of its non-negativity and probability constraints.

© Cambridge University Press 2019 0269-9648/18 $25.00

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

3) The RNN model has rigorously established mathematical properties which simplify the related computations.
4) Owing to the highly-distributed nature, the RNN model can potentially be deployed in either energy-efficient customized or neuromorphic hardware.

The RNN has been used to model and explain that cortico-thalamic oscillations observed in the rat somatosensory system are caused by positive feedback in cortex [56] and is an example of a mathematical model for natural computation [53] that has also been used for protein alignment [122].

The RNN has many engineering applications: it represents image textures [9] to obtain accurate segmentation of brain Magnetic Resonance Images [71]. It achieves high video compression ratios at low computational cost [26,27,72], and has been used for vehicle classification [5], the estimation of Quality of Experience (QoE) [120], and real-time cognitive radio deployment [6]. As a fast heuristic for task assignment in distributed systems [7], it also optimizes a network's connectivity [73] and can also be used to generate error correcting codes [3]. With reinforcement learning, it improves the Quality of Service experienced by network users [15,39,40,52], by adapting the routing to the traffic characteristics of the traffic in packet networks [59,147], to improve web search [134], to dynamically allocate tasks in Cloud Computing systems [148], to predict the toxicity of compounds from data on their physical–chemical characteristics [85], to detect network attacks [16,132], and to control heating ventilation and air conditioning (HVAC) systems (that incorporate Internet of Things (IoT) and Cloud Computing) [97].

The RNN is actually a special case of a class of G-Networks, a general purpose probability models for large-scale networks [38,46–48,63,69] which were shown to have the desirable product form solution [62] to efficiently compute their mathematical characteristics. While all the potential of such models for machine learning has still not been exploited [50], G-Networks have had applications in areas different from neuronal networks, such as for load balancing in distributed systems and communication networks [61], to model Gene Regulatory Networks [51,101], and to represent systems that operate with intermittent sources of energy [54,55,60,99].

Inspired by the structure and function of the brain, deep learning (DL) functions as a machine-learning approach [88,107]. Most computational models in DL exploit a feed-forward neural-network architecture composed of multi-processing layers; this architecture allows the model to extract high-level representations from the raw data. In this manner, essential and useful information can be extracted from these raw data so that the problem of building complex mapping from the raw inputs to the desired outputs becomes more solvable. DL has achieved major advances in solving many historic issues in machine learning.

Since the proposal of the RNN in 1989, there have been numerous investigations into the RNN's varieties, applications, and learning algorithms [11,79,97,144]; however, there has not been any research regarding the property of the RNN for DL until the first related work of the authors in 2016 [66]. Table 1 attempts to present a brief time table of the RNN developments over three decades. The main objective of the following work of the authors in [66–68,154–156] is to connect the RNN with DL, which is novel in the following two aspects: 1) the research could provide powerful and DL tools that not only achieve state-of-the-art performance but can potentially be used with less energy expenditure than existing methods; 2) the research attempts to provide a better understanding of the relationship between DL and the brain.

The details of the progresses in this direction has been summarized into the PhD thesis of the author Yonghua Yin [153].

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

TABLE 1. Time table of RNN developments since 1989.

Time	Development
1989	Proposal of RNN [42]
1991	Bipolar RNN [70]
1993	Gradient-descent learning algorithm of RNN [49]
1999	Multi-class RNN [57]
1999	RNN function approximator [75]
2005	RNN with state-dependent firing [98]
2008	RNN with synchronized interactions [65]
2016	Dense RNN and deep learning [66]
2017	Deep learning with standard RNN [156]

This paper is organized into the following sections.

1) Section 2 presents the backgrounds of neural networks, DL, and the RNN.
2) Section 3 illustrates the multi-layer non-negative RNN autoencoders proposed in [155] for dimension reduction, which are robust and effective in handling various types of data.
3) The work on dense RNN [66–68,154] that includes mathematical modeling and DL algorithms have been summarized in Section 4.
4) Section 5 presents the investigation into the standard RNN and demonstrates its power for DL [156]. The resultant DL tool is demonstrated to be effective and is arguably the most efficient of five different DL approaches.
5) Section 6 presents applications of the RNN and its DL algorithms for the detection of anomalies and attacks on IoT devices and object recognition via images.
6) The conclusions and future work are given in Section 7.

2. BACKGROUND

2.1. Neural Networks

In 1943, McCulloch and Pitts [115] built mathematical models of neural networks to describe how the neurons in the brain might work, where the neural behaviors are treated by means of propositional logic. This is the first computational model of neuron activity. In order to test theories on how the brain works, Rochester conducted neural-network simulations using computers. The theory in [128] was developed for a hypothetical nervous system, called a perceptron. The Rosenblatt perceptrons are considered as the basis of neural networks, and the related work had caused a heated controversy among the AI community at that time [151].

In 1989, Hornik [91] proved that a multi-layer feed-forward neural network, in other terms, MLP, with as few as one hidden layer can approximate any continuous function uniformly. This work provides theoretical basis for the learning capability of neural networks and had provoked lots of interests from researchers toward this direction [133]. Specifically, Stinchcombe and White [138] showed that sigmoid functions are not necessary for universal approximation. Further work by [90] investigated the approximation capability of the MLP with bounded and non-constant activation functions. A generalized result from [90] in [109]

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

relaxes the condition, i.e., an MLP with non-polynomial activation functions is a universal approximator.

In addition to the MLP, there are other types of neural networks. The RBF-based neural networks are also widely used. The work in [118,119] proved the universal approximation property of the RBF-based neural networks. Another type of neural network uses a product-based orthogonal-polynomial (POP) activation function in [163,164]. It is also proved that the POP-based single-hidden-layer neural network (SLNN) in [163] has lower computational complexity than the MLP.

With recent improvements, back propagation (BP) has become a standard method to train neural networks. For example, the developments of the rectified linear unit (ReLU) units and weight-sharing strategies facilitate the training of multi-layer neural networks with the BP algorithms [82]. This involves the direction of DL, which contributes to the recent success in solving many complex problems and will be further reviewed in Subsection 2.3. However, training a neural network with BP well can still be time-consuming and require huge computation load because of the possible slow convergence and potential instability inherent in the training process. This issue has provided the rationale for recent research focusing on the subject of neural networks that adopted the perspectives of both the approximation property and training efficiency.

Huang [94] showed that, using randomly-generated input weights and hidden biases, a single-hidden-layer feed-forward neural network can be described as a linear system whose output weights can be analytically determined. This is the ELM concept. Learning systems based on the ELM can be trained more than a hundred times faster than the BP-based ones without compromising accuracy in many cases [140]. Zhang [163,164] focused on the POP neural network. Theoretical results in [163,164] present its universal approximation property. Similar to the ELM concept, the POP neural network can then be described as a linear system with the output weights analytically determined, which is the weights-direct-determination concept.

2.2. Spiking Neural Network and Neuromorphic Computing

Von Neumann architecture [19] has provided the basis for many major advances in modern computing [25,150], allowing for the rapid decrease in transistor size, and improved performance without increasing costs. However, the rate of progress experienced by this type of architecture has recently slowed down, and with various seemingly insurmountable obstacles preventing further significant gains, the end of Moore's Law is in sight [121]. One major obstacle is that, due to quantum-mechanical effects, transistors smaller than 65 nanometers can no longer switch between "on" and "off" reliably, which means that a digital computer cannot distinguish one from zero. The frequency wall is also a major obstacle, which means that the hardware could melt down because of excess heat if the CPU frequency exceeds four billion times per second. Therefore, a new machine that goes beyond the boundaries imposed by von Neumann architecture needs to be developed, and the areas of neuromorphic computing and spiking neural network (SNN) are starting to attract significant attention from researchers in both academic and commercial institutions.

A non-von Neumann architecture is the TrueNorth cognitive computing system, which has been developed by IBM and is inspired by how organic brains handle complex tasks easily while being extremely energy-efficient [34,35,123]. The TrueNorth chip has received a lot of attention from wider society since it is the first large-scale commercial spiking-based neuromorphic computing platform [25]. A first major step occurred when IBM researchers presented an architectural simulator, called Compass, for large-scale network models of TrueNorth cores [123]. Since the Compass simulator is functionally equivalent to TrueNorth

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

and runs on a von Neumann computer, it facilitates research and development for TrueNorth at this stage. Based on this simulator, in [35], applications – including speaker recognition, digital recognition, and so on – are integrated to demonstrate the feasibility of ten different algorithms, such as convolution networks, liquid state machines, and so on, to be run in TrueNorth. The conventional neural network algorithms require high precision while the synaptic weights and spikes in TrueNorth have low quantization resolution, which is the issue investigated in [25,150]. Considering that TrueNorth is spiking-based, naturally, mapping models of SNNs into TrueNorth could be a reasonable choice [25,150]. In [150] and [25], a SLNN was deployed into TrueNorth with the learning algorithm presented, where the non-linearity is brought by the cumulative probability function. The research in [31] implemented a sentiment-analysis algorithm on both the spiking-neural-network simulator and TrueNorth system. First, a fully-connected single-hidden-layer neural network (FCNN) with ReLU was trained using the stochastic gradient descent. Then, this FCNN was converted into a SNN model with a discussion on the related issues needed to be addressed, e.g., the precision and the negative inputs. There is a performance drop in this conversion. Finally, the SNN was mapped into TrueNorth. It was mentioned that performance loss in this step is more significant than that through the FCNN-SNN conversion.

The SNN architecture (SpiNNaker) project is another spike-based neuromorphic computing platform [34,41,102]. The SpiNNaker aims to model a large-scale SNN in biological real time. The paper [41], published in 2014, presented an overview of the SpiNNaker architecture and hardware implementation, introduced its system software and available API and described typical applications. A research project on SpiNNaker in [102] presented an application for audio classification by mapping a trained one-hidden-layer leaky integrate-and-fire SNN into the SpiNNaker, utilizing the PyNN library [30] that is a common interface for different SNN simulators including NEURON [22], NEST [80], and Brian [84].

Recently, SNN-based neuromorphic computing has made rapid progress by combining its own inherent benefits and DL [37]. By enforcing connectivity constraints, the work in [37] applied the deep convolutional network structure into neuromorphic systems. Conventional convolutional networks require high precision while neuromorphic design uses only low-precision one-bit spikes. However, it was demonstrated that, by introducing constraints and approximation for the neurons and synapses, it is possible to adapt the widely-used backpropagation training technique. This enables the creation of a SNN that is directly implementable into a neuromorphic system with low-precision synapses and neurons [36]. Comparisons based on eight typical datasets demonstrated that this neuromorphic system achieves comparable or higher accuracies compared with start-of-the-art DL methods. High-energy efficiency is maintained throughout this process.

2.3. Deep Learning

DL [88,107] has achieved great success in machine learning and is currently contributing to huge advances in solving complex practical problems compared with traditional methods. For example, based on deep convolutional neural networks (CNN), which are powerful in handling computer vision tasks, DL systems could be designed to realize the dream of self-driving cars [13]. Another example is the game of Go, invented in ancient China more than 2,500 years ago, which was one of the most challenging classic games for artificial intelligence since the number of possible moves is greater than the total number of atoms in the visible universe [93]. However, a major step was recently taken when a computer system based on deep neural networks, a product of the Google DeepMind laboratories, defeated some of the world's top human players [135].

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

Many computational models used in DL exploit a feed-forward neural-network architecture that is composed of multi-processing layers, which allows the model to extract high-level representations from raw data. The feed-forward fully-connected multi-layer neural network, also known as MLP, has already been investigated for decades [129]. However, progress on the MLP was slow since researchers found difficulties in training MLPs with more than three layers [81]. Furthermore, adding more layers did not seem to help improve the performance. Pre-training the MLP layer by layer is a great advance [88,89], and this breakthrough demonstrates that it is possible to train the MLP and adding more layers can actually improve the network performance. This method is still very useful due to its wide adaptability. Recent literature shows that utilising the ReLU reduces the difficulty in the training of deep neural networks [82]. The typical training procedure called stochastic gradient descent (SGD) provides a practical choice for handling large datasets [14].

A considerable body of successful work has been amassed thanks to LeCun's inspirational work on CNN [105]. The weight-sharing property in the convolutional structure significantly reduces the number of weights to be learned and the local connectivity pattern allows good feature extraction, which facilitates the training of a deep CNN. Recent work on deep residual learning has made it possible to train extremely deep neural networks with thousands of layers [86].

The task of supervised learning in the DNN can be handled using the greedy layer-wise unsupervised learning, convolution structures, and residual learning. Given the ease with which humans are able to learn in an unsupervised manner, and the ready availability of raw datasets – which is not the case with labeled datasets – unsupervised learning has become a key issue upon which the DL community has begun to focus. A popular topic in unsupervised learning is the generative adversarial nets (GAN) system [83], which includes a discriminative and generative models. The generative model that takes noise as input tries to generate fake samples to cheat the discriminative model such that it cannot distinguish between fake and true samples. The GAN can be used to model natural images and handle image generation tasks [125]. The adversarial autoencoders(AAE) are proposed based on the GAN [114]. The work in [114] shows how the AAE can be used in different applications, e.g., semi-supervised classification, unsupervised clustering, dimensionality reduction and data visualization.

2.4. Random Neural Network

The RNN proposed by E. Gelenbe in 1989 [42,45,49] is a stochastic integer-state "integrate and fire" system and was developed to mimic the behavior of biological neurons in the brain. The RNN is also a recurrent neural network. In an RNN, an arbitrarily large set of neurons interact with each other via excitatory and inhibitory spikes which modify each neuron's action potential in continuous time. The power of the RNN is derived from the fact that, in steady state, the stochastic spiking behaviors of the network have a remarkable property called "product form" and that the state probability distribution is given by an easily solvable system of non-linear equations. In addition, the RNN models the biological neuronal network with greater accuracy, and offers an improved representation of signal transmission in the human brain, when compared with alternative neural network models, e.g., the MLP [130], ELM [94], and OPN [163,164].

2.4.1. Model description. Within the RNN system with L neurons, all neurons communicate with each other with stochastic unit amplitude spikes while receiving external spikes. The potential of the l-th neuron, denoted by $k(t) \geq 0$, is dynamically changing in

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

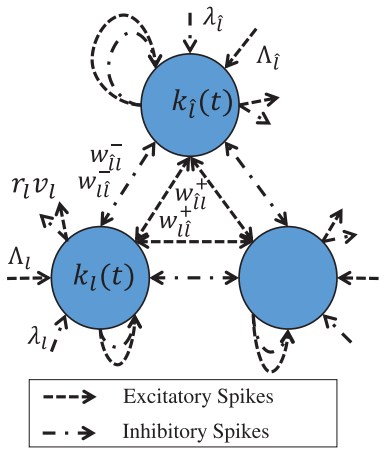


FIGURE **1.** Schematic representation of a spiking RNN system.

continuous time, and the l-th neuron is said to be excited if its potential $k(t)$ is larger than zero. An excitatory spike arrival to the l-th neuron increases its potential by 1, denoted by $k(t^+) \leftarrow k(t) + 1$; while an inhibitory spike arrival decreases its potential by 1 if it is larger than zero, denoted by $k(t^+) \leftarrow \max(k(t) - 1, 0)$, where $\max(a, b)$ produces the larger element between a and b.

For better illustration, Figure 1 presents the schematic representation of the RNN system. Excitatory spikes arrive at the l-th neuron from the outside world according to Poisson processes of rate Λ_l, which means that the probability that there are ρ excitatory spike arrivals in time interval Δt is [87]: $\text{Prob}(\rho \text{ spike arrivals in interval } \Delta t) = (\Lambda_l \Delta t)^\rho e^{-\Lambda_l \Delta t}/\rho!$, where ρ is a non-negative integer. In addition, the rate of inhibitory Poisson spike arrivals from the outside world to the l-th neuron is λ_l. When the l-th neuron is excited, it may fire excitatory or inhibitory spikes toward other neurons with the inter-firing interval $\bar{\rho}$ being exponentially distributed, which means that the probability density function of $\bar{\rho}$ is $\text{Prob}(\bar{\rho} = \Delta t) = r_l e^{-r_l \Delta t}$, where r_l is the firing rate of the l-th neuron. When the l-th neuron fires a spike, its potential is decreased by 1. The fired spike heads for the $\hat{l}$-th neuron as an excitatory spike with probability $p^+_{l,\hat{l}}$ or as an inhibitory spike with probability $p^-_{l,\hat{l}}$, or it departs from the network/system with probability ν_l. The summation of these probabilities is 1: $\sum_{\hat{l}=1}^{L}(p^+_{l,\hat{l}} + p^-_{l,\hat{l}}) + \nu_l = 1$.

Evidently, the potentials of the L neurons in the system are dynamically changing over time due to the stochastic spikes and firing events. Let $\text{Prob}(k_l(t) > 0)$ denote the probability that the l-th neuron is excited at time t. Accordingly, let $q_l = \lim_{t\to\infty} \text{Prob}(k_l(t) > 0)$ denote stationary excitation probability of the l-th neuron. Due to the stochastic and distributed nature of the behaviors of the whole spiking neural system, it is difficult to obtain the value of q_l. For a system with fixed configurations and inputs, a straightforward method is to estimate the value of q_l by using the Monte Carlo method. However, this method may not enable us to obtain a good estimation of q_l or be applicable when the number of neurons becomes very large.

In [42], Gelenbe presented important results on the excitation probabilities of the neurons of this RNN system. It is proven in [42,45,49] that $q_l = \lim_{t\to\infty} \text{Prob}(k_l(t) > 0)$ can be directly calculated by the following system of equations:

$$q_l = \min\left(\frac{\lambda^+_l}{r_l + \lambda^-_l}, 1\right), \tag{1}$$

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

where $\lambda_l^+ = \Lambda_l + \sum_{\hat{l}=1}^{N} q_{\hat{l}} w_{\hat{l},l}^+$, $\lambda_l^- = \lambda_l + \sum_{\hat{l}=1}^{N} q_{\hat{l}} w_{\hat{l},l}^-$, $w_{\hat{l},l}^+ = r_{\hat{l}} p_{\hat{l},l}^+$, $w_{\hat{l},l}^- = r_{\hat{l}} p_{\hat{l},l}^-$ and $l = 1, \ldots, L$. Here Λ_l and λ_l are respectively the arrival rates of external excitatory and inhibitory spikes and r_l is the firing rate of the l-th neuron. In addition, $\sum_{\hat{l}=1}^{L}(p_{l,\hat{l}}^+ + p_{l,\hat{l}}^-) + \nu_l = 1$. Operation $\min(a, b)$ produces the smaller one between a and b. In [49], it was shown that the system of N non-linear eq. (1) have a solution which is unique. Therefore, the states of the RNN can be efficiently obtained by solving a system of equations without requiring the Monte Carlo method [116].

The RNN's approximation property shows that it is a function approximator with the universal approximation property (UAP). In particular, in [74,75,78], it is shown that, for any continuous real-valued and bounded function $f(X) : [0, 1]^{1\times N} \to \mathbb{R}$, there exists an RNN that approximates $f(X)$ uniformly on $[0, 1]^{1\times N}$. The work in [157] presents a constructive proof for this UAP theorem that lays a theoretical basis for the learning capability of the RNN and its capability for DL. The RNN function approximator is demonstrated to have a lower computational complexity than the orthogonal–polynomial function approximator in [163] and the one-hidden-layer MLP. The RNN function approximator, equipped with the proposed configuration/learning procedure, is then applied as a tool for solving pattern-classification problems. Numerical experiments on various datasets demonstrate that the RNN classifier is more efficient than the Chebyshev-polynomial neural network [163], ELM [94], MLP equipped with the Levenberg–Marquardt (LM) algorithm, radial-basis-function neural networks [152,160], and support vector machine [24].

2.4.2. Learning in random neural network. Solving supervised learning problems is a key issue in machine learning and artificial intelligence [8,165], as this is a requirement for the majority of practical applications of neural networks. In general, supervised learning of a system requires it to learn mapping from the input data (integers or real values) to the output data (labels or real values) in a given training dataset. Investigations have been carried out to facilitate the development of the RNN learning methods since 1993 [49], with the goal of solving the RNN's learning problems. In the work of [49], Gelenbe developed a gradient-descent learning algorithm for the RNN, which seeks non-negative weights in terms of the gradient descent of a quadratic cost function and is the most widely-used learning algorithm for the RNN. Likas [111] utilizes quasi-Newton methods and the derivative results of gradient descent in [49] for the RNN learning. The proposed algorithm was tested on the RNN with two different architectures, where the fully recurrent architecture yielded poor results while the feed-forward architecture yielded smaller errors with fewer steps than the pure gradient-descent algorithm. It is also reported that the proposed algorithm has higher computational complexity and cannot be applied to online learning.

Based on [49], Basterrech [12] exploited the LM optimization procedure and adaptive momentum. Its performance was evaluated when applied to several problems. Compared with the gradient-descent algorithm, it achieved faster convergence speed in almost all cases but proved less robust in solving the XOR problem. The RPROP learning algorithm introduced in [127], which performs a direct adaptation of the weight step based on the local gradient information, is combined with the gradient-descent algorithm [49] for the RNN learning [12]. Georgiopoulos gave a brief description and some critical comments on this RPROP algorithm in [79]. It is reported in [12] that, in the learning of the geometrical images, this RPROP algorithm outperformed the gradient-descent algorithm. The gradient-descent-based learning algorithms may have some inherent weaknesses, such as slow convergence and being trapped into poor local minima. Instead of using the gradient descent, Georgiopoulos applied the adaptive inertia weight particle swarm optimization (AIW-PSO) [124] and differential evolution (DE) approach [139] for the RNN learning in [79]. Then,

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

the performances of the gradient-descent algorithm, RPROP algorithm, AIW-PSO, and DE approach for classifying 12 datasets were compared, and there is no clear overall winner among them. In addition, Timotheou tackled the learning problem of the RNN from a different perspective [142,143]. Specifically, the learning problem was first approximated to obtain a non-negative least-square (NNLS) problem. The next step is to apply the solution to the NNLS problem directly to the RNN learning problem. It is reported in [142,143] that this algorithm achieved better performance than the gradient-descent algorithm in a combinatorial optimization problem emerging in disaster management. Javed et.al [96] proposed to combine the artificial bee colony/particle swarm optimization with the sequential quadratic programming optimization algorithm to train the RNN. The RNN was investigated for DL, and efficient DL algorithms based on the RNN were proposed [66,156].

2.4.3. Applications of random neural network. The RNN has been successfully used in numerous applications [11,79,126,144]. In [64], the RNN is applied in encoding an image into fewer bits in order to achieve image compression by exploiting a feed-forward architecture to learn the mapping from the image to the same image through a narrow passage with fewer hidden-layer neurons, where the extension to video compression applied similar principles [28,29]. In [10], multi RNNs are exploited to learn from training data and recognize shaped objects in clutter by combining the results of all RNNs. In [77], the feed-forward RNN is applied to learning a mapping from the scanning electron microscopy intensity waveform to the cross-section shape (i.e., the profile) such that the profile can be reconstructed from the intensity waveform and the destruction caused by acquiring a cross-section image can be avoided. In [117], the task of Denial of Service (DoS) detection is formulated as a pattern classification problem, and then, with useful input features measured and selected, the RNNs with both the feed-forward and recurrent architectures are exploited to fulfil the task.

Gelenbe proposes the multi-class RNN model in [57] as a generalization following the work of the standard RNN [42]. This multiple-class RNN can be described as a mathematical framework to represent networks that simultaneously process information of various types. Then, by setting the excitatory input rates and the desired outputs of the neuron states as the normalized pixel values, this multiple-class RNN is utilized to learn a given color texture and generate a synthetic color texture that imitates the original one [58]. The earlier work in [76] that uses the standard RNN for black and white texture generation utilizes similar principles. The RNN learning is also applied to image segmentation [113], vehicle classification [95], and texture classification and retrieval [141]. The work by Vlontzos [145] tested the RNN-based DL algorithms on two image recognition applications. Recent study in [97] applied the RNN to controlling the HVAC systems. More applications related to utilizing the learning capability of the RNN can be seen from the critical review in [79]. The many other areas in which the RNN has already been applied are detailed in the reviews in [11,144].

2.4.4. Implementation of random neural network. There are three main approaches to implement the RNN [4]. The most accessible approach is to use computer software tools based on CPU to simulate the RNN spiking behaviors or conduct the RNN equations computations. However, this approach can be of low efficiency because it does not utilize the fact that the RNN can be computed or simulated in a parallel manner. The second approach is still to implement the RNN in computers but in a parallel manner. Specifically, the computations related to the RNN can be conducted simultaneously by a very large number of simple processing elements, where the rapid development of chip multiprocessor and Graphics Processing Unit (GPU) make it more feasible. The third approach is to implement the

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

RNN model in hardware using special-purpose digital or analog components [4]. The RNN's high parallelism can be exploited fully via specially designed hardware, and significant speed increases can be achieved when applying the RNN.

For the first approach, Abdelbaki's work [2] implemented the RNN model using MATLAB software. To the best of the authors' knowledge, no investigation has hitherto been undertaken to implement the RNN using the second approach. In terms of hardware, another of Abdelbaki's studies in [4] shows that the RNN can be implemented by a simple continuous analog circuit that contains only the addition and multiplication operations. In addition, the Cerkez's study [23] proposed a digital realization for the RNN using discrete logic integrated circuits. Kocak's work [103] implemented the RNN-based routing engine of the cognitive packet networks (CPN) in hardware. This innovation significantly decreased the number of weight terms (from $2n^2$ to $2n$) needing to be stored for each RNN in the engine, where the CPN is an alternative to the IP-based network architectures proposed by Gelenbe [52].

3. NON-NEGATIVE AUTOENCODERS WITH SIMPLIFIED RANDOM NEURAL NETWORK

The authors' work in [155] combines the knowledge from three machine-learning fields, which are: the RNN (i.e., the RNN), DL, and non-negative matrix factorization (NMF), to develop new multi-layer non-negative autoencoders that have the potential to be implemented in a highly-distributed manner.

3.1. Technical Background

The mathematical tool of the RNN has existed since 1989 [42,45,49], but is less well known in the machine-learning community. The RNN is developed to mimic the behavior of biological neurons in the brain. In the RNN, an arbitrarily large set of neurons interact with each other via stochastic spikes which modify each neuron's action potential in continuous time. Though the stochastic spiking behaviors are complex, the state probability distribution of the RNN can be calculated by an easily solvable system of non-linear equations. This powerful property of the RNN provides the feasibility of utilizing the techniques from the other machine-learning fields, e.g., the DL and NMF described as the following, for the development of new learning tools based on the RNN.

DL has achieved great success in machine learning [88,107]. It is attractive to investigate whether the network architecture and the training techniques in this area could be combined with the RNN or not. For example. in DL, the feed-forward neural-network architecture, composed of multi-processing layers, allows a model to extract high-level representations from raw data. Pre-training a multi-layer network layer by layer is an effective and widely-adaptable technique to tackle the training difficulty in the network [88,89]. In addition, the typical training procedure called SGD provides a practical choice for handling large datasets [14].

Non-negative matrix factorisation (NMF) is also a popular topic in machine learning [32,92,108,112,149], and it provides another perspective of optimizing the parameters of the RNN with the constraints of non-negativity. Lee [108] suggested that the perception of the whole in the brain may be based on the part-based representations (based on the physiological evidence [146]) and proposed simple yet effective update rules. Hoyer [92] combined sparse coding and NMF that allows control over sparseness. Ding investigated the equivalence between the NMF and K-means clustering in [32,33] and presented simple update rules for orthogonal NMF. Wang [149] provided a comprehensive review on recent processes in the NMF area.

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

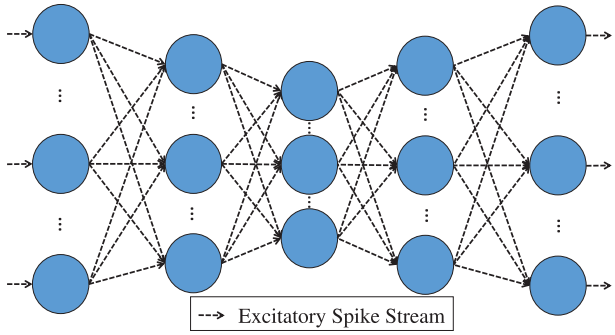


FIGURE **2.** Structure of a multi-layer non-negative RNN autoencoder.

3.2. Network Architecture, Training Strategy and Verification

In [155] and Chapter 4 of [153], the authors first exploit the structure of the RNN equations as a quasi-linear structure. Using it in the feed-forward case, an RNN-based shallow non-negative autoencoder is constructed. This shallow autoencoder is then stacked into a multi-layer feed-forward autoencoder following the network architecture in the DL area [88,89, 107]. The structure of the resultant multi-layer non-negative RNN autoencoder is given in Figure 2.

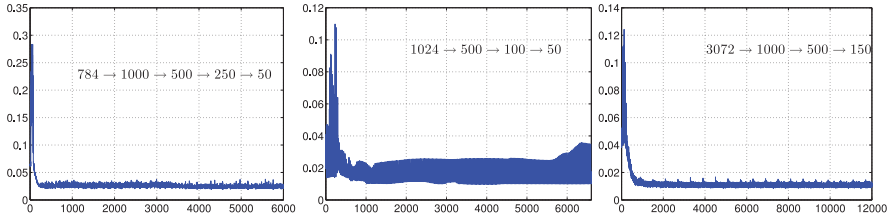
Since connecting weights in the RNN are products of firing rates and transition probabilities, they are subject to the constraints of non-negativity and that the sum of probabilities is no larger than 1, which are called the RNN constraints. In view of this, the conventional gradient descent is not applicable for training such an autoencoder. By adapting the update rules from non-negative graph embedding [112] that is closely related to NMF, applicable update rules are developed for the autoencoder, which satisfy the first RNN constraint of non-negativity. For the second RNN constraint, a check-and-adjust procedure is imposed into the iterative learning process of the learning algorithms. The training procedure of the SGD is also adapted into the algorithms.

Figure 3 presents the curves of reconstruction error versus iteration number by using multi-layer RNN autoencoders to learn the MNIST [105], Yale face [20], and CIFAR-10 [104] datasets. It demonstrates the convergence of proposed RNN autoencoders. In [155] and Chapter 4 of [153], the learning efficacy of the non-negative autoencoders equipped with the learning algorithms is well verified via numerical experiments on both typical image datasets including the MNIST, Yale face, and CIFAR-10 datasets and 16 real-world datasets in different areas from the UCI machine-learning repository [110], in terms of convergence and reconstruction performance and dimensionality-reduction capability for pattern classification.

4. DEEP LEARNING WITH DENSE RANDOM NEURAL NETWORK

The work by the authors in [66–68,154] and Chapter 5 of [153] explores the idea that the human brain contains many important areas that are composed of dense clusters of cells, such as the basal ganglia and various nuclei. These clusters may be composed of similar or identical cells, or of cells of different varieties; however, due to the density of their arrangement, it may be that such clusters allow for a substantial or at least increased amount of direct communication between somata, in addition to the commonly exploited signaling through dendrites and synapses. Thus, based on recurrent RNN [42], a mathematical model

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press



(a) MNIST (b) Yale face (c) CIFAR-10

FIGURE **3.** Reconstruction error (Y-axis) versus iteration number (X-axis) in training multi-layer RNN autoencoders for the MNIST, Yale face, and CIFAR-10 datasets.

of dense clusters/nuclei is developed that models both synapses and direct soma-to-soma interactions, creating a dense RNN (Dense RNN). Each cluster is modeled as a recurrent spiking RNN. Each neuron in each nucleus has a statistically identical interconnection structure with the other cells in the same nucleus; this statistical regularity allows for great individual variability among neurons both with regard to spiking times and interconnection patterns. In the model of the dense nucleus, the authors define the number of cells as N, the probability for repeated firing as p, the firing rate as r, the external excitatory input as λ^+, and external inhibitory input as λ^-. Let q denote the stationary excitation probability of a cell in the nucleus. Mathematical deduction has demonstrated that $q = \zeta(x, N, p, r, \lambda^+, \lambda^-)$, where x is the external inputs from the cells outside the nucleus. The explicit expression of $\zeta(\cdot)$ and details of the mathematical deduction can be found in Chapter 5 of [153]. Theoretical analyses of the nucleus activation function $q = \zeta(x, N, p, r, \lambda^+, \lambda^-)$ are conducted to find the conditions of the values $N, p, r, \lambda^+, \lambda^-$ to maintain the stability of the nucleus. The conditions are summarized as $N \geq 2,\ 0 \leq p \leq 1,\ \lambda^+ > 0, \lambda^- > 0, r > 0$ and $\lambda^- \geq \lambda^+$.

The model of dense clusters allows consideration of a multi-layer architecture (MLA) network wherein each layer is composed of a finite number of dense nuclei. The MLA of dense nuclei, modeled by the Dense RNN and named as MLDRNN, is given schematically in Figure 4. Within each nucleus, the cells communicate with each other in fully-connected recurrent structures that use both synapse and direct soma-to-soma interaction [65]. The communication structure between the various layers of nuclei is a conventional multi-layer feedforward structure: the nuclei in the first layer receive excitation signals from external sources, while each cell in each nucleus creates an inhibitory projection to the layer up. Further, an efficient training procedure is developed for the MLDRNN, which combines unsupervised and supervised learning techniques. Note that most of the work related to the training of the deep neural networks, as well as the RNN, uses the gradient-descent algorithms, where there are inherent disadvantages of slow convergence and the possibilities of being trapped into poor local minimums. The proposed training algorithm by the authors converts the training of the MLDRNN into multiple convex optimization problems that can be solved with comparable or better solutions in a much faster speed than the gradient-descent algorithms. Specifically, let us denote the connecting weight matrices between layers of a L-hidden-layer MLDRNN by $W_l = [w_{h_l,h_{l+1}}] \in \mathbb{R}_{\geq 0}^{H_l \times H_{l+1}}$ with $l = 1, 2, \ldots, L$. In addition, let us define a readout matrix as $W_{L+1} \in \mathbb{R}^{H_{L+1} \times H_{L+2}}$, where H_{L+2} is the dimension of the outputs. The connection weights W_l with $l = 1, 2, \ldots, L-1$ between the first and $L-1$ layers are determined by constructing a series of reconstruction optimization problems, subject to the non-negative constraints of the connection weigths. These optimization problems are designed as convex problems and therefore the solving process can be very

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

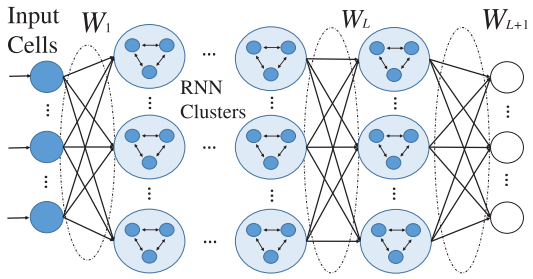


FIGURE **4.** Schematic representation of the MLDRNN.

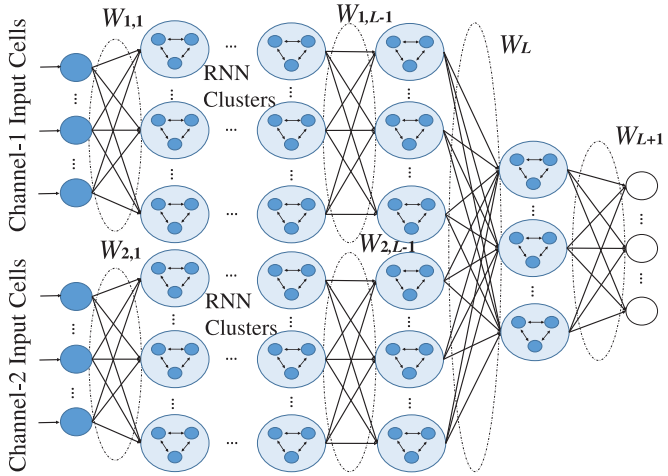


FIGURE **5.** Schematic representation of the MCMLDRNN.

fast. The connection weights W_L is randomly generated within their feasible region. Finally, the determination of the matrix W_{L+1} is formulated as a convex optimization problem that can be solved directly by the Moore–Penrose pseudo-inverse [66,100,140,158,162,163].

In the work of [68,154], the multi-channel MLDRNN (MCMLDRNN), extended from the MLDRNN, is developed to handle multi-channel classification datasets. The schematic representation of the MCMLDRNN is given in Figure 5. The MCMLDRNN is applied to recognizing 3D objects, distinguishing different chemical gases and detecting daily and sports activities of humans. The corresponding numerical results on these three applications indicate that the MCMLDRNN is more accurate than state-of-the-art methods in most cases with high training efficiency, including the MLP, the CNN, and hierarchical extreme learning machine (H-ELM) [140]. The related results can be found in [68,154] and Chapter 5 of [153]. Note that, in addition to the above applications, the work in [159] has successfully applied the dense RNN to inferring the states of servers within the Cloud system.

Finally, in Chapter 5 of [153], the case where the number of cells in a nucleus is very large is investigated, by which the transfer functions of dense nuclei are significantly simplified. Numerical results show that a dense-enough nucleus produces a more efficient algorithm than a not-that-dense nucleus.

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

5. DEEP LEARNING WITH STANDARD RANDOM NEURAL NETWORK

The authors' recent work in [156] and Chapter 6 of [153] goes back to the original simpler structure of the RNN and investigates the power of single-standard RNN cells for DL in the two aspects. This section presents the refined descriptions and procedures.

In the first part, it is shown that, with only positive parameters, the RNN implements image convolution operations similar to the CNN [105,107,136]. The authors examine single-cell, twin-cell, and cluster approaches that model the positive and negative parts of a convolution kernel as the arrival rates of excitatory and inhibitory spikes to receptive cells. The relationship between the RNN and ReLU activation [82] is also investigated and an approximate ReLU-activated convolution operation is presented.

In the second part, a MLA of the standard RNN (MLRNN) is built. The computational complexity of the MLRNN is lower than the MLDRNN model developed in [68,154], and it can also be generalized to handle multi-channel datasets. Numerical results in [156] and Chapter 6 of [153] regarding multi-channel classification datasets show that the MLRNN is effective and that it is arguably the most efficient among five different DL approaches.

5.1. Image Convolution with Single-Cell Approach

Suppose a single RNN cell receives excitatory and inhibitory Poisson spike trains from other cells with rates x^+ and x^-, respectively, and it also receives an excitatory Poisson spike train from the outside world with rate λ^+. The firing rate of the cell is r. Then, based on the RNN theory [42,45,49] and RNN eq. (1), the probability in the steady state that this cell is excited can be calculated by

$$q = \min\left(\frac{\lambda^+ + x^+}{r + x^-}, 1\right). \tag{2}$$

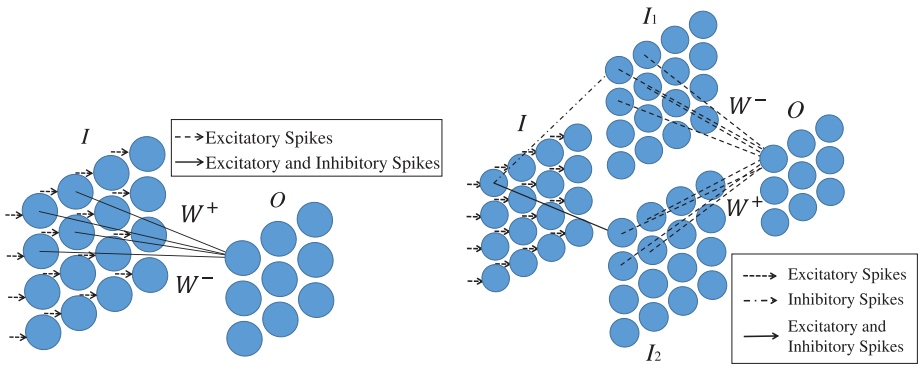
For notational ease, let $\phi(x^+, x^-)|_{\lambda,r} = \min((\lambda + x^+)/(r + x^-), 1)$ and $\phi(\cdot)$ is used as a term-by-term function for vectors and matrices.

The convolution operation of this approach with single cells is shown schematically in Figure 6(a). As see from the figure, the quasi-linear RNN cells (called the LRNN-E cell) [155] presented in Section 3 are utilized as input cells for the convolution inputs I. The convolution kernel W is first normalized to satisfy the RNN probability constraint and separated as $W^+ = \max(W, 0) \geq 0$ and $W^- = \max(-W, 0) \geq 0$ to avoid negativity, where operation $\max(a, b)$ produces the larger element between a and b. In this case, the stationary excitation probabilities (outputs) of the output cells denoted by O can be obtained as $O = \phi(\text{conv}(I, W^+), \text{conv}(I, W^-))$ or $O = \phi(\text{conv}(I, W^-), \text{conv}(I, W^+))$.

5.2. Image Convolution with Twin-Cell Approach

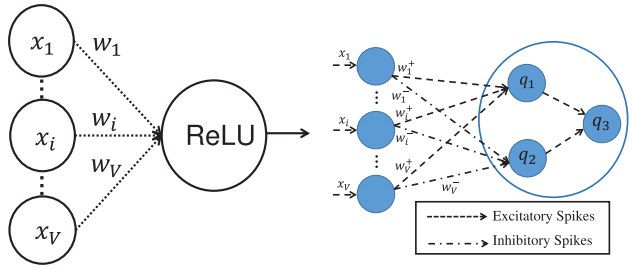
The convolution operation of this approach with twin cells is shown schematically in Figure 6(b). Inputs I pass through the twin arrays and produce I_1 and I_2. The convolution kernel W is then normalized and split into $W^+ \geq 0$ and $W^- \geq 0$, as connecting weights between twin arrays and output cells. By setting the excitatory arrival rate of each output cell as $1 - \text{sum}(W^-)$, the output of the output cells are then obtained as $O = \min(\text{conv}(I_1, W^+) + \text{conv}(I_2, W^-) + \mathbf{1} - \text{sum}(W^-), \mathbf{1}) = \min(\text{conv}(I_1, W) + \mathbf{1}, \mathbf{1})$, where the operation $\text{sum}(\cdot)$ produces the summation of all input elements and $\mathbf{1}$ is an all-one matrix with appropriate dimensions.

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press



(a) Single-cell approach (b) Twin-cell approach

FIGURE **6.** Image convolution operation with RNN using the single-cell and twin-cell approaches.



(a) ReLU unit (b) RNN cluster

FIGURE **7.** A ReLU unit and a cluster of the LRNN-I and LRNN-E cells to approximate the ReLU activation.

5.3. Image Convolution with a Cluster Approach

This approach is based on a type of multi-cell cluster that approximates the ReLU, where the ReLU unit has been widely used in the DL area [82,107]. The cluster is constituted by the LRNN-E cell [155] and another different quasi-linear RNN cell that is deduced from the first-order approximation of the RNN eq. (1).

5.3.1. Relationship between RNN and ReLU. The relationship between the RNN and the ReLU activation function [82,107] is investigated; and it is shown that a cluster of the RNN cells produces approximately the ReLU activation.

ReLU Activation. A single unit with ReLU activation is considered. Suppose the input to the unit is a non-negative vector $X = [x_{1,v}] \in \mathbb{R}^{1\times V}$ in the range $[0, 1]$, and the connecting weights is a vector $W = [w_{v,1}] \in \mathbb{R}^{V\times 1}$ whose elements can be both positive and negative. Then, the output of this unit is described by $\mathrm{ReLU}(XW) = \max(0, XW)$. Figure 7(a) illustrates the corresponding schematic representation of the ReLU.

A Cluster of RNN Cells. The schematic representation of the cluster for producing approximately the ReLU activation is given in Figure 7(b). For illustration, let $W^+ =$

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

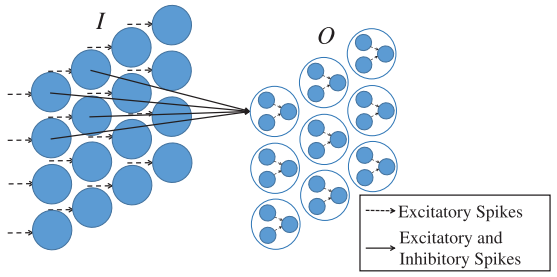


FIGURE **8.** An RNN convolution operation with the cluster approach.

$[w^+_{v,1}] = \max(0, W)$ and $W^- = [w^-_{v,1}] = -\min(0, W)$. It is evident that $W^+ \geq 0$, $W^- \geq 0$, $W = W^+ - W^-$ and $\text{ReLU}(XW) = \max(0, XW^+ - XW^-)$. The input X is imported into the 1st cell with connecting weights W^-, whose output is $q_1 = \min(XW^-, 1)$. The input X is also imported into the 2nd cell with W^+, and its cell output is $q_2 = 1/(1 + XW^+) \approx \min(1 - XW^+, 1)$. Second, the 1st and 2nd cells of q_1 and q_2 are connected to the 3rd cell (or say, the output cell) with connecting weight being 1. The output of the 3rd/output cell is $q_3 = \min(q_2 + q_1, 1) \approx 1 - \text{ReLU}(XW)$. The conditions for the approximation are $XW^- \leq 1$ (that can be loosed as $\sum_{v=1}^{V} w^-_{v,1} = \text{sum}(W^-) \leq 1$ if $X \leq 1$) and $\sum_{v=1}^{V} w^+_{v,1} = \text{sum}(W^+) << 1$.

5.3.2. Cluster-based convolution. The convolution operation with the RNN clusters is shown schematically in Figure 8. As seen from the figure, the input cells receive excitatory spikes with rates I and produce stationary excitation probabilities I. Then, input cells are connected to a set of receptive RNN clusters with local connectivity. To satisfy the conditions for the approximation in the clusters, the convolution kernel is normalized via $W \leftarrow W/\text{sum}(|W|)/10$. In addition, let us split the kernel as $W^+ = \max(W, 0) \geq 0$ and $W^- = \max(-W, 0) \geq 0$. By assuming that $0.1 \ll 1$, it is evident that $\text{sum}(W^+) \leq 0.1 \ll 1$ and $\text{sum}(W^-) \leq 0.1 \ll 1$, which satisfy the approximation conditions. Then, the weights W^+ and W^- are utilized respectively as the rates of the excitatory and inhibitory spikes from the input cells to the clusters. The stationary excitation probabilities of the output cells of the receptive clusters are obtained as

$$O \approx \mathbf{1} - \text{ReLU}(\text{conv}(I, W)). \tag{3}$$

5.4. Individual-Cell-Based Multi-Layer RNN

In [156] and Chapter 6 of [153], the authors then present a MLA of individual RNN cells described in (2) (called the MLRNN). The MLRNN is then adapted to handle multi-channel datasets (MCMLRNN). The schematic representations of the MLRNN and MCMLRNN are respectively given in Figures 9 and 10. The proposed MLRNN and MCMLRNN achieve similar performance as the MLDRNN and MCMLDRNN presented in Section 4 but cost less computation time.

5.4.1. Mathematical model formulation in matrix-form. Suppose there is a dataset represented by a non-negative matrix $X \in \mathbb{R}_{\geq 0}^{D \times N_1}$, where D is the number of instances and each instance has N_1 attributes.

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

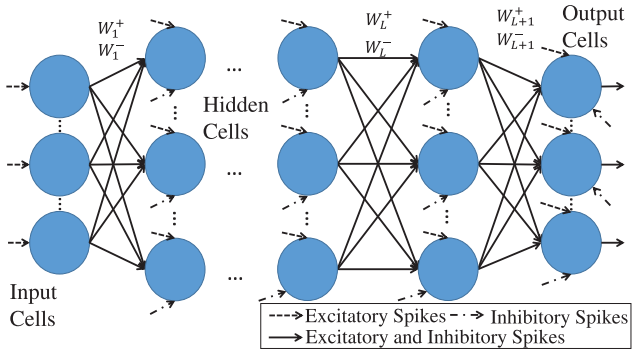


FIGURE **9.** Schematic representation of the MLRNN.

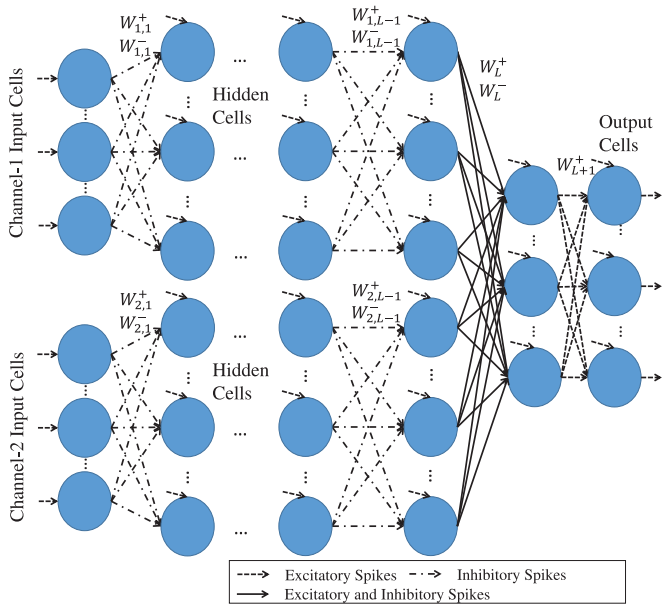


FIGURE **10.** Schematic representation of the MCMLRNN.

1) Let a matrix $Q_l \in \mathbb{R}_{\geq 0}^{D \times N_l}$ $(1 \leq l \leq L+2)$ denote the outputs of the lth layer.
2) Let a matrix $R_l \in \mathbb{R}_{\geq 0}^{D \times N_l}$ denote the firing-rate matrix for cells in the lth layer $(1 \leq l \leq L+2)$. Note that, the fire rate of a cell is the same for all instances.
3) Let two matrices $W_l^+ \in \mathbb{R}_{\geq 0}^{N_l \times N_{l+1}}$ and $W_l^- \in \mathbb{R}_{\geq 0}^{N_l \times N_{l+2}}$ denote excitatory and inhibitory connecting weight matrices between the lth and $(l+1)$th layers for $l = 1, \ldots, L+1$.
4) Let two matrices $\Lambda_l^+ \in \mathbb{R}_{\geq 0}^{D \times N_l}$ and $\Lambda_l^- \in \mathbb{R}_{\geq 0}^{D \times N_l}$ denote the external arrival rate matrices of excitatory and inhibitory spikes for the lth layer $(1 \leq l \leq L+2)$. Note that, the elements in each row in Λ_l^+ and Λ_l^- are the same.

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

In the matrix form, the excitation probability matrix of the MLRNN can be described as:

$$Q_1 = \min\left(\frac{X}{R_1}, 1\right), \tag{4}$$

$$Q_l = \min\left(\frac{\Lambda_l^+ + Q_{l-1}W_{l-1}^+}{R_l + \Lambda_l^- + Q_{l-1}W_{l-1}^-}, 1\right), \tag{5}$$

where $l = 2, \ldots, L+2$ and the division operation between matrices is element-wise. The parameters in the MLRNN are required to satisfy the RNN constraints: $R_1, \ldots, R_{L+2} \geq 0$; $\Lambda_2^+, \ldots, \Lambda_{L+2}^+ \geq 0$; $\Lambda_2^-, \ldots, \Lambda_{L+2}^- \geq 0$; $W_1^+, \ldots, W_{L+1}^+ \geq 0$; $W_1^-, \ldots, W_{L+1}^- \geq 0$ and $\sum_{n_{l+1}=1}^{N_{l+1}}(w_{n_l,n_{l+1}}^+ + w_{n_l,n_{l+1}}^-) \leq r_{n_l}$ with $l = 1, \ldots, L+1$.

5.4.2. Learning problem and training procedure. Suppose there is a labeled dataset with D instances $\{(X, Y)\}$, where $X = [x_{d,n_1}] \in \mathbb{R}_{\geq 0}^{D \times N_1}$ is the input-attribute matrix and $Y = [y_{d,n_{L+2}}] \in [0\ 1]^{D \times N_{L+2}}$ is the desired-output matrix. For the dth instance (x_d, y_d), the input-attribute vector is denoted by $x_d = [x_{d,1}\ x_{d,2}\ \ldots\ x_{d,N_1}] \in \mathbb{R}^{1 \times N_1}$ and the desired-output vector is denoted by $y_d = [y_{d,1}\ y_{d,2}\ \ldots\ y_{d,N_{L+2}}] \in [0\ 1]^{1 \times N_{L+2}}$. The mapping from X to Y is $f : \mathbb{R}_{\geq 0}^{2 \times N_1} \to [0\ 1]^{N_{L+2}}$ and $Y = f(X)$. The objective is to find appropriate values for parameters R_1 and $R_l, \Lambda_l^+, \Lambda_l^-, W_{l-1}^+, W_{l-1}^-$ with $l = 2, \ldots, L+2$ that satisfy the RNN constraints, so that the MLRNN learns the input–output mapping f with the given dataset $\{(X, Y)\}$.

The training/configuration procedure developed in [156] includes the following steps:
Step 1. Configure Λ_l^- with $l = 2, \ldots, L+2$, W_{L+1}^-, R_1 via $\Lambda_l^- \leftarrow 0$ for $l = 2, \ldots, L+2$, $W_{L+1}^- \leftarrow 0$ and $R_1 \leftarrow 1$.
Step 2. Configure W_l^+, W_l^-, Λ_{l+1}^+, R_{l+1} with $l = 1, \ldots, L-1$:

1) $W_l^+ \leftarrow 0$ with $l = 1, \ldots, L-1$;
2) W_l^- with $l = 1, \ldots, L-1$ are configured by solving a series of reconstruction problems;
3) Λ_{l+1}^+, R_{l+1} are configured via $\Lambda_{l+1}^+ \leftarrow \max(X_l W_l^-)/5$ and $R_{l+1} \leftarrow \max(X_l W_l^-)/5$ with $l = 1, \ldots, L-1$.

Step 3. Configure W_L^+, W_L^-, Λ_{L+1}^+, R_{L+1}, W_{L+1}^+, Λ_{L+2}^+, R_{L+2}: a single-hidden-layer RNN-based artificial neural network is first constructed using the ELM concept [100,140] (with $N_{L+1}/2$ hidden units and N_{L+2} output units); and then its weights are utilized to configure W_L^+, W_L^-, Λ_{L+1}^+, R_{L+1}, W_{L+1}^+, Λ_{L+2}^+, R_{L+2}.

5.4.3. Multi-channel MLRNN. The MLRNN can be adapted to handle multi-channel datasets (denoted as the MCMLRNN), shown in Figure 10, where the connecting weights between layers for only Channel-c ($c = 1, \ldots, C$) are $W_{c,l}^+, W_{c,l}^- \geq 0$ ($l = 1, \ldots, L-1$), those between the $(L-1)$th and Lth hidden layers are $W_L^+, W_L^- \geq 0$ and output weights are $W_{L+1}^+ \geq 0$. The training procedure of the MCMLRNN can be generalized from that of the MLRNN.

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

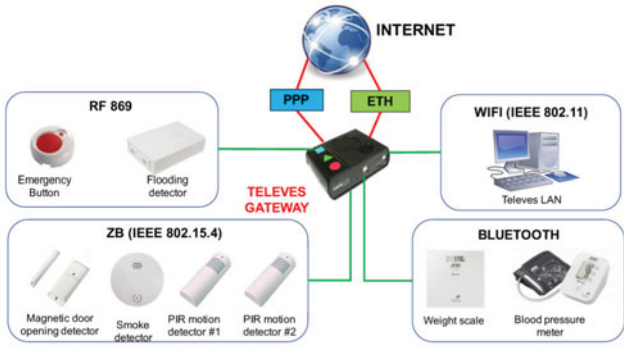


FIGURE **11.** Configuration of the IoT system used for the experiment.

6. APPLICATIONS TO SENSING AND IOT

This section presents how the RNN and its deep algorithm is used for 1) the detection of anomalies and attacks on IoT devices in the previous work [17,18], and 2) object recognition in [68,156] that is relevant, since many IoT devices (such as cameras) use DL to identify object in images.

6.1. Detecting Attacks Against IoT-Connected Home Environments

In an IoT-connected home environment, there may be dozens or even hundreds of sensors with various functions. Most of the sensors are connected to the IoT gateway, and the sensor data are aggregated and processed by the gateway before being sent to the Internet. As IoT gateways sit at the intersection of edge devices (sensors and actuators) and the Internet, they are vulnerable to both traditional IP attacks and to attacks against wireless sensor networks. Therefore, the authors' previous work in [17,18] presented the application of the RNN and its deep algorithm for the detection of network attacks within a Carelife IoT system.

6.1.1. Experiment setup. The experiments are based on a Carelife IoT system, whose overall framework is given in Figure 11. In the system, the Televes gateway was connected to the Internet using a 3G SIM card, and various sensors were connected and registered by the gateway. Software modules were installed on the gateway in order to capture and parse the data packets exchanged with the sensors and Internet servers. Packets were captured for a complete weekend on all the network interfaces of the gateway. Note that the captured data packets were stored in the .pcap file format.

Specifically, we focus on the packets captured on the point-to-point interface (a wide-area-network interface based on 3G data communication). In total, 100, 653 frames were captured on this interface during the experiment, 50, 296 IP packets were received by the gateway, and 41, 938 IP packets were sent by it. As a whole, the IP traffic exchanged with the gateway is composed of 93.8% of TCP packets, 4.1% of UDP packets, and 2.1% of ICMP packets.

6.1.2. Dataset construction. First, starting from the captured packets, statistical data in time series, which are metric values closely related to network attacks, can be obtained. Second, samples are extracted from the time-series statistical data by setting a sliding

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

window with length l. If a sample $X_d \in \mathbb{R}^{1\times l}$ is extracted in the non-attack case, then we assign the label of this sample denoted as y_d as 0; otherwise, if it is extracted in the attack case, the label of this sample is assigned as $y_d = 1$. Then, we have a dataset $\{(X_d, y_d)|d = 1, \ldots, D\}$, where the input is the statistical data extracted from captured packet data and the output is a binary value.

6.1.3. Attack detection results. This subsection presents the empirical results obtained for the detection of TCP SYN attacks.

To achieve this objective, a training dataset is required. Using Scapy, a Python script was written for generating such attacks. The resulting .pcap files contain packet flows of both normal and attack cases, and the corresponding labels are explicit. According to the procedures in Subsection 6.1.2, the training dataset is constructed with these .pcap files and used to train a MLDRNN.

In order to test the trained MLDRNN, the utility tool mergecap was used to insert a SYN attack into the .pcap files containing the packets captured during the experiment of the IoT system. As an example, Figure 12(a) plots the time-series for the difference between the numbers of initiated and established TCP connections per time slot (10 second), which was extracted from a .pcap file obtained using the insertion procedure. As can be observed in Figure 12(b), the MLDRNN correctly predicts that there was an attack.

6.2. Recognizing 3D Objects

This section presents the application of the RNN and its DL algorithms in [68,153] for recognizing objects via images, which is a common task in a IoT system with cameras.

6.2.1. NORB dataset. The small NORB dataset [106] is intended for experiments in 3D object recognition from shape. The objects belong to 5 generic categories: four-legged animals, human figures, airplanes, trucks, and cars. The instance numbers for both training and testing are 24,300. There are two 96×96 images in each instance which are downsampled into 32×32. Examples of the two images of each class are given in Figure 13. All images are whitened using the code provided by [140]. The objective is to recognize the generic category of a given instance using the corresponding two images. The number of channels is 2.

6.2.2. Experiment settings. The structures for the MLDRNN, MCMLDRNN, MCMLDRNN1, MRCMLDRNN2, H-ELM, and MLP (Tanh activation) are respectively 2048-500-2000-5, 2x1024-2x500-2000-5, 2x1024-80x2x100-10x160x10-8000-5, 2x1024-2x2x500-2x4x500-5, 2048-1000-1000-12000-5, and 2048-500-500-5. The CNN with Tanh activation is constituted by two convolution layers, a pooling layer, a fully-connected layer, and an output layer and each convolution layer has 30 filters with kernel size 2x2. The dropout technique developed to prevent from overfitting [137] is exploited for the MLP and CNN. The Adadelta optimizer [161] is used to train the MLP and CNN; the training method of the H-ELM is specified in [140]. For each method, five trials are conducted. For the MLP and CNN, there are 50 epochs in each trial and both Theano and Tensorflow are used; in the cases of early stopping, the number of epochs is 4.

6.2.3. Experiment results. The results of different methods on the training and testing accuracies and the training and testing time are summarized in Table 2. The MCMLDRNN

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

https://doi.org/10.1017/S026999648180005 8X Published online by Cambridge University Press

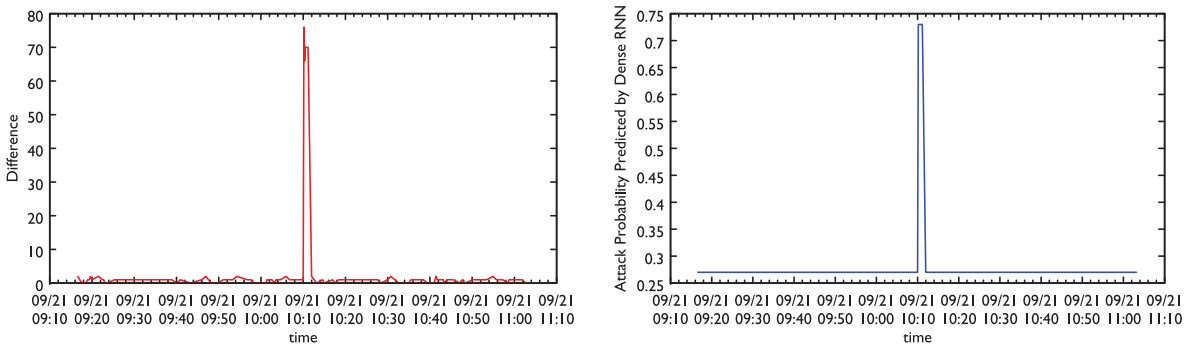


FIGURE **12.** Scenario where a SYN attack, starting at 10:10 AM and lasting for 40 seconds, was inserted into the normal traffic captured from 9:15 AM to 11:03 AM on September 21, 2017: (a) time-series of the difference between the numbers of initiated and established TCP connections per time slot (10 s), and (b) attack probability predicted by the MLDRNN.

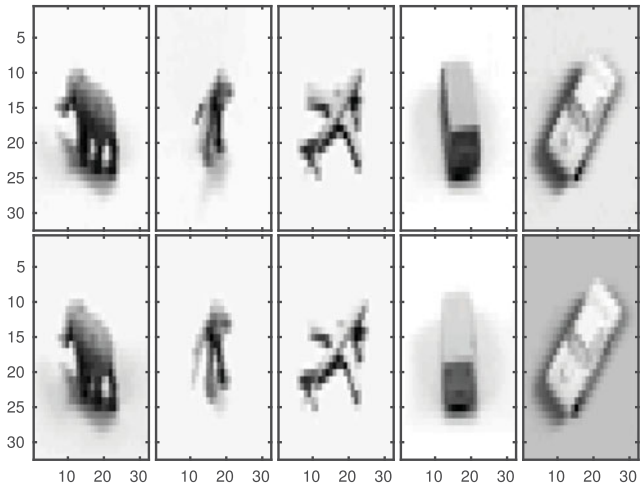


FIGURE **13.** Examples of two images of each class in the NORB dataset: four-legged animals, human figures, airplanes, trucks, and cars.

obtains the second highest testing accuracy (91.19%), which is very close to the one reported in [140]. With the aid of the dropout technique, the CNN achieves a comparable testing accuracy (91.09%) with the MCMLDRNN. However, the computational complexity is much higher. Specifically, using only the CPU, the MCMLDRNN can be trained and applied (tested) respectively around 100 and 10 times faster than the CNN. Aided by the computational power of the GPU, the training of the CNN is still more than 15 times slower than that of the MCMLDRNN. The testing accuracy of the improved MLDRNN is slightly lower than that of the MCMLDRNN, as are the training and testing times. Therefore, both the improved MLDRNN and MCMLDRNN are good classifiers for the NORB dataset with high accuracy and training efficiency.

7. CONCLUSIONS AND FUTURE WORK

7.1. Summary of Achievements

This paper has presented recent progresses on connecting the RNN methods and DL in [66–68,153–156,159] since the first related work was published in 2016 [66]:

1) Based on the RNN function approximator proposed in 1999 [74,75,78], the approximation capability of the RNN has been investigated and an efficient classifier has been developed.
2) Multi-layer non-negative RNN autoencoders have been proposed for robust dimension reduction.
3) The model of the dense RNN that incorporates both soma-to-soma interactions and conventional synaptic connections among neurons has been proposed and an efficient DL tool based on the dense RNN has been developed.
4) The power of the standard RNN for DL has been investigated. The ability of the RNN with only positive parameters to conduct image convolution operations has been demonstrated. The MLRNN equipped with the developed training algorithm achieves

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

TABLE 2. Training and testing accuracies (%) and time (s) of different methods on NORB dataset for recognizing 3D objects.

Method	Accuracies (%)		Times (s)	
	Training	Testing	Training	Testing
MLP	89.43 ± 10.49	75.70 ± 9.07	424.60 ± 37.47	1.46 ± 0.09
MLP+dropout	82.19 ± 17.54	64.61 ± 19.08	457.85 ± 2.64	2.07 ± 0.08
CNN	100.00 ± 0	89.59 ± 0.24	14754.6 ± 17.21	64.59 ± 0.21
CNN+dropout	100.00 ± 0	90.61 ± 2.06	15001.3 ± 28.68	65.67 ± 0.36
MLP*	99.40 ± 0.52	83.29 ± 3.24	119.87 ± 5.12	0.80 ± 0.17
MLP+dropout*	99.41 ± 0.38	83.28 ± 1.08	123.44 ± 3.16	0.93 ± 0.08
CNN*	100.00 ± 0	90.31 ± 0.35	282.46 ± 3.26	2.29 ± 0.08
CNN+dropout*	100.00 ± 0	91.09 ± 0.49	292.23 ± 1.64	2.31 ± 0.06
CNN*‡	99.55 ± 0.39	89.07 ± 1.57	139.47 ± 2.01	6.16 ± 0.03
CNN+dropout*‡	99.64 ± 0.22	88.18 ± 1.15	142.32 ± 1.06	6.27 ± 0.05
H-ELM	99.80 ± 0.10	87.36 ± 0.85	30.79 ± 1.41	5.80 ± 0.21
Original MLDRNN	99.94 ± 0.03	87.11 ± 0.44	12.12 ± 0.22	3.85 ± 0.19
Improved MLDRNN	99.99 ± 0	90.46 ± 0.60	12.91 ± 0.31	4.00 ± 0.27
MCMLDRNN	99.99 ± 0.01	91.19 ± 0.63	15.11 ± 0.26	5.95 ± 0.21
MCMLDRNN1	99.82 ± 0.06	90.64 ± 0.58	357.11 ± 7.60	123.25 ± 5.91
MCMLDRNN2	99.51 ± 0.11	90.83 ± 0.34	54.89 ± 0.39	8.65 ± 0.28
H-ELM⋆	–	**91.28**	–	–

*These experiments are conducted using the GPU.
‡Early-stopping is used to reduce over-fitting.
⋆These data are obtained directly from [140].

comparable or better classification at a lower computation cost than conventional DL methods.

The achievements in the RNN methods and DL can be summarized into the following aspects:

1) The properties of the RNNs have been investigated from different aspects, such as its approximation properties, various types of structures, and numerous learning strategies.
2) Based on these RNN-property investigations, the RNN has been successfully connected with DL and the capabilities of the RNN as a DL tool are well investigated.
3) Efficient and effective neural-network learning tools based on the RNN have been developed for handling various data challenges arising from various systems, such as the over-fitting effect, diversified attribute characteristics, speed requirements, and so forth.

7.2. Future Work

In the directions of DL with the RNN, future work could focus on the following aspects:

1) The capabilities of the DL tools based on the RNN and dense RNN could be further evaluated in more practical applications when solving real-world problems.
2) Extensive comparisons of the proposed RNN tools with other related machine-learning tools could be further investigated.

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

3) The training and testing phases of the multi-layer RNN could be implemented using the existing DL platforms, such as Tensorflow [1].
4) The feasibility of applying the RNN theory to modeling the existing neuromorphic computing hardware platforms may be worthy of investigation.
5) The approach to utilizing linearization facilitates the analysis of deep neural networks from a theoretical perspective [153], and as such this may be worth investigating further in order to better understand the deep neural networks from the theoretical perspective and develop improved DL tools.
6) The size of dataset in the issues of DL can be very large. Transforming the DL problem into a moment-learning problem [153] using the statistical estimations of raw moments from the dataset could significantly reduce the problem's complexity. Thus, this could be a promising direction for future work on DL.

Acknowledgment

The author would like to thank the support of the President's PhD Scholarship from Imperial College London.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., & Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/.
2. Abdelbaki, H.E. (1999). Random neural network simulator for use with matlab. *Technical report*.
3. Abdelbaki, H., Gelenbe, E., & El-Khamy, S.E. (1999). Random neural network decoder for error correcting codes. *Neural Networks, 1999. IJCNN'99. International Joint Conference on*. Vol. 5. IEEE, 3241–3245.
4. Abdelbaki, H., Gelenbe, E., & El-Khamy, S.E. (2000). Analog hardware implementation of the random neural network model. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*. Vol. 4, 197–201.
5. Abdelbaki, H.M., Hussain, K., & Gelenbe, E. (2001). A laser intensity image based automatic vehicle classification system. *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*. IEEE, 460–465.
6. Adeel, A., Larijani, H., & Ahmadinia, A. (2015). Resource management and inter-cell-interference coordination in lte uplink system using random neural network and optimization. *IEEE Access* 3: 1963–1979.
7. Aguilar, J. & Gelenbe, E. (1997). Task assignment and transaction clustering heuristics for distributed systems. *Information Sciences* 97(1–2): 199–219.
8. Anthony, M., Bartlett, P.L. (2009) *Neural network learning: theoretical foundations*. New York: Cambridge University Press.
9. Atalay, V., Gelenbe, E., & Yalabik, N. (1992). The random neural network model for texture generation. *International Journal of Pattern Recognition and Artificial Intelligence* 6(01): 131–141.
10. Bakircioglu, H. & Gelenbe, E. (1998). Random neural network recognition of shaped objects in strong clutter. *Applications of artificial neural networks in image processing III*. Vol. 3307. International Society for Optics and Photonics, 22–29.
11. Bakırcıoğlu, H. & Koçak, T. (2000). Survey of random neural network applications. *European Journal of Operational Research* 126(2): 319–330.
12. Basterrech, S., Mohammed, S., Rubino, G., & Soliman, M. (2009). Levenberg-marquardt training algorithms for random neural networks. *The Computer Journal* 54(1): 125–135.

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

13. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., & Zieba, K. (Apr. 2016). End to End Learning for Self-Driving Cars, *ArXiv e-prints*.
14. Bousquet, O. & Bottou, L. (2008). The tradeoffs of large scale learning. *Advances in neural information processing systems*, 161–168.
15. Brun, O., Wang, L., & Gelenbe, E. (2016). Big data for autonomic intercontinental overlays. *IEEE Journal on Selected Areas in Communications* 34(3): 575–583.
16. Brun, O., Yin, Y., Gelenbe, E., Kadioglu, Y.M., Augusto-Gonzalez, J., & Ramos, M. (2018). Deep learning with dense random neural networks for detecting attacks against iot-connected home environments. *Security in Computer and Information Sciences: First International ISCIS Security Workshop 2018, Euro-CYBERSEC 2018, London, UK, February 26-27, 2018*. Lecture Notes CCIS No. 821, Springer Verlag.
17. Brun, O., Yin, Y., Gelenbe, E., Kadioglu, Y., Augusto-Gonzalez, J., & Ramos, M. (2018). Deep learning with dense random neural networks for detecting attacks against iot-connected home environments. In Gelenbe, E., Campegiani, P., Czachorski, T., Katsikas, S., Komnios, I., Romano, L., & Tzovaras, D., (eds.), *Recent Cybersecurity Research in Europe: Proceedings of the 2018 ISCIS Security Workshop, Imperial College London*. Lecture Notes CCIS No. 821, Springer Verlag.
18. Brun, O., Yin, Y., & Gelenbe, E. Deep learning with dense random neural network for detecting attacks against iot-connected home environments, *Procedia Computer Science*. Vol. 134, 458 – 463, 2018, the 15th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2018) / The 13th International Conference on Future Networks and Communications (FNC-2018) / Affiliated Workshops. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050918311487.
19. Burks, A.W., Goldstine, H.H., & von Neumann, J. (1946). Preliminary discussion of the logical design of an electronic computing instrument. *Report to the US Army Ordenance Department*.
20. Cai, D., He, X., Hu, Y., Han, J., & Huang, T. (2007). Learning a spatially smooth subspace for face recognition. *2007 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1–7.
21. Cambria, E., Huang, G.-B., Kasun, L.L.C., Zhou, H., Vong, C.M., Lin, J., Yin, J., Cai, Z., Liu, Q., Li, K., Leung, V.C.M., Feng, L., Ong, Y.-S., Lim, M.-H., Akusok, A., Lendasse, A., Corona, F., Nian, R., Miche, Y., Gastaldo, P., Zunino, R., Decherchi, S., Yang, X., Mao, K., Oh, B.-S., Jeon, J., Toh, K.-A., Teoh, A.B.J., Kim, J., Yu, H., Chen, Y., & Liu, J. (2013). Extreme learning machines [trends & controversies]. *IEEE Intelligent Systems* 28(6): 30–59. [Online]. Available: http://dx.doi.org/10.1109/MIS.2013.140.
22. Carnevale, N.T. & Hines, M.L. (2006). *The NEURON book*. New York: Cambridge University Press.
23. Çerkez, C., Aybay, I., & Halici, U. (1997). A digital neuron realization for the random neural network model. *Neural Networks, 1997. International Conference on*. Vol. 2. IEEE, 1000–1004.
24. Chang, C.-C. & Lin, C.-J. (2011). Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)* 2(3): 27.
25. Cheng, H.-P., Wen, W., Song, C., Liu, B., Li, H., & Chen, Y. (2016). Exploring the optimal learning technique for ibm truenorth platform to overcome quantization loss. *Nanoscale Architectures (NANOARCH), 2016 IEEE/ACM International Symposium on*. IEEE, 185–190.
26. Cramer, C. & Gelenbe, E. (2000). Video quality and traffic qos in learning-based subsampled and receiver-interpolated video sequences. *IEEE Journal on Selected Areas in Communications* 18(2): 150–167. [Online]. Available: https://doi.org/10.1109/49.824788.
27. Cramer, C., Gelenbe, E., & Bakircloglu, H. (1996). Low bit-rate video compression with neural networks and temporal subsampling. *Proceedings of the IEEE* 84(10): 1529–1543.
28. Cramer, C., Gelenbe, E., & Gelenbe, P. (1998). Image and video compression. *IEEE Potentials* 17(1): 29–33.
29. Cramer, C., Gelenbe, E., & Bakircioglu, H. (1996). Video compression with random neural networks. *Neural Networks for Identification, Control, Robotics, and Signal/Image Processing, 1996. Proceedings., International Workshop on*. IEEE, 476–484.
30. Davison, A., Brüderle, D., Kremkow, J., Muller, E., Pecevski, D., Perrinet, L., & Yger, P. (2009). Pynn: a common interface for neuronal network simulators.
31. Diehl, P.U., Pedroni, B.U., Cassidy, A., Merolla, P., Neftci, E., & Zarrella, G. (2016). Truehappiness: Neuromorphic emotion recognition on truenorth. *Neural Networks (IJCNN), 2016 International Joint Conference on*. IEEE, 4278–4285.
32. Ding, C., Li, T., Peng, W., & Park, H. (2006). Orthogonal nonnegative matrix t-factorizations for clustering. *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 126–135.

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

33. Ding, C.H., He, X., & Simon, H.D. (2005). On the equivalence of nonnegative matrix factorization and spectral clustering. *SDM*. Vol. 5. SIAM, 606–610.
34. Dominguez-Morales, J.P., Jimenez-Fernandez, A., Rios-Navarro, A., Cerezuela-Escudero, E., Gutierrez-Galan, D., Dominguez-Morales, M.J., & Jimenez-Moreno, G. (2016). Multilayer spiking neural network for audio samples classification using spinnaker. *International Conference on Artificial Neural Networks*. Springer, 45–53.
35. Esser, S.K., Andreopoulos, A., Appuswamy, R., Datta, P., Barch, D., Amir, A., Arthur, J., Cassidy, A., Flickner, M., Merolla, P. *et al.* (2013). Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores. *Neural Networks (IJCNN), The 2013 International Joint Conference on*. IEEE, 1–10.
36. Esser, S.K., Appuswamy, R., Merolla, P., Arthur, J.V., & Modha, D.S. (2015). Backpropagation for energy-efficient neuromorphic computing. *Advances in Neural Information Processing Systems*, 1117–1125.
37. Esser, S.K., Merolla, P.A., Arthur, J.V., Cassidy, A.S., Appuswamy, R., Andreopoulos, A., Berg, D.J., McKinstry, J.L., Melano, T., Barch, D.R. *et al.* (2016). Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences* 113: 201604850.
38. Fourneau, J.-M. & Gelenbe, E. (2017). G-networks with adders. *Future Internet* 9(3), 34.
39. François, F. & Gelenbe, E. (2016). Towards a cognitive routing engine for software defined networks. *ICC 2016*. IEEE, 1–6.
40. François, F. & Gelenbe, E. (2016). Optimizing secure sdn-enabled inter-data centre overlay networks through cognitive routing. *MASCOTS 2016, IEEE Computer Society*. IEEE, 283–288.
41. Furber, S.B., Galluppi, F., Temple, S., & Plana, L.A. (2014). The spinnaker project. *Proceedings of the IEEE* 102(5): 652–665.
42. Gelenbe, E. (1989). Random neural networks with negative and positive signals and product form solution. *Neural Computation* 1(4): 502–510.
43. Gelenbe, E. (1989). Réseaux stochastiques ouverts avec clients négatifs et positifs, et réseaux neuronaux. *Comptes-Rendus Acad. Sciences de Paris, Série 2* 309: 979–982.
44. Gelenbe, E. (1990). Reseaux neuronaux aléatoires stables. *Comptes Rendus de l'Académie des Sciences. Série 2* 310(3): 177–180.
45. Gelenbe, E. (1990). Stability of the random neural network model. *Neural Computation* 2(2): 239–247.
46. Gelenbe, E. (1991). Product-form queueing networks with negative and positive customers. *Journal of Applied Probability* 28(3): 656–663.
47. Gelenbe, E. (1993). G-networks by triggered customer movement. *Journal of Applied Probability* 30(3): 742–748.
48. Gelenbe, E. (1993). G-networks with signals and batch removal. *Probability in the Engineering and Informational Sciences* 7(3): 335–342.
49. Gelenbe, E. (1993). Learning in the recurrent random neural network. *Neural Computation* 5: 154–164.
50. Gelenbe, E. (1994). G-networks: a unifying model for neural and queueing networks. *Annals of Operations Research* 48(5): 433–461.
51. Gelenbe, E. (2007). Steady-state solution of probabilistic gene regulatory networks. *Physical Review E* 76: 031903-1–031903-8.
52. Gelenbe, E. (2009). Steps toward self-aware networks. *Communications of the ACM* 52(7): 66–75.
53. Gelenbe, E. (2012). Natural computation. *The Computer Journal* 55(7): 848–851.
54. Gelenbe, E. & Abdelrahman, O.H. (2018). An energy packet network model for mobile networks with energy harvesting. *Nonlinear Theory and Its Applications, IEICE* 9(3): 1–15.
55. Gelenbe, E. & Ceran, E.T. (2016). Energy packet networks with energy harvesting. *IEEE Access* 4: 1321–1331.
56. Gelenbe, E. & Cramer, C. (1998). Oscillatory corticothalamic response to somatosensory input. *Bio Systems* 48(1–3): 67–75.
57. Gelenbe, E. & Fourneau, J.-M. (1999). Random neural networks with multiple classes of signals. *Neural Computation* 11(4): 953–963.
58. Gelenbe, E. & Hussain, K.F. (2002). Learning in the multiple class random neural network. *IEEE Transactions on Neural Networks* 13(6): 1257–1267.
59. Gelenbe, E. & Kazhmaganbetova, Z. (2014). Cognitive packet network for bilateral asymmetric connections. *IEEE Trans. Industrial Informatics* 10(3): 1717–1725.
60. Gelenbe, E. & Marin, A. (2015). Interconnected wireless sensors with energy harvesting. *Analytical and Stochastic Modelling Techniques and Applications - 22nd International Conference, ASMTA 2015, Albena, Bulgaria, May 26–29, 2015. Proceedings*, 87–99.

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

61. Gelenbe, E. & Morfopoulou, C. (2010). A framework for energy-aware routing in packet networks. *The Computer Journal* 54(6): 850–859.
62. Gelenbe, E. & Pujolle, G. (1987). *Introduction to Networks of Queues*. Translation of "Introduction aux Réseaux de Files d'Attente", Eyrolles, Paris, 1982, published by John Wiley Ltd, New York and Chichester.
63. Gelenbe, E. & Schassberger, R. (1992). Stability of product form g-networks. *Probability in the Engineering and Informational Sciences* 6(3): 271–276.
64. Gelenbe, E. & Sungur, M. (1994). Random network learning and image compression. *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*. Vol. 6. IEEE, 3996–3999.
65. Gelenbe, E. & Timotheou, S. (2008). Random neural networks with synchronized interactions. *Neural Computation* 20(9): 2308–2324.
66. Gelenbe, E. & Yin, Y. (2016). Deep learning with random neural networks. *2016 International Joint Conference on Neural Networks (IJCNN)*, 1633–1638.
67. Gelenbe, E. & Yin, Y. (2016). Deep learning with random neural networks. *SAI Intelligent Systems Conference 2016*, 907–912.
68. Gelenbe, E. & Yin, Y. (2017). Deep learning with dense random neural networks. *International Conference on Man–Machine Interactions*. Springer, 3–18.
69. Gelenbe, E., Glynn, P., & Sigman, K. (1991). Queues with negative arrivals. *Journal of Applied Probability* 28(1): 245–250.
70. Gelenbe, E., Stafylopatis, A., & Likas, A. (1991). Associative memory operations of the random neura network. *Proceedings of the International Conference on Artificial Neural Networks*, 307–312.
71. Gelenbe, E., Feng, Y., & Krishnan, K.R.R. (1996). Neural network methods for volumetric magnetic resonance imaging of the human brain. *Proceedings of the IEEE* 84(10): 1488–1496.
72. Gelenbe, E., Sungur, M., Cramer, C., & Gelenbe, P. (1996). Traffic and video quality with adaptive neural compression. *Multimedia Systems* 4(6): 357–369. [Online]. Available: https://doi.org/10.1007/s005300050037.
73. Gelenbe, E., Ghanwani, A., & Srinivasan, V. (1997). Improved neural heuristics for multicast routing. *IEEE Journal on Selected Areas in Communications* 15(2): 147–155.
74. Gelenbe, E., Mao, Z., & Li, Y. (Aug 1999). Approximation by random networks with bounded number of layers. *Neural Networks for Signal Processing IX: Proceedings of the 1999 IEEE Signal Processing Society Workshop (Cat. No.98TH8468)*, 166–175.
75. Gelenbe, E., Mao, Z., & Li, Y. (1999). Function approximation with spiked random networks. *IEEE Transactions on Neural Networks* 10(1): 3–9.
76. Gelenbe, E., Hussain, K.F., & Abdelbaki, H. (2000). Random neural network texture model. *Applications of Artificial Neural Networks in Image Processing V*. Vol. 3962. International Society for Optics and Photonics, 104–112.
77. Gelenbe, E., Koçak, T., & Wang, R. (2004). Wafer surface reconstruction from top–down scanning electron microscope images. *Microelectronic Engineering* 75(2): 216–233.
78. Gelenbe, E., Mao, Z.-H., & Li, Y.-D. (2004). Function approximation by random neural networks with a bounded number of layers. *Differential Equations and Dynamical Systems* 12(1): 143–170.
79. Georgiopoulos, M., Li, C., & Kocak, T. (2011). Learning in the feed-forward random neural network: a critical review. *Performance Evaluation* 68(4): 361–384, g-Networks and their Applications. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166531610000970.
80. Gewaltig, M.-O. & Diesmann, M. (2007). Nest (neural simulation tool). *Scholarpedia* 2(4): 1430.
81. Glorot, X. & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Aistats*. Vol. 9, 249–256.
82. Glorot, X., Bordes, A., & Bengio, Y., Deep sparse rectifier neural networks, In Gordon, G. J. & Dunson, D. B., (eds.), *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, vol. 15. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011, 315–323. [Online]. Available: http://www.jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf.
83. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., & Weinberger, K. Q., (eds.), *Advances in Neural Information Processing Systems 27*, Curran Associates, Inc., 2672–2680. [Online]. Available: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.
84. Goodman, D. & Brette, R. (2008). Brian: a simulator for spiking neural networks in python.

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

85. Grenet, I., Yin, Y., Comet, J.-P., & Gelenbe, E. (2018). Machine learning to predict toxicity of compounds. *27th Annual International Conference on Artificial Neural Networks, ICANN18, accepted for publication*. Springer Verlang.
86. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.
87. Heeger, D. (2000). Poisson model of spike generation. *Handout, University of Standford* 5: 1–13.
88. Hinton, G.E. & Salakhutdinov, R.R. (2006). Reducing the dimensionality of data with neural networks. *Science* 313(5786): 504–507.
89. Hinton, G.E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation* 18(7): 1527–1554.
90. Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks* 4(2): 251–257.
91. Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks* 2(5): 359–366.
92. Hoyer, P.O. (2002). Non-negative sparse coding. *Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on*. IEEE, 557–565.
93. https://en.wikipedia.org/wiki/go_(game).
94. Huang, G.-B., Zhu, Q.-Y., & Siew, C.-K. (2006). Extreme learning machine: theory and applications. *Neurocomputing* 70(1): 489–501.
95. Hussain, K.F. & Moussa, G.S. (2005). Laser intensity vehicle classification system based on random neural network. *Proceedings of the 43rd annual Southeast regional conference-Volume 1*. ACM, 31–35.
96. Javed, A., Larijani, H., Ahmadinia, A., & Emmanuel, R. (2017). Random neural network learning heuristics. *Probability in the Engineering and Informational Sciences* 31(4): 436–456.
97. Javed, A., Larijani, H., Ahmadinia, A., & Gibson, D. (2017). Smart random neural network controller for hvac using cloud computing technology. *IEEE Transactions on Industrial Informatics* 13(1): 351–360.
98. Jo, S., Yin, J., & Mao, Z.-H. (2005). Random neural networks with state-dependent firing neurons. *IEEE Transactions on Neural Networks* 16(4): 980–983.
99. Kadioglu, Y.M. & Gelenbe, E. (2018). Product form solution for cascade networks with intermittent energy. *IEEE Systems Journal*. doi: 10.1109/JSYST.2018.2854838.
100. Kasun, L.L.C., Zhou, H., & Huang, G.-B. (2013). Representational learning with extreme learning machine for big data. *IEEE Intelligent Systems* 28(6): 31–34.
101. Kim, H. & Gelenbe, E. (2012). Stochastic gene expression modeling with hill function for switch-like gene responses. *IEEE/ACM Trans. Comput. Biology Bioinform.* 9(4): 973–979. [Online]. Available: https://doi.org/10.1109/TCBB.2011.153.
102. Knight, J., Voelker, A.R., Mundy, A., Eliasmith, C., & Furber, S. (2016). Efficient spinnaker simulation of a heteroassociative memory using the neural engineering framework. *Neural Networks (IJCNN), 2016 International Joint Conference on*. IEEE, 5210–5217.
103. Kocak, T., Seeber, J., & Terzioglu, H. (2003). Design and implementation of a random neural network routing engine. *IEEE Transactions on Neural Networks* 14(5): 1128–1143.
104. Krizhevsky, A. & Hinton, G. (2009). Learning multiple layers of features from tiny images.
105. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11): 2278–2324.
106. LeCun, Y., Huang, F.J., & Bottou, L. (2004). Learning methods for generic object recognition with invariance to pose and lighting. *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*. Vol. 2. IEEE, II–97–104.
107. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature* 521(7553): 436–444.
108. Lee, D.D. & Seung, H.S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature* 401(6755): 788–791.
109. Leshno, M., Lin, V.Y., Pinkus, A., & Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks* 6(6): 861–867.
110. Lichman, M. (2013). UCI machine learning repository. [Online]. Available: http://archive.ics.uci.edu/ml.
111. Likas, A. & Stafylopatis, A. (2000). Training the random neural network using quasi-newton methods. *European Journal of Operational Research* 126(2): 331–339. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221799004828.
112. Liu, X., Yan, S., & Jin, H. (2010). Projective nonnegative graph embedding. *Image Processing, IEEE Transactions on* 19(5): 1126–1137.

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

113. Lu, R. & Shen, Y. (2006). Image segmentation based on random neural network model and gabor filters. *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the*. IEEE, 6464–6467.
114. Makhzani, A., Shlens, J., Jaitly, N., & Goodfellow, I.J. (2015). Adversarial autoencoders. *CoRR*, abs/1511.05644, 1–16. [Online]. Available: http://arxiv.org/abs/1511.05644.
115. McCulloch, W.S. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics* 5(4): 115–133. [Online]. Available: https://doi.org/10.1007/BF02478259.
116. Metropolis, N. & Ulam, S. (1949). The monte carlo method. *Journal of the American Statistical Association* 44(247): 335–341.
117. Öke, G. & Loukas, G. (2007). A denial of service detector based on maximum likelihood detection and the random neural network. *The Computer Journal* 50(6): 717–727. [Online]. Available: http://dx.doi.org/10.1093/comjnl/bxm066.
118. Park, J. & Sandberg, I.W. (1991). Universal approximation using radial-basis-function networks. *Neural Computation* 3(2): 246–257.
119. Park, J. & Sandberg, I.W. (1993). Approximation and radial-basis-function networks. *Neural Computation* 5(2): 305–316.
120. Paudel, I., Pokhrel, J., Wehbi, B., Cavalli, A., & Jouaber, B. (Sept 2014). Estimation of video qoe from mac parameters in wireless network: A random neural network approach. *2014 14th International Symposium on Communications and Information Technologies (ISCIT)*, 51–55.
121. Pavlus, J. (2015). The search for a new machine. *Scientific American* 312(5): 58–63.
122. Phan, H.T.T., Sternberg, M.J.E., & Gelenbe, E. (2012). Aligning protein-protein interaction networks using random neural networks. *2012 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2012, Philadelphia, PA, USA, October 4–7, 2012*, 1–6. [Online]. Available: https://doi.org/10.1109/BIBM.2012.6392664.
123. Preissl, R., Wong, T.M., Datta, P., Flickner, M., Singh, R., Esser, S.K., Risk, W.P., Simon, H.D., & Modha, D.S. (2012). Compass: a scalable simulator for an architecture for cognitive computing. *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 54.
124. Qin, Z., Yu, F., Shi, Z., & Wang, Y. (2006). Adaptive inertia weight particle swarm optimization. *International conference on Artificial Intelligence and Soft Computing*. Springer, 450–459.
125. Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 1–16. [Online]. Available: http://arxiv.org/abs/1511.06434.
126. Radhakrishnan, K. & Larijani, H. (2011). Evaluating perceived voice quality on packet networks using different random neural network architectures. *Performance Evaluation* 68(4): 347–360, g-Networks and their Applications. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166531611000101.
127. Riedmiller, M. & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: the rprop algorithm. *IEEE International Conference on Neural Networks*. Vol. 1, 586–591.
128. Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review* 65(6), 386.
129. Rosenblatt, F. (1961). Principles of neurodynamics. perceptrons and the theory of brain mechanisms, DTIC Document, Tech. Rep.
130. Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1985). Learning internal representations by error propagation, California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep.
131. Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). Parallel distributed processing: Explorations in the microstructure of cognition. In Rumelhart, D. E., McClelland, J. L., & C. PDP Research Group, (eds.), ch. Learning Internal Representations by Error Propagation. Vol. 1, Cambridge, MA, USA: MIT Press, 318–362. [Online]. Available: http://dl.acm.org/citation.cfm?id=104279.104293.
132. Sakellari, G. & Gelenbe, E. (2010). Demonstrating cognitive packet network resilience to worm attacks. *17th ACM conference on Computer and Communications Security, Proceedings of the*. ACM, 636–638.
133. Scarselli, F. & Tsoi, A.C. (1998). Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural Networks* 11(1): 15–37.
134. Serrano, W. & Gelenbe, E. (2018). The random neural network in a neurocomputing application for web search. *Neurocomputing* 280: 123–134.
135. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. *et al.* (2016). Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587): 484–489.

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

136. Springenberg, J.T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net, *arXiv preprint arXiv:1412.6806*.
137. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15(1), 1929–1958.
138. Stinchcombe, M. & White, H. (1989). Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. *International 1989 Joint Conference on Neural Networks*. Vol. 1, 613–617.
139. Storn, R. & Price, K. (1997). Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization* 11(4): 341–359. [Online]. Available: https://doi.org/10.1023/A:1008202821328.
140. Tang, J., Deng, C., & Huang, G.-B. (2016). Extreme learning machine for multilayer perceptron. *IEEE Transactions on Neural Networks and Learning Systems* 27(4): 809–821.
141. Teke, A. & Atalay, V. (2006). Texture classification and retrieval using the random neural network model. *Computational Management Science* 3(3): 193–205.
142. Timotheou, S. (2008). Nonnegative least squares learning for the random neural network. *International Conference on Artificial Neural Networks*. Springer, 195–204.
143. Timotheou, S. (2009). A novel weight initialization method for the random neural network. *Neurocomputing* 73(1–3): 160–168.
144. Timotheou, S. (2010). The random neural network: a survey. *The Computer Journal* 53(3): 251–267.
145. Vlontzos, A. (May 2017). The rnn-elm classifier. *2017 International Joint Conference on Neural Networks (IJCNN)*, 2702–2707.
146. Wachsmuth, E., Oram, M., & Perrett, D. (1994). Recognition of objects and their component parts: responses of single units in the temporal cortex of the macaque. *Cerebral Cortex* 4(5): 509–522.
147. Wang, L. & Gelenbe, E. (2016). Real-time traffic over the cognitive packet network, 3–21.
148. Wang, L. & Gelenbe, E. (2018). Adaptive dispatching of tasks in the cloud. *IEEE Transactions on Cloud Computing* 6(1): 33–45.
149. Wang, Y.-X. & Zhang, Y.-J. (2013). Nonnegative matrix factorization: a comprehensive review. *IEEE Transactions on Knowledge and Data Engineering* 25(6): 1336–1353.
150. Wen, W., Wu, C., Wang, Y., Nixon, K., Wu, Q., Barnell, M., Li, H., & Chen, Y. (2016). A new learning method for inference accuracy, core occupation, and performance co-optimization on truenorth chip. *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*. IEEE, 1–6.
151. Wikipedia contributors, "Perceptron," 2018, [Online; accessed 25-Sep-2018]. [Online]. Available: https://en.wikipedia.org/wiki/Perceptron.
152. Wilson, D.R. & Martinez, T.R. (1996). Heterogeneous radial basis function networks. *Proceedings of the International Conference on Neural networks (ICNN 96)*, 1263–1267.
153. Yin, Y. (2018). Random neural networks for deep learning, *Imperial College London, PhD Thesis, available in* https://san.ee.ic.ac.uk/publications/PhDThesis_Yonghua_Yin_v31.pdf.
154. Yin, Y. & Gelenbe, E. (2016). Deep learning in multi-layer architectures of dense nuclei, *arXiv preprint arXiv:1609.07160*.
155. Yin, Y. & Gelenbe, E. (Sept. 2016). Nonnegative autoencoder with simplified random neural network, *ArXiv e-prints*.
156. Yin, Y. & Gelenbe, E. (May 2017). Single-cell based random neural network for deep learning. *2017 International Joint Conference on Neural Networks (IJCNN)*, 86–93.
157. Yin, Y. & Gelenbe, E. (2018). A classifier based on spiking random neural network function approximator, *Preprint available in ResearchGate.net*.
158. Yin, Y. & Zhang, Y. (2012). Weights and structure determination of chebyshev-polynomial neural networks for pattern classification. *Software* 11, 048.
159. Yin, Y., Wang, L., & Gelenbe, E. (May 2017). Multi-layer neural networks for quality of service oriented server-state classification in cloud servers. *2017 International Joint Conference on Neural Networks (IJCNN)*, 1623–1627.
160. Yunong, Z., Kene, L., & Ning, T. (2009). An rbf neural network classifier with centers, variances and weights directly determined. *Computing Technology and Automation* 3, 002.
161. Zeiler, M.D. (2012). ADADELTA: an adaptive learning rate method. *CoRR* abs/1212.5701, 1–6. [Online]. Available: http://arxiv.org/abs/1212.5701.
162. Zhang, Y., Yin, Y., Yu, X., Guo, D., & Xiao, L. (2012). Pruning-included weights and structure determination of 2-input neuronet using chebyshev polynomials of class 1. *Intelligent Control and Automation (WCICA), 2012 10th World Congress on*. IEEE, 700–705.

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press

163. Zhang, Y., Yin, Y., Guo, D., Yu, X., & Xiao, L. (2014). Cross-validation based weights and structure determination of chebyshev-polynomial neural networks for pattern classification. *Pattern Recognition* 47(10): 3414–3428.
164. Zhang, Y., Yu, X., Guo, D., Yin, Y., & Zhang, Z. (2014). Weights and structure determination of multiple-input feed-forward neural network activated by chebyshev polynomials of class 2 via cross-validation. *Neural Computing and Applications* 25(7–8): 1761–1770.
165. Zhu, X. (2005). Semi-supervised learning literature survey.

https://doi.org/10.1017/S026996481800058X Published online by Cambridge University Press