



# Deep Learning with Dense Random Neural Networks for Detecting Attacks Against IoT-Connected Home Environments

Olivier Brun<sup>1,2</sup>, Yonghua Yin<sup>1</sup>, Erol Gelenbe<sup>1(✉)</sup>, Y. Murat Kadioglu<sup>1</sup>,  
Javier Augusto-Gonzalez<sup>3</sup>, and Manuel Ramos<sup>3</sup>

<sup>1</sup> Imperial College, London SW7 2AZ, UK

[e.gelenbe@imperial.ac.uk](mailto:e.gelenbe@imperial.ac.uk)

<sup>2</sup> LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

<sup>3</sup> TELEVES, 17 Rua B. de Conxo, 15706 Santiago de Compostela, Spain

**Abstract.** In this paper, we analyze the network attacks that can be launched against IoT gateways, identify the relevant metrics to detect them, and explain how they can be computed from packet captures. We also present the principles and design of a deep learning-based approach using dense random neural networks (RNN) for the online detection of network attacks. Empirical validation results on packet captures in which attacks were inserted show that the Dense RNN correctly detects attacks.

**Keywords:** Cybersecurity · IoT · Dense random neural network

## 1 Introduction

With the proliferation of network attacks aiming at accessing sensitive information without authorisation, or at rendering computer systems unreliable or unusable, cybersecurity has become one of the most vibrant of today research areas. Whereas most work has been done in the context of traditional TCP/IP networks, IoT systems have specific vulnerabilities which need to be addressed. In this paper, we analyze the cybersecurity threats against an IoT-connected home environment and present the principles and design of a learning-based approach for detecting network attacks.

The paper is organized as follows. In Sect. 2, we analyze the vulnerabilities of IoT gateways and identify the relevant metrics for detecting some of the attacks against them. In Sect. 3, we present the experiment performed in order to obtain some initial packet captures and explain how the previous metrics can be extracted from them. Section 4 is devoted to the description of the learning algorithm, whereas Sect. 5 presents empirical validation results.

## 2 Network Attacks

In an IoT-connected home environment, there may be dozens or even hundreds of sensors with various functions, e.g., measuring temperature, light, noise, etc. These environments usually also include some actuators for controlling systems such as the heating, ventilation, and air conditioning system. Each of these devices may use different protocols to connect (Wi-Fi, Bluetooth, Ethernet, Zig-Bee and others) and most of them are not able to connect directly to the Internet. A crucial component is then the IoT gateway, which is a device capable of aggregating and processing sensor data before sending it to Internet servers.

IoT gateways sit at the intersection of edge devices (sensors and actuators) and the Internet, and are therefore vulnerable to both traditional IP attacks and to attacks against wireless sensor networks. In this section, we focus on the security of IoT gateways and consider both types of attacks. As there is a myriad of different computer and network attack methods, we focus on some of the most common and most damaging ones: Denial-of-Service attacks for TCP/IP networks, and Denial-of-Sleep attacks for wireless sensor networks.

### 2.1 Denial-of-Service Attacks

A denial-of-service attack (DoS attack) is typically accomplished by flooding the targeted machine or resource with superfluous requests in an attempt to overload systems and prevent some or all legitimate requests from being fulfilled. In a distributed denial-of-service attack (DDoS attack), the incoming traffic flooding the victim originates from many different sources, making it impossible to stop the attack simply by blocking a single source.

Some DoS attacks aim at remotely stopping a service on the victim host. The basic method for remotely stopping a service is to send a malformed packet. Below, are two standard examples of this type of attacks:

- **Ping-of-death attack:** the attacker tries to send an over-sized ping packet to the destination with the hope to bring down the destination system due to the system’s lack of ability to handle huge ping packets.
- **Jolt2 attack:** the attacker sends a stream of packet fragments, none of which have a fragment offset of zero. The target host exhausts its processor capacity in trying to rebuild these bogus fragments.

Other well known examples of this type of attacks include *Land attacks*, *Latierra attacks* and *Rose attacks*, but there are many more.

Another form of DoS attack aims at remotely exhausting the resources of the victim host. This form of attacks involves flooding the remote victim with a huge number of packets. Below are some well-known examples:

- **TCP SYN attacks:** This type of attacks exploits a flaw in some implementations of the TCP three-way handshake. When an host receives the SYN request from another host, it must keep track of the partially opened connections in a “listening queue” for a given number of seconds. The attacker

exploits the small size of the listen queue by sending multiple SYN requests to the victim, never replying to the sent back SYN-ACK. The victim's listening queue is quickly filled up, and it stops accepting new connections.

- **UDP flood:** The attacker sends a large number of UDP packets to random ports on a remote host. The victims checks for the application listening on this port. After seeing that no application listens on the port, it replies with an ICMP “Destination Unreachable” packet. In this way, the victimized system is forced to send many ICMP packets, eventually leading it to be unreachable by other clients, or even to go down.

There are of course many other forms of flooding attacks, including *ICMP floods* and *HTTP POST DoS attacks*, and many more.

## 2.2 Denial-of-Sleep Attacks

In the context of the Internet of Things, low-rate wireless personal area networks are a prevalent solution for communication among devices. As discussed in [2], tight limitations on hardware cost, memory use and power consumption have given rise to a number of security vulnerabilities, including traffic eavesdropping, packet replay, and collision attacks, straightforward to conduct<sup>1</sup>. A simple form of attack is to deplete the energy available to operate the wireless sensor nodes [4, 6, 7]. For instance, vampire attacks are routing-layer resource exhaustion attacks aiming at draining the whole life (energy) from network nodes, hence their name [12]. In this section, we shall focus on another form of energy attacks, which are MAC-layer attacks known as Denial-of-Sleep attacks. Below are some examples of denial-of-sleep attacks:

- **Sleep Deprivation Attack:** the ability of sensor nodes to enter a low power sleep mode is very useful for extending network longevity. The attacker launches a sleep deprivation attack by interacting with the victim in a manner that appears to be legitimate; however, the purpose of the interactions is to keep the victim node out of its power conserving sleep mode, thereby dramatically reducing its lifetime [5, 10, 11].
- **Barrage Attack:** As in the sleep deprivation attack, the attacker seeks to keep the victim out of its sleep mode by sending seemingly legitimate requests. However, the requests are sent at a much higher rate and aim at making the victim performs energy intensive operations. Barrage attacks are more easily detected than sleep deprivation attacks, which are carried out solely through the use of seemingly innocent interactions.
- **Broadcast Attack:** malicious nodes can broadcast unauthenticated traffic and long messages which must be received by other nodes before being possibly discarded for lack of authentication [1]. Such attacks are hard to detect since they have no effect on system throughput, and nodes that receive them waste energy.

---

<sup>1</sup> For instance, these attacks can be conducted with KillerBee, a python-based framework for attacking ZigBee and other 802.15.4 networks.

Other forms of denial-of-sleep attacks include *Synchronization attacks* [9], *Replay attacks* [3], and *Collision attacks* [8].

### 2.3 Relevant Metrics to Detect Attacks

Table 1 presents the relevant metrics for detecting the attacks described above.

**Table 1.** Selected attacks and relevant metrics to detect them.

Attack	Metric
UDP flood	Number of destination UDP ports per second
	Number of outgoing ICMP “destination unreachable” packets
TCP SYN	Difference between the numbers of initiated and established connexions
Sleep deprivation attack	Number of data packets over a long time scale
Barrage attack	Number of data packets over a short time scale
Broadcast attack	Number of broadcast messages

## 3 Metering Cybersecurity Metrics from Packet Captures

In this section, we explain how the metrics identified in Sect. 2 can be obtained from packet capture files. We first present the experiment performed in order to obtain some initial packet captures. We then briefly describe Scapy, a python package for packet manipulation, which was used to analyse the packet capture files and extract the time-series of various metrics.

### 3.1 Packet Captures

A standard installation of the Carelife system was done to obtain packet captures. Several software modules were installed on the Televes gateway in order to capture and parse (in a PCAP format) the data packets exchanged with various sensors which were previously paired and registered by the gateway. The gateway was also connected to the Internet using a 3G SIM card. Packets were captured for a complete weekend on all the network interfaces of the gateway (see Fig. 1).

The packet captured during the experiment were analyzed using Scapy. Scapy is a packet manipulation tool for computer networks, written in Python by Philippe Biondi. It can forge or decode packets, send them on the wire, capture them, and match requests and replies. It can also handle tasks like scanning, tracerouting, probing, unit tests, attacks, and network discovery.

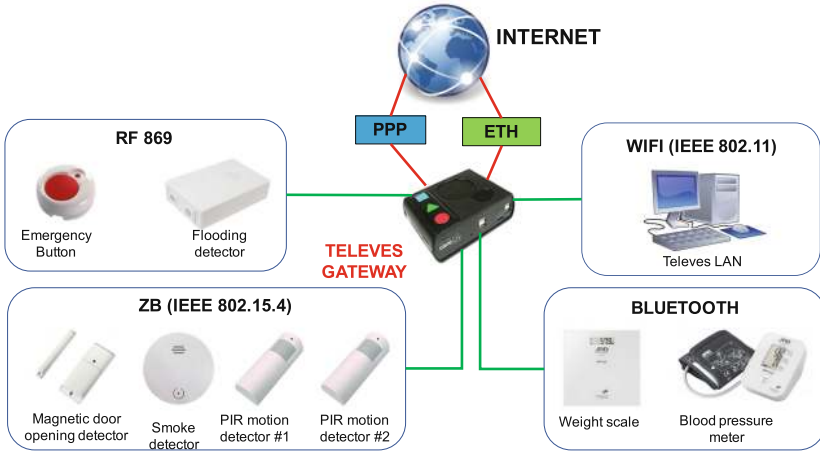


Fig. 1. Configuration used for the experiment.

### 3.2 Analysis of Packet Captures

In the following, we shall illustrate how some metrics are extracted from packet capture files. Instead of providing results for all network interfaces, we shall focus on the PPP interface, but we emphasize that the metering of metrics is similar for the other technologies. Scapy was used to extract some general packet-level characteristics from pcap files, in order to have a better understanding of what is “normal traffic”, but also for computing the time-series associated to the various cybersecurity metrics.

**General Packet-Level Characteristics.** In total, 100,653 frames were captured during the experiment on the PPP interface. The total number of IP packets received by the gateway is 50,296, whereas it sent in total 41,938 packets. Analysis reveals that packets were sent to 158 distinct destination IP addresses, and that packets were received from 2,375 distinct origin IP addresses. Figure 2 shows the locations of packet origins and destinations. 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.

We have analysed several other packet-level characteristics of the traffic, distinguishing between the traffic sent by the gateway, and the traffic it receives, as well as between different protocols (ICMP, DNS, UDP, TCP):

- **Packet-size distributions:** The analysis of the packets emitted (resp. received) by the gateway reveals that there are 56 (resp. 130) different sizes. Small IP packets are the most frequent ones. Figure 3a shows the packet-size distribution of the incoming traffic.
- **Inter-arrival and inter-departure times:** Figure 3b shows the inter-arrival distribution of incoming packets at the gateway. The inter-departure

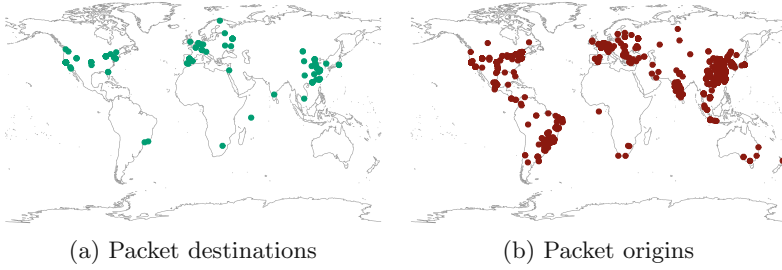


Fig. 2. Locations of packet origins and destinations.

time distribution of packets from the gateway is similar, but UDP and ICMP outgoing traffics follow an almost periodic pattern (one packet every 5 min).

- **Packet throughput:** Although there are some peaks, the non-TCP traffic exchanged with the gateway is usually in the order of a few hundred of Bytes per second. The throughput of TCP traffic is slightly higher, but does not exceed a few kilobytes per second.

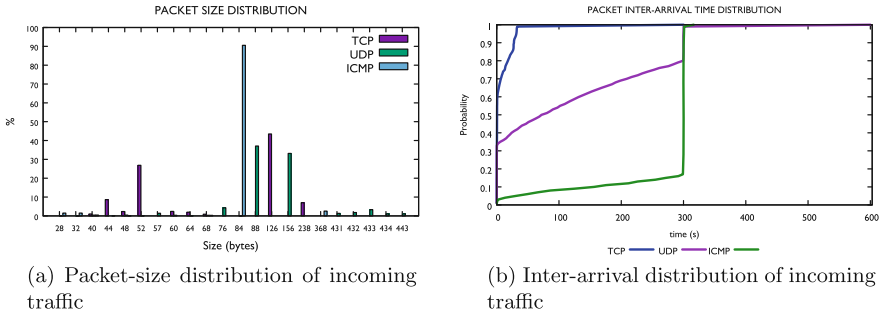
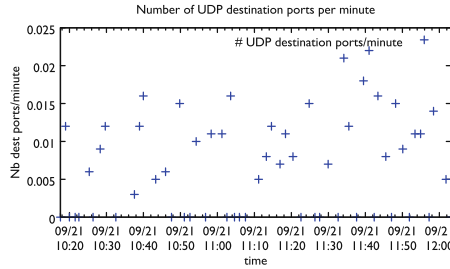


Fig. 3. Packet-size distributions.

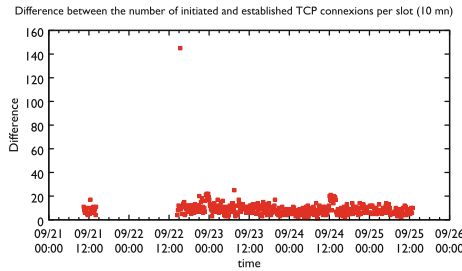
### Cybersecurity Metrics

- **UDP flood attacks:** A UDP flood can be detected by looking either at the number of UDP destination ports or at the number of ICMP “Destination Unreachable” packet sent by the victim, or preferably at both metrics. Under an UDP flood, the “Destination Unreachable” messages sent by the victim will usually have their error code set to the value 3, to indicate a port unreachable error. Figure 4 shows the number of UDP destination ports opened per minute during the first 80 min of the experiment. Overall, the average number of UDP destination port opened on the gateway is 0.011, and the standard deviation is 0.065.

- **SYN flood attacks:** a SYN flood can be detected by looking at the difference between the numbers of initiated and established TCP connexions. A TCP connexion is initiated when the first SYN packet is received. It is established when the 3-way handshake is completed, that is, when the destination has sent a SYN-ACK packet which was acknowledged by the source with a ACK packet. Figure 5 shows the difference between the numbers of initiated and established TCP connexions per 10-min time slot. Except for a peak around 15:21 on 09/22/2017, this difference is usually quite low. Its average value is 4 and its standard deviation is 8.43.



**Fig. 4.** Number of UDP destination ports opened per minute (first 80 min).



**Fig. 5.** Difference between the numbers of initiated and established TCP connexions per time slot (10 min)

## 4 Network-Attack Detection with Random Neural Network

This section describes the use of random neural networks (RNN) [17,18] developed for deep learning recently [13–16] to detect network attacks, which can be viewed as a binary classification problem. First, we show how to construct training datasets from captured packets. Then, the dense RNN is presented to learn given datasets so as to conduct classification.

## 4.1 Dataset Construction

Starting with the captured packets, statistical data (e.g., the number of UDP ports opened per time slot) in time series can be obtained as explained in Sect. 3. We extract samples from the time-series statistical data by setting a sliding window with length  $l$ . If a sample  $X_n \in R^{l \times 1}$  is extracted in the non-attack case, then we assign the label of this sample denoted as  $y_n$  as 0; otherwise, if it is extracted in the attack case, the label of this sample is assigned as  $y_n = 1$ . Then, we have a dataset  $\{(X_n, y_n) | n = 1, \dots, N\}$ , where the input is the statistical data extracted from captured packet data and the output is a binary value.

## 4.2 Dense Random Neural Network for Deep Learning

A dense cluster in a dense RNN [13, 14, 16] is composed of  $n$  statistically identical cells. Each cell receives inhibitory spike trains from external cells with rate  $x$ , whose spike behaviours follow the pattern of random selection of soma-to-soma interactions. Let  $q$  denote the probability of the activation state of a cell in the cluster in the steady state. Previous work shows that a numerical solution can be obtained for  $q$  such that

$$q = \zeta(x) = \frac{-(c - nx) - \sqrt{(c - nx)^2 - 4p(n - 1)(\lambda^- + x)d}}{2p(n - 1)(\lambda^- + x)},$$

with  $d = n\lambda^+$  and  $c = \lambda^+p + rp - \lambda^-n - r - \lambda^+pn - npr$ , where  $p$  is the repeated-firing probability when a cell fires,  $r$  is the firing rate of a cell, and a cell receives excitatory and inhibitory spikes from external world with rates  $\lambda^+$  and  $\lambda^-$  respectively. For notation ease,  $\zeta(\cdot)$  is used as a term-by-term function for vectors and matrices.

Dense RNN in multi-layer architectures (DenseRNN) are constructed in the following manner.

The first layer (input layer) of the DenseRNN is made up of RNN cells that receives excitatory spike trains from external sources, resulting in a quasi-linear cell activation  $q(x) = \min(x, 1)$  [19]. The successive  $L$  layers are hidden layers composed of dense clusters that receive inhibitory spike trains from cells in the previous layer, with a resultant activation function  $q(x) = \zeta(x)$ . The last layer is an RNN-ELM. Let us denote the connecting weight matrices between layers of a  $L$ -hidden-layer ( $L \geq 2$ ) DenseRNN by  $W_1, \dots, W_L \geq 0$  and output weight matrix by  $W_{L+1}$ . Given input matrix  $X$ , a forward pass of  $X$  in the DenseRNN can be described as:

$$\begin{cases} Q_1 = \min(X, 1), \\ Q_l = \zeta(Q_{l-1}W_{l-1}) \text{ for } l = 2, \dots, L + 1, \\ O = Q_{L+1}W_{L+1}. \end{cases}$$

where  $Q_1$  is the 1st layer output,  $Q_l$  is the  $l$ th layer output ( $l = 2, \dots, L + 1$ ) and  $O$  is the final DenseRNN output.

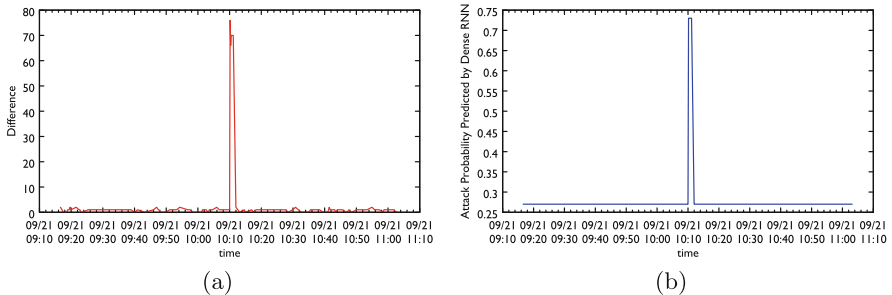


Given a training dataset  $\{(X_n, y_n) | n = 1, \dots, N\}$ , the works in [13, 14, 16] have developed an efficient training procedure for DenseRNN to determine the values of  $W_1, \dots, W_L, W_{L+1}$ , which combines unsupervised and supervised learning techniques.

## 5 Experimental Results

In this section, we present the empirical results obtained for the detection of TCP SYN attacks. Using Scapy, we wrote a Python script for generating such attacks. The resulting pcap files can be used to train the learning algorithm, in addition to the “normal traffic” captured during the experiment described in Sect. 3. Moreover, using the utility tool *mergcap*, it is possible to insert a SYN attack into the files containing the packets captured during the experiment, thereby allowing to test the learning algorithm.

As an example, Fig. 6a plots the time-series for the difference between the numbers of initiated and established TCP connections per time slot (10s) which was extracted from a pcap file obtained using the above procedure. As can be observed in Fig. 6b, the Dense RNN models correctly predicts that there was an attack.



**Fig. 6.** Scenario where a SYN attack was inserted into the normal traffic captured from 9:15 AM to 11:03 AM on Sep. 21st, 2017: (a) time-series of the difference between the numbers of initiated and established TCP connexions per time slot (10s), and (b) attack probability predicted by the Dense RNN.

## 6 Conclusion

In this paper, we presented a methodology for the online detection of network attacks against IoT gateways. The methodology, which is based on a deep-learning approach with dense random neural networks, can predict the probability that a network attack is ongoing from a set of metrics extracted from packet captures. As future work, we intend to apply our methodology to a broad range of network attacks, including Denial-of-Sleep attacks against ZigBee and Bluetooth-connected devices, and to investigate the design of a one-class classification algorithm for network attack detection.

**Acknowledgements.** This work was partially funded by the European Union's Horizon 2020 Research and Innovation Programme through the GHOST project (<https://www.ghost-iot.eu/>) under Grant Agreement No. 740923.

## References

1. Brownfield, M., Gupta, Y., Davis, N.: Wireless sensor network denial of sleep attack. In: Proceedings of the 2005 IEEE Workshop on Information Assurance and Security. United States Military Academy, West Point, NY (2005)
2. Dalrymple, S.D.: Comparison of ZigBee replay attacks using a universal software radio peripheral and USB radio. Master's thesis, AFIT, USAF (2014)
3. Di Mauro, A., Fafoutis, X., Mödersheim, S., Dragoni, N.: Detecting and preventing beacon replay attacks in receiver-initiated MAC protocols for energy efficient WSNs. In: Riis Nielson, H., Gollmann, D. (eds.) NordSec 2013. LNCS, vol. 8208, pp. 1–16. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-41488-6\\_1](https://doi.org/10.1007/978-3-642-41488-6_1)
4. Dubey, A., Jain, V., Kumar, A.: A survey in energy drain attacks and their countermeasures in wireless sensor networks. *Int. J. Eng. Res. Technol.* **3**(2), 1206–1210 (2014)
5. Falk, R., Hof, H.-J.: Fighting insomnia: a secure wake-up scheme for wireless sensor networks. In: 3rd International Conference on Emerging Security Information, Systems and Technologies, IEEE SECURWARE 2009, pp. 191–196 (2009)
6. Francois, F., Abdelrahman, O.H., Gelenbe, E.: Impact of signaling storms on energy consumption and latency of LTE user equipment. In: 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, pp. 1248–1255, August 2015
7. Gelenbe, E., Murat Kadioglu, Y.: Energy life-time of wireless nodes with and without energy harvesting under network attacks. In: IEEE International Conference on Communications (ICC), Kansas City, MO, USA, 20–24 May 2018
8. Law, Y.-W., Palaniswami, M., Van Hoesel, L., Doumen, J., Hartel, P., Havinga, P.: Energy-efficient link-layer jamming attacks against wireless sensor network MAC protocols. *ACM Trans. Sen. Netw.* **5**(1), 6:1–6:38 (2009)
9. Lu, X., Spear, M., Levitt, K., Matloff, N.S., Wu, S.F.: A synchronization attack and defense in energy-efficient listen-sleep-slotted MAC protocols. In: 2008 2nd International Conference on Emerging Security Information, Systems and Technologies (2008)
10. Pirretti, M., Zhu, S., Vijaykrishnan, N., McDaniel, P., Kandemir, M., Brooks, R.: The sleep deprivation attack in sensor networks: analysis and methods of defense. *Int. J. Distrib. Sens. Netw.* **2**(3), 267–287 (2006)
11. Stajano, F., Anderson, R.: The resurrecting duckling: security issues for ad-hoc wireless networks. In: Christianson, B., Crispo, B., Malcolm, J.A., Roe, M. (eds.) Security Protocols 1999. LNCS, vol. 1796, pp. 172–182. Springer, Heidelberg (2000). [https://doi.org/10.1007/10720107\\_24](https://doi.org/10.1007/10720107_24)
12. Vasserman, E.Y., Hopper, N.: Vampire attacks: draining life from wireless ad hoc sensor networks. *IEEE Trans. Mob. Comput.* **12**(2), 318–332 (2013)
13. Gelenbe, E., Yin, Y.: Deep learning with dense random neural networks. In: Gruca, A., Czachórski, T., Harezlak, K., Kozielski, S., Piotrowska, A. (eds.) ICMMI 2017. AISC, vol. 659, pp. 3–18. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-67792-7\\_1](https://doi.org/10.1007/978-3-319-67792-7_1)

14. Yin, Y., Gelenbe, E.: Deep learning in multi-layer architectures of dense nuclei. arXiv preprint [arXiv:1609.07160](https://arxiv.org/abs/1609.07160) (2016)
15. Yin, Y., Gelenbe, E.: Single-cell based random neural network for deep learning. In: 2017 International Joint Conference on Neural Networks (IJCNN), pp. 86–93. IEEE (2017)
16. Gelenbe, E., Yin, Y.: Deep learning with random neural networks. In: Bi, Y., Kapoor, S., Bhatia, R. (eds.) IntelliSys 2016. LNNS, vol. 16, pp. 450–462. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-56991-8\\_34](https://doi.org/10.1007/978-3-319-56991-8_34)
17. Gelenbe, E.: Learning in the recurrent random neural network. *Neural Comput.* **5**(1), 154–164 (1993)
18. Gelenbe, E.: Random neural networks with negative and positive signals and product form solution. *Neural Comput.* **1**(4), 502–510 (1989)
19. Yin, Y., Gelenbe, E.: Nonnegative autoencoder with simplified random neural network. CoRR, abs/1609.08151 (2016). <http://arxiv.org/abs/1609.08151>

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

