Machine Learning for Wireless Networks with Artificial Intelligence: A Tutorial on Neural Networks

Article · October 2017

5 authors, including:

Mingzhe Chen
Beijing University of Posts and Telecommunications
29 PUBLICATIONS   **397** CITATIONS

SEE PROFILE

Ursula Challita
The University of Edinburgh
21 PUBLICATIONS   **215** CITATIONS

SEE PROFILE

Walid Saad
Virginia Polytechnic Institute and State University
442 PUBLICATIONS   **9,232** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Internet of Things: Wireless Communications, Learning, and Security View project

Beyond 5G Networks: Context-Awareness, mmW communications, and Caching View project

Machine Learning for Wireless Networks with Artificial Intelligence: A Tutorial on Neural Networks

Mingzhe Chen[*], Ursula Challita[†], Walid Saad[‡], Changchuan Yin[*], and Mérouane Debbah [§]

[*]Beijing Laboratory of Advanced Information Network, Beijing University of Posts and Telecommunications, Beijing, China 100876, Email: chenmingzhe@bupt.edu.cn and ccyin@bupt.edu.cn.

[†] School of Informatics, The University of Edinburgh, Edinburgh, UK. Email: ursula.challita@ed.ac.uk.

[‡]Wireless@VT, Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA, USA, Email: walids@vt.edu.

[§]Mathematical and Algorithmic Sciences Lab, Huawei France R & D, Paris, France, Email: merouane.debbah@huawei.com.

Abstract

Next-generation wireless networks must support ultra-reliable, low-latency communication and intelligently manage a massive number of Internet of Things (IoT) devices in real-time, within a highly dynamic environment. This need for stringent communication quality-of-service (QoS) requirements as well as mobile edge and core intelligence can only be realized by integrating fundamental notions of artificial intelligence (AI) and machine learning across the wireless infrastructure and end-user devices. In this context, this paper provides a comprehensive tutorial that introduces the main concepts of machine learning, in general, and artificial neural networks (ANNs), in particular, and their potential applications in wireless communications. For this purpose, we present a comprehensive overview on a number of key types of neural networks that include feed-forward, recurrent, spiking, and deep neural networks. For each type of neural network, we present the basic architecture and training procedure, as well as the associated challenges and opportunities. Then, we provide an in-depth overview on the variety of wireless communication problems that can be addressed using ANNs, ranging from communication using unmanned aerial vehicles to virtual reality and edge caching.For each individual application, we present the main motivation for using ANNs along with the associated challenges while also providing a detailed example for a use case scenario and outlining future works that can be addressed using ANNs. In a nutshell, this article constitutes one of the first holistic tutorials on the development of machine learning techniques tailored to the needs of future wireless networks.

I. Introduction

The wireless networking landscape is undergoing a major revolution. The smartphone-centric networks of yesteryears are gradually morphing into a massive *Internet of Things (IoT)* ecosystem [1]–[4] that integrates a heterogeneous mix of wireless-enabled devices ranging from smartphones, to drones, connected vehicles, wearables, sensors, and virtual reality apparatus. This unprecedented transformation will not only drive an exponential growth in wireless traffic in the foreseeable future, but it will also lead to the emergence of new and untested wireless service use cases, that substantially differ from conventional multimedia or voice-based services. For instance, beyond the need for high data rates – which has been the main driver of the wireless network evolution in the past decade – next-generation wireless networks must be able to deliver ultra-reliable, low-latency communication [5]–[8], that is adaptive, in real-time to a rich and dynamic IoT environment. For example, drones and connected vehicles [9]–[12] will place autonomy at the heart of the IoT. This, in turn, will necessitate the deployment of ultra-reliable wireless links that can provide real-time, low-latency control for such autonomous systems. Meanwhile, in tomorrow's wireless networks, large volumes of data must be collected, periodically and in real-time, across a massive number of sensing and wearable devices. Such massive short-packet transmissions will lead to a substantial traffic over the wireless uplink, which has traditionally been much less congested than the downlink. This same wireless network must also support cloud-based gaming [13], immersive virtual reality services, real-time HD streaming, and conventional multimedia services. This ultimately creates a radically different networking environment whose novel applications and their diverse quality-of-service (QoS) and reliability requirements mandate a fundamental change in the way in which wireless networks are modeled, analyzed, designed, and optimized.

The need to cope with this ongoing and rapid evolution of wireless services has led to much research that investigates what the optimal cellular network architecture will be within the context of the emerging fifth generation (5G) wireless networks (e.g., see [14] and references therein). While the main ingredients for 5G – such as dense small cell deployments, millimeter wave (mmWave) communications, and device-to-device (D2D) communications – have been identified, integrating them into a truly harmonious wireless system that can meet the IoT challenges requires instilling *intelligent functions across both the edge and core of the network.*

These intelligent functions must be able to adaptively exploit the wireless system resources and generated data, in order to optimize network operation and guarantee, in real-time, the QoS needs of emerging wireless and IoT services. Such mobile edge and core intelligence can only be realized by integrating fundamental notions of *artificial intelligence (AI)* [15] across the wireless infrastructure and end-user devices.

While the notion of artificial intelligence can be traced back to the mythological Greek bronze man Talos [16] – an artificially intelligent man-machine created to protect the island of Crete from invaders, its true potential has only been recently realized owing to the substantial developments in *machine learning* [17], in general, and *neural networks* [18], in particular. Indeed, machine learning tools are undoubtedly the primary vessel that is carrying artificial intelligence to fruition across a myriad of applications [19]–[26] that range from computer vision to natural language processing, robotics, and autonomous systems. As such, creating AI-enabled wireless networks is contingent upon developing the right set of machine learning and neural network tools that can provide the plethora of AI functions needed in future wireless networks to unleash the true potential of the IoT. Such tools must naturally be tailored to the unique features of the wireless environment, which is evidently quite different from the traditional applications of AI, such as in robotics and computer vision [27]–[30].

For instance, artificial intelligence is expected to play several roles in the next-generation of wireless networks [31]. First, the most natural application of AI and machine learning is to exploit *big data analytics* to enhance situational awareness and overall network operation. In this context, AI will provide the wireless network with the ability to parse through massive amounts of data, generated from multiple sources that range from wireless channel measurements and sensor readings to drones and surveillance images, in order to create a comprehensive operational map of the massive number of devices within the network. This map can, in turn, be exploited to optimize various functions, such as fault monitoring and user tracking, across the wireless network.

Second, beyond its powerful prediction and data analytics functions, AI will be a major driver of intelligent and data-driven *wireless network optimization*. For instance, machine learning tools will enable the introduction of intelligent resource management tools, that can be used to address a variety of problems ranging from cell association and radio access technology selection to frequency allocation, spectrum management, power control, and intelligent beamforming. In

contrast to conventional distributed optimization technique, that are often done iteratively in an offline or semi-offline manner, AI-guided resource management mechanisms will be able to operate in a fully online manner by learning, in real time, the states of the wireless environment and the network's users. Such mechanisms will therefore be able to continuously improve their own performance over time which, in turn, will enable more intelligent and dynamic network decision making. Such intelligent decision making is essential for much of the envisioned IoT and 5G services, particularly those that require real-time, low latency operation, such as autonomous driving, drone guidance, and industrial control. In fact, if properly designed, machine learning-based AI optimization algorithms will provide inherently *self-organizing, self-healing, and self-optimizing* solutions for a broad range of problems within the context of network optimization and resource management. Such AI-driven self-organizing solutions are particularly apropos for ultra dense wireless networks in which classical centralized and distributed optimization approaches can no longer cope with the scale and heterogeneity of the network.

Third, beyond its system-level functions, AI can play a key role at the *physical layer* of a wireless network. As shown in recent works [32]–[37], neural networks and machine learning tools can be used to redefine the way in which physical layer functions, such as coding and modulation, are designed and operated, at both transmitter and receiver levels, within a generic communication system. Such an AI-driven approach has been shown [32]–[37] to have a lot of promise in delivering lower bit error rates and better robustness to the wireless channel impediments.

Last, but not least, the rapid deployment of highly *user-centric wireless services*, such as virtual reality [38]–[40], in which the gap between the end-user and the network functions is almost minimal, strongly motivate the need for wireless networks that can track and adapt to their human user behavior. In this regard, machine learning is perhaps the only tool that is capable to learn and mimic human behavior, which will help in creating the wireless network to adapt its functions to its human users, thus creating a truly immersive environment and maximizing the overall quality-of-experience (QoE) of the users.

Clearly, AI-based system operation is no longer a privilege, but rather a necessity for future wireless networks. AI-driven wireless network designs will pave the way towards an unimaginably rich set of new network functions and wireless services. For instance, even though 5G networks may not be fully AI capable, we envision that the subsequent, sixth generation (6G)

of wireless cellular networks will be almost completely reliant on AI and machine learning, as evidenced by recent developments of AI-enabled mobile networks proposed by Huawei [41] and "big innovation house" proposed by Qualcomm [42]. *As such, the question is no longer if machine learning tools are going to be integrated into wireless networks but rather when such an integration will happen.* In fact, the importance of an AI-enabled wireless network has already been motivated by a number of recent wireless networking paradigms such as mobile edge caching, context-aware networking, big data analytics, location-based services, and mobile edge computing [43]–[50], the majority of which rely on machine learning or AI-inspired tools. However, despite their importance, these works remain rather focused on the wireless challenges, rather than on the synergistic integration of wireless and AI. In consequence, these works are of narrow focus and do not provide any broad, tutorial-like material that can shed light on the challenges and opportunities associated with the use of machine learning for designing wireless networks with AI.

Meanwhile, a number of surveys on machine learning and neural network applications in wireless networking have emerged, such as in [2], [4], [32], and [51]–[64]. Despite being interesting, these surveys are limited in a number of ways. First, they are often focused on one type of machine learning techniques (often deep learning such as in [32], [51], and [54]) and, as such, they do not capture the rich spectrum of available of AI and machine learning frameworks. Second, they mostly restrict their scope to a single wireless application such as sensor networks [55]–[57], cognitive radio networks [53], machine-to-machine (M2M) communication [4], or physical layer design [32] and, hence, they do not cover the broad range of applications that can adopt machine learning in future networks. Third, a large number of the existing surveys, such as [2], [4], and [52]–[57], are highly qualitative and do not provide an in-depth technical and quantitative description on the variety of existing machine learning tools that are suitable for wireless communications. Last, but not least, a number of surveys [58]–[62] discuss the basics of neural networks with applications outside of wireless communications. However, these surveys are largely inaccessible to the wireless community, due to their reliance on examples from rather orthogonal disciplines such as computer vision. Moreover, most of the existing tutorials or surveys [62]–[64] do not provide concrete guidelines on how, when, and where to use different machine learning tools in the context of wireless networks. Finally, the introductory literature on machine learning for wireless networks such as in [2], [4], [32], and

[51]–[64], is largely sparse and fragmented, hence, making it difficult to understand the intrinsic details of this broad and far reaching area. In fact, there is a clear lack of a comprehensive and holistic tutorial that can explain, pedagogically and, in detail, how to develop machine learning solutions to endow wireless networks with artificial intelligence and realize the full potential of IoT systems, and beyond.

The main contribution of this paper is, thus, to provide one of the first holistic tutorials on the topic of machine learning for wireless network design. The overarching goal is to gather the state-of-the-art and emerging research contributions, from machine learning and wireless communications, that address the major opportunities and challenges in developing machine learning frameworks for understanding and designing wireless systems with artificial intelligence. In particular, we will provide a comprehensive treatment of *artificial neural networks*, which are one of the most important pillars of machine learning, with an emphasis on both new analytical techniques from neural networks and novel wireless application scenarios. After providing a substantial introduction to the basics of machine learning using neural networks, we introduce a classification of the various types of neural networks. For each type, we provide an introduction on their basic components, their training processes, and their use cases with specific example neural networks. Then, we overview a broad range of wireless applications that can make use of neural network designs. These applications include drone-based communications, spectrum management with multiple radio access technologies, wireless virtual reality, mobile edge caching and computing, and the IoT, among others. For each application, we first outline the main rationale for applying machine learning while pinpointing illustrative scenarios. Then, we overview the challenges and opportunities brought forward by the use of neural networks in the specific wireless application. We complement this overview with a detailed example drawn from the state-of-the-art and, then, we conclude by shedding light on the potential future works within each specific area. Overall, we anticipate that this tutorial will provide an invaluable reference on the use of neural networks in wireless communications and, as a result, will help in driving further research in this area.

The rest of this paper is organized as follows. In Section II, we introduce the basic of machine learning and artificial neural networks. Section III presents several key types of artificial neural networks such as feed-forward neural networks, recurrent neural networks, spiking neural networks, and deep neural networks. In Section IV, we discuss the use of artificial neural networks

for wireless communication and corresponding challenges and opportunities. Finally, conclusions are drawn in Section V.

II. ARTIFICIAL NEURAL NETWORKS: PRELIMINARIES

In this section, we first provide a brief overview on the basics of machine learning, while motivating the importance of neural networks. Then, we expose the fundamentals of a suite of neural network algorithms and techniques.

A. Machine Learning: A Brief Overview

Machine learning was born from pattern recognition and it is essentially based on the premise that machines should be endowed with artificial intelligence that enables them to learn from previous computations and adapt to their environment through experience [19]–[26]. Due to growing volumes of generated data – across critical infrastructure, communication networks, and smart cities – and the need for intelligent data analytics, the use of machine learning algorithms has become ubiquitous [65] across many sectors such as financial services, government, health care, technology, marketing, and entertainment. Using machine learning algorithms to build models that uncover connections and predict dynamic system or human behavior, system operators can make intelligent decisions without human intervention. For example, using machine learning enables a system to grasp the entire knowledge of social relationships between individuals and can recognize individuals' speech, face, and writing. Similarly, in a wireless system such as the IoT, machine learning tools can be used for big data analytics and edge intelligence. Machine learning tasks often depend on the nature of their training data. In machine learning, *training* is the process that teaches the machining learning framework to achieve a specific goal, such as for speech recognition. In other words, training enables the machine learning framework to discover potentially relationships between the input data and output data of this machining learning framework. There exists, in general, four key classes of learning approaches [66]: a) supervised learning, b) unsupervised learning, c) semi-supervised learning, and d) reinforcement learning.

Supervised learning algorithms are trained using labeled data. When dealing with labeled data, both the input data and its desired output data are known to the system. Supervised learning is commonly used in applications that have enough historical data. In contrast, training

of unsupervised learning tasks is done without labeled data. The goal of unsupervised learning is to explore the data and infer some structure directly from the unlabeled data. Semi-supervised learning is used for the same applications as supervised learning but it uses both labeled and unlabeled data for training. This type of learning can be used with methods such as classification, regression and prediction. Semi-supervised learning is useful when the cost of a fully labeled training process is relatively high. In contrast to the previously discussed learning methods that need to be trained with historical data, reinforcement learning (RL) is trained by the data from implementation. The goal of RL is to learn an environment and find the best strategies for a given agent, in different environments. RL algorithms are used for robotics, gaming, and navigation [67]. To perform these learning tasks, several frameworks have been developed. Among those frameworks, artificial neural networks (ANNs) [60] constitute one of the most important pillars of machine learning, as they are able to mimic human intelligence, to model complex relationships between inputs and outputs, to find patterns in data, or to extract the statistical structure in an unknown joint probability distribution from the observed data.

ANNs are inspired by the structure and functional aspects of biological neural networks, that can learn from observational complicated or imprecise data. ANNs process information in a manner that is analogous to the human brain. A given ANN is composed of a large number of highly interconnected processing elements working in parallel to solve a specific problem. ANNs can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. An ANN can create its own organisation or representation of the information it receives during learning time. Moreover, an ANN can be used in a self-organizing manner to learn how to perform tasks based on the data given for training or initial experience. Within the context of wireless communications, as will be clearer from the latter sections, ANNs can be used to investigate and predict network and user behavior so as to provide users' information for solving diverse wireless networking problems such as cell association, spectrum management, computational resource allocation, and cached content replacement. Moreover, recent developments of smart devices and mobile applications have significantly increased the level at which human users interact with mobile systems. A trained ANN can be thought of as an "expert" in dealing with human-related data. Therefore, using ANNs to exact information from the user environment can provide a wireless network with the ability to predict the users' future behaviors and, hence, design an optimal strategy to improve
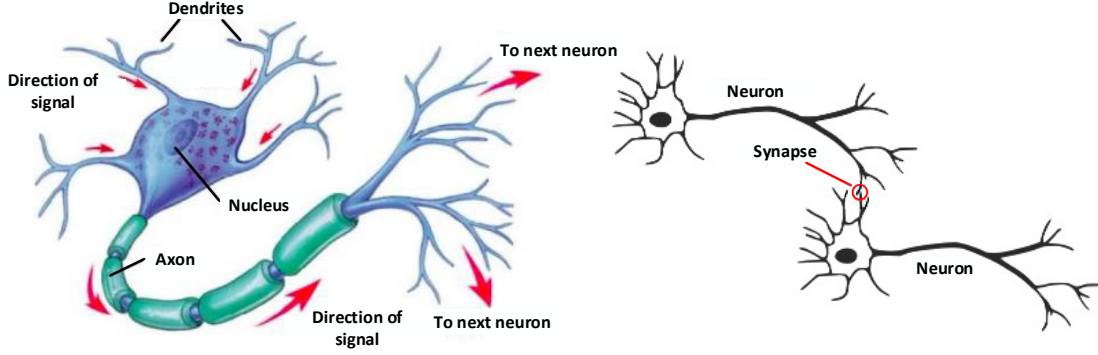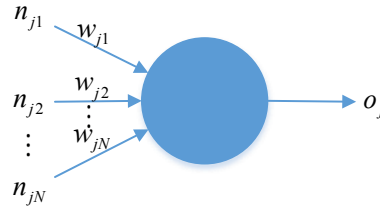
Fig. 1. A neuron model as shown in [68].

Fig. 2. A mathematical neuron model.

the QoS and reliability. Next, we introduce the basic concepts an general architecture of ANNs.

B. General Architecture of Artificial Neural Networks

The architecture of ANNs consists of a number of simple, highly interconnected processing elements known as *neurons*, which are used to mimic how the human brain learns. ANNs are essentially an artificial model of a human nervous system whose base elements are also neurons used to process information in the sense of cognition and transmit this information signal in the nervous system [69]. A neuron consists of nucleus, dendrites, and axons [69]. Neurons are connected to each other by dendrites and axons as shown in Fig. 1. The connection point between two neurons is known as a synapse. The information signal transmitting to a neuron will change its membrane potential. If this change makes the neuron's membrane potential exceed a certain value, the neuron will send a signal to all of its connected neurons. This is how signals propagate through the human nervous system. ANNs use artificial neurons to replicate this operation of the human nervous system, thus enabling artificial intelligence. Mathematically, an artificial neuron consists of the following components: (a) A number of incoming connections, analogous to synapses on the dendrites; (b) A number of outcoming connections, analogous to synapses on the axon; and (c) An activation value assigned to each neuron, analogous to a biological neuron's membrane potential. To capture the strengths of the synapses for biological

neurons, each connection will have a connection strength. The connection strength between two neurons is captured by a weight value. The basic model for a neuron j is shown in Fig. 2 and mathematically given by:

$$o_j \left(\boldsymbol{w}_j, b_j, \boldsymbol{n}_j \right) = f \left(b_j + \sum_{i=k}^{N} n_{jk} \cdot w_{jk} \right), \tag{1}$$

where n_{jk} is the input signal from neuron j to neuron i, $\boldsymbol{n}_j = [n_{j1}, n_{j2}, \ldots, n_{jN}]$ is a vector of the input signal of neuron j, w_{jk} is the corresponding input weight value, $\boldsymbol{w}_j = [w_{j1}, w_{j2}, \ldots, w_{jN}]$ is a vector of input weight of neuron j, o_j is the output signal of neuron j, b_j is the bias of neuron j, and $f(\cdot)$ is a nonlinear activation function. A bias value can shift the activation function, which is critical for successful learning. The activation function in a neural network will represent the rate of action potential firing in the cell of a neuron. The simplest form of an activation function [70] is binary such as a Heaviside step function, which is used to indicate whether a neuron is firing or not. However, using linear activation functions, many neurons must be used in the computation beyond linear separation. Meanwhile, an ANN constructed using linear activation functions in (1) cannot reach a stable state after training since the value of the activation function will increase without bound. To avoid this drawback, one can choose, $f(\cdot)$ in (1) to be a normalizable activation function such as a sigmoid activation function rather than a linear activation function. The selection of a type of activation functions in ANNs depends on the sought objectives such as analytic tractability, computational power, and the type of the desired output signal (logistic or continuous). For example, if $f(\cdot)$ in (1) is a linear function, we will have $o_j \left(\boldsymbol{w}_j, b_j, \boldsymbol{n}_j \right) = b_j + \sum_{i=k}^{N} n_{jk} \cdot w_{jk}$, which can be used to analyze the performance of ANNs due to its analytical tractability. In essence, an ANN is a composition of multiple neurons connected in different ways and operating using different activation functions.

In general, the main components of an ANN that consists of multiple neurons will include the following:

- *Input layer* that consists of a number of neurons used to represent the input signal which will be transmitted in the neurons.
- *Output layer* that consists of a number of neurons used to represent the output signal.
- *Hidden layer* that consists of a number of neurons used to mimic the human brain.
- *Input weight matrix* that represents the strengths of the connections between the neurons in the input layer and the neurons in the hidden layer.

- *Neuron weight matrix* that represents the strengths of the connections between the neurons in the hidden layer.

- *Output weight matrix* that represents the strengths of the connections between the neurons in the output layer and the neurons in the hidden layer.

The connection strength in all weight matrices can be used to calculate the value of the activation function as per (1). For example, if $\boldsymbol{n}_j = [n_{j1}, n_{j2}, \ldots, n_{jN}]$ in (1) is an input signal vector, then $\boldsymbol{w}_j = [w_{j1}, w_{j2}, \ldots, w_{jN}]$ will represent the value of the input weight, and, thus, the value of the activation function can be calculated by (1). The hidden layer is used to analyze the relationship between the input signal in the input layer and the output signal in the output layer. We can view the neurons in the hidden layer as a black box that can be used to find the relationship between the input signal and the output signal without having any specifics on their activations in the hidden layer. This is why this layer is known as a "hidden" layer. Having defined the basic components of an ANN, next, we use an ANN example to explain specifically how these components can be used to form a given ANN.

One of the simplest forms of an artificial neural network is the *feed-forward neural network (FNN)* [71], as shown in Fig. 3. An FNN consists of the following components: (a) Input layer, (b) Hidden layer(s), and (c) Output layer. In this FNN architecture, the strengths of the connections between the neurons in the input layer and the hidden layer are captured by the input weight matrix. The strengths of the connections between the neurons in the hidden layer are captured by the neuron-by-neuron weight matrix and the strengths of the connections between the neurons in the output layer and hidden layer are included in the output weight matrix. In an FNN, the connection between the neurons is unidirectional and there is no connection between the neurons in a layer. Each neuron in the hidden layer calculates its output using an activation function such as the function in (1). Moreover, each neuron in the hidden layer has incoming connections only from the previous layer and outgoing connections only to the next layer, and, hence, this architecture is named *feed-forward neural network*. For this connection method, the output of each neuron propagates from the input layer through the hidden layers to the output layer.

In fact, an ANN model is typically determined by the connection method, activation function, and layers. For instance, if each neuron in the hidden layer has incoming connections only from the previous layer and outgoing connections only to the next layer, the network is an
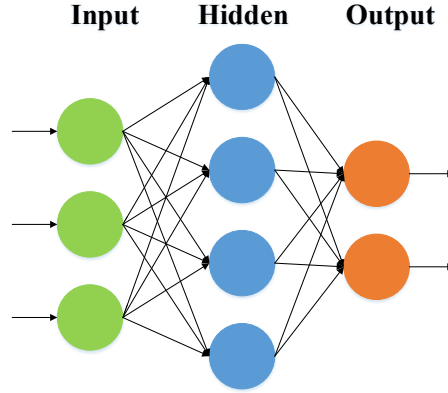
Fig. 3.   Feed-forward neural network.

FNN. If the connections between neurons form a loop, the network is called a recurrent neural networks (RNNs) [72]. The network is called deep neural network [73], if there is more than one hidden layer in any given ANN model such as FNNs or RNNs. Having introduced the general architecture of an ANN, next, we discuss the training methods that ANNs can use to perform their learning tasks.

C. Training in Artificial Neural Networks

To learn information from their input data, ANNs must adjust the weights of the connections between the neurons in the system. The process of adjusting and updating the weights is known as the *training process*. Different learning tasks require different training algorithms. For example, to perform supervised learning tasks such as mobility prediction, ANNs can be trained using labeled data. For unsupervised learning tasks such as clustering data into different groups, ANNs can be trained without the use of labeled data. In other words, for the unsupervised case, the desired output for each input is not known. Next, we mainly explain the general training algorithms for supervised learning tasks, as they constitute the foundation for other types.

For supervised learning tasks, the objective of training ANNs is to minmize the errors between the desired output signal and actual output signal. This error can be typically defined as:

$$E\left(\boldsymbol{W}, \boldsymbol{b}\right) = 0.5 \cdot \sum \left(\|\boldsymbol{y}\left(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{x}\right) - \boldsymbol{y}_D\|\right), \tag{2}$$

where $\boldsymbol{x}$ is a vector of input signals, $\boldsymbol{W}$ is the weight matrix that is a combination of the input weight matrix, hidden weight matrix, and output weight matrix, $\boldsymbol{b}$ is a vector of bias factor, and $\boldsymbol{y}_D$ is the desired output. $\boldsymbol{y}\left(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{x}\right)$ is the actual output signal for each neuron, which can be

calculated based on (1). In (1), the error is scaled by $\frac{1}{2}$ to facilitate differentiation. In general, the most commonly used supervised learning algorithms for ANNs include gradient descent and backpropagation [74], which is a special case of gradient descent. Hence, we will now overview these learning algorithms as they constitute a building block for any other learning algorithm.

In order to minimize $E\left(\boldsymbol{W}, \boldsymbol{b}\right)$, we need to update the weights related to each neuron. One common approach to do this weight update is via the use of a *gradient descent algorithm*, as explained next. For a given neuron j, the error between the desired output signal $o_{D,j}$ and actual output signal o_j will be given by $E_j\left(\boldsymbol{w}_j, b_j\right) = 0.5 \cdot \sum \left(\|o_j\left(\boldsymbol{w}_j, b_j, \boldsymbol{n}_j\right) - o_{D,j}\|\right)$. The gradient descent algorithm is used to minmize $E_j\left(\boldsymbol{w}_j, b_j\right)$. For every element w_{jk} of vector $\boldsymbol{w}_j$, the minimization of $E_j\left(\boldsymbol{w}_j, b_j\right)$ using gradient descent algorithms follows from the following equations:

$$w_{jk,\text{new}} = w_{jk,\text{old}} - \gamma \frac{\partial E_j\left(\boldsymbol{w}_j, b_j\right)}{\partial w_{jk}}, \tag{3}$$

$$b_{j,\text{new}} = b_{j,\text{old}} - \gamma \frac{\partial E_j\left(\boldsymbol{w}_j, b_j\right)}{\partial b_j}, \tag{4}$$

where γ is the learning rate. The first order derivative $\frac{\partial E_j(\boldsymbol{w}_j,b_j)}{\partial w_{jk}}$, will enable us to determine whether the error $E_j\left(\boldsymbol{w}_j, b_j\right)$ is decreasing or increasing when the weight value is w_{jk}. Based on (3) and (4), ANNs can update the weight matrix and bias to find the optimal $\boldsymbol{w}_j$ and b_j that will minimize $E_j\left(\boldsymbol{w}_j, b_j\right)$. From (3) and (4), we can see that, the update of w_{jk} and b_j is easy to compute and, hence, the gradient descent algorithm is known to be computationally fast, even on large datasets [75]. The gradient descent algorithm mentioned above only focuses on the update of a single neuron. However, in an ANN, the signal is transmitted from one neuron to another neuron and, hence, we must design a rule to train all of these neurons. Backpropagation is the most widely used algorithm to calculate the gradient of the error such as $\frac{\partial E_j(\boldsymbol{w}_j,b_j)}{\partial w_{jk}}$ in (3) and $\frac{\partial E_j(\boldsymbol{w}_j,b_j)}{\partial b_j}$ in (4), so as to effectively minimize $E\left(\boldsymbol{W}, \boldsymbol{b}\right)$ for an ANN. In fact, backpropagation is a method to compute the gradient of each neuron for an ANN, which is just a chain rule, which is just a chain rule,.

Next, we provide a step-by-step explanation of the backpropagation algorithm. We first assume that neuron j is at layer L and neuron i is at layer $L + 1$, which is closer to the output layer than layer L. The backpropagation procedure that can be used to update the weight value of w_{ij} proceeds as follows:

- An input signal is transmitted from the input layer to the hidden layer of the ANN, until it reaches the output layer. If an ANN does not have hidden layers, (e.g., a perceptron), the input signal will be directly transmitted to the output layer. Then, the activations of the neurons in all of the layers will be computed based on (1).

- The ANN will next compute the error between the targeted output and actual output based on (2) and will derive an error propagation value δ for each neuron. δ_i of neuron i can be given by [76]:

$$\delta_i = \frac{\partial E\left(\boldsymbol{W}_i, \boldsymbol{b}\right)}{\partial o_i} \cdot \frac{\partial o_i}{\partial n_{\text{sum},i}}, \tag{5}$$

where o_i is the output of neuron i and $n_{\text{sum},i} = b_i + \sum_{k=1}^{N} n_{ik} \cdot w_{ik}$ is the summation of the input signal of neuron i and its bias. In particular, if the activation function is a logistic function, $f(x) = \frac{1}{1+e^{-x}}$, then the error propagation of neuron j can be given by [74]:

$$\delta_i = \begin{cases} \left(o_i - o_{D,i}\right) o_i \left(1 - o_i\right), & \text{neuron } i \text{ in the output layer}, \\ \left(\sum_{l \in \mathcal{L}_{L+1}} \delta_l w_{li} \right) o_i \left(1 - o_i\right), & \text{neuron } i \text{ in the hidden layer}, \end{cases} \tag{6}$$

where $\mathcal{L}_{L+1}$ represents the set of neurons at layer $L+1$ (layer $L+1$ is closer to the output layer than layer L). From (6), we can see that error propagation δ_i of a neuron in layer L depends on the error propagation $\delta_l, l \in \mathcal{L}_{L+1}$, of a neuron at layer $L+1$. Therefore, each neuron must transmit its error propagation parameter to the neurons at the former layer. This is the central definition of *backpropagation*.

- Next, the ANN updates the weight value of w_{ij}, which can be given by $w_{ij} = w_{ij} - \gamma \delta_i o_j$.

- The above process is repeated until all weight values reach the minimum of $E\left(\boldsymbol{W}, \boldsymbol{b}\right)$. Note that backpropagation is not guaranteed to find a global minimum as it typically converges to a local optimum since the dataset used for training ANNs is finite and, hence, it must have some blindness in exploration.

In backpropagation, the gradient is computed based on the complete labeled data. However, if the size of the labeled data is very large, then using backpropagation may be time consuming. To reduce the time used for training when the size of the labeled data is very large, *a stochastic gradient descent (SGD)* algorithm [77] can be employed to update the weight values and bias. The stochastic gradient descent algorithm performs a weight value update for each training example. However, the SGD algorithm will often update frequently, which can lead to overshooting – the

weight values are larger or smaller than the optimum. To overcome these drawbacks of SGD, the *mini batch gradient descent* [78] can be used. The mini batch gradient descent is an algorithm that strikes a balance between stochastic gradient descent and batch gradient descent [78]. In mini-batch gradient descent, the gradient is computed based on a small number of samples, e.g., of around 10-500. One benefit of mini-batch gradient descent is that it can be performed in a distributed manner and, hence, it can train ANNs in a time efficient manner.

In summary, gradient descent algorithms enable an ANN to be trained in a computationally simple manner, and hence, they can quickly converge to a local minimum value for the error, even on a large dataset. However, choosing a proper learning rate for the update of the weights and bias can be difficult. In fact, the learning rate determines the step size that the algorithm uses to reach the minimum and, thus, it has an impact on the convergence rate. In particular, a learning rate that is too large can cause the algorithm to diverge from the optimal solution. This is due to the fact that choosing very large initial learning rates will decay the loss function faster thus not allowing the model to explore better the optimization space. On the other hand, a learning rate that is too small will result in a slow speed of convergence. In particular, the optimal value of the initial learning rate is dependent on the dataset under study, whereby for each dataset, there exists an interval of good learning rates at which the performance does not vary much [79]. Moreover, gradient descent algorithms can often converge to a sub-optimal local minimum rather than the global minimum. To solve these challenges, several algorithms have been proposed, such as momentum SGD, RMSProp [78], nesterov accelerated gradient [80], Adagrad [81], and AdaDelta [82]. For instance, Adagrad and Adam are independent of the initial value of the learning rate while RMSProp relies heavily on a good choice of an initial learning rate. Moreover, one can use pruning techniques [83] to minimize the number of neurons in ANNs and make the ANNs become smaller and faster. The basic concept of pruning is that to eliminate ANN neurons that may not contribute to the output as they are not relevant to the learning task.

It is worth noting that two central problems in training ANNs are *overfitting* and *underfitting*. Overfitting corresponds to the case in which the model learns the random fluctuations and noise in the training dataset to the extent that they negatively impact the model's ability to generalize when fed with new data. This occurs mainly when the dataset is too small compared to the number of model parameters that must be learned. On the other hand, underfitting occurs when
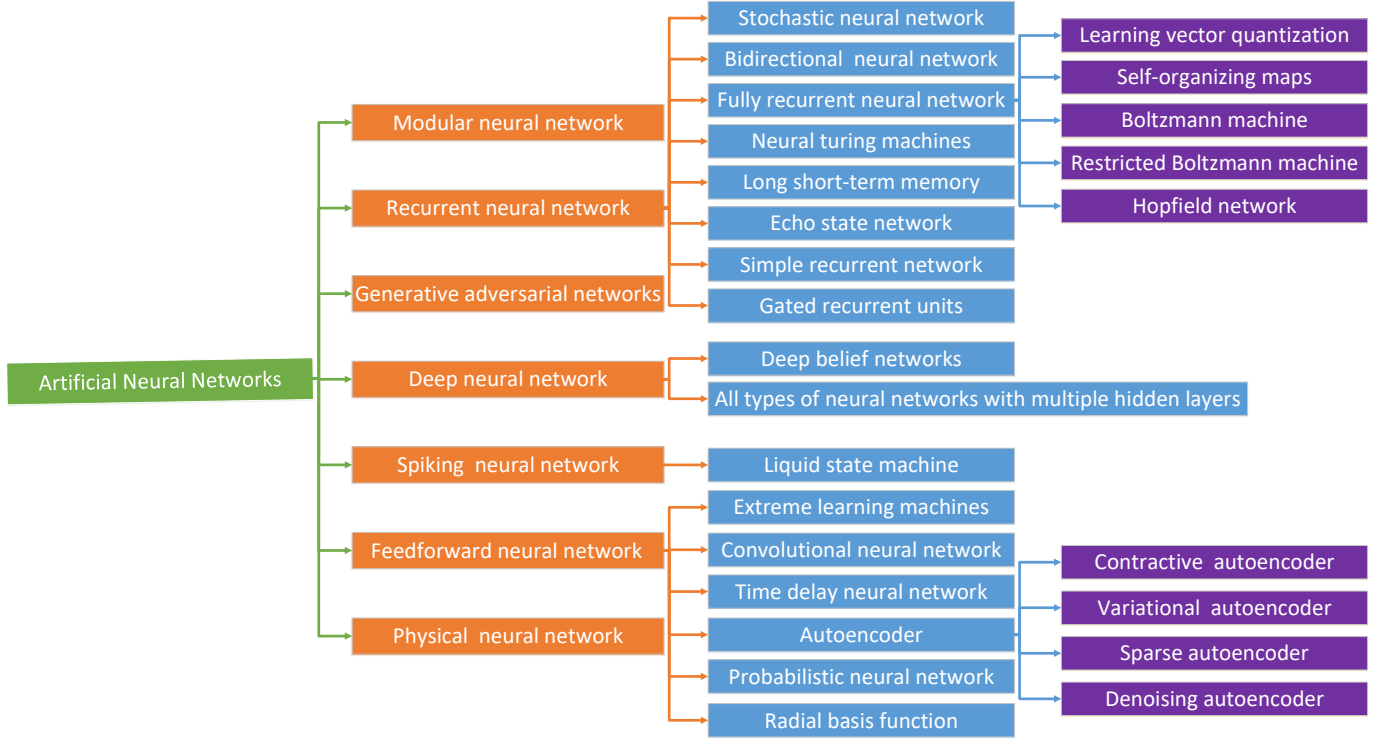
Fig. 4. Summary of a recurrent neural network..

a learning algorithm cannot capture the underlying trend of the data. Intuitively, underfitting occurs when the learning algorithm does not fit the data well enough. Therefore, one must carefully choose the architecture of an ANN along with the proper training methods to avoid overfitting and underfitting.

Using the aforementioned training algorithms, the values of the weight matrix and bias can be updated to their optimal values, and, hence, a trained ANN can output the desired output signal. However, each type of ANNs, as shown in Fig. 4, is suitable for a particular type of data. For instance, RNNs are more convenient for time series data while spiking neural networks are good at modeling continuous data. Therefore, in Section III, we will introduce specific types of ANNs and we discuss their properties and use cases.

III. TYPES OF ARTIFICIAL NEURAL NETWORKS

Beyond the simple FNN architecture introduced earlier, in this section, we specifically discuss three more advanced types of ANNs: recurrent neural networks, spiking neural networks, and deep neural networks. For each kind of neural network, we will present its architecture,
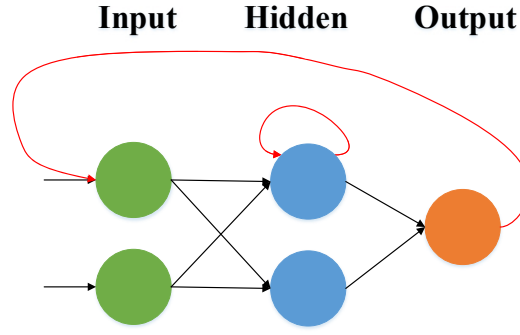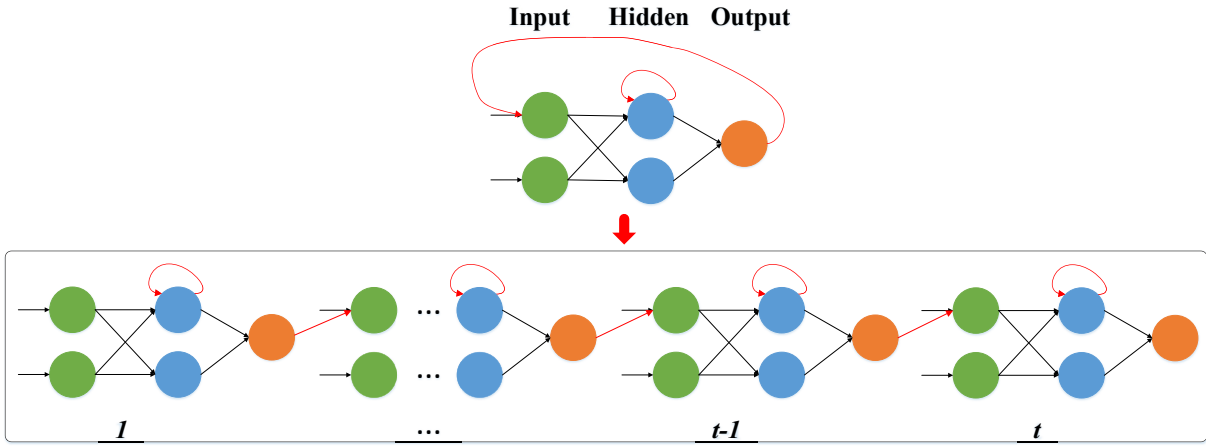
Fig. 5. Recurrent neural network.

Fig. 6. Architecture of an unfolded recurrent neural network.

advantages and properties, training and learning approach, and specific example architectures. We conclude this section with an overview on other types of ANNs that can also be of potential interest for diverse wireless networking applications.

A. Recurrent Neural Networks

1) Architecture of Recurrent Neural Networks: In a traditional ANN, it is assumed that all inputs are independent from each other or all outputs are independent from each other. However, for many tasks, the inputs (outputs) are related. For example, for mobility pattern prediction, the input data that is the users' locations are actually related. To this end, *recurrent neural networks* [84] , which are ANN architectures that allow neuron connections from a neuron in one layer to neurons in previous layers, have been introduced, as shown in Fig. 5. This seemingly simple change enables the output of a neural network to depend, not only on the current input, but also on the historical input, as shown in Fig. 6. This allows RNNs to make use of sequential

information and exploit dynamic temporal behaviors such as those faced in mobility prediction, handwriting recognition, or speech recognition. For example, an RNN can be used to recognize an individual person's speech when they pronounce only one word at each step. Clearly, this task cannot be done in one step without combining different words from different steps. Another application example for RNN is mobility prediction. An RNN can be used to predict the mobility patterns of certain vehicles. These patterns are related to the historical locations that the vehicles have been visited. This task also cannot be done in one step without combing historical locations from previous steps. Therefore, ANNs whose output depends only on the current input, such as FNNs, cannot perform tasks such as speech recognition. RNNs can also be seen as an ANN that have a "memory", which allows RNNs to store historical information. Note that, in theory, RNNs can make use of historical information in arbitrarily long sequences, but in practice they are limited to only a subset of historical information [72]. Given their ability to store historical information, RNNs can more easily perform time-related tasks such as users' mobility pattern predictions compared to traditional ANNs (e.g., FNNs). However, RNNs can require more time to train since each value of the activation function depends on the series data recorded in RNNs.

In terms of architecture, the key components of a given RNN can be specified as follows:

- *Input signal x_t*: this signal represents the input data to a given RNN at time t.
- *Input weight matrix W_{in}*: this matrix represents the strengths of the connections between the neurons in the input layer and the neurons in the hidden layers.
- *Output weight matrix W_{out}*: this matrix is used to represent the strengths of the connections between the neurons in the output layer and the neurons in the hidden layers.
- *Recurrent weight matrix W*: The hidden weight matrix is defined as the recurrent weight matrix, which captures the strengths of the connections between the neurons in the hidden layers of the RNN.
- *Hidden state s_t*: this is effectively the hidden state of a neuron in the hidden layer at time t. The hidden state represents the value of the activation function at time t, which is calculated based on the previous hidden state s_{t-1} and the input at time t. s_t can be computed using different methods for different recurrent neural networks. For most commonly used RNNs, we have $s_t = f(W s_{t-1} + W_{in} x_t)$ where $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ or $f(x) = \max(0, x)$. However, in more elaborate types of RNNs, such as long short-term memory algorithm [85] which will be detailed in Subsection III-C, each neuron needs to decide what to keep in and what

to erase from the hidden state.

- *Output signal*: $\boldsymbol{y}_t$ is the output of a RNN at time t, representing the output signal.

Clearly, we can see that the basic architecture of RNNs is similar to that of FNNs except for the generation of the input, output, and recurrent weight matrices. Moreover, the hidden state in RNNs depends on both current and historical inputs, which enables RNNs to store the historical information. However, when the architecture of an ANN changes from FNNs to RNNs, traditional training methods may not be applicable to RNNs. Hence, next, we introduce training methods suitable for RNNs.

2) Training in Recurrent Neural Networks: In the RNN architecture, the connections between units will form a directed cycle and, hence, the feedforward gradient descent algorithms such as backpropagation cannot be directly used. This is due to the fact that the error backpropagation pertaining to a backpropagation algorithm requires no cycles in the connections between the ANN neurons. In consequence, the *backpropagation through time (BPTT) algorithm* [86] is more commonly used for training RNNs. The BPTT approach unfolds the recurrent network in time, by stacking identical copies of it, and redirecting connections within the network to obtain connections between subsequent copies, as shown in Fig. 6. In consequence, the BPTT algorithm actually transforms an RNN into an FNN, which is amenable for training by a backpropagation algorithm. Since BPTT is similar in operation to the backpropagation algorithm, it is significantly faster for training RNNs than general-purpose optimization algorithms such as evolutionary optimization which some works such as in [87] have used. However, due to the cycle connections in RNNs, BPTT may get more often trapped in numerous sub-optimal local minima compared to the gradient descent algorithms used for training FNNs. Moreover, like backpropagation, the gradient in BPTT is computed based on the complete training set, which may become time consuming if the size of the training set is very large.

To overcome these drawbacks in BPTT training, real-time recurrent learning (RTRL) [88] can be used to compute the exact error gradient at every time step, which is suitable for online learning tasks. In contrast to the BPTT that unfolds RNNs in time, RTRL propagates error forward in time. From (3), we can see that the gradient value with respect to w at time t is $\frac{\partial E(t)}{\partial w}$. In RTRL, the update of weight w depends not only on the gradient value at time t but also on the gradient value at the previous time instant, i.e., $w_{t+1} = w_t - \gamma \sum_{k=0}^{t} \frac{\partial E(k)}{\partial w}$. In RTRL, the gradient of errors propagates forward in time rather than backward in time as in the BPTT

algorithm and, therefore, there is no need to unfold RNNs as needed by the BPTT algorithm. However, the time complexity of RTRL is $O\left(N_w^4\right)$ where N_w is the number of neurons in the considered RNN. In contrast, BPTT has a time complexity of $O\left(N_w^2 G\right)$ where G is length of the input data.

Training the weights in RNNs can also be modeled as a nonlinear global optimization problem. A target function can be formed to evaluate the error of a particular weight vector. In particular, the sum-squared-difference between the predictions and the target values specified in the training sequence is used to represent the error of the current weight vector. Arbitrary global optimization techniques may then be used to minimize the sum-squared-difference. The most common global optimization method for training RNNs is via the use of genetic algorithms [89], especially in unstructured networks. Other global optimization algorithms may be used to seek a good set of weights, such as simulated annealing [90] or particle swarm optimization [91]. Such global optimization algorithms can maximize all weight values at the same time and, hence, can avoid to reach sub-optimal local minimum. However, the complexity of these algorithms for training RNNs can be very high, particularly, when an RNN has a large number of neurons. For instance, when an RNN has 1000 neurons, then a typical global optimization algorithm needs to optimize 1000×1000 weight values at the same time, which is clearly more complex than in the case of BPTT.

In summary, training RNNs can be done using a variety of techniques. Each technique has its own advantages and disadvantages in terms of the particular learning tasks (e.g., supervised vs. unsupervised), training data size, training time, and data storage space. Depending on the scenario and target performance needed, one can select the most suitable RNN training algorithm.

3) Example RNN – Echo State Networks: Next, to shed more light on how RNNs operate, we introduce a type of RNNs that is conceptually simple and easy to implement, called *echo state networks (ESNs)* [92]. Since their inception, ESNs proved to be a highly practical type of RNNs due to their effective approach for training the neural network [93]. In fact, ESNs reinvigorated interest in RNNs [94] by making them accessible to wider audiences due to their apparent simplicity. In ESN, the input weight matrix and hidden weight matrix are randomly generated without any specific training. Therefore, ESN needs to only train the output weight matrix. Moreover, ESNs belong to a class of algorithms in the framework of *reservoir computing* [72]. In general, an ANN algorithm is considered part of the framework of reservoir computing

if its input signals are mapped to a higher dimensional dynamics of the randomly generated hidden layers, known as a *reservoir*, and the dynamics of the reservoir are mapped to the desired output signals by using a simple training method such as backpropagation. The main benefit of reservoir computing is that the neural network training is performed only at the readout stage while the input and hidden weight matrices are fixed. ESNs can, in theory, approximate arbitrary nonlinear dynamical system with arbitrary precision, they have an inherent temporal processing capability, and are therefore a very powerful enhancement of linear blackbox modeling techniques in nonlinear domain. Due to ESN's appealing properties such as training simplicity and ability to record historical information, it has been widely applied for several supervised learning tasks such as audio, time series, mobility, water flow, and content request predictions [95]–[98], reinforcement learning tasks such as navigating through a maze [99], classification, and regression. In wireless networks, ESNs admit many natural applications, such as content prediction, resource management, and mobility pattern estimation, as will be clear from Section IV. Next, the specific architecture and training methods for ESNs are introduced.

- *Architecture of an Echo State Network:* ESNs use an RNN architecture with only one hidden layer [1]. The generation of an ESN can be given as follows:

  - *Generation of a Reservoir:* Generate a large random reservoir that is represented by the tuple $(\boldsymbol{W}_{\text{in}}, \boldsymbol{W}, \alpha)$ where α is known as the leaking rate which can be seen as the speed of the reservoir update dynamics, discretized in time. As we mentioned previously, the dynamical system in reservoir computing is known as a reservoir. In ESN, the input and hidden weight matrixes are jointly known as the reservoir. Setting the leaking rate α must match the speed of the dynamics of hidden state $\boldsymbol{s}_t$ and output $\boldsymbol{y}_t$. Here, $\boldsymbol{W}_{\text{in}}$ and $\boldsymbol{W}$ is generated randomly. In particular, $\boldsymbol{W}$ is a sparse matrix while $\boldsymbol{W}_{\text{in}}$ is a dense matrix. The generation of $\boldsymbol{W}_{\text{in}}$ and $\boldsymbol{W}$ are determined by the training data and other ESN parameters. If one ESN uses discrete bi-valued distribution, i.e., $(-0.5, 0.5)$, to generate $\boldsymbol{W}_{\text{in}}$ and $\boldsymbol{W}$, then the ESN tends to have a slightly less rich signal space (there is a non-zero probability of identical neurons), but might render the analysis of what is happening in the reservoir easier. To allow ESNs to store historical information, the reservoir should satisfy the so-called *echo state property* which means that the hidden state $\boldsymbol{s}_t$ should be uniquely defined

[1]Deep generalizations of ESNs also exist [100]

by the fading history of the input $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_t$. This is in contrast to traditional ANNs such as FNNs that need to adjust the weight values of the neurons in hidden layers, ESNs only need to guarantee the echo state property. To guarantee the echo state property of an ESN, the spectral radius of $\boldsymbol{W}$ should be smaller than 1. However, for some training data, the echo state property will hold even when the spectral radius of $\boldsymbol{W}$ is larger than 1. The scaling of $\boldsymbol{W}_{\text{in}}$ is another key method to optimize an ESN. In order to have a small number of freely adjustable parameters, all elements in $\boldsymbol{W}_{\text{in}}$ are scaled using a single scaling value. If the input signals are of different types and have different contributions to the learning task, one should separately optimize the scalings of $\boldsymbol{W}_{\text{in}}$. For very linear tasks, $\boldsymbol{W}_{\text{in}}$ should be small, letting the neruons operate around 0. For large $\boldsymbol{W}_{\text{in}}$, the neurons will get easily close to their -1 and 1 values acting in a more nonlinear, binary switching manner.

- *ESN Implementation:* Run ESN using the training input $\boldsymbol{x}_t$ and collect the corresponding hidden states $\boldsymbol{s}_t$. The input $\boldsymbol{x}_t$ can be a vector or a scalar, which depends on the training data. Normalization of input data $\boldsymbol{x}_t$ can keep the input $\boldsymbol{x}_t$ bounded and avoid the hidden weight matrix being infinity. In general, the input data from the beginning of the training will be discarded and not used for training $\boldsymbol{W}_{\text{out}}$ since it may introduce an unnatural starting state which is not normally visited once the network has learnt the task. The amount of input data to discard depends on the memory of the ESN.

- *Training Output weight matrix:* Compute the output weight matrix $\boldsymbol{W}_{\text{out}}$ from the reservoir using a training algorithm such as gradient descent or ridge regression (explained next) to minimize the mean square error (MSE) between the targeted output and action output.

- *Generate Output:* Use the trained network on new input data $\boldsymbol{x}$ computing $\boldsymbol{y}_t$ by employing the trained output weights $\boldsymbol{W}_{\text{out}}$.

Given the components of ESNs, we will describe the activation value of each neuron. Even though the input and hidden weight matrices are fixed (randomly), all neurons of an ESN will have their own activation values (hidden state). As opposed to the classical RNNs in which the hidden state depends only on the current input, in ESNs, the hidden state will be given by:

$$\tilde{\boldsymbol{s}}_t = f\left(\boldsymbol{W}[1; \boldsymbol{s}_{t-1}] + \boldsymbol{W}_{\text{in}}\boldsymbol{x}_t\right), \tag{7}$$

$$\boldsymbol{s}_t = (1 - \alpha) + \alpha\tilde{\boldsymbol{s}}_t, \tag{8}$$

where $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and $[\cdot; \cdot]$ represents a vertical vector (or matrix) concatenation. The model is

also sometimes used without the leaky integration, which is a special case for $\alpha = 1$ which yields $\tilde{s}_t = s_t$. From (7), we can see that the scaling of $\boldsymbol{W}_{\text{in}}$ and $\boldsymbol{W}$ determines the proportion of how much the current state $\boldsymbol{s}_t$ depends on the current input $\boldsymbol{x}_t$ and how much on the previous state $\boldsymbol{s}_{t-1}$. Here, a feedback connection from $\boldsymbol{y}_{t-1}$ to $\boldsymbol{s}_t$ can be applied to the ESNs, which is defined as a weight matrix $\boldsymbol{W}_{\text{fb}}$. Hence, (7) can be rewritten as $\tilde{s}_t = f\big(\boldsymbol{W}[1; \boldsymbol{s}_{t-1}] + \boldsymbol{W}_{\text{in}}\boldsymbol{x}_t + \boldsymbol{W}_{\text{fb}}\boldsymbol{y}_{t-1}\big)$.

Based on the hidden state $\boldsymbol{s}_t$, the output signal of ESN can be given by:

$$y_t = \boldsymbol{W}_{\text{out}}\left[1; \boldsymbol{s}_t; \boldsymbol{x}_t\right]. \tag{9}$$

Here, an additional nonlinearity can be applied to (9), i.e., $\boldsymbol{y}_t = \tanh\left(\boldsymbol{W}_{\text{out}}\left[1; \boldsymbol{s}_t; \boldsymbol{x}_t\right]\right)$.

• *Training in Echo State Networks*: The objective of the training process in ESNs is to minimize the MSE between the targeted output and actual output. When this MSE is minimized, the actual output will be the target output which can be given by $\boldsymbol{y}_t^{\text{D}} = \boldsymbol{W}_{\text{out}}\left[1; \boldsymbol{s}_t; \boldsymbol{x}_t\right]$ where $\boldsymbol{y}_t^{\text{D}}$ is the targeted output. Therefore, the training goal is to find an optimal $\boldsymbol{W}_{\text{out}}$ that enables $\boldsymbol{W}_{\text{out}}\left[1; \boldsymbol{s}_t; \boldsymbol{x}_t\right]$ is equal to $\boldsymbol{y}_t^{\text{D}}$. In contrast to conventional recurrent neural networks that require gradient-based learning algorithms, such as BPTT mentioned in Subsection III-A2 to adjust all input, hidden, and output weight matrices, ESNs only need to train the output weight matrix with simple training methods such as ridge regression. The most universal and stable solution to this problem is via the so-called *ridge regression* approach also known as *regression with Tikhonov regularization* [101], which can be given by:

$$\boldsymbol{W}_{\text{out}} = \boldsymbol{y}_t^{\text{D}}\left[1; \boldsymbol{s}_t; \boldsymbol{x}_t\right]^{\text{T}}\left(\left[1; \boldsymbol{s}_t; \boldsymbol{x}_t\right]\left[1; \boldsymbol{s}_t; \boldsymbol{x}_t\right]^{\text{T}} + \theta\boldsymbol{I}\right)^{-1}, \tag{10}$$

where $\boldsymbol{I}$ is an identity matrix and θ is a regularization coefficient which should be selected individually for a concrete reservoir based on validation data. When $\theta = 0$, the ridge regression will become a generalization of a regular linear regression. However, ridge regression is an offline training method for ESNs. In fact, ESNs can be also trained by using an online method.

The simplest way to train $\boldsymbol{W}_{\text{out}}$ online is using the well-known *least mean squares* (LMS) algorithm in [102], which is a stochastic gradient descent algorithm. At every time step t, the LMS algorithm changes $\boldsymbol{W}_{\text{out}}$ in the direction of minimizing the instantaneous squared error $\|\boldsymbol{y}_t^{\text{D}} - \boldsymbol{y}_t\|^2$. LMS is a first-order gradient descent method, locally approximating the error surface with a hyperplane. However, this approximation in LMS is not always accurate. In particular, the curvature of the error surface is very different in different directions. To overcome this

disadvantage, a learning algorithm named the recursive least squares (RLS), can be used for training ESNs. RLS, is insensitive to the detrimental effects of eigenvalue spread and exhibits a much faster convergence. Demonstrations of RLS for ESNs are presented in [96] and [103]. The backpropagation-decorrelation in [104] and the so-called FORCE learning algorithm in [105] are two other powerful methods for online training of single-layer output with feedback connections. Hence, the output weight matrix of each ESN can be optimized using different training methods. One can select the most suitable ESN training algorithm according to the scenario and target performance needed.

In summary, RNNs (including ESNs) are a type of ANNs in which the connections between neurons will form a directed cycle. This small change enables RNNs to store historical input informations. Therefore, RNNs can be used to process the tasks that cannot be done in one step such as mobility pattern prediction and speech recognition. However, since the RNNs can store historical information, the hidden state of each neuron depends both the current and historical inputs, which makes the training of RNNs much more complex and time consuming than traditional ANNs such as FNNs. To reduce the training complexity, one can adopt the ESN model, which is a type of RNNs that will typically have only one hidden layer and needs to only train an output weight matrix. Due to ESN's properties such as training simplicity, it has been widely used for several supervised learning tasks such as predictions, as well as reinforcement learning tasks such as robotic control, classification, and regression.

B. Spiking Neural Network

Another important type of ANNs is the so-called *spiking neural networks (SNNs)*. In contrast to other ANNs such as FNNs and RNNs that simply use a single value to denote the activations of neurons, SNNs use a more accurate model of biological neural networks to denote the activations of neurons. Meanwhile, the change of the neurons' activations will directly affect the training methods for SNNs. Next, we will first introduce the architecture of SNNs. Then, the training methods are formulated. Finally, we give an example for SNNs, called liquid state machine.

1) Architecture of a Spiking Neural Network: The architecture of neurons in SNNs is similar to the neurons in biological neural networks. Therefore, we will first discuss how the neurons operate in a real-world biological neural networks. Then, we formulate the model of neurons in SNNs.
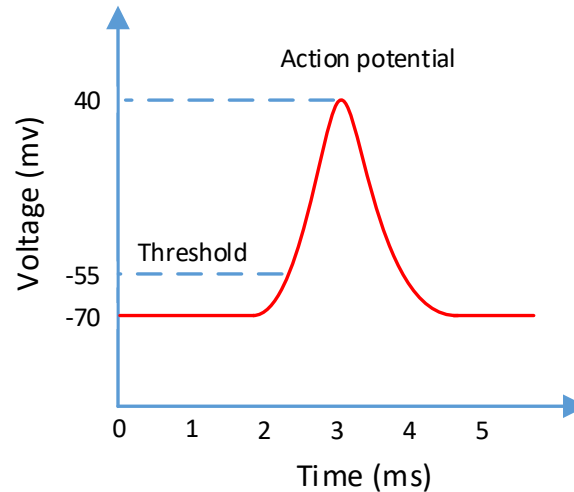
Fig. 7. Action potential of each spiking neuron.

In biological neural networks, neurons use spikes to communicate with each other. Incoming signals alter the voltage of a neuron and when the voltage reaches above a threshold value, the neuron sends out an action potential which is a short (1 ms) and sudden increase in voltage that is created in the cell body or soma. Due to the form and nature of this process, as shown in Fig. 7, we refer to it as a *spike* or a pulse. This spike in the form of a 1-2 ms pulse travels through the axon, which is linked to the dendrite of another neuron via synapses. The incoming spike influences the receiving neuron's membrane potential, which causes the neuron to fire a spike. A spike can have either a positive or a negative impact on the receiving neuron, also called postsynaptic neuron. The positive one is called postsynaptic potential (PSP) and the negative one is inhibitory postsynaptic potential. After sending out a spike, the neuron enters a short moment of rest, the refractory period, during which it cannot send out a spike again. For SNNs, the use of such spikes can significantly improve the dynamics of the network. Therefore, SNNs can model a central nervous system and study the operation of biological neural circuits. Since the neurons in SNNs are modeled based on the spike, as opposed to other ANNs such as FNNs and RNNs, SNNs have two major advantages over traditional neural networks: fast real-time decoding of signals and high information carriage capacity by adding a temporal dimension. Therefore, an SNN can use fewer neurons to accomplish the same task compared to traditional neural networks and it can also be used for real-time computations on continuous streams of data which means that both the inputs and outputs of an SNN are streams of data in continuous time. However, the training of SNNs will be more challenging (and potentially more time consuming)

than traditional ANNs due to their complex spiking neural models.

The basic architecture of an SNN consists of the following components:

- *Input layer* that consists of a number of neurons, representing the signal transmitted in the neurons.

- *Output layer* that consists of a number of neurons, representing the desired output signal.

- *Input weight matrix* that represents the strengths of the connections between the neurons in the input layer and the neurons in the hidden layer;

- *Neuron weight matrix* that represents the strengths of the connections between the neurons.

- *Output weight matrix* that represents the strengths of the connections between the neurons in the output layer and the neurons in the hidden layer.

- *An activation value* that represents a biological neuron's membrane potential.

In the previously discussed ANNs, such as RNNs and FNNs, the weight matrix values are non-negative which means that if two neurons are connected, then the activation value of each neuron will increase. However, in SNNs, the weight matrix values of SNNs can be negative due to the inhibitory neurons, which improves the dynamics of the neuron system. A spiking neural model is used to represent how a neuron processes the spikes. There are many models to describe the spiking neurons such as Hodgkin-Huxley model in [106], integrate-and-fire model in [107], theta model in [108], Lzhikevich model in [109], and spike response model in [110]. The most common spiking neuron models are based on the *integrate-and-fire* and *leaky-integrate-and-fire (LIF)* paradigms due to their low complexity. Hence, we will present the LIF model [111] here in detail as it also serves as a building block for understanding other types.

- *LIF Spiking Neuron Model:* For different spiking neuron models, the spike transmission process will be different. We first study the spike transmission process in the LIF model. Then, the mathematical LIF model is formulated. In the LIF model, the activation value of a neuron normally starts at a resting value, similar to the resting potential in biological neural networks. Then, whenever the neuron receives a spike, the activation value is raised. Once the activation value crosses the firing threshold, the neuron fires, sending a signal to its all associated neurons. After firing, the neuron observes a relative refractory period during which the activation value is reset to a value lower than its resting value, named the reset value [112].

The LIF model is based on (and most easily explained by) principles of electronics as shown in Fig. 8. From Fig. 8, we can see that a spike travels down the axon and is transformed by a
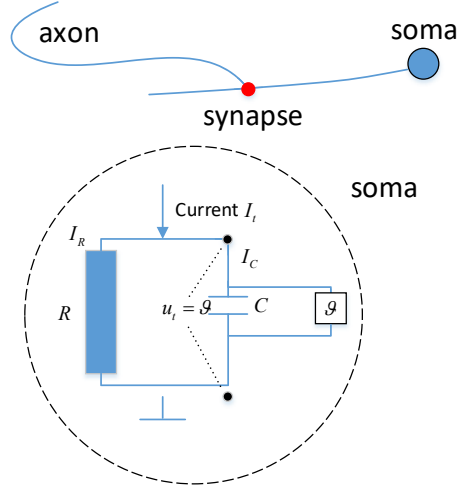
Fig. 8. Illustration of an integrate-and-fire model.

low-pass filter, which converts the short pulse into a current I_t. The driving current can be split into two components, $I = I_R + I_C$. The first component is the resistive current I_R that passes through the linear resistor R. Here, R models the membrane resistance. I_R can be calculated from Ohm's law as $I_R = \frac{u_R}{R}$ where u_R is the voltage across the resistor, analogous to the membrane potential. The second component I_C charges the capacitor C. Here, C models the membrane capacitance. Based on the definition of the capacity, $C = \frac{q}{u}$ with q being the charge, the capacitive current is given by $I_C = C\frac{du}{dt}$ and, hence, the total current will be given by:

$$I_t = C\frac{du}{dt} + \frac{u - u_{\text{rest}}}{R}, \tag{11}$$

where $u - u_{\text{rest}} = u_R$ with u_{rest} being the resting value. In an LIF model, the driving current I_t represents the input signals of each spiking neuron at time t. We introduce the time constant $\tau = RC$ of the neuron membrane, modeling the voltage leakage and multiply (11) by R, which can be given by:

$$\tau\frac{du}{dt} = I_t - u. \tag{12}$$

Based on (12), we can calculate the activation potential at time t. In particular, given a constant input current $I_t = I$ and $u_0 = u_{\text{rest}}$, (12) can be solved by:

$$u_t = u_{\text{rest}} + RI\left[1 - \exp\left(-\frac{t}{\tau}\right)\right], \tag{13}$$

where u_t is the voltage across the capacitor, analogous to the activation value of a spiking neuron.

In the LIF model, the form of an action potential u_t is not explicitly described. Instead, when u_t reaches a certain threshold ϑ (spiking threshold), it is instantaneously reset to a lower value

u_{rest} (reseting value) which can be given by:

$$u_t = u_{\text{rest}}, \ t_f < t \leq t_f + \Delta_{\text{abs}}, \tag{14}$$

where t_f is the time that a spiking neuron triggers a spike, $u_{t_f} = \vartheta$, and Δ_{abs} is the absolute refractory period. During the absolute refractory period, s_t might be clamped to u_{rest} and the leaky integration process is re-initiated following a delay of Δ_{abs} after the spike. The leaky integration process described by (11) will start again with the initial value u_{rest}. Whenever the activation value of a neuron is either higher or lower than its resting potential, its activation value slowly increases or decreases towards the resting value. This is where the leaky part in the LIF name comes from. In fact, equations (11)-(14) describe how the activation of a spiking neuron changes once the neuron receives the signals. From (14), we can see that the activation value u_t of a spiking neuron is time-varying which is different from the previously discussed ANNs such as FNNs in which the activation value is calculated based on (1). In fact, this is the reason that SNNs can model biological neural networks more accurately than traditional RNNs since in biological neural networks, the activation value is also time varying. However, the varied activation value will result in new challenges for the training of SNNs, as discussed next.

*2) **Training in Spiking Neural Networks:*** As previously mentioned, the activation value of each spiking neuron is time varying and, therefore, in SNNs, the connection strength (weight value) between two neurons is also time varying. The case of weight strengthening (weakening) is called potentiation (depression). If the impact of a change lasts up to a few hours, it is called *long term*. Meanwhile, the impact of a change that lasts up to a few minutes scale is called *short term*. Therefore, the classification of an impact of a weight change can be given by: long-term potentiation (LTP) or depression (LTD), short-term potentiation (STP) or depression (STD). Neurobiological research has also increasingly demonstrated that connection strength adjustment in networks of spiking neurons is sensitive to the presence and precise timing of spikes [113].

The most commonly used algorithm to adjust the connection strength between neurons is the so-called *spike-timing dependent plasticity (STDP)*. In STDP, it is first assumed that the weight change between presynaptic neuron i and postsynaptic neuron j is Δw_{ij}, which depends on the relative timing between presynaptic spike arrivals and postsynaptic spikes. Let t_i^z be the presynaptic spike arrival times at synapse, where $z = 1, 2, 3, \ldots$, is the index of the presynaptic spike. Similarly, let t_j^n be the postsynaptic spike arrival time at synapse, where $n = 1, 2, 3, \ldots$,

is the index of the postsynaptic spike. For instance, t_i^2 represents the arrival time of the second spike. The weight change Δw_{ij} can be given by:

$$\Delta w_{ij} = \sum_{f=1}^{N} \sum_{n=1}^{N} Z\left(t_i^f - t_j^n\right), \tag{15}$$

which yields:

$$\Delta w_{ij} = \begin{cases} F_+\left(w_{ij}\right) \exp\left(-\frac{|x|}{\tau_+}\right), & \text{if } x > 0, \\ -F_-\left(w_{ij}\right) \exp\left(-\frac{|x|}{\tau_-}\right), & \text{if } x \leq 0, \end{cases} \tag{16}$$

where $t_i^f - t_j^n$ is the temporal difference between the postsynaptic and the presynaptic spikes. $F_{\pm}\left(w_{ij}\right)$ is the dependence of the update on the current weight of the synapse w_{ij}, and N is the number of spikes considered for updating the weight. $\tau_{\pm}$ is the time constant (i.e., $\tau_+ = 10$ ms and $-\tau_+ = -10$ ms). From (15), we can see that if the presynaptic spikes of a neuron occur immediately before the postsynaptic spikes, then the weight value will be larger and the connection strength will be stronger. Based on the STDP rule, one can control the weight values of LTP and LTD.

In summary, STDP is the most commonly used training algorithm in SNNs which uses temporal windows for controlling the weight LTP and LTD. Its apparent simplicity has led to some to propose that it is a universal "first rule" for training SNNs. Different shapes of STDP windows have been used in recent literature such as in [114]–[116]. However, different forms of STDP are often used in a synapse-specific manner which means that one STDP algorithm may be only available for one type of spiking neuron model. Moreover, STDP only focuses on spike timing and ignores other factors that contributes to weight adjustment such as firing rate and dendritic depolarization.

To overcome the disadvantages of STDP, it is possible to develop training algorithms for adjusting weights of the spiking neurons in SNN. In contrast to the STDP algorithm that only focuses on the temporal domain of SNNs, these training algorithms jointly consider both temporal domain and the complexity of SNNs. The best-known learning rules for training SNNs are the class of error-backpropagation rules for supervised learning. SpikeProp [117], a supervised learning algorithm for SNNs and derived from backpropagation, which uses spike response model neurons to transfer the information in spike-times. Theta neuron learning, another supervised learning method creates a continuous and cyclic model via mapping quadratic-integrate-and-fire neurons to theta neuron, a one-dimensional nonlinear neuron model [117]. There also exist
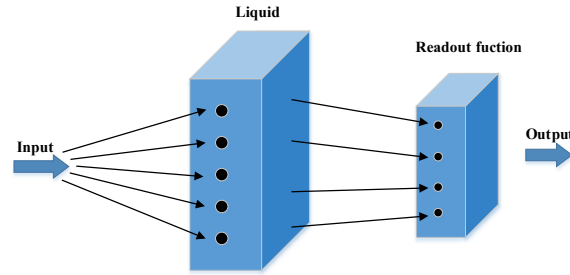
Fig. 9. Architecture of a LSM.

training rules for unsupervised learning of SNNs. A number of approaches for unsupervised learning in spiking neural networks have been developed, based mostly on variants of Hebbian learning [113], such as winner-takes-all learning algorithm in [118]. In [119], a training algorithm that performs unsupervised clustering in a simple two-layer network of spike response model neurons is proposed. Due to the dynamic activation values of spiking neurons, the complexity for training SNNs is much higher than that of training traditional ANNs such as FNNs or RNNs. To reduce the training complexity of SNNs and keep the dynamics of spiking neurons, one promising solution is to develop a spiking neuron network that needs to only train the output weight matrix, like ESNs in RNNs. Next, we specifically present this type of SNNs, named liquid state machine.

3) Example SNN - Liquid State Machine: A *liquid state machine (LSM)* is a reservoir computing algorithm that is conceptually simple and easy to implement as part of an SNN. The architecture of an LSM consists of only two components: liquid and readouts, as shown in Fig. 9. Here, the *liquid* represents a spiking neural network with LIF model neurons and *the readout* function is a number of FNNs. The term liquid stems from the analogy between the operation of the LSM and dropping a stone into water. The falling stone will generate ripples in the water. The input in LSM, that represents the falling stone, is converted into a spatio-temporal pattern activation that is representative of ripples. For an LSM, the connections between the neurons in the liquid is randomly generated which allows LSM to possess a recurrent nature that turns the time-varying input into a spatio-temporal pattern. In contrast to general SNNs that need to adjust the weight values of all neurons, LSMs need to only train the comparatively simple FNN of the readout function. In particular, simple training methods for FNNs such as the feedforward propagation algorithm can be used for for training SNNs to minimize the errors between the desired output signal and the actual output signal, which enables LSM to be widely applied

for practical applications such as [112] and [120]–[124]. Due to LSM's spiking neurons, it can perform machine learning tasks on continuous data like general SNNs but, it is possible to train it using effective and simple algorithms. Next, we specifically introduce the LSM architecture that consists of a liquid model and FNN readout functions.

• **Liquid Model**

In LSM, the liquid is made up of a large number of spiking LIF model neurons, located in a virtual three-dimensional column. For instance, Maass in [121] used an architecture composed of 3 by 3 by 15 neurons. The liquid has two important functions in the classification of time-series data. First, its fading memory is responsible for collecting and integrating the input signal over time. Each one of the neurons in the liquid keeps its own state, which gives the liquid a strong fading memory. The activity in the network and the actual firing of the neurons can also last for a while after the signal has ended, which can be viewed as another form of memory. Second, in the liquid of an LSM, different input signals are separated, allowing for the readout to classify them. This separation is hypothesized to happen by increasing the dimensionality of the signal. For example, if the input signal has 20 input channels, this is transformed into 135 ($3 \times 3 \times 15$) signals and states of the neurons in the liquid. The generation of a liquid for LSM can be done as follows:

- *Neurons*: The liquid consists of both excitatory and inhibitory neurons. Excitatory neurons will increase the membrane potential of their postsynaptic neurons, making them more likely to fire. Inhibitory neurons on the other hand will decrease the membrane potential making the postsynaptic neurons less likely to fire. In particular, one can set 20% randomly selected inhibitory neurons and 80% excitatory neurons.

- *Connections from input to liquid*: for every pair of input signal and liquid neuron there is a certain chance of being connected, e.g., 30% in [121].

- *Connections between neurons in the liquid*: The connections between the neurons are allocated in a stochastic manner, with a probability of a connection being placed between neurons a and b as given by [121]:

$$P_{ab} = C \exp\left(-\frac{d_{ab}^2}{\lambda^2}\right),\tag{17}$$

where C is a constant that depends on the type of both neurons. For example, for excitatory-excitatory connections, $C = 0.3$, for inhibitory-excitatory connections 0.2, for excitatory-

inhibitory connections $C = 0.4$ and $C = 0.1$ for inhibitory-inhibitory connections. d_{ab} is the distance between neuron a and b. λ is a parameter that controls the average number of connections as well as the average distance between connected neurons. The stochastic nature of the creation of connections between neurons makes the connections of spiking neurons highly form a loop. This loop of connections between spiking neurons allow us to view SNNs as a special type of RNNs.

- *Connections from liquid to readout functions*: All neurons in a liquid will connect to the readout functions.

• **Readout Model**

The readout of an LSM consists of one or more FNNs that use the activity state of the liquid to approximate a specific function. The purpose of the readout is to build the relationship between the dynamics of the spiking neurons and the desired output signals. The inputs of the readout networks are so-called readout-moments. These are snapshots of the liquid activity taken at a regular interval. The readout can include membrane potentials or spikes or both. Whatever measure is used, the readout represents the state of the liquid at some point in time. In general, in LSM, FNNs are used as the readout function. FNNs will use the liquid dynamics (i.e., membrane potentials and spikes) as their input and the desired output signals as their output. Then, the readout function can be trained using traditional training methods used for FNNs, mainly backpropagation. There are also some variations on backpropagation such as Levenberg-Marquardt [125], which Matlab recommends for speed and performance. Hence, LSM can be trained using different training methods for FNNs. One can select the most suitable LSM training algorithm according to the specific situation. Once the readout function has been trained, the LSM can be used to perform corresponding tasks.

In summary, SNNs are a type of ANNs that model the biological neurons more accurately than other types of ANNs. This change enables SNNs to have two advantages over traditional ANNs: fast real-time decoding of signals and high information carriage capacity. SNNs are particularly effective in dealing with continuous data such as spike trains [121]. However, the complexity of general training methods for SNNs can be much higher than traditional ANNs such as FNNs due to the dynamic spiking neurons. To alleviate this training complexity, a special time of SNNs, known as LSM can be used. LSM consists of randomly connected LIF model neurons and the FNNs readout functions, and is one type of reservoir computing algorithms. LSM is particularly
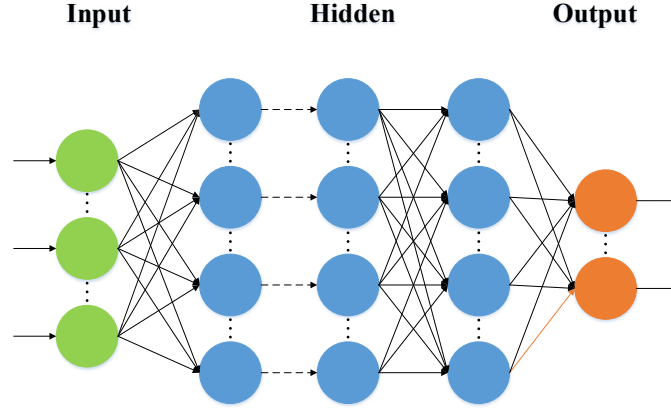
Fig. 10. Architecture of a DNN.

popular due to its simple training, since it needs only to train the readout functions. Due to spiking neurons in LSM, it can deal with continuous-time data. Moreover, the readout function that consists of multiple FNNs enables an LSM to perform multiple tasks at the same time.

C. Deep Neural Networks

Thus far, all of the discussed ANNs, including FNN, RNN, and SNN, have assumed a single hidden layer. Such an architecture is typically referred to as a shallow neural network. In contrast, a *deep neural network (DNN)* is an ANN with multiple hidden layers between the input and output layers [126], as shown in Fig. 10. Therefore, a DNN models high-level abstractions in data through multiple nonlinear transformations and thus learning multiple levels of representation and abstraction [126]. Several types of DNNs exist such as deep convolutional networks, deep RNNs, deep belief networks, deep feedforward networks, deep SNNs, deep Q-learning, deep ESN, deep residual network, and long-short term memory (LSTM) [126]. In fact, the reasons that have made the move from conventional, shallow ANNs, towards DNN possible and desirable are summarized as follows:

- *Improved computing capacity:* The advances in hardware development and data processing capabilities are at the core of the recent renewed interest in developing DNNs. In particular, graphics processing units (GPUs), originally designed to support computer graphics in video games, have been recently used for speeding up the execution of machine learning algorithms. This in turn has resulted in a faster and more parallelized computation thus decreasing the required processing time.

- *Improved datasets:* The availability of a large amount of data has made the training of DNNs possible.

- *Improved training algorithms and network architectures:* Deep network architectures and training algorithms have evolved over the years. For example, the use of rectified linear units (ReLU) instead of sigmoid or tanh has made training faster. The RelU activation function is defined as $f(x) = x$ for $x > 0$ and $f(x) = 0$ otherwise. It is shown in [127] that the ReLU activation function outperforms the sigmoidal one in DNNs and is six times faster than tanh [128] for reaching the same training error. In fact, one major benefit of the ReLU activation function is the reduced likelihood of the gradient to vanish. Unlike the derivative of the sigmoid function which is always smaller than one, the gradient of the ReLU function is either 0 for input less than 0 or 1 for input greater than 0. Therefore, one can stack as much layers as needed without having the gradient neither vanish nor explode, thus yielding a faster learning. Another benefit of ReLUs is sparsity which arises when $x < 0$. Sigmoids on the other hand are always likely to generate some non-zero value resulting in dense representations. Sparse representations have been shown to be more beneficial than dense representations.

As opposed to shallow ANNs that have only one hidden layer, a DNN having multiple layers is more beneficial due to the following reasons:

- *Number of neurons:* Generally, a shallow network would require a lot more neurons than a DNN for the same level of performance. In fact, the number of units in a shallow network grows exponentially with the complexity of the task.

- *Task learning:* While shallow networks can be effective to solve small-scale problems, they can be ineffective when dealing with more complex problems such as image recognition. In fact, the main issue is that shallow networks are very good at memorization, but not so good at generalization. As such, DNNs are more suitable for many real-world tasks which often involve complex problems that are solved by decomposing the function that needs to be learned into a composition of several simpler functions thus making the learning process effective.

It is worth noting that, although DNNs have a large capacity to model a high degree of nonlinearity in the input data, a central challenge is that of overfitting. While overfitting can

be a challenge in any type of ANN (as explained in Section III), typically, typically, it can be overcome by simple regularization methods [129]. However, in DNNs, it becomes particularly acute due to the presence of a very large number of parameters. To overcome this issue, several approaches, known as regularization, have been designed. These methods modify the learning algorithm so that the test error is reduced at the expense of increased training error. Some of the most commonly used techniques for regularization in DNNs are summarized as follows [129]:

- *Dataset augmentation:* An overfitting model can perform better if the learning algorithm processes more training data. While an existing dataset might be limited, for some machine learning problems there are relatively easy ways of creating synthetic data. Therefore, dataset augmentation refers to the expansion of the available dataset by applying operations which reflect real world variations as close as possible.

- *Early stopping:* Intuitively, as more data is fed to the model, both training and test errors go down. After enough passes over training data, the model might start overfitting and learning noise in the given training set. In this case, the training error will decrease while the test error gets worse. Therefore, early stopping overcomes overfitting by interrupting the training process once the performance of the model on a validation set gets worse. A validation set is a set of examples that are not used for neither the training nor the testing process. They are considered to be representative of future test examples. Early stopping is effectively tuning the hyperparameter pertaining to the "number of epochs/steps".

- *Dropout layer:* At each training iteration, a dropout layer randomly removes some nodes in the network along with all of their incoming and outgoing connections. Dropout can be applied to either the hidden or the input layer. The intuition behind this approach is that the nodes become more insensitive to the weights of other nodes, and therefore the model becomes more robust. If a hidden unit has to be working well with different combinations of other hidden units, it's more likely to do work well individually.

- *Weight penalty L1 and L2:* Weight penalty, also known as "weight decay", is a technique that relies on the implicit assumption that a model with small weights is somehow simpler than a network with large weights. Therefore, the penalties try to keep the weights small or non-existent (zero) unless there are large gradients to counteract the penalties values. The L2 norm penalizes the square value of the weight and tends to drive all the weights to

smaller values. On the other hand, the L1 norm penalizes the absolute value of the weight and tends to drive some weights to exactly zero, while allowing some weights to be large.

*1) **Training Deep Neural Networks**:* DNNs are often much harder to train than shallow ANNs due to the instability of their gradient that occurs when training them with gradient-based methods such as those described in Section II. In such conventional methods, each one of the ANN's weights receives an update proportional to the gradient of the error function with respect to the current weight in each iteration of training. In particular, the weights and the activation functions (or more precisely, their derivatives) that the gradient passes through will affect the magnitude of the gradients. Here, note that the gradient by the backpropagation algorithm is computed by the chain rule. Therefore, multiplying n of the gradients at each layer makes the gradients at the "front" layers, in an n-layer DNN, exponentially decrease or increase with n for small gradient values within range (-1, 1) or for large gradient values, respectively. This is obviously not a major problem in conventional shallow ANNs, as they have only one single layer. For example, the tanh derivative is < 1 for all inputs except 0 and the sigmoid is always ≤ 0.25 when used as activation functions $f(\cdot)$ in (1). These two problems are known as the *vanishing gradient problem* and the *exploding gradient problem*, respectively, and they can result in having different layers in DNNs learn at vastly different speeds. For instance, for a vanishing gradient problem, when latter layers in the DNN are learning well, early layers often learn almost nothing during training. To overcome this instability, several techniques for training DNNs have been proposed in the literature [85], [130]–[132]. In what follows, we summarize some of these techniques such as the adaptive learning rate algorithms, multi-level hierarchy, LSTM and residual networks (ResNets):

- *Adaptive learning rate training algorithms:* As mentioned in Subsection II-C, several optimization algorithms have been proposed in order to allow more flexibility during the learning process. In particular, optimization algorithms such as Adagrad, Adam, AdaDelta, RMSProp have been recently proposed for overcoming the vanishing gradient problem in DNNs [81], [130], [133]. These algorithms provide adaptive learning rates and use different rates for different model parameters. For instance, in the RMSprop gradient descent optimization algorithm [130], the learning rate (i.e., gradient) of a particular weight is divided by a running average of the magnitudes of recent gradients for that weight. Here, note that these

rates are considered as hyper-parameters and should be tuned on a subset of training data.

- *Multi-level hierarchy:* Schmidhuber's multi-level hierarchy of DNNs [131] is based on pre-training one level at a time through unsupervised learning and then fine-tuning through backpropagation. Here, each level learns a compressed representation of the observations that is fed to the next level.

- *Long short-term memory:* The LSTM architecture is a special type of deep RNN that was introduced in 1997 by Hochreiter and Schmidhuber [85]. In the recurrency of the LSTM, the activation function is the identity function with a derivative of 1. Therefore, in LSTM, the backpropagated gradient neither vanishes nor explodes when passing through, but remains constant. Moreover, unlike traditional RNNs in which each hidden node consists of a single activation function, a hidden node in an LSTM structure is a memory cell with three different gates which either pass or block information, i.e., 1 or 0, and thus allowing it to maintain its own cell state. These gates consist of the forget gate, the input gate, and the output gate and a combination of these gates is trained. These gates control the extent to which new content should be memorized, old content should be erased, and current content should be exposed. Normal RNNs take as an input their previous hidden state and the current input, and output a new hidden state. An LSTM does the same, except that it also takes in its old cell state and outputs its new cell state. Therefore, no matter how 'deep' your network is, or how 'long' the input sequence is, the network can remember those values, as long as those gates are all 1 along the path. Consequently, iterative gradient descent such as backpropagation through time can be used for training LSTMs. In reinforcement learning applications, LSTM can be trained by policy gradient methods, evolution strategies, or genetic algorithms. More details on LSTM will be given in the next subsection.

- *Residual networks:* Residual neural networks have been introduced by Microsoft [132]. They yield lower training error (and test error) than their shallower counterparts simply by reintroducing outputs from shallower layers in the network to compensate for the vanishing data. Therefore, as opposed to traditional activation functions that are defined as $y = f(x)$, these functions are defined as $y = f(x) + x$ in ResNets. ResNets have been recently applied for image recognition and classification [132].

Alongside the aforementioned techniques for overcoming the instability of the gradient while

training DNNs, some useful general guidelines for training DNNs are summarized as follows [134], [135]:

- *Training data:* Use a large training dataset to avoid overfitting; one can possibly rely on data augmentation to create new examples if needed.

- *Activation function:* Depending on the task that needs to be learned, one can choose a sigmoid, tanh, ReLu or SoftSign function.

- *Number of hidden units:* There is an inherent tradeoff that must be factored in when choosing the number of hidden units. On the one hand, using too few neurons in the hidden layers results in underfitting which could make it hard for the DNN to detect the signals in a complicated dataset. On the other hand, using too many neurons in the hidden layers can either result in overfitting or an increase in the training time that could prevent the training of the DNN.

- *Number of layers:* Selecting the optimal number of layers can be done using an experimental approach in which one would keep on adding layers until the test error stops to improve.

- *Weight initialization:* Weights should be initialized with small random numbers. For instance, when using sigmoid activation functions, if weights are initialized to very large numbers, then, the sigmoid will saturate resulting into dead neurons. On the other hand, if the weights are very small, then the gradients will also be small. Therefore, it is preferable to choose weights in an intermediate range having an even distribution around a mean value, typically following a uniform distribution.

- *Hyperparameter tuning:* As the number of hyperparameters keeps on increasing, the computations required for grid search increase exponentially. Therefore, given the large number of hyperparameters in DNNs, one should use random search/sampling instead of grid search, for choosing the optimal hyperparameters. A combination of hyperparameters is generally chosen from a uniform distribution within a particular desired range.

- *Stochastic vs. mini-batch learning:* In a stochastic learning approach, the gradients of the weights are tuned after each training sample. This approach introduces noise into the gradients thus making the model less prone to overfitting. Nevertheless, a stochastic learning approach might be relatively slow and inefficient especially given the availability of machines with high computational power capable of performing parallel computations.
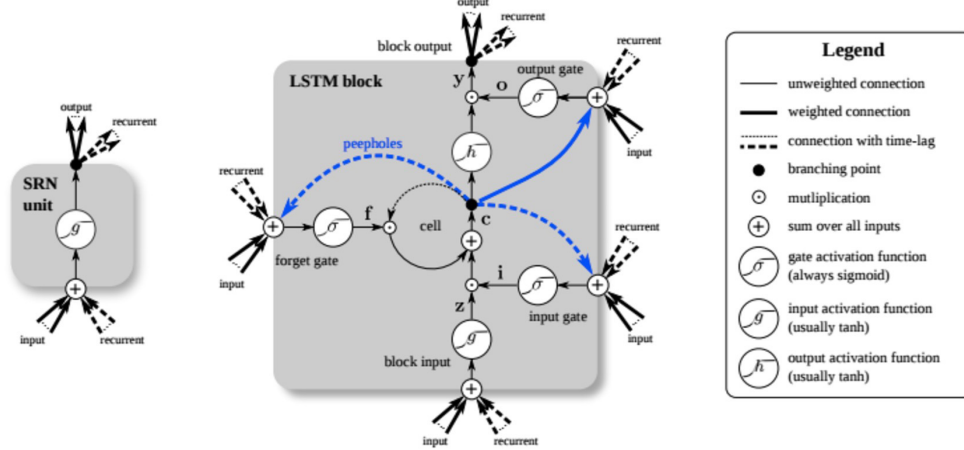
Fig. 11. Architecture of an LSTM as shown in [136].

Therefore, for greater throughput (and faster learning), it is recommended to use mini-batches instead of stochastic learning, as discussed previously in Section II-C. An appropriate batch size should be capable of retaining some noise and simultaneously using the computation power of machines more effectively. Typically, a batch size between 16 and 128 examples is a good choice (exponential of 2). Here, we note that, in online learning, where the model gets the training data as a stream, one has to resort to stochastic learning.

- *Shuffling training examples:* Randomizing the order of training examples (in different epochs, or mini-batches) results in a faster convergence.

- *Number of epochs/training iterations:* In general, training a DNN for multiple epochs will result in a better model. To know the optimal number of required epochs, one could compare the test error with train error; if the gap is decreasing, then keep on training.

Therefore, the above discussion gives a brief overview on DNNs and some guidelines for training them. Next, we elaborate more on LSTM, a special kind of DNN that is capable of storing information for long periods of time and we also overview convolutional neural networks, a popular DNN in many applications.

2) Example DNN - Long Short Term Memory: As mentioned earlier in Subsection III-C1, LSTMs are a special kind of "deep learning" RNNs that are capable of storing information for either long or short periods of time. In particular, the activations of an LSTM network correspond to short-term memory, while the weights correspond to long-term memory. Therefore, if the activations can preserve information over long duration of time, then this makes them long-term short-term memory. Although both ESN and LSTM are good at modeling time series data, LSTM

Input gate	Forget gate	Behavior
0	1	remember the previous value
1	1	add to the previous value
0	0	erase the value
1	0	overwrite the value

cells have the capability of dealing with long term dependencies. An LSTM network contains LSTM units each of which having a cell with a state c_t at time t. Access to this memory unit, shown in Fig.11, for reading or modifying information is controlled via three gates:

- *Input gate (i_t):* controls whether the input to is passed on to the memory cell or ignored.

- *Output gate (o_t):* controls whether the current activation vector of the memory cell is passed on to the output layer or not.

- *Forget gate (f_t):* controls whether the activation vector of the memory cell is reset to zero or maintained.

Therefore, an LSTM cell makes decisions about what to store, and when to allow reads, writes, and erasures, via gates that open and close. At each time step t, an LSTM receives inputs from two external sources, the current frame x_t and the previous hidden states of all LSTM units in the same layer h_{t-1}, at each of the four terminals (the three gates and the input). These inputs get summed up, along with bias factors b_f, b_i, b_o, and b_c. The gates are activated by passing their total input through the logistic function. Table I summarizes the various behaviors an LSTM cell can achieve depending on the values of the input and forget gates. Moreover, the update steps of a layer of LSTM units are summarized in the following equations:

$$g_t = f_g(\boldsymbol{W}_f \boldsymbol{x}_t + \boldsymbol{U}_f \boldsymbol{s}_{t-1} + \boldsymbol{b}_f), \tag{18}$$

$$i_t = f_g(\boldsymbol{W}_i \boldsymbol{x}_t + \boldsymbol{U}_i \boldsymbol{s}_{t-1} + \boldsymbol{b}_i), \tag{19}$$

$$o_t = f_g(\boldsymbol{W}_o \boldsymbol{x}_t + \boldsymbol{U}_o \boldsymbol{s}_{t-1} + \boldsymbol{b}_o), \tag{20}$$

$$c_t = g_t \odot c_{t-1} + i_t \odot f_c(\boldsymbol{W}_c \boldsymbol{x}_t + \boldsymbol{U}_c \boldsymbol{h}_{t-1} + \boldsymbol{b}_c), \tag{21}$$

$$\boldsymbol{s}_t = o_t \odot f_h(c_t), \tag{22}$$

where g_t, i_t, and o_t are the forget, input and output gate vectors at time t, respectively. x_t is the input vector, h_t is the hidden/output vector and c_t is the cell state vector (i.e., internal memory) at time t. $\boldsymbol{W}_f$ and $\boldsymbol{U}_f$ are the weight and transition matrices of the forget gate, respectively. f_g, σ_c, and f_h are the activation functions and correspond respectively to the sigmoid, tanh and tanh functions. $\odot$ denotes the Hadamard product. Compared to a standard RNN, LSTM uses additive memory updates and separates the memory c from the hidden state $\boldsymbol{s}$, which interacts with the environment when making predictions. To train an LSTM network, the stochastic gradient descent algorithm that was introduced in Subsection II-C can be used along with the general guidelines given in Subsection III-C1.

LSTM is thus suitable for applications involving sequential learning; it can classify, process and predict time series given time lags of unknown size and duration between important events. In what follows, we summarize a number of other variants of LSTM:

- *Bidirectional LSTM:* Unlike conventional LSTMs, bidirectional LSTMs utilize both the previous and future context, by processing the data from two directions with two separate hidden layers. One layer processes the input sequence in the forward direction, while the other processes the input in the reverse direction. This allows the bidirectional LSTM, at each point in the sequence, to have complete and sequential information about all points before and after it. The output of the current time step is then generated by combining both layers' hidden vector [137].

- *Sequence-to-sequence LSTM (a.k.a. encoder-decoder LSTM):* An encoder LSTM reads the input sequence and transforms it into a rich fixed-length vector representation, which, in turn, is used as the initial hidden state of a decoder LSTM to generate the output sequence from that vector. Therefore, a straightforward application of the LSTM architecture is that it can solve general sequence-to-sequence problems. Unlike other DNNs that can only be applied to problems whose inputs and targets can be sensibly encoded with vectors of fixed dimensionality, sequence-to-sequence LSTM can solve problems with sequences whose lengths are not known a-priori [138].

- *Peephole LSTM:* This is similar to traditional LSTM, however, with extra connections between the memory cell and the gates, called peepholes [139]. These peepholes allow the gates to depend, not only on the previous hidden state h_{t-1}, but also on the previous internal state c_{t-1}.

- *Gated Recurrent Unit (GRU):* GRU was introduced by Cho et al. [140]. It combines the forget and input gates into a single "update gate" and merges the cell state and hidden state. Therefore, each GRU cell consists of a reset gate r and an update gate z. In particular, the reset gate determines how to combine the new input with the previous memory and the update gate defines how much of the previous memory to keep around. The basic equations that describe its operation are the following:

$$\boldsymbol{z}_t = \sigma'_g(\boldsymbol{W}_z\boldsymbol{x}_t + \boldsymbol{U}_z\boldsymbol{h}_{t-1} + \boldsymbol{b}_z), \tag{23}$$

$$\boldsymbol{r}_t = \sigma'_g(\boldsymbol{W}_r\boldsymbol{x}_t + \boldsymbol{U}_r\boldsymbol{h}_{t-1} + \boldsymbol{b}_r), \tag{24}$$

$$\boldsymbol{s}_t = \boldsymbol{z}_t \odot h_{t-1} + (1 - \boldsymbol{z}_t) \odot f'_h(\boldsymbol{W}_h\boldsymbol{x}_t + \boldsymbol{U}_h(\boldsymbol{r}_t \odot \boldsymbol{s}_{t-1}) + \boldsymbol{b}_h), \tag{25}$$

where $\boldsymbol{z}_t$ and $\boldsymbol{r}_t$ correspond to the update and reset gate vectors respectively. f'_g and f'_h denote, respectively, the sigmoid and hyperbolic tangent functions. Therefore, the basic idea of using a gating mechanism to learn long-term dependencies is similar to that of LSTM, but with the following key differences:

  - A GRU has two gates while an LSTM has three gates.
  - GRUs do not possess and internal memory (c_t) that is different from the exposed hidden state. Moreover, they do not have the output gate that is present in LSTMs.
  - The input and forget gates are coupled by an update gate z and the reset gate r is applied directly to the previous hidden state. Thus, the responsibility of the reset gate in a LSTM is split up into both r and z.
  - One does not have to apply a second nonlinearity when computing the output.

The authors in [141] compare the performance of different variants of LSTM. Results show that existing variants of LSTM do not show any significant improvement in the LSTM performance. Nevertheless, some modifications such as coupling the input and the forget gates or removing peephole connections simplify the LSTM architecture without significantly degrading its performance. Results also show that the forget gate and the output activation function are the most critical components of an LSTM unit.

Next, we give an overview on other types of DNN, in particular, convolutional neural networks and deep spiking neural networks.

3) Other Types of DNN - Convolutional Neural Networks and Deep Spiking Neural Networks: Convolutional neural networks (CNNs) and deep spiking neural networks are other types of deep ANN architectures that have been proposed recently in order to provide better performance over existing DNN architectures in various applications. In particular, CNNs have been applied to various domains such as in image and video recognition, recommender systems, and natural language processing (and more recently wireless communications [142]). They constitute a class of deep, feed-forward ANNs that consist of one input layer, one output layer, and multiple hidden layers [143]. In CNNs, the hidden layers have neurons arranged in three dimensions: width, height and depth. The neurons inside a layer are connected to only a small region of the layer before it, called a receptive field. These hidden layers are either convolutional, pooling, or fully connected, and, hence, if one hidden layer is convolutional (pooling/fully connected), then it is called convolutional (pooling/fully connected) layer. As opposed to traditional ANNs such as RNNs and FNNs, convolutional layers apply a convolution operation to the input, passing the result to the next layer. Each convolutional neuron processes data only for its receptive field thus allowing CNNs to tolerate translation of the input image (eg. translation, rotation, and perspective distortion). In fact, the biases and weights of the convolutional layers are shared thus reducing the number of parameters involved in a CNN. On the other hand, the pooling layers are mainly used to simplify the information from the convolutional layer while fully connected layers connect every neuron in one layer to every neuron in another layer. Therefore, local receptive fields, shared weights, and pooling allow CNNs to achieve better generalization on vision problems compared to the previously discussed ANNs. Nevertheless, the high computation cost, slow training speed, and the need for a larger training data pose a limit on the practicality of CNNs. Here, note that, as opposed to LSTM that are good at temporal modeling, CNNs are appropriate at reducing frequency variations which therefore makes them suitable for applications that deal with spatial data. Moreover, CNN can be combined with LSTM, resulting in a CNN LSTM architecture that can be used for sequence prediction problems with spatial inputs, like images or videos [144].

Another interesting DNN architecture is deep SNN. Deep SNNs have a great potential for improving the latency and energy efficiency of DNNs through event-based computation. In fact, deep SNNs are attractive for real-time applications where speed and power consumption are important factors. Although deep SNNs have been shown theoretically to be computationally

powerful [145], a major issue is the lack of adequate training algorithms suitable for deep SNNs. This follows from the fact that spike signals (i.e., discrete events produced by a spiking neuron whenever its internal state crosses a threshold condition) are not differentiable, but differentiable activation functions are fundamental for using error backpropagation, which is still by far the most widely used algorithm for training DNNs [146]. This in turn has limited the use of deep SNNs in different domains. However, we expect these limitations to be overcome in the near future.

In summary, DNNs have been shown recently to be powerful for solving various real-world tasks and have won numerous contests in machine learning. Nevertheless, there is still a lot of opportunities for research in this domain for improving their performance.

D. Other Types of ANNs

Beyond our previous, detailed discussion of several types of ANNs, in this subsection, we briefly overview other types of ANNs that can potentially be useful for wireless networking problems, such as self-organizing maps, radial basis function neural networks, modular neural networks, and physical neural networks.

• *Self-Organizing Maps:* A self-organizing map (SOM) [147] is a type of ANN that can use a small number of dimensions (typically one or two dimensions) to represent a multidimensional input data. In contrast to the traditional ANNs such as RNNs and FNNs that are generally trained by supervised learning algorithms, SOM is trained using unsupervised learning algorithms. SOMs differ from other ANNs since they use competitive learning as opposed to error-correction learning such as backpropagation. Moreover, SOMs use a neighborhood function to store the topological properties of the input signals. These unique features of SOMs make them suitable for visualizing low-dimensional views of high-dimensional data and multidimensional scaling. However, SOMs can only focus on one type of data. When using categorical data or several types of data as input, SOMs will have a lower performance than traditional ANNs. In wireless networks, SOMs can be useful for signal processing such as signal recognition, users context processing, and users clustering.

• *Radial Basis Function:* A radial basis function (RBF) network [148] is a type of ANN that uses radial basis functions as activation functions for each neuron. Each neuron in the RBF stores an example from the training set. The output of RBF is a linear combination of radial

basis functions of the inputs and neuron parameters. The linearity of RBF prevents it from suffering from local minima. RBF networks have been applied across many applications [149]–[153] including function approximation, stock prediction, classification, and face recognition. However, RBF networks require good coverage of the input space by radial basis functions. Moreover, the centers of RBF functions are determined by the input data, but not by the data related to the prediction task. Therefore, the resources that are used to store useful training data may be wasted by irrelevant input signals that are not related to the task. In wireless networks, this can be useful for antenna array signal processing, channel equalization, coding/decoding, and system identification.

- *Modular Neural Networks:* A modular neural network [154] is a type of ANN that consists of a number of independent ANNs. Each one of these independent ANN is moderated by an intermediary and works with separate inputs. In a modular neural network, the task is divided into several subtasks and each ANN will complete one subtask. The ANNs in a modular neural network will not interact with one another when they process the subtasks. The intermediary is used to collect the outputs of ANNs and use them to produce the output of the entire task. The major advantage of a modular neural network is that it can divide a large task into several subtasks and perform the subtasks using different ANNs. By dividing a task into several subtasks one can improve efficiency, training complexity, and robustness. However, in turn, dividing a task into several subtasks may yield inaccurate learning. Moreover, dividing a learning task into subtasks can be done in numerous ways which makes it complex to determine the optimal division. This will increase the complexity of a modular neural network. Modular neural networks admit several applications such as pattern profiling problem, classification, prediction and identification [155]–[158].

- *Physical Neural Networks:* Physical neural networks (PNNs) belong to a group of ANN methods in which simulating an ANN depends on a physical hardware rather than only on software, as is the case with other ANNs. In PNNs, an adjustable resistance material can be used to emulate the activation function of a neural synapse. Therefore, in PNNs, the physical hardware emulates the neurons and the software emulates the connections between different neurons. PNNs can be used for universal computing and general-purpose machine learning [159]. However, one of the most important challenge for PNN is the scaling problem. A PNN is designed for a specific task. When the tasks change, the physical circuit for the PNN must be designed again, which

significantly increase the complexity of implement of PNNs. PNN have several applications such as classification and regression.

In summary, different types of ANNs will have different architectures, activation functions, connection methods, and data storage capacities. Each specific type of ANNs is suitable for dealing with a particular type of data. For example, RNNs are good at dealing with time-related data while SNNs are good at dealing with continuous data. Moreover, each type of ANNs has its own advantages and disadvantages in terms of learning tasks, specific tasks such as time-related tasks or space-related tasks, training data size, training time, and data storage space. Given all of their advantages, ANNs are ripe to be exploited in a diverse spectrum of applications in wireless networking, as discussed next.

IV. APPLICATIONS OF NEURAL NETWORKS IN WIRELESS COMMUNICATIONS

In this section, we first overview the motivation behind developing ANN solutions for wireless communications and networking problems. Then, we introduce the use of ANNs for various types of wireless applications. In particular, we discuss how to use neural networks for unmanned aerial vehicles (UAVs), virtual reality (VR), mobile edge caching and computing, multiple radio access technologies, and internet of things.

A. Artificially Intelligent Wireless Networks using ANNs: An Overview

One of the main advantages of ANNs is their ability to extract, predict, and characterize regular patterns from massive datasets [160]. Since ANNs can capture many kinds of relationships between a variety of time-varying inputs and outputs such as complex nonlinear relationships or highly dynamic relationships, they are well suited to tackle problems such as content predictions, pattern recognition, classification, regression, clustering, and fault detection [160]. Owing to these advantages, ANNs have been widely applied within financial services, government, healthcare, transportation, marketing, and entertainment [161]. More recently, ANNs have started to attract significant attention in the context of wireless communications and networking [2], [4], [5], [31] and [32], since the development of smart devices and mobile applications has significantly increased the level at which human users interact with the wireless communication system. Moreover, the development of mobile edge computing and caching technologies makes it possible

for base stations (BSs) to store and analyze the behavior of the human users of a wireless network. In addition, the emergence of smart cities motivates the use of ANNs to improve the quality of life for residents. For instance, ANNs can be used for the prediction of the road traffic and intelligent control of the traffic light so as to reduce route congestion and optimize the overall efficiency of a city's transportation network.

In particular, within wireless communication domains, ANNs admit two major applications. First, they can be used for *prediction, inference, and big data analytics purposes*. Within this application domain, the artificial intelligence feature that is enabled by ANNs pertains to the ability of the wireless network to learn from the datasets generated by its users, environment, and network devices. For instance, ANNs can be used to analyze and predict the wireless users' mobility patterns and content requests therefore allowing the BSs to optimize the use of their resources, such as frequency, time, or the files that will be cached across the network. Here, behavioral patterns pertaining to the users such as their mobility patterns and content requests will significantly affect which content will be cached, at which node in the network, and at which time. If a network is able to learn and predict the various *context information* of its users, such as their locations and content requests, then it can intelligently cache the contents that users will request at the mobile edge, in order to reduce the data traffic from the core network. Such predictions lie at the heart of the notion of proactive wireless networking [162], in which a wireless network can predict its users' behavior and adapt its operation to such behavior.

Beyond mobile edge caching, predictions and inference will be a primary enabler of the emerging IoT and smart cities paradigms. Within an IoT or smart city ecosystem, sensors will generate massive volumes of data that can be used by the wireless network to optimize its resources usage, understand its network operation, monitor failures, or simply deliver smart services, such as intelligent transportation. In this regard, the use of machine learning, in general, and ANNs, in particular, as tools for optimized predictions is imperative. In fact, ANNs will equip the network with the capability to process massive volumes of data and to parse useful information out of this data, as a pre-cursor to delivering smart city services. For example, road traffic data gathered from IoT sensors can be processed using ANN tools to predict road traffic status at various locations in the city. This can then be used by the wireless network that connects road traffic signals, apparatus, and autonomous/connected vehicles to inform the vehicles of the traffic state and potentially re-route some traffic to respond to the current state of

the system. Moreover, ANNs can be used to predict the requirement of computational resources that are needed to process a computational task. This will therefore allow BSs to pre-allocate computational resources so as to reduce processing latency and overhead, while also enabling the system to complete a larger number of computational tasks. Moreover, ANNs can be beneficial for integrating different data from multiple sensors thus facilitating more interesting and complex applications of wireless communication. In particular, ANNs can identify nonintuitive features largely from cross-sensor correlations which can result in a more accurate estimation of a wireless network's conditions and an efficient allocation of the available resources. Finally, ANNs can be used for spectrum usage predictions so as to minimize the occurrence of a congestion event or the under-utilization of the available resources thus enabling a better usage of the radio spectrum. Clearly, by enabling smart data analytics, ANNs will provide wireless networks with a plethora of smart applications and smart approaches to manage their resources.

Second, a key application of ANNs in wireless networks is for enabling *self-organizing network operation* by instilling artificial intelligence at the edge of the network, as well as across its various components (e.g., base stations and end-user devices). Such edge intelligence is a key enabler of self-organizing solutions for resource management, user association, and data offloading. In this context, ANNs can be developed to serve as RL tools [163] that can be used by a wireless network's devices to learn the wireless environment and make intelligent decision. For example, in UAV-based wireless networks, each UAV can use an ANN-based RL algorithm to adjust its three dimensional location and flying path so as to optimize the total number of users that can be served within its coverage area.

An ANN-based RL algorithm can be used to learn the users information such as their locations and data rate, and determine the UAV's path based on the learned information. Compared to traditional learning algorithms such as Q-learning that use tables or matrices to record historical data that do not scale well for a network with dense users, ANNs use a nonlinear function approximation method to find the relationship from the historical information. Therefore, ANN-based learning algorithms can be applied for a network with dense users which makes them suitable for tomorrow's ultra dense wireless networks.

Naturally, ANNs can be simultaneously employed for both prediction and intelligent/self-organizing operation, as the two functions are largely interdependent. For instance, data can help in decision making, while decision making can generate new data. For example, when
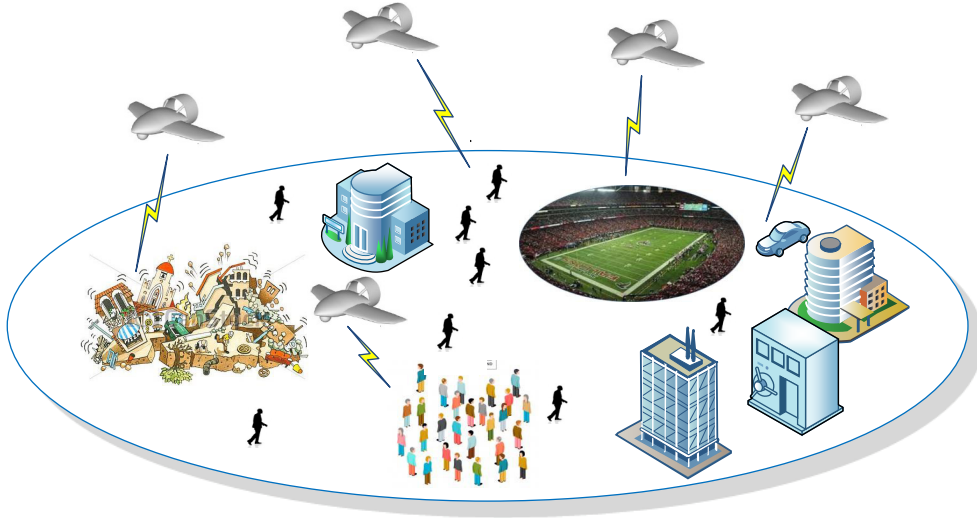
Fig. 12. UAV-enabled wireless networks.

considering virtual reality applications over wireless networks, one can use ANN to predict the behavior of users, such as head movement and content requests. These predictions can help an ANN-based RL algorithm to allocate computational and spectral resources to the users hence improving their QoS. Next, we discuss specific applications that use ANNs for wireless communications.

B. Wireless Communications and Networking with Unmanned Aerial Vehicles

1) UAVs for Wireless Communication: Providing connectivity from the sky to ground wireless users is an emerging trend in wireless communications [164], as shown in Fig. 12. High and low altitude platforms, drones, and small aircrafts are being considered as candidates for servicing wireless users and, thus, complementing the terrestrial communication infrastructure. Such communication from the sky is expected to be a major component of beyond 5G cellular networks. Compared to terrestrial communications, a wireless system with low-altitude UAVs is faster to deploy, more flexibly reconfigured, and likely to have better communication channels due to the presence of short-range, line-of-sight (LoS) links. In a UAV-based wireless system, UAVs can have three key functions: aerial BSs, aerial relays, and cellular-connected UAVs (i.e., user equipment (UE) UAVs) for information dissemination/data collection. Aerial BSs can be used for coverage extension, capacity improvement, mission critical services, and in an event of damaged/overloaded terrestrial BSs. They can also be used as part of a fully-fledged flying

wireless cellular network. For instance, UAVs can be used as flying BSs that can dynamically change their locations and, then, service ground users. Cellular connected UE UAVs, on other hand, open up a large number of IoT applications in agriculture, military, mining operations and industrial inspection services. Such applications include real time video streaming, delivery, surveillance and transmitting telematics. Moreover, UAVs can act as relays between a source and a destination in which a LoS link does not exist or for backhaul connectivity of BSs [165].

However, the use of highly mobile and energy-constrained UAVs for wireless communications also introduces many new challenges such as network modeling, backhaul (fronthaul) limitation for UAV-to-UAV communication when UAVs act as flying BSs, optimal deployment, air-to ground channel modeling, energy efficiency, path planning, and security. In particular, compared to the deployment of terrestrial BSs that are static, mostly long-term, and two-dimensional, the deployment of UAVs is flexible, short-term, and three-dimensional. Therefore, there is a need to investigate the optimal deployment of UAVs for coverage extension and capacity improvement. Moreover, UAVs can be used for data collection, delivery, and transmitting telematics. Hence, there is a need to develop intelligent self-organizing control algorithms to optimize the flying path of UAVs. In addition, the scarcity of the wireless spectrum, that is already heavily used for terrestrial networks, is also a big challenge for UAV-based wireless communication. Due to the UAVs' channel characteristics (less blockage and high probability for LoS link), the use of mmWave spectrum bands will be a promising solution for UAV-based communication. Therefore, one can consider resource management problems in the context of mmW-equipped UAVs, given their potential benefits for air-to-ground communications. Finally, one must consider the resource allocation, interference management, and routing strategy when the UAVs act as users.

2) Neural Networks for UAV-Based Wireless Communication: Due to the flying nature of UAVs, they can track the users' behavior and collect information such as users' and vehicles' data within any distance, at any time or any place, which provides an ideal setting for implementing neural networks techniques. ANNs have two major use cases for UAV-based wireless communication. First, using ANN-centric RL algorithms, UAVs can be operated in a self-organizing and manner. For instance, using ANNs as RL, UAVs can dynamically adjust their locations, flying directions, resource allocation decisions, and path planning to serve their ground users and adapt to these users' dynamic environment. Second, UAVs can be used to map the ground environment as well as the wireless environment itself to collect data and leverage ANN algorithms to exploit

the collected data and perform data analytics to predict the ground users' behavior. For example, ANNs can exploit collected mobility data to predict the users' mobility patterns. Based on the behavioral patterns of the users, battery-limited UAVs can determine their optimal locations and design an optimal flying path to service ground users. Meanwhile, using ANNs enables more advanced UAV applications such as environment identification, $360°$ VR image generation, and content caching. In particular, UAVs can use ANNs to investigate the collected data and predict the users' movements. Based on this prediction, UAVs can generate $360°$ images from high altitude and deliver them to ground users. Meanwhile, UAVs can be equipped with storage units to store popular content. Cache-enabled UAVs can directly send the contents that are stored at the cache to the users that request the cached contents to offload data traffic from the core network. To improve caching efficiency, UAVs can use ANNs to predict the users' content request behavior. Clearly, within a wireless environment, most of the data of interest, such as that pertaining to the human behavior, UAV movement, and data collected from wireless devices will be time related. For instance, certain users will often go to the same office for work at the same time during weekdays. ANNs can effectively deal with time-dependent data which makes them a natural choice for the applications of UAV-based wireless communication.

However, using ANNs for UAVs faces many challenges such as limited flight time to collect data, limited power and computational resources for training ANNs, and data errors due to the air-to-ground channel. First, UAVs have limited battery life and limited computational power which can significantly constrain the use of ANNs. This stems from the fact that ANNs require a non-negligible amount of time and computational resources for training. For instance, UAVs must consider a tradeoff between the energy used for training ANNs and that used for other applications such as servicing users. Moreover, due to their flight time constraints [166], UAVs can only collect data within a limited time period. In consequence, UAVs may not have enough collected data for training ANNs. In addition, the air-to-ground channels of UAVs will be significantly affected by the weather, environment, and their movement. Therefore, the collected data can include errors that may affect the accuracy of the outcomes of the ANNs.

• *Related works:* The existing literature has studied a number of problems related to using ANNs for UAVs such as in [167]–[171]. In [167], the authors studied the use of neural networks for trajectory tracking of UAVs. The authors in [168] used a modified grossberg neural network as a self-organizing algorithm that allows UAVs to avoid obstacles and collision. The work

in [169] proposed a multilayer perceptron based learning algorithm that uses aerial images and aerial geo-referenced images to estimate the positions of UAVs. In [170], a deep neural network based algorithm was proposed for the classification of images collected by drones. In essence, the recent works in [167]–[171] have used ANNs to address problems such as UAV control, position estimation, image classification. However, none of these works [167]–[171] investigate the use of ANNs within the context of UAV-based wireless communications. For UAV-based wireless communications, ANNs can be used many applications such as path planning, deployment, resource management, and channel modeling, as mentioned previously. Next, we explain a specific ANN application for UAV-based wireless communication.

3) Example: An elegant and interesting use of neural networks for UAV-based communication systems is presented in [171] for the study of the proactive deployment of cache-enabled UAVs. The model in [171] considers the downlink of a wireless cloud radio access network (CRAN) servicing a set of mobile users via terrestrial remote radio heads and flying cache-enabled UAVs. Terrestrial remote radio heads (RRHs) transmit over the cellular band and are connected to the cloud's pool of the baseband units (BBUs) via capacity-constrained fronthaul links. Since each user has its own QoE requirement, the capacity-constrained fronthaul links will directly limit the data rate of the users that request contents from the cloud. Therefore, cache-enabled UAVs are introduced to service the mobile users along with terrestrial RRHs. Each cache-enabled UAV can store a limited number of popular contents that users request. By caching predicted content, the transmission delay from the content server to the UAVs can be significantly reduced as each UAV can directly transmit its stored content to the users.

In [171], the transmissions from the UAVs to the users are over mmWave frequencies. This is due to the fact that, the UAVs-to-users links have much less blocking effect than terrestrial RRHs-to-users links which makes mmWave suitable. The transmissions from the cloud to the UAVs occur over wireless fronthaul links using the licensed cellular band at sub-6 GHz. Consequently, the UAVs' wireless fronthaul links may interfere with the transmission links from the RRHs to the users.

A realistic model for periodic, daily, and pedestrian mobility patterns is considered according to which each user will regularly visit a certain location of interest. The QoE of each user is formally defined as function of each user's data rate, delay, and device type. The impact of the device type on the QoE is captured by the screen size of each device. The screen size will also

affect the QoE perception of the user, especially for video-oriented applications. The goal of [171] is to find an effective deployment of cache-enabled UAVs to satisfy the QoE requirement of each user while minimizing the transmit powers of the UAVs. This problem involves predicting, for each user, the content request distribution and the periodic locations, finding the optimal contents to cache at the UAVs, determining the users' associations, as well as adjusting the locations and transmit power of the UAVs. ANNs can be used to solve the prediction tasks due to their effectiveness in dealing with time-varying data (e.g., mobility data). Moreover, ANNs can extract the relationships between the users' locations and the users' context information such as gender, occupation, and age. In addition, ANN-based RL algorithms can find the relationship between the UAVs' location and the data rate of each user which enables UAVs to find the locations that maximize the users' data rates.

Consequently, a prediction algorithm using the framework of ESN with conceptors is developed to find the users' content request distributions and their mobility patterns. The predictions of the users' content request distribution and their mobility patterns will then be used to find the user-UAV association, optimal locations of the UAVs and content caching at the UAVs. Since the data of the users' behaviors such as mobility and content request are time related, an ESN-based approach previously discussed in Subsection III-A3 can quickly learn the mobility pattern and content request distributions without requiring significant training data. Conceptors defined in [172] enable an ESN to perform a large number of mobility and content request patterns predictions. Moreover, new patterns can be added to the reservoir of the ESN without interfering with previously acquired ones. The architecture of the conceptor ESN-based prediction approach is based on the ESN model specified in Subsection III-A3. For content request distribution prediction, the cloud's BBUs must implement one conceptor ESN algorithm for each user. The input is defined as each user's context that includes gender, occupation, age, and device type. The output is the prediction of a user's content request distribution. The generation of the reservoir is done as explained in Subsection III-A3. The conceptor is defined as a matrix that is used to control the learning of an ESN. For predicting mobility patterns, the input of the ESN-based algorithm is defined as the user's context and current location. The output is the prediction of a user's location in the next time slots. Ridge regression is used to train the ESNs. The conceptor is also defined as a matrix used to control the learning of an ESN. During the learning stage, the conceptor will record the learned mobility patterns and content request distribution patterns.
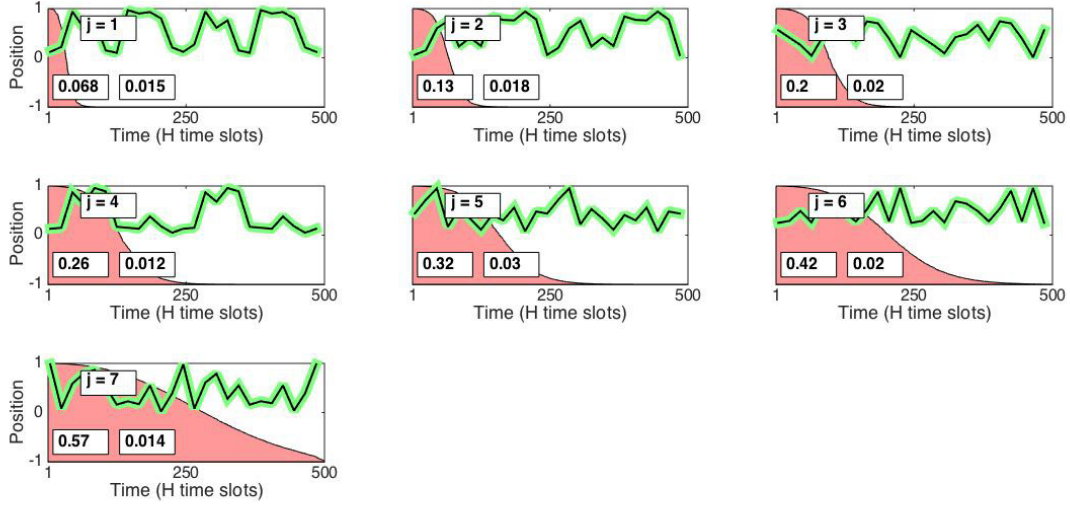
Fig. 13. Mobility patterns predictions of conceptor ESN algorithm from [171]. In this figure, the green curve represents the conceptor ESN prediction, the black curve is the real positions, top rectangle j is the index of the mobility pattern learned by ESN, the legend on the bottom left shows the total reservoir memory used by ESN and the legend on the bottom right shows the normalized root mean square error of each mobility pattern prediction.

When the conceptor ESN-based algorithm encounters a new input pattern, it will first determine whether this pattern has been learned. If this new pattern has been previously learned, the conceptor will instruct the ESN to directly ignore it. This can allow ESN to save some of its memory only for the unlearned patterns.

Based on the users' mobility pattern prediction, the BBUs can determine the users association using a K-mean clustering approach. By implementing a K-mean clustering approach, the users that are close to each other will be grouped into one cluster. In consequence, each UAV services one cluster and the user-UAV association will be determined. Then, based on the UAV association and each user's content request distribution, the optimal contents to cache at each UAV and the optimal UAVs' locations can be found. When the altitude of a UAV is much higher (lower) than the size of its corresponding coverage, the optimal location of the UAV can be found by [171, Theorems 2 and 3]. For more generic cases, it can be found by the ESN-based RL algorithm given in [173].

In Fig. 13, based on [171], we show how the memory of the conceptor ESN reservoir changes as the number of mobility patterns that were learned varies. Here, the mobility data is from *Beijing University of Posts and Telecommunications*. In Fig. 13, one mobility pattern represents the users' trajectory in one day and the colored region is the memory used by the ESN. Fig. 13
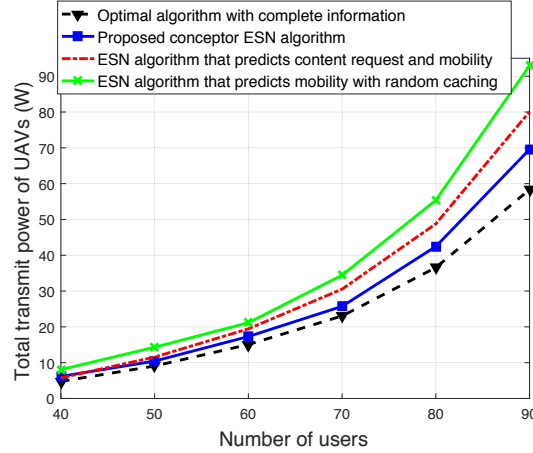
Fig. 14. Simulation result from [171] showing transmit power as the number of users varies.

shows that the usage of the memory increases as the number of the learned mobility patterns increases. Fig. 13 also shows that the conceptor ESN uses less memory for learning mobility pattern 2 compared to pattern 6. In fact, compared to pattern 6, mobility pattern 2 has more similarities to mobility pattern 1, and, hence, the conceptor ESN requires less memory to learn pattern 2. This is because the proposed approach can be used to only learn the difference between the learned mobility patterns and the new ones rather than to learn the entirety of every new pattern.

Fig. 14 shows how the total transmit power of the UAVs changes as the number of users varies. From Fig. 14, we can observe that the total UAV transmit power resulting from all algorithms increases with the number of users. This is due to the fact that the number of users associated with the RRHs and the capacity of the wireless fronthaul links are limited. Therefore, the UAVs must increase their their transmit power to satisfy the QoE requirement of each user. From Fig. 14, we can also see that the conceptor based ESN approach can reduce the total transmit power of the UAVs of about 16.7% compared to the ESN algorithm used to predict the content request and mobility for a network with 70 users. This is because the conceptor ESN that separates the users' behavior into multiple patterns and uses the conceptor to learn these patterns, can predict the users' behavior more accurately compared to the ESN algorithm.

From this example, we illustrated two key uses of ESN: mobility pattern prediction and content request distribution prediction. In particular, ESNs aere particularly effective when dealing with time-dependent users behaviors predictions. Based on the information of the predictions, UAVs

can move to optimal location and select the most popular content to store at the UAV cache in advance. Simulation results show that the ESNs can predict the users behaviors accurately and the proactive caching and UAVs movement can effectively service the mobile users.

4) Future Works: Clearly, ANNs are an important tool for addressing key challenges in UAV-based communication networks. In fact, different types of ANNs can be suitable for various UAV applications. For instance, given their effectiveness in dealing with time-dependent data, RNNs can be used for predicting the users' locations and traffic demand. This allows UAVs to optimize their location based on the dynamics of the network. Meanwhile, DNN-based RL algorithms can be used for resource management such as resource allocation, power control, interference management, specturm band selection, and user association. In particular, DNN-based RL algorithms can be used to find the relationship between the resource allocation schemes and their corresponding utility values. Subsequently, using DNN-based RL algorithms, the network can optimize the resource allocation to improve the network performance such as the users' throughput. Moreover, DNN-based RL algorithms can be used to determine the time duration that the UAVs need to service the ground users and how to service the ground users (e.g., stop or fly to service the users). Since DNNs have the ability to store large amount of data, DNN-based RL algorithms can also be used to store the data related to the users' historical context and, then, predict each ground user's locations, content requests, and latency requirement. Based on these predictions, the UAVs can find their optimal trajectory and, as a result, determine which area to serve at any given time. In addition, SNNs can be used for modeling the air-to ground channel, in general, and over mmWave frequencies, in particular. This is because SNNs are good at dealing with continuous data and the wireless channel is time-varing and continuous [174]. For instance, UAVs can use SNNs to analyze the data data that they can collect from the radio environment, such as the received signal strength, UAVs' positions, and users' positions, and then generate an air-to-ground channel model to fit the collected data. In summary, for UAV-based wireless communications, RNNs can be used for human behavior prediction, SNNs can be used for signal detection and channel modeling, and DNNs can be used for data classification and path planning. A summary of key problems that can be solved by using ANNs in the context of UAV-based communications is presented in Table II along with the challenges and future works.
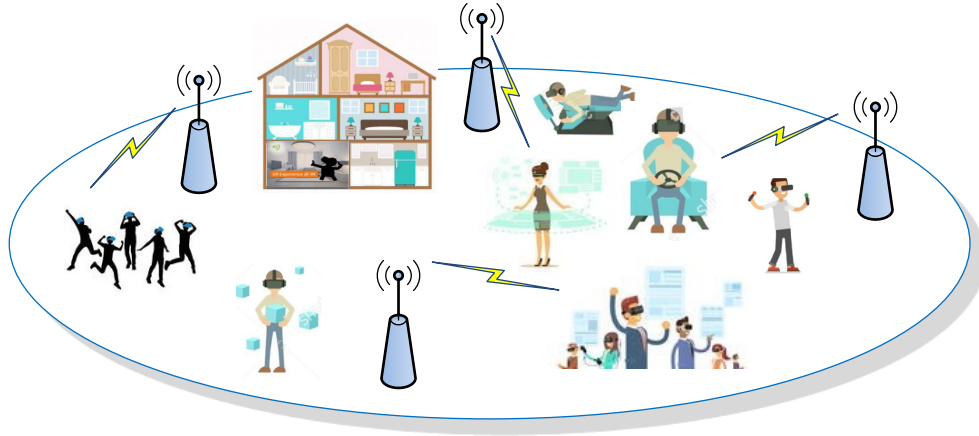
Fig. 15. Wireless VR networks.

C. Wireless Virtual Reality

1) Virtual Reality over Wireless Networks: Virtual reality is seen as one of the most promising new technologies that will enable users to experience and interact with a wealth of virtual and immersive environments through a first-person view. However, if VR devices such as HTC Vive [175] continue to use wired connections to a VR control center, such as a computer, for processing VR information, then the users will be significantly restricted in the type of actions that they can take and VR applications that they can experience. Moreover, the reliance on wired connectivity can lead to many breaks in presence for the VR users which can degrade the overall user experience. Therefore, there is a need to enable wireless VR technologies, as shown in Fig. 15, to allow better flexibility and more immersive user experience. In particular, when a VR device is operated over a wireless link, the users must send the tracking information that includes the users' locations and orientations to the BSs and, then, the BSs will use the tracking information to construct $360°$ images and send these images to the users. Therefore, for wireless VR applications, the uplink and downlink transmissions must be jointly considered. Moreover, in contrast to traditional video that consists of $120°$ images, a VR video consists of high-resolution $360°$ vision with three-dimensional surround stereo. This new type of VR video requires a much higher data rate than that of traditional mobile video. In addition, as VR images are constructed according to the the users' movement such as head movement or eye movement, the tracking accuracy of the VR system will directly affect the user experience. In summary, the challenges of operating VR devices over wireless networks [40] include tracking accuracy, low

delay, high data rate, user experience modeling, and effective image compression.

2) Neural Networks for Wireless Virtual Reality: The use of ANNs is a promising solution for a number of problems related to wireless VR. This is due to the fact that, compared to other applications such as UAV or caching, VR applications depend more on the users' environment, e.g., head and eye movements and their behavior vis-a-vis the VR environment. In particular, the VR environment (e.g., the corresponding images) is often constructed based on the users' locations, orientations, and eye movements. This environment must also change in response to the users' actions. ANNs are effective at identifying and predicting users' movements and their actions. Based on the predictions of the users' environment, actions, and movements, the BSs can improve the generation of the VR images and optimize the resource management for wireless VR users. ANNs have two major applications for wireless VR. First, ANNs can be used to predict the users' movement such as head movement and eye gaze movement, as well as their future interactions with the VR environment. For example, $360°$ videos provide users with a panoramic view and allow them to freely control their viewing direction. However, a user displays only the visible portion of a $360°$ video and, hence, transmitting the entire $360°$ video frame can waste the capacity-limited bandwidth. Since all images are constructed based on the users' movements, using ANNs, one can predict the users' movement such as locations, orientations, eye and head movements, and, hence, enable the wireless BSs to generate only the portion of the VR image that a user wants to display. Moreover, the predictions of users' movement can also improve the tracking accuracy of VR sensors. In particular, the BSs will jointly consider the users' movement predicted by ANNs and the users' movements collected by VR sensors to determine the users' movements.

Second, ANNs can be used to develop self-organizing algorithms to dynamically control and manage the wireless VR network thus addressing problems such as dynamic resource management. In particular, ANNs can be used for adaptively optimizing resource allocation and adjusting the quality and format of VR images according to the cellular network environment. In fact, ANNs can be used jointly for prediction and distributed optimization. For example, the same ANN can be used to predict the users' data rate requirements first and, then, it can be used as a reinforcement learning algorithm that optimizes the allocation of resources to each VR user.

However, using ANNs for VR faces many challenges. First, in wireless VR networks, the

data collected from the users may contain errors that are unknown to the BSs. In consequence, the BSs may need to use erroneous data to train the ANNs and, hence, the prediction accuracy of the ANN will be significantly affected. Second, due to the large data size of each $360°$ VR image, the BSs must spend a large amount of computational resources to process VR images. Meanwhile, the training of ANNs will also require a large amount of computational resources. In consequence, how to effectively allocate the computational resources for processing VR images and training neural networks is an important challenge. In addition, VR applications require ultra-low latency while the training of ANNs can be time consuming. Hence, how to effectively train ANNs in a limited time is an important question for wireless VR. In this regard, training the ANNs in an offline manner or using ANNs that converge quickly can be two promising solutions for speeding up the training process of ANNs .

• *Related works:* The existing literature has studied a number of problems related to using ANNs for VR such as in [176]–[183]. In [176] and [177], the authors investigated the use of different ANNs for recognition of hand gestures in VR applications. The authors in [178] developed an algorithm that uses ANNs to enable users to perform interactive shape changes and view the resulting deformation changes in a virtual environment. In [179], a deep neural network based algorithm is proposed to automatically convert 2D videos and images to a stereoscopic 3D format. The work in [180] proposed a linear regression learning algorithm to predict the users' head movement in VR applications. In [181], a decision forest learning algorithm is proposed for gaze prediction. The work in [183] developed a neural network based transfer learning algorithm for data correlation aware resource allocation. In essence, the existing VR literature such as [176]–[183] has used ANNs to solve a number of VR problems such as hand gestures recognition, interactive shape changes, video conversion, head movement prediction, and resource allocation. However, with the exception of our work in [182], all of the other works that use ANNs for VR applications are focused on wired VR. Therefore, they do not consider the challenges of wireless VR such as scarce spectrum resources, limited data rates, and how to transmit the tracking data accurately and reliably. In fact, ANNs can be used for wireless VR to solve the problems such as users movement prediction, computational resource allocation, spectrum management, and VR image generation. Next, a specific ANNs' application for VR over wireless network is introduced.

3) Example: One key application of using ANNs for wireless VR systems is presented in [182] for the study of resource allocation in cellular networks that support VR users. In this model, BSs act as the VR control centers that collect the tracking information from VR users over the cellular uplink and then send the generated images (based on the tracking information) and accompanying surround stereo audio to the VR users over the downlink. Therefore, this resource allocation problem in wireless VR must jointly consider both the uplink and downlink transmissions. To capture the VR users' QoS in a cellular network, the model in [182] jointly accounts for VR tracking accuracy, processing delay, and transmission delay. The tracking accuracy is defined as the difference between the tracking vector transmitted wirelessly from the VR headset to the BS and the accurate tracking vector obtained from the users' force feedback. The tracking vector represents the users' positions and orientations. The transmission delay consists of the uplink transmission delay and the downlink transmission delay. The uplink transmission delay represents the time that a BS uses to receive the tracking information while the downlink transmission delay is the time that a BS uses to transmit the VR contents. The processing delay is defined as the time that a BS spends to correct the VR image from the image constructed based on the inaccurate tracking vector to the image constructed according to the accurate tracking vector. In [182], the relationship between the delay and tracking is not necessarily linear nor independent and, thus, multi-attribute utility theory [184] is used to construct a utility function assigns a unique value to each tracking and delay components of the VR QoS.

The goal of [182] is to develop an effective resource blocks allocation scheme to maximize the users' utility function that captures the VR QoS. This maximization jointly considers the coupled problems of user association, uplink resource allocation, and downlink resource allocation. Moreover, the VR QoS of each BS depends not only on its resource allocation scheme but also on the resource allocation decisions of other BSs. Consequently, the use of centralized optimization for such a complex problem is largely intractable and yields significant overhead. In addition, for VR resource allocation problems, we must jointly consider both uplink and downlink resource allocation, and, thus, the number of actions will be much larger than conventional scenarios that consider only uplink or downlink resource allocation. Therefore, as the number of actions significantly increases, each BS may not be able to collect all information needed to calculate the average utility function.

To overcome these challenges, an ANN-based RL algorithm can be used for self-organizing VR

resource allocation. In particular, an ANN-based RL algorithm can find the relationship between the users association, resource allocation, and users' data rates, and, then, it can, directly select the optimal resource allocation scheme after the training process. Compared to traditional RL algorithms such as Q learning that use tables to record a given agent's utility values and actions, an ANN-based RL algorithm uses its hidden neurons, such as the dynamic reservoir mentioned in Section III, to approximate the relationships between the utility values and actions, which can significantly speed up the training process. Therefore, for the downlink and uplink resource allocation problem in [182], an ANN-based RL algorithm can use less exploration time to build the relationship between the actions and their corresponding utilities and then optimize resource allocation.

To simplify the generation and training process of an ANN-based RL algorithm, an ESN-based RL algorithm is selected for VR resource allocation in [182]. The ESN-based learning algorithm enables each BS to predict the value of VR QoS resulting from each resource allocation scheme without having to traverse all resource allocation schemes. The architecture of the ESN-based self-organizing approach is based on the ESN model specified in Subsection III-A3. To use ESNs for RL, each row of the ESN's output weight matrix is defined as one action. Here, one action represents one type of resource allocation. The input of each ESN is the current strategies of the action selection of all BSs. The generation of the reservoir follows Subsection III-A3. The output is the estimated utility value. In the learning process, at each time slot, each BS will implement one action according to the current action selection strategy. After the BSs perform their selected actions, they can get the actual utility values. Based on the actual utility values and the utility values estimated by ESN, each BS can adjust the values of the output weight matrix of an ESN according to (10). As time elapses, the ESN can accurately estimate the utility values for each BS and can therefore find the relationship between the resource allocation schemes and the utility values. Based on this relationship, each BS can find the optimal action selection strategy that maximizes the average VR QoS for its users.

Using the simulation results of [182], Fig. 16 shows how the average delay of each user varies as the number of BSs changes. From Fig. 16, we can see that, as the number of BSs increases, the transmission delay for each served user increases. This is due to the fact that, as the number of BSs increases, the number of users located in each BS's coverage decreases and, hence, the average delay increases. However, as the number of BSs increases, the delay increase
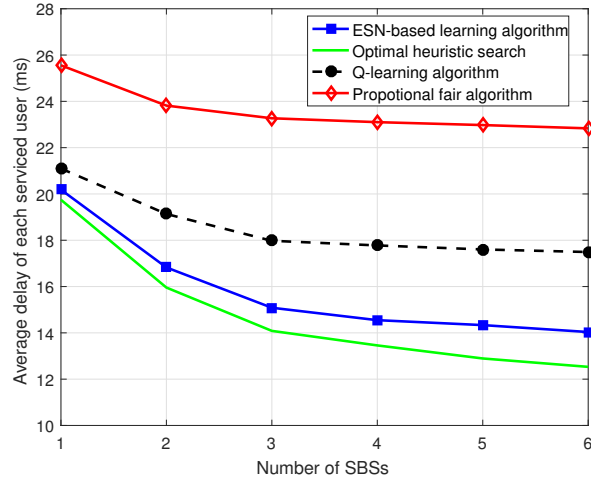
Fig. 16. Delay for each serviced user vs. the number of BSs.
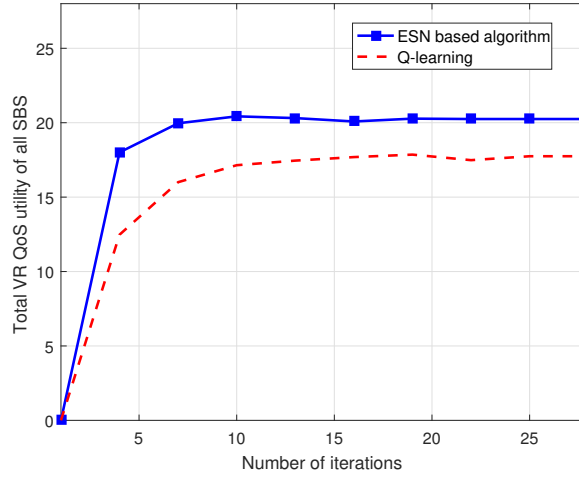

Fig. 17. Convergence of the proposed algorithm and Q-learning.

becomes slower due to the additional interference. This stems from the fact that, as the number of BSs continues to increase, the number of the users associated with each BS decreases and more spectrum will be allocated to each user. Hence, the delay of each user will continue to decrease. However, as the number of the BSs increases, the increasing interference will limit the reduction in the delay. Fig. 16 also shows that the ESN-based algorithm achieves up to 19.6% gain in terms of average delay compared to the Q-learning algorithm for the case with 6 BSs. Fig. 16 also shows that the ESN-based approach allows the wireless VR transmission to meet the VR delay requirement that includes both the transmission and processing delay (typically 20

ms [185]). These gains stem from the adaptive nature of ESNs. Fig. 17 shows the number of iterations needed till convergence for both the ESN-based learning approach and Q-learning. In this figure, we can see that, as time elapses, the total VR QoS utilities for both the ESN-based algorithm and Q-learning increase until convergence to their final values.

Fig. 17 also shows that the ESN-based algorithm needs 19 iterations to converge while Q-learning needs 25 iterations to converge. Hence, the ESN-based learning algorithm achieves 24.2% gain in terms of the number of the iterations needed to reach convergence compared to Q-learning. This is due to the fact that ESN can store the BSs' actions and their corresponding total utility values.

From this example, we illustrated the use of ESN as an RL algorithm for self-organizing resource allocation in wireless VR. An ESN-based RL algorithm enables each BS to allocate downlink and uplink spectrum resource in a self-organizing manner that adjusts the resource allocation according to the dynamical environment. Moreover, an ESN-based RL algorithm can use an approximation method to find the relationship between each BS's actions and its corresponding utility values, and, hence, an ESN-based RL algorithm can speed up the training process. Simulation results show that an ESN-based RL algorithm enables each BS to achieve the delay requirement of VR transmission.

4) Future Works: Clearly, ANNs are a promising tool to address challenges in wireless VR applications. In fact, the above application of ANNs for spectrum resource allocation can be easily extended to manage other types of resources such as transmit power, computational resources, and video formats. In particular, DNNs are a good choice for implementing RL algorithms to to manage resources in a wireless VR network. This is due to the fact that DNNs have multiple hidden layers that can store more actions and utility values compared to other ANNs. Hence, DNNs can use stored information to find the relationship between the actions and utility values. Moreover, RNNs can be used to predict and detect the VR users' movement such as eye movement and head movement and their interactions with the environment. Then, the network can pre-construct VR images based on these predictions which can reduce the time spent to construct the VR images. The user-VR system interactions are all time-dependent and, hence, RNNs are a good choice for performing such tasks. Note that, the prediction of the users' movement will directly affect the VR images that are sent to the users at each time slot and, hence, the learning algorithm must complete the training process during a short time period. In consequence, we
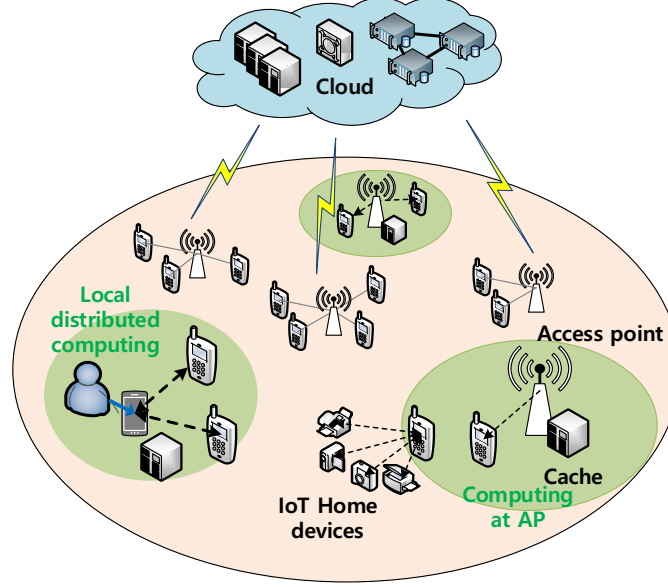
Fig. 18. Mobile edge caching and computing wireless networks.

should use RNNs that are easy to train for the prediction of the users' movement. Finally, CNNs can be used for VR video compression, recovery, coding, and decoding so as to reduce the data size of each transmitted VR video and improve the QoS for each VR user. This is because CNNs are good at storing large amount of data in spatial domain and learn the features of VR images. A summary of key problems that can be solved by using ANNs in wireless VR system is presented in Table II along with the challenges and future works.

D. Mobile Edge Caching and Computing

1) Mobile Edge Caching and Computing: Due to the drastic increase in the number of devices connected via wireless networks, mobile traffic is expected to increase by 60%. The majority of this mobile traffic will stem from video and streaming applications [186]. However, for current cellular networks, the centralized information management and the limited-capacity cellular transmission links cannot handle such a substantial growth in wireless data traffic, which motivates the use of mobile edge caching and computing technologies [187], as shown in Fig. 18. Caching at the edge of wireless networks enables the network devices (BSs and end-user devices) to store the most popular contents to reduce the data traffic (content transmission), delay, and bandwidth usage, as well as to improve the energy efficiency and the utilization of users' context information and social information. It has recently become possible to jointly

consider cache placement and content delivery, using coded caching. Coded caching enables network devices to create multicasting opportunities for a certain content via coded multicast transmissions thus can significantly improving bandwidth efficiency [188]. However, designing effective caching strategies for wireless networks faces many challenges that include optimized cache placement, cache update, and content popularity analytics.

In addition to caching, the wireless network's edge devices can be used for performing effective, low-latency computations, using the emerging paradigm of *mobile edge computing* [189]. The basic premise of mobile edge computing is to exploit local resources for computational purposes (e.g., for VR image generation or sensor data processing), in order to avoid high-latency transmission to remote cloud servers. Mobile edge computing, which includes related concepts such as fog computing [190], can reduce the overall computational latency by reducing the reliance on the remote cloud while effectively offloading computational resources across multiple local and remote devices. The key challenge in mobile edge computing is to optimally allocate computational tasks across both edge devices (e.g., fog nodes) and the remote data servers, in a way to optimize latency. Finally, it is worth noting that some recent works [191] have jointly combined caching and computing. In this case, caching is used to store the most popular and basic computational tasks. Based on the caching results, the network will have to determine the optimal computational resource allocation to globally minimize latency. However, optimizing mobile edge computing faces many challenges such as computing placement, computational resource allocation, computing tasks assignment, end-to-end latency minimization, and minimization of the energy consumption for the devices.

2) Neural Networks for Mobile Edge Caching and Computing: ANNs can play a central role in the design of new mobile edge caching and computing mechanisms. For instance, the problems of optimal cache placement and cache update are all dependent on the predictions of the users' behaviors such as users' content request. For example, the cache placement depends on the users' locations while the cache update depends on the frequency with which a user requests a certain content. Human behavior is highly predictable by ANNs and, hence, ANNs are a promising solution for effective mobile edge caching and computing.

In essence, ANNs have three major applications for mobile edge caching and computing. First, ANNs can be used for prediction and inference purposes. For example, ANNs can be used to predict the users' content request distributions, mobility patterns, and content request

frequency. The content request distribution and content request frequency can be used to determine which contents to store at the end-user devices or BSs. The mobility patterns can be used to determine the users' association and, hence, affect the cache replacement and the stored contents. Furthermore, ANNs can also be used to find social information from the collected data. In particular, ANNs can learn the users' interests, activities, and their interactions. By exploiting the correlation between users' data, their social interests, and their common interests, the accuracy of predicting future events such as users' geographic locations, next visited cells, and requested contents can be dramatically improved [162]. For example, ANNs can be used to predict the users' interests. The users that have the same interests are highly likely to request the same content. Therefore, the system operator can cluster the users that have the same interests and store the popular contents they may request. Similarly, ANNs can be used to predict the computational requirements of tasks. The prediction of the computational requirements of tasks enables the network devices to schedule the computational resources in advance to minimize the global latency.

Second, ANNs can be used as an effective clustering algorithm to classify the users based on their activities such as content request, which enables the system operator to determine which contents to store at a storage unit and improve the usage of cached contents. For instance, the content requests of users can change over time while the cached content will be updated for a long time (i.g., one day) and, hence, the system operator must determine which contents to cache by reviewing all the collected content requests. ANNs such as CNNs can be used to store the content request information and classify the large amount of content requests for cache update. In fact, predictions and clustering are interrelated and therefore ANNs can be used for both applications simultaneously. For example, ANNs can first be used to predict the users' content request distributions, and then, ANNs can be used to classify the users that have the similar content request distributions. Meanwhile, ANN-based clustering algorithm can also be used to classify the computing tasks. Then, the computing tasks that are clustered into a group will be assigned to a certain computing center. In this case, each computing center will process one type of computing tasks, which will reduce the time consumed by computation. Finally, ANNs can also be used for intelligently scheduling the computing tasks to different computing centers. In particular, ANNs can be used as an RL algorithm to learn each computing center's state such as its computational load, and then, allocate computing tasks based on the learned information

to reduce the time consumed by computation.

However, using ANNs for mobile edge caching and computing faces many challenges. Data cleaning is an essential part of the data analysis process for mobile edge processing. For example, to predict the users' content requests, the data processing system should be capable of reading and extracting useful data from huge and disparate data sources. For example, one user's content request depends on this user's age, job, and locations. The data processing system should be able to extract the useful data to train the ANN. In fact, the data cleaning process usually takes more time than the learning process. For example, the number of contents that users may request can be in the order of millions and, hence, the data processing system should select appropriate contents to analyze and predict the users' content request behaviors. For caching, the most important use of ANNs is to predict the users' content requests which directly determines the caching update. However, each user can request a large amount of types of contents such as video, music, and news. For each type of contents, it may have different formats and different resolutions. Hence, for each user, the total number of the requested contents will be significantly large. However, the memory of an ANN is limited and, hence, each ANN can record only a limited number of requested contents. In consequence, an ANN must be able to select the most important contents for content request prediction so as to help the network operator to determine which contents to store at mobile edge cache. Similarly, for computing tasks predictions, limited-memory ANNs can only store a finite number of the computing tasks and, hence, they must select suitable computing tasks to store and predict. Moreover, compared to mobile edge caching that can update cache content in a period, mobile edge computing needs to process the tasks as soon as possible. Therefore, the ANNs used for mobile edge computing must complete their training process in a short time.

• *Related works:* The existing literature has studied a number of problems related to using neural networks for caching such as in [98], [171], [192]–[196]. The authors in [192] proposed a big data-enabled architecture to investigate proactive content caching in 5G wireless networks. In [193] and [194], neural networks are used to determine the cache replacement. The authors in [195] used neural networks for the predictions of the web pages that are most likely to be re-accessed. The authors in [196] developed a data extraction method using the Hadoop platform to predict content popularity. The works in [171] and [98] developed an ESN-based learning algorithm to predict the users' mobility patterns and content request distributions.

In general, existing works such as in [98], [171], [192]–[196] have used ANNs to solve the caching problems such as cache replacement, content popularity prediction, and content request distribution prediction. However, ANNs can also be used to analyze the content correlation in the temporal and spatial domains. Meanwhile, ANN-based algorithms can be used to optimize the content transmission and users' clustering. For mobile edge computing, in general, there is no existing work that uses ANNs to solve the relevant problems. In fact, ANNs can be used to treat mobile edge computing problems such as prediction of the computational resource demand, computational resource allocation, and estimation of the computing time for a task. Next, we introduce a specific ANNs' application for mobile edge caching.

3) Example: One illustrative application for the use of ANNs for mobile edge caching is presented in [98] which studies the problem of proactive caching in CRANs. In this model, users are served by the RRHs which are connected to the cloud pool of the BBUs via capacity-constrained wired fronthaul links. The RRHs and users are all equipped with storage units that can be used to store the most popular contents that users request. RRHs which have the same content request distributions are grouped into a virtual cluster and serve their users using zero-forcing dirty paper coding. The content request distribution for a particular user represents the probabilities with which the user requests different contents. Virtual clusters are connected to the content servers via capacity-constrained wired backhaul links. The total transmission rate of backhaul (fronthaul) links is equally allocated to the contents that must be transmitted over the backhaul (fronthaul) links. Each user has a periodic mobility pattern and will regularly visit a certain location. Since cache-enabled RRHs and BBUs can store the requested contents, can be done over four possible links: a) content server-BBUs-RRH-user, b) cloud cache-BBUs-RRH-user, c) RRH cache-RRH-user, and d) remote RRH cache-remote RRH-BBUs-RRH-user. We used the notion of *effective capacity* [197] to capture the maximum content transmission rate of a channel under a certain QoS requirement. The effective capacity is a link-layer channel model that can be used to measure a content transmission over multiple hops. In particular, the effective capacity can be used to measure a content transmission from the BBUs to the RRHs, then from RRHs to the users. The effective capacity of each content transmission depends on the link that is used to transmit the content and the actual link capacity between the user and associated RRHs.

The goal of [98] is to develop an effective framework for content caching and RRH clustering

in an effort to reduce the network's interference and offload the traffic of the backhaul and fronthaul based on the predictions of the users' content request distributions and mobility patterns. To achieve this goal, a QoS and delay optimization problem is formulated, whose objective is to maximize the long-term sum effective capacity. This optimization problem involves predicting the content request distribution and periodic location for each user, and finding optimal contents to cache at the BBUs and RRHs. To predict the content request distribution and mobility patterns for each user, an ESN-based learning algorithm is used similar to the one described in Subsection III-A3. For each user, the BBUs must implement one ESN algorithm for content request distribution prediction and another ESN algorithm for mobility pattern prediction.

For content request distribution prediction, the input of the developed ESN is a user's context which includes content request time, week, gender, occupation, age, and device type. The output is the predicted content request distribution. The ESN model consists of the input weight matrix, output weight matrix, and the recurrent weight matrix that have been explained earlier. A linear gradient descent approach is used to train the output weight matrix. For mobility pattern prediction, the input of the developed ESN is current location of each user and the output is the vector of locations that a user is predicted to visit for the next steps. In contrast to the recurrent matrix that is a sparse matrix and generated randomly, as explained in Subsection III-A3, the recurrent matrix of the ESN used for mobility prediction contains only W non-zero elements where W is the dimension of the recurrent matrix. This simplified recurrent matrix can speed up the training process of ESNs. An offline manner using ridge regression is used to train the output weight matrix.

Based on the users' content request distribution and locations, the cloud can estimate the users' RRH association, determine each RRH's content request distribution, and, then, cluster the RRHs into several groups. The content request distribution that is used to determine the contents stored at RRH cache is referred to RRH content request distribution. Similarly, the content request distribution that is used to determine the contents stored at the cloud's cache is referred to cloud content request distribution. To determine the contents that must be cached at the cloud, the cloud needs to calculate the cloud content request distribution of each user. To calculate this cloud content request distribution for each user, the BBUs must record all of the RRH content request distributions within a given time period. However, the BBUs cannot record the RRH content request distributions for all of the users as this will lead to a massive amount
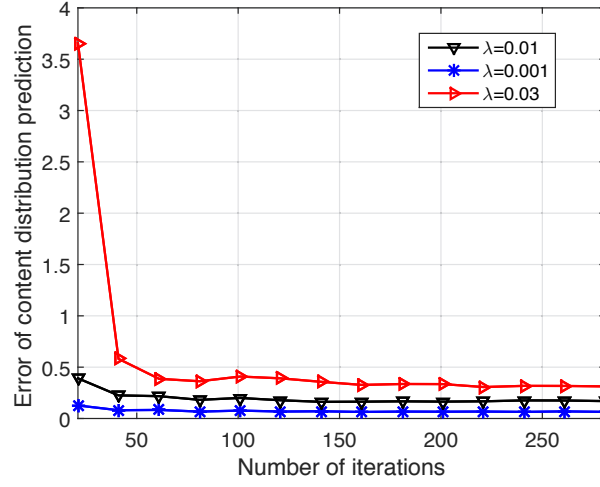
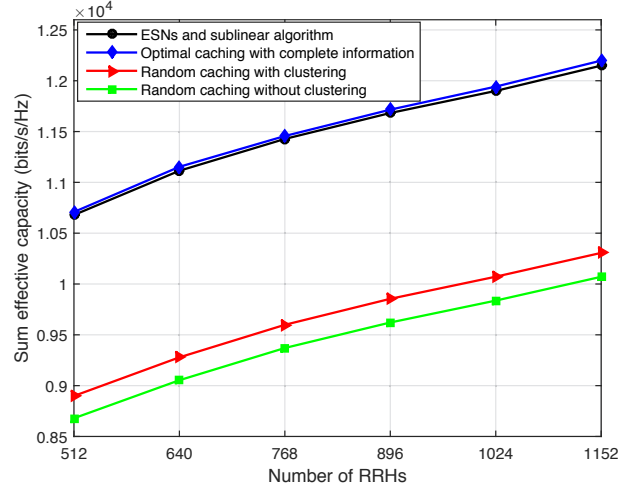Fig. 19. Prediction errors as the number of iteration varies.

Fig. 20. Sum effective capacity as function of the number of RRHs.

of data. Therefore, a sublinear approach [198] is proposed to determine the contents stored at cloud cache. Our result in [98, Theorem 2] proves that the ESN-based algorithm will reach an optimal solution to the content caching problem.

Using the simulation results of [98], in Fig. 19, we show how the error of the ESN-based estimation changes as the number of iterations varies. In Fig. 19, λ is the learning rate of the ESN-based algorithm. From Fig. 19, we can see that, as the number of iterations increases, the error of the ESN-based estimation decreases. Fig. 19 also shows that the ESN approach requires less than 50 iterations to estimate the content request distribution for each user because ESNs

need to only train the output weight matrix, as explained in Subsection III-A3. Fig. 19 also shows that, for all learning rates, an ESN-based algorithm yields a very accurate prediction, with an error that does not exceed 0.43% for $\lambda = 0.03$. Fig. 20 shows how the sum of the effective capacities of all users in a period changes with the number of RRHs. From Fig. 20, we can see that, as the number of the RRHs increases, the effective capacities of all algorithms increase as the users become closer to their RRHs. From Fig. 20, we can also see that the ESN approach can yield up to 21.6% and 24.4% of improvements in the effective capacity compared to random caching with clustering and random caching without clustering, respectively, for a network with 512 RRHs. This is due to the fact that the ESN-based algorithm can effectively use the predictions of the ESNs to determine which content to cache.

This example illustrated the use of ESN for content request distribution prediction and mobility patterns prediction. An ESN-based learning algorithm can quickly learn the users mobility patterns and content requests by adjusting only the output weight matrix. Moreover, it enables the BBUs to quickly obtain the locations of the users and their content request distributions due to its simple training process. Based on the predictions, the BBUs can determine the user association and the contents that must be cached in advance.

4) Future Works: Clearly, ANNs will be an important tool for solving challenges in mobile edge caching and computing applications, especially for content request prediction and computing tasks prediction. In fact, CNNs, that are good at storing voluminous data in spatial domains, can be used to investigate the content correlation in the spatial domains. Based on the content correlation, each BS can store the contents that are the most related to other contents to improve the caching efficiency and hit ratio. Moreover, RNNs can be used as a self-organizing RL algorithms to allocate computational resources. RNNs are suitable here because they can record the utility values resulting from different computational resources allocation schemes as time elapses. Then, the RNN-based RL algorithms can find the optimal computational resource allocation after several implementations. Meanwhile, in contrast to the user association in cellular network that each user can only associate with one BS, one computing task can be assigned to several computing centers and one computing center can process different computing tasks. Therefore, the problem of computing task assignment is a many-to-many matching problem [199]. RNN-based RL algorithms can also be used to solve the computing task assignment problem due to their advantages in analyzing historical data pertaining to past assignments of computing tasks.
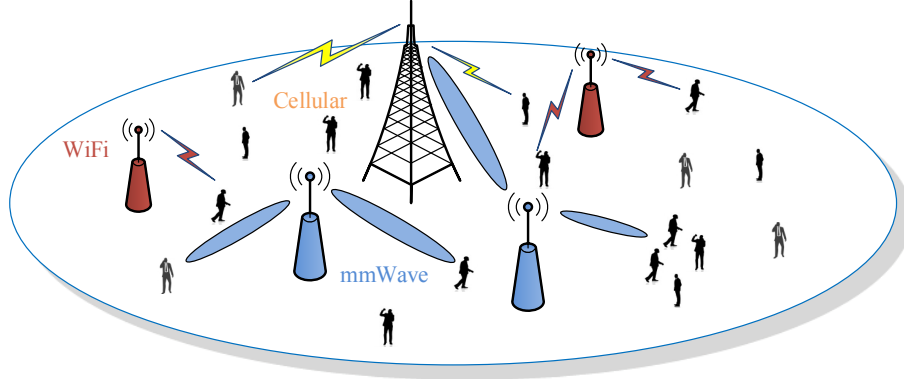
Fig. 21. Multiple Radio Access Networks.

In addition, DNN-based RL algorithm can be used to jointly optimize the cache replacement and content delivery. To achieve this purpose, each action of the DNN-based RL algorithm must contain one content delivery method as well as one cache update scheme. This is because DNNs are good at storing large amounts of utility values resulting from different content delivery and cache update schemes. Last but not as least, RNNs can also be used to predict the computational resource demands for each user due to their advantages in dealing with time-dependent data. A summary of key problems that can be solved by using ANNs in wireless VR system is presented in Table II along with the challenges and future works.

E. Spectrum Management and Co-existence of Multiple Radio Access Technologies (Multi-RAT)

1) Co-existence of Multiple Radio Access Technologies: To cope with the unprecedented increase in mobile data traffic and realize the envisioned 5G services, significant enhancement of per-user throughput and overall system capacity is required [200]. Such an enhancement can be achieved through advanced PHY/MAC/network technologies and efficient methods of spectrum management. In fact, one of the main advancements in the network design for 5G networks relies on the integration of multiple different radio access technologies (RATs), as shown in Fig. 21. With multi-RAT integration, a mobile device can potentially transmit data over multiple radio interfaces such as LTE and WiFi, at the same time, thus improving its performance [201]. Moreover, a multi-RAT network allows fast handover between different RATs and, thus, providing seamless mobility experience for users. Therefore, the integration of different RATs results in an improvement in the overall utilization of the available radio resources and, thus, an increase in the system's capacity. It also guarantees a consistent service experience for

different users irrespective of the served RAT and facilitates the management of the resulting network.

Spectrum management is also regarded as another key component of 5G networks. Unlike early generations of cellular networks that operate exclusively on the sub-6 GHz (microwave) licensed band, 5G networks are expected to transmit over the conventional sub-6 GHz band, unlicensed spectrum and the 60 GHz mmWave frequency band [202]–[204]. We note that, on the other hand, the classical LTE microwave licensed band is reliable, however, limited and hence is a scarce resource. On the other hand, the unlicensed bands can be used to serve best effort traffic only since the operation over this spectrum should account for the presence of other coexisting technologies. The mmWave spectrum has a large amount of available bandwidth and can provide multi-gigabit communication services, however, the uncertainty and dynamic channel variations of the mmWave band due to blockage make it unreliable. Therefore, a multi-mode BS operating over the licensed, unlicensed, and mmWave frequency bands can exploit the different characteristics and availability of the frequency bands thus providing robust and reliable communication links for the end users [204]. However, to reap the benefits of multi-mode BSs, spectrum sharing is crucial. A particular example of spectrum sharing is LTE in unlicensed spectrum (LTE-U) or licensed-assisted access using LTE (LTE-LAA) [205]. The goal of LTE-U is to allow the opportunistic transmission of LTE over the unlicensed bands. It will leverage unlicensed spectrum as a complement to licensed spectrum to offload best-effort traffic through the carrier aggregation framework, while critical control signalling, mobility, voice and control data will always be transmitted on the licensed band. Nevertheless, the problems of traffic balancing across the licensed and unlicensed bands [206], coexistence of LTE with WiFi [173], [207], and coexistence of different LTE-U operators need to be addressed.

2) Neural Networks for Spectrum Management and Multi-RAT: ANNs are regarded as an attractive solution approach for tackling various challenges that arise in multi-RAT scenarios. To leverage the advantage of such multi-RAT networks, ANNs can allow the smart use of different RATs wherein a BS can learn when to transmit on each type of frequency band based on the network conditions. For instance, ANNs allow multi-mode BSs to steer their traffic flows between the mmWave, the microwave, and the unlicensed band based on the availability of a LoS link, congestion on the licensed band and the availability of the unlicensed band [206]. Moreover, in LTE-WiFi link aggregation (LWA) scenarios, ANNs allow cellular devices to learn when to

operate on each band or utilize both links simultaneously. For instance, delay-tolerant data traffic can be offloaded to wireless local area network (WLAN) in case of congestion on the licensed band.

Moreover, ANNs can provide multi-mode BSs with the ability to learn the appropriate resource management over different RATs or spectrum bands in an online manner and thus offering an autonomous and self-organizing operation with no explicit communication among different BSs, once deployed. For instance, ANNs can be trained over large datasets which take into account the variations of the traffic load over several days for scenarios in which the traffic load of WiFi access points (WAPs) can be characterized based on a particular traffic model [208]. Here, note that cellular data traffic networks exhibit statistically fluctuating and periodic demand patterns, especially for applications such as file transfer, video streaming, and browsing [208]. ANNs can also accommodate the users' mobility patterns to predict the availability of a LoS link thus allowing the transmission over the mmWave band. In particular, they can be trained to learn the antenna tilting angle based on the environment changes in order to guarantee a LoS communication link with the users and, thus, enabling an efficient communication over the mmWave spectrum. Moreover, ANNs enable multiple BSs to learn how to form multi-hop, mmWave links over backhaul infrastructure, while properly allocating resources across those links in an autonomous manner [165], [209]. To cope with the changes in the traffic model and/or users' mobility pattern, ANNs can be combined with online machine learning [210] by properly re-training the weights of the developed learning mechanisms. Multi-mode BSs can thus learn the traffic pattern over time and thus predict the future channel availability status. With proper network design, ANNs can allow operators to improve their network's performance by reducing the probability of occurrence of a congestion event while ensuring a degree of fairness to other technologies in the network, such as WLAN.

A proactive resource management of the radio spectrum for multi-mode BSs can also be achieved using ANNs. In a *proactive* approach, rather than reactively responding to incoming demands and serving them when requested, multi-mode BSs can predict traffic patterns and determine future off-peak times on different spectrum bands so that incoming traffic demand can be properly allocated over a given time window. In an LTE-U system, for instance, a proactive coexistence mechanism enables future delay-intolerant data demands to be served within a given prediction window ahead of their actual arrival time thus avoiding the underutilization of the

unlicensed spectrum during off-peak hours [207]. This will also lead to an increase in the LTE-U transmission opportunity as well as a decrease in the collision probability with WAPs and other BSs in the network.

• *Related works:* Several existing works have adopted different learning techniques in order to tackle various challenges that arise in multi-RAT networks [173], [211]–[216]. The authors in [211] propose a fuzzy-neural system for resource management among different access networks. In [212], the authors propose a cross-system learning framework, based on Boltzmann-Gibbs learning, in order to optimize the long-term performance of multi-mode BSs, by steering delay-tolerant traffic towards WiFi. A reinforcement learning approach was proposed in [213] to deal with the problem of RAT selection. The authors in [214] propose a supervised neural network approach, based on a multi layer perceptron (MLP), for the classification of the users' transmission technology in a multi-RAT system. In [215], the authors propose a hopfield neural network scheme for multi-radio packet scheduling. The problem of resource allocation with uplink-downlink decoupling in an LTE-U system has been investigated in [173] in which the authors propose a decentralized scheme based on ESN. The authors in [216] propose a distributed approach based on Q-learning for the problem of channel selection in an LTE-U system. Nevertheless, this prior work [173], [211]–[216] consider a reactive approach in which data requests are first initiated and, then, resources are allocated based on their corresponding delay tolerance value. In particular, existing works do not consider the predictable behavior of the traffic and thus do not account for future off-peak times during which data traffic can be steered among different RATs.

Here, note that, ANNs are suitable for learning the data traffic variations over time and thus predicting the future traffic load. In particular, LSTM cells are capable of storing information for long periods of time and hence learning the long-term dependency within a given sequence. Predictions at a given time step are influenced by the network activations at previous time steps thus making LSTMs an attractive solution for proactively allocating the available resources in multi-RAT systems. In what follows, we summarize our work in [217], in which we developed a deep reinforcement learning scheme, based on LSTM memory cells, for allocating the resources in an LTE-U network over a fixed time window T.

3) Example: An interesting application of DNNs in the context of LTE-U and WiFi coexistence is presented in [217]. The work in [217] considers a network composed of several LTE-U

BSs belonging to different LTE operators, several WAPs and a set of unlicensed channels on which LTE-U BSs and WAPs can operate on. The LTE carrier aggregation feature, using which the BSs can aggregate up to five component carriers belonging to the same or different operating frequency bands, is adopted. We consider a time domain divided into multiple time windows of duration T, each of which consisting of multiple time epochs t. Our objective is to proactively determine the spectrum allocation vector for each BS at $t = 0$ over T while guaranteeing long-term equal weighted airtime share with WLAN. In particular, each BS learns its channel selection, carrier aggregation, and fractional spectrum access over T while ensuring long-term airtime fairness with WLAN and other LTE-U operators. A contention-based protocol is used for channel access over the unlicensed band. The exponential backoff scheme is adopted for WiFi while the BSs adjust their CW size (and thus the channel access probability) on each of the selected channels based on the network traffic conditions while also guaranteeing a long-term equal weighted fairness with WLAN and other BSs.

The proactive resource allocation scheme in [217] is formulated as a noncooperative game in which the players are the BSs. Each BS must choose which channels to transmit on along with the corresponding channel access probabilities at $t = 0$ for each t of the next time window T. This, in turn, allows BSs to determine future off-peak hours of WLAN on each of the unlicensed channels thus transmitting on the less congested channels. Each BS can therefore maximize its total throughput over the set of selected channels over T while guaranteeing long-term equal weighted fairness with WLAN and other BSs. To solve the formulated game, a DNN framework based on LSTM cells was used. To allow a sequence-to-sequence mapping, we considered an encoder-decoder model as described in Section III-C. In this model, the encoder network maps an input sequence to a vector of a fixed dimensionality and then the decoder network decodes the target sequence from the vector. In this scheme, the input of the encoder is essentially a time series representation of the historical traffic load of the BSs and WAPs on all the unlicensed channels. The learned vector representation is then fed into a multi-layer perceptron (MLP) that summarizes the input vectors into one vector, thus accounting for the dependency among all the input time series vectors. The output of the MLP is then fed into different separate decoders which allows each BS to reconstruct its predicted action sequence over T.

To train the proposed network, the REINFORCE algorithm in [218] is used to compute the gradient of the expected reward with respect to the policy parameters, and the standard gradient
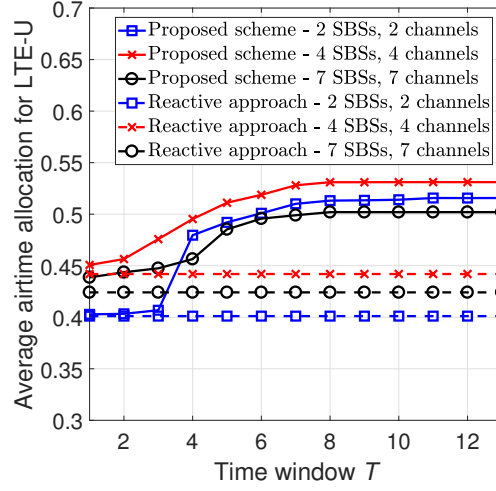
Fig. 22. The average throughput gain for LTE-U upon applying a proactive approach (with varying T) as compared to a reactive approach [217].

descent optimization algorithm [219] described in Subsection II-C is adopted to allow the model to generate optimal action sequences for input history traffic values. In particular, we consider the RMSprop gradient descent optimization algorithm [130], an adaptive learning rate approach, wherein the learning rate of a particular weight is divided by a running average of the magnitudes of recent gradients for that weight.

The proposed proactive resource allocation scheme in [217] is compared with a reactive approach for three different network scenarios. Fig. 22 shows that for very small values of T, the proposed scheme does not yield any significant gains. However, as T increases, BSs have additional opportunities for shifting part of the traffic into the future and thus the gains start to become more pronounced. For example, from Fig. 22, we can see that, for 4 BSs and 4 channels, the proposed proactive scheme achieves an increase of 17% and 20% in terms of the average airtime allocation for LTE-U as compared to the reactive approach, respectively. Here, note that the gain of the proposed scheme, with respect to the reactive approach, keeps on increasing until it reaches the maximum achievable value, after which it remains almost constant.

4) Future Works: In summary, we have shown that ANNs can improve the total network performance for a multi-RAT system. They allow different technologies to predict the behavior of other nodes in the network and thus manage their resources based on the future network state. Moreover, ANNs can allow multi-mode BSs to learn on which spectrum band to transmit on

based on different network conditions. Here, note that, the above application of ANNs to LTE-U systems can be easily extended to a multi-mode network in which the BSs transmit on the licensed, unlicensed, and mmWave spectrum. In fact, given their capability of dealing with time series data, RNNs can enhance mobility and handover in highly mobile wireless environments by learning the mobility patterns of users thus decreasing the ping-pong effect among different RATs. For instance, a predictive mobility management framework can address critical handover issues, including frequent handover, handover failure, and excessive energy consumption for seamless handover in emerging dense multi-RAT wireless cellular networks. ANNs can also predict the QoS requirements, in terms of delay and rate, for future offered traffic. Moreover, they can predict the transmission links' conditions and thus schedule users based on the links' conditions and the QoS requirements. Therefore, given the mobility patterns, transmission links' conditions and QoS requirements for each user, BSs can learn how to allocate different users on different bands such that the total network performance, in terms of delay and throughput, is optimized.

An interesting application of DNNs for mmWave is antenna tilting. In particular, DNNs are capable of learning several features of the network environment and thus predicting the optimal tilt angle based on the availability of a LoS link and data rate requirements. This in turn improves the users' throughput thus achieving high data rate. Moreover, LSTMs are capable of learning long time series and thus can allow BSs to predict the link formation for the mmWave backhaul network. In fact, the formation of this backhaul network is highly dependent on the network topology and traffic conditions. Therefore, given the dynamics of the network, LSTMs enable BSs to dynamically update the formation of the links among each others based on the changes in the network. Moreover, SNNs can be used for mmWave channel modeling since they can process and predict continuous-time data effectively. A summary of key problems that can be solved by using ANNs in multi-RAT system is presented in Table II along with the challenges and future works.

F. Internet of Things

1) The Internet of Things: In the foreseeable future, it is envisioned that trillions of machine-type devices such as wearables, sensors, connected vehicles, or mundane objects will be connected to the Internet, forming a massive IoT ecosystem [220], as shown in Fig. 23. The IoT
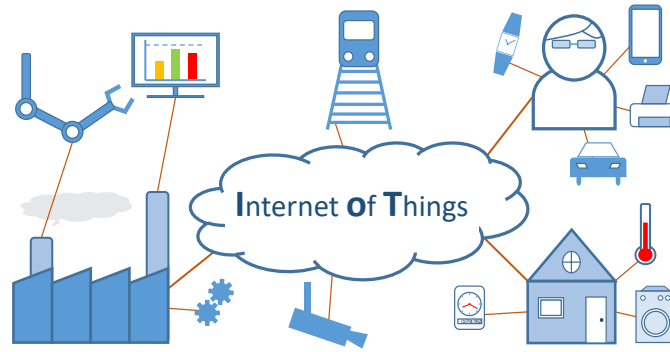
Fig. 23. An illustration of the Internet of Things.

will enable machine-type devices to connect with each other over wireless links and operate in a self-organizing manner. Therefore, IoT devices can collect and exchange real-time information to provide smart services. In this respect, the IoT will allow delivering innovative services and solutions in the realms of smart cities, smart grids, smart homes, and connected vehicles that could provide a significant improvement in people's lives. For example, IoT devices can be used to intelligently manage all the city's systems such as local departments' information, schools, libraries, transportation, hospitals, water supply, and electricity systems hence improving service efficiency. Meanwhile, retailers, restaurant chains and makers of consumer goods can use data from smartphones, wearable devices and in-home devices to do targeted marketing and promotions. However, the practical deployment of the IoT system sill faces many challenges such as data analytics, computation, transmission capabilities, connectivity end-to-end latency, security, and privacy. In particular, how to provide massive device connectivity with stringent latency requirement will be one of the most important challenges. The current centralized communication models and corresponding technologies may not be able to provide such massive connectivity. Therefore, there is a need for new communication architecture such as fog computing models for IoT devices connectivity. Moreover, for each IoT device, energy and computational resources are limited. Hence, how to allocate computational resources and power for all IoT devices to achieve the data rate and latency requirements is another challenge.

2) Neural Networks for Internet of Things: ANNs can be used to address some of the key challenges within the context of the IoT. ANNs have four major applications for IoT. First, ANNs enable the IoT system to leverage big data analytics to extract important patterns and relationship from the data sent by the IoT devices. For example, ANNs can be used to discover

important correlations among data to improve the data compression and data recovery. Second, using ANN-based RL algorithms, IoT devices can operate in a self-organizing manner and adapt their strategies (i.e., channel selection) based on the wireless and users environments. For instance, an IoT device that uses an ANN-based RL algorithm can dynamically select the most suitable frequency band for communication according to the network state. Third, IoT devices that use ANN-based algorithms can identify and classify the data collected from the IoT sensors. Finally, one of the main goals of the IoT is to improve the life quality of humans and reduce the interaction between human and IoT devices, and, hence, ANNs can be used to predict the users behavior to provide advanced information for IoT devices. For example, ANNs can be used to predict the time that an individual will come home, and, hence, adjust the control strategy for IoT devices at home.

However, using ANNs for IoT faces many challenges. First, in IoT, both energy and computational resources are limited. Therefore, one should consider the tradeoff between the energy and computational needs of training ANNs and the accuracy requirement of a given ANN-based learning algorithm. In particular, the higher the required accuracy, the higher the computational and energy requirements. Second, within an IoT ecosystem, the collected data may have different structure and even contain several errors. Therefore, when we use data to train ANNs, we should consider how to classify the data and deal with the flaws in the data. In other words, ANNs in IoT must tolerate erroneous data. Third, in the IoT system, ANNs can exploit thousands types of data for prediction and self-organizing control. For a given task, the data collected from the IoT devices may be not all related to the task. Hence, ANNs must select suitable data for the task.

• *Related works:* The existing literature such as in [221]–[226] has studied a number of problems related to using ANNs for IoT. In [221], the authors use a framework to treat an IoT network holistically as an ANNs to reduce delivery latency. The authors in [222] and [223] used a backpropagation neural network for sensor failure detection in IoT network. In [224], eight machine learning algorithms such as DNNs and FNNs are tested for human activities classification, robot navigation, body postures and movements. In [225], the authors used Laguerre neural network-based approximate dynamic programming scheme to improve the tracking efficiency in IoT network. The authors in [226] develped a streaming hardware accelerator for convolutional neural network to improve the accuracy of image detection in IoT network. In summary, the

prior art in [221]–[226] used ANNs to solve a number of IoT problems such as IoT network modeling, failure detection, human activities classification, and tracking accuracy improvement. However, ANNs can also be used to analyze the data correlation for data compression and data recovery, identify human, predict human activities, and manage the resources of IoT devices. Next, we introduce a specific ANNs' application for IoT.

3) Example: One illustrative application for the use of ANNs within the context of the IoT is presented in [221] which studies how to improve the communication quality by mapping IoT networks to ANNs. In [221], the considered IoT network is primarily a wireless sensor network. Two objective functions are considered : a) minimizing the overall cost of communication between the devices mapped to the neurons in the input layer and the devices mapped to he neurons in the output layers. Here, the overall cost represents the total transmit power of all devices used to transmit the information signals, and b) minimizing the expected transmit time to delivery the information signals.

To minimize the total transmit power and expected transmit time for the IoT, the basic idea of [221] is to train an ANN so as to approximate the objective functions discussed above and, then, map the IoT network to the ANN. FNNs, discussed in Section III, are used for this mapping since they transmit the information in only one direction, forward, from the input nodes, through the hidden nodes, and to the output nodes. First, one must identify devices that want to send signals as well as the devices that will receive signals. The IoT devices that want to send signals are mapped to the neurons in the input layers. The IoT devices that want to receive the signals are mapped to the neurons in the output layers. The other IoT devices are mapped to the neurons in the hidden layers. Some of the devices that are mapped to the hidden layers will be used to forward the signals. Second, the FNN is trained in an offline manner to approximate the objective functions. In [221], the IoT network devices are mapped into neurons and wireless links into connections between neurons, and, hence, a method is needed to map the trained FNN to the IoT network. Since the computational resources of each IoT device is limited, IoT devices with different computational resources will map to different number of neurons. For example, an IoT device that has more computational resources can map to a larger number of neurons. Moreover, to ensure the integrity of the mapping model, each neuron can only map to one of the IoT devices. Given that there are several ways to map the IoT network to the trained FNN, the optimal mapping is formulated as an integer linear program which is then solved

using CPLEX [221]. When the optimal mapping between the IoT network and the trained FNN is found, the optimal connections between IoT devices are built. Hence, if the IoT network can find the optimal connections for all devices based on the objective functions, the transmit power and expected transmit time can be reduced. Simulation results in [221] show that the mapping algorithm can achieve significant gains in terms of total transmit power and expected transmit time compared to a centralized algorithm. This is because the IoT network uses FNNs to approximate the objective functions and find the optimal devices' connections.

From this example, we have illustrated the use of FNN to minimize the total transmit power and expected transmit time for IoT network. Since FNNs transmit information in only one direction from the input layer to the hidden layer, then to the output layer, the IoT devices that want to send signals can be mapped to the neurons in the input layer. Meanwhile, the devices that will receive wireless signals can be mapped to the neurons in the output layer. Then, one can train the FNN and find the optimal mapping. The optimal mapping represents the optimal connections between the devices. Simulation results show that the FNN mapping algorithm can reduce the total transmit power and expected delivery time for all IoT devices.

4) Future Works: ANNs are undoubtedly an important tool for solving a variety of problems in the IoT, particularly in terms of big data analytics and smart operation. In fact, beyond using FNNs to map the IoT devices hence optimizing connections between IoT devices as discussed above, FNNs can also be used to map other systems. For example, one can map the input layer of an FNN to the IoT devices and the output layer to the computing centers. Then, one can find an optimal allocation of computational tasks via FNN mapping. Moreover, ANNs can be used for data compression and recovery so as to reduce both the size of the transmitted data and end-to-end devices latency. To compress the data, an ANN needs to extract the most important features from the data and these features can be used to present the compressed data. In particular, CNNs can be used for data compression and recovery in the spatial domain while RNNs can be used for data compression and recovery in the time domain. This is because CNNs are effective at extracting patterns and features from large amounts of data and RNNs are suitable for extracting the relationships from time-dependent series data. In addition, DNNs can be used for human identification. An IoT ecosystem that can identify different individuals can pre-allocate spectral or computational resources to the IoT devices that a certain individual often uses. DNNs are suitable here because they have multiple hidden layers to store more information related to a user

TABLE II
SUMMARY OF THE USE OF ANNs FOR SPECIFIC APPLICATION

Applications	Existing Works	Challenges	Future Works and Suggested Solutions
UAV	• UAV control [167], [168]. • Position estimation [169]. • Image classification [170]. • Deployment and caching [171].	• Limited power and computation for training ANNs. • Limited time for data collection. • Errors in training data.	• UAV path planning $\Rightarrow$ RNN-based RL algorithm. • Resource management $\Rightarrow$ DNN-based RL algorithm. • Channel modeling for air-to-ground $\Rightarrow$ SNN prediction algorithm. • Handover for cellular connected UAVs $\Rightarrow$ RNN prediction algorithm. • Design multi-hop aerial network $\Rightarrow$ CNN-based algorithm.
VR	• Hand gesture recognition [176]. • Interactive shape change [178]. • Video conversion [179]. • Gaze prediction [181]. • Resource allocation [182].	• Errors in collected data. • Limited computational resources. • Limited time for training ANNs.	• VR users' movement $\Rightarrow$ RNNs prediction algorithm. • Manage computational resources and video formats $\Rightarrow$ DNN-based RL algorithm. • VR video coding and decoding $\Rightarrow$ CNN-based algorithm. • Correction of inaccurate VR images $\Rightarrow$ CNN-based algorithm. • Joint wireless and VR user environment prediction $\Rightarrow$ RNNs prediction algorithm. • Content correlation $\Rightarrow$ CNN-based algorithm.
Caching and Computing	• Architecture for caching [192]. • Cache replacement [193], [194]. • Content prediction [195], [171], [98].	• Data cleaning. • Content classification. • Limited storage of ANNs for recording all types of contents.	• Analysis of content correlation. • Content transmission methods $\Rightarrow$ RNN-based RL algorithm. • Clustering of users and tasks $\Rightarrow$ CNN-based algorithm. • Computing demand and time prediction $\Rightarrow$ RNN-based algorithm. • Computational resource allocation $\Rightarrow$ RNN-based RL algorithm. • Computational caching $\Rightarrow$ RNN-based RL algorithm.
Multi-RAT	• Resource management [211], [173]. • RAT selection [213], [216]. • Transmission technology classification [214]. • Multi-radio packet scheduling [215].	• Channel selection. • Mobility predictions. • Channel load estimation. • Load balancing.	• Detection of LoS links $\Rightarrow$ CNN-based algorithm. • Antenna tilting $\Rightarrow$ DNN-based RL algorithm. • Channel estimation $\Rightarrow$ SNN-based algorithm. • Handover among multi-RAT BSs $\Rightarrow$ RNN-based algorithm. • Links for multi-RAT with mmWave $\Rightarrow$ DNN-based algorithm. • MmWave channel modeling $\Rightarrow$ SNN-based algorithm.
IoT	• Model IoT as ANNs [221]. • Failure detection [222], [223]. • User activities classification [224]. • Image detection [226]. • Tracking accuracy improvement [225].	• Massive amounts of data and large number of devices. • Limited computation and energy resources. • Errors in collected data. • Real-time training for ANNs.	• Data compression and recovery $\Rightarrow$ CNN-based algorithm. • Resource management $\Rightarrow$ RNN-based RL algorithm. • User identification $\Rightarrow$ DNN-based algorithm. • IoT devices management $\Rightarrow$ SNN-based algorithm. • Data relationship extraction $\Rightarrow$ RNN-based RL algorithm. • Modeling autonomous M2M communication $\Rightarrow$ FNN and SNN based algorithm.

compared to other ANNs and, hence, DNNs can use one user's information such as hairstyle, clothes, and oral patterns to identify that individual so as to provide services tailored to this user. A summary of key problems that can be solved by using ANNs in IoT system is presented in Table II along with the challenges and future works.

G. Other Applications

Beyond our previous, detailed discussion of several applications of ANNs, in this subsection, we briefly introduce other applications of ANNs such as wireless physical layer design, vehicular networks, and smart cities.

• *Wireless Physical Layer Design:* The wireless physical layer is responsible for the various features of the physical medium, signals, and bits and manages services such as modulation, bit-by-bit delivery, coding, forward error correction, and transmission mode control. To improve the performance of the wireless physical layer, ANNs can be used for various purposes [33]. For example, FNNs can be used to model the end-to-end transmission, especially when the channel model of the transmission is unknown. This is due to the fact that an FNN can map the transmitter to the input layers, receiver to the output layers, and the values of channel gain to the hidden layers, and hence, there is no need to know the specific mathematical channel model. Moreover, SNNs that are good at dealing with continuous data can be used for signal detection and correction so as to improve the quality of the signal transmission. SNNs can be used to learn the signal patterns and use the learned patterns to detect the received signals. Then, signal errors can be detected and corrected. In addition, DNNs and CNNs can be used for modulation classification due to their capability of storing and extracting information from the data. The existing works such as [33]–[37] have studied a number of problems related to the use of ANNs for physical layer design such as signal detection in [33] and [37], learning transmitted symbols in [34], channel decoding in [35] and [36]. In summary, ANNs can be used for physical layer design in a number of applications such as channel decoding and coding, signal detection and correction, modulation control and classification, and carrier sensing and collision detection.

• *Vehicular Networks:* Vehicular networks leverage wireless connectivity to provide on-road services such as forwarding safety messages as well as traffic and congestion monitoring [227]. In vehicular networks, some vehicles can be selected to act as temporary content providers for delivering popular contents to potential nearby vehicles using vehicle-to-vehicle (V2V)

communication. Recently, with the development of AI applications [228], autonomous vehicles are considered as an inevitable trend for future vehicles development. Autonomous vehicles use a number of techniques to detect their surroundings, such as radar, GPS, and computer vision, and intelligently identify suitable navigation paths, obstacles and relevant signage from the collected information. In fact, ANNs have the potential to be highly beneficial when integrated into an autonomous vehicle's software architecture. For example, ANNs can use the data collected by autonomous vehicles to learn the behaviors of other vehicles on the road and recognize the navigation paths, obstacles, and signages. Moreover, ANNs can also be used for speech recognition, gesture recognition, eye tracking, virtual assistance, and natural language interfaces in autonomous vehicles. In addition, ANNs can also be used for measuring the QoS of the wireless communication links so as to select the most suitable transmission links for content delivery (e.g., using vehicle-to-vehicle communication). In particular, ANNs can be used to predict the vehicles' speed, the distance, the power, the contents that have been stored, and the route. Based on this predicted information, each vehicle can select the optimal target vehicles for content transmission. The existing literature [229]–[233] has studied a number of problems pertaining to the use of ANNs for autonomous vehicles, such as driver behavior modeling [229], mobility speed prediction [230], objective classification [231], vehicles classification [232], and steering autonomous vehicles [233]. However, most of these existing works [229]–[233] focus only on issues of autonomy and do not consider the challenges of wireless communications.

- *Smart Cities:* A smart city is a complex ecosystem that uses a number of communications technologies such as the IoT to enable cities to operate in a sustainable, clean, and efficient manner so as to improve the quality of life for individuals [234]. Wireless communication technologies can allow the city officials to monitor the various critical infrastructure and in their city and coordinate the various critical infrastructure systems. A city's critical infrastructure systems [235] such as transportation, healthcare, and water supply systems will generate tremendous amounts of data that can be leveraged for safe, efficient applications and services for city residents. To find the relationships between various infrastructure systems and improve the services of these systems, one must find a method to efficiently use and manage this voluminous data. ANN-based algorithms may be a promising method for the management and investigation of the voluminous data in smart cities. In particular, ANNs such as RNNs can be used to predict the traffic activities such as vehicular traffic and pedestrian traffic using the collected data in

smart cities. Based on these predictions, a DNN-based RL algorithm can dynamically adjust the signals so as to relieve congestion and clear isolated backups caused by collisions. Meanwhile, DNN-based RL algorithms can also be used to control the street lights so as to reduce the energy consumption. For example, a DNN-based RL algorithm can learn the time of day and the location where traffic activities are low, and, hence, turn off the lights to save energy. Moreover, we can consider the communication devices as an IoT network and, hence, the solutions that use ANNs to solve IoT problems can also be extended to the context of a large-scale, smart city. Existing works such as [234], [236]–[238] studied several problems related to the use of ANNs for solving smart cities problems such as data management [234], urban traffic flow prediction [236], water demand prediction [237], and parking availability prediction [238]. However, the area remain ripe for many innovations.

V. CONCLUSION

In this paper, we have provided one of the first comprehensive tutorials on the use of machine learning, in general, and neural networks, in particular, for enabling a variety of applications in tomorrow's wireless networks. In particular, we have presented a comprehensive overview on a number of key types of neural networks, that include feed-forward, recurrent, spiking, and deep neural networks. For each type, we have overviewed the basic architecture and training procedure, as well as the associated challenges and opportunities. Then, we have provided a panoramic overview on the variety of wireless communication problems that can be addressed using ANNs. In particular, we have investigated many emerging applications that include unmanned aerial vehicles, wireless virtual reality, mobile edge caching and computing, Internet of Things, and multi-RAT wireless networks. For each application, we have provided the main motivation for using ANNs along with their associated challenges while also providing a detailed example for a use case scenario. Last, but not least, for each individual application, we have provided a broad overview on future works that can be addressed using ANNs. Clearly, the future of cellular networks will inevitably rely on artificial intelligence and, thus, this paper provides a first stepping stone towards understanding the analytical machinery needed to develop such a new bread of wireless networks. This is therefore expected to spur further research in this burgeoning area.

REFERENCES

[1] I. Yaqoob, E. Ahmed, I. A. T. Hashem, A. I. A. Ahmed, A. Gani, M. Imran, and M. Guizani, "Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges," *IEEE Wireless Communications*, vol. 24, no. 3, pp. 10–16, June 2017.

[2] N. C. Luong, D. T. Hoang, P. Wang, D. Niyato, D. I. Kim, and Z. Han, "Data collection and wireless communication in internet of things (IoT) using economic analysis and pricing models: a survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2546–2590, June 2016.

[3] Z. Dawy, W. Saad, A. Ghosh, J. G. Andrews, and E. Yaacoub, "Toward massive machine type cellular communications," *IEEE Wireless Communications*, vol. 24, no. 1, pp. 120–128, Nov. 2017.

[4] T. Park, N. Abuzainab, and W. Saad, "Learning how to communicate in the internet of things: Finite resources and heterogeneity," *IEEE Access*, vol. 4, pp. 7063–7073, Nov. 2016.

[5] F. Boccardi, R. W. Heath, A. Lozano, T. L. Marzetta, and P. Popovski, "Five disruptive technology directions for 5G," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 74–80, Feb. 2014.

[6] P. Popovski, "Ultra-reliable communication in 5G wireless systems," in *Proc. of International Conference on 5G for Ubiquitous Connectivity (5GU)*, Akaslompolo, Finland, Feb. 2014.

[7] N. A. Johansson, Y. P. E. Wang, E. Eriksson, and M. Hessler, "Radio access for ultra-reliable and low-latency 5G communications," in *Proc. of IEEE International Conference on Communication Workshop (ICCW)*, London, UK, Sept. 2015.

[8] O. N. C. Yilmaz, Y. P. E. Wang, N. A. Johansson, N. Brahmi, S. A. Ashraf, and J. Sachs, "Analysis of ultra-reliable and low-latency 5G communication for a factory automation use case," in *Proc. of IEEE International Conference on Communication Workshop (ICCW)*, London, UK, Sept. 2015, pp. 1190–1195.

[9] N. H. Motlagh, M. Bagaa, and T. Taleb, "UAV-based IoT platform: A crowd surveillance use case," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 128–134, Feb. 2017.

[10] N. H. Motlagh, M. Bagaa, and T. Taleb, "UAV selection for a UAV-based integrative IoT platform," in *Proc. of IEEE Global Communications Conference (GLOBECOM)*, Washington DC, USA, Dec. 2016.

[11] Y. Kawamoto, H. Nishiyama, N. Kato, N. Yoshimura, and S. Yamamoto, "Internet of things (IoT): Present state and future prospects," *IEICE Transactions on Information and Systems*, vol. 97, no. 10, pp. 2568–2575, Oct. 2014.

[12] J. Zhou, X. Cao, Z.and Dong, and A. V. Vasilakos, "Security and privacy for cloud-based IoT: Challenges," *IEEE Communications Magazine*, vol. 55, no. 1, pp. 26–33, Jan. 2017.

[13] D. G. Gopal and S. Kaushik, "Emerging technologies and applications for cloud-based gaming: Review on cloud gaming," *Emerging Technologies and Applications for Cloud-Based Gaming*, vol. 41, no. 07, pp. 79–89, March 2016.

[14] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. K. Soong, and J. C. Zhang, "What will 5G be?," *IEEE Journal on selected areas in communications*, vol. 32, no. 6, pp. 1065–1082, June 2014.

[15] J. Ferber, *Multi-agent systems: An introduction to distributed artificial intelligence*, vol. 1, Addison-Wesley Reading, 1999.

[16] "Talos," http://www.theoi.com/Gigante/GiganteTalos.html.

[17] T. Segaran, *Programming collective intelligence: Building smart web 2.0 applications*, O'Reilly Media, Inc., 2007.

[18] B. Yegnanarayana, *Artificial neural networks*, PHI Learning Pvt. Ltd., 2009.

[19] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan, "An introduction to MCMC for machine learning," *Machine Learning*, vol. 50, no. 1-2, pp. 5–43, Jan. 2003.

[20] W. T. Freeman, E. C. Pasztor, and O. T. Carmichael, "Learning low-level vision," *International Journal of Computer Vision*, vol. 40, no. 1, pp. 25–47, Oct. 2000.

[21] F. Sebastiani, "Machine learning in automated text categorization," *ACM Computing Surveys (CSUR)*, vol. 34, no. 1, pp. 1–47, March 2002.

[22] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proc. of the International Conference on Machine Learning*, New York, NY, USA, July 2008, pp. 160–167.

[23] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up?: Sentiment classification using machine learning techniques," in *Proc. of the Conference on Empirical Methods in Natural Language processing*, Stroudsburg, PA, USA, July 2002.

[24] C. M. Bishop, *Pattern recognition and machine learning*, Springer, 2006.

[25] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," *Autonomous Robots*, vol. 8, no. 3, pp. 345–383, June 2000.

[26] F. J. Varela and P. Bourgine, *Toward a practice of autonomous systems: Proceedings of the First European Conference on Artificial Life*, MIT press, 1992.

[27] R. Szeliski, *Computer vision: Algorithms and applications*, Springer Science & Business Media, 2010.

[28] L. Dorst, D. Fontijne, and S. Mann, *Geometric algebra for computer science: an object-oriented approach to geometry*, Morgan Kaufmann Publishers Inc., 2009.

[29] J. R. Parker, *Algorithms for image processing and computer vision*, John Wiley & Sons, 2010.

[30] A. K. Jain and S. Z. Li, *Handbook of face recognition*, Springer, 2011.

[31] S. Bi, R. Zhang, Z. Ding, and S. Cui, "Wireless communications in the era of big data," *IEEE Communications Magazine*, vol. 53, no. 10, pp. 190–199, Oct. 2015.

[32] T. J. O'Shea and J. Hoydis, "An introduction to machine learning communications systems," *available online arXiv:1702.00832*, July 2017.

[33] T. J. O'Shea, K. Karra, and T. C. Clancy, "Learning approximate neural estimators for wireless channel state information," *available online arXiv:1707.06260*, June 2017.

[34] T. J. O'Shea, T. Erpek, and T. C. Clancy, "Deep learning based mimo communications," *available online arXiv:1707.07980*, July 2017.

[35] F. Liang, C. Shen, and F. Wu, "An iterative BP-CNN architecture for channel decoding," *available online arXiv:1707.05697*, July 2017.

[36] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Beery, "Deep learning methods for improved decoding of linear codes," *available online arXiv:1706.07043*, June 2017.

[37] N. Samuel, T. Diskin, and A. Wiesel, "Deep MIMO detection," *available online arXiv:1706.01151*, June 2017.

[38] J. Steuer, "Defining virtual reality: Dimensions determining telepresence," *Journal of Communication*, vol. 42, no. 4, pp. 73–93, Dec. 1992.

[39] C. Burdea Grigore and P. Coiffet, *Virtual reality technology*, London: Wiley-Interscience, 1994.

[40] E. Baştuğ, M. Bennis, M. Médard, and M. Debbah, "Towards interconnected virtual reality: Opportunities, challenges and enablers," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 110–117, June 2017.

[41] R. Yu, "Huawei reveals the future of mobile ai at IFA 2017," http://www.businesswire.com/news/home/20170902005020/en/Huawei-Reveals-Future-Mobile-AI-IFA-2017, 2017.

[42] S. Kovach, "Qualcomm CEO Steve Mollenkopf: What the big innovation house that powered the mobile boom is betting on next," http://www.businessinsider.com/qualcomm-ceo-steve-mollenkopf-interview-2017-7, 2017.

[43] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing-A key technology towards 5G," *ETSI White Paper*, vol. 11, no. 11, pp. 1–16, Sept. 2015.

[44] A. Ahmed and E. Ahmed, "A survey on mobile edge computing," in *Proc. of International Conference on Intelligent Systems and Control*, Coimbatore, India, Jan. 2016.

[45] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, June 2015.

[46] S. Nunna, A. Kousaridas, M. Ibrahim, M. Dillinger, C. Thuemmler, H. Feussner, and A. Schneider, "Enabling real-time context-aware collaboration through 5G and mobile edge computing," in *Proc. of International Conference on Information Technology-New Generations*, Las Vegas, NV, USA, June 2015.

[47] G. Lee, W. Saad, and M. Bennis, "Decentralized cross-tier interference mitigation in cognitive femtocell networks," in *Proc. of IEEE International Conference on Communications (ICC)*, Paris, France, May 2017.

[48] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, Sep. 2016.

[49] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[50] O. Semiari, W. Saad, S. Valentin, M. Bennis, and H. V. Poor, "Context-aware small cell networks: How social metrics improve wireless resource allocation," *IEEE Transactions on Wireless Communications*, vol. 14, no. 11, pp. 5927–5940, July 2015.

[51] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos, "Learning to optimize: Training deep neural networks for wireless resource management," *available online arXiv:1705.09412*, May 2017.

[52] C. Jiang, H. Zhang, Y. Ren, Z. Han, K. C. Chen, and L. Hanzo, "Machine learning paradigms for next-generation wireless networks," *IEEE Wireless Communications*, vol. 24, no. 2, pp. 98–105, April 2017.

[53] M. Bkassiny, Y. Li, and S. K. Jayaweera, "A survey on machine-learning techniques in cognitive radios," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1136–1159, Oct. 2013.

[54] N. Kato, Z. M. Fadlullah, B. Mao, F. Tang, O. Akashi, T. Inoue, and K. Mizutani, "The deep learning vision for heterogeneous network traffic control: Proposal, challenges, and future perspective," *IEEE Wireless Communications*, vol. 24, no. 3, pp. 146–153, Dec. 2017.

[55] M. A. Alsheikh, S. Lin, D. Niyato, and H. P. Tan, "Machine learning in wireless sensor networks: Algorithms, strategies, and applications," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1996–2018, April 2014.

[56] Y. Zhang, N. Meratnia, and P. Havinga, "Outlier detection techniques for wireless sensor networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 2, pp. 159–170, April 2010.

[57] M. Di and E. M. Joo, "A survey of machine learning in wireless sensor netoworks from networking and application perspectives," in *Proc. of International Conference on Information, Communications & Signal Processing*, Singapore, Dec. 2007.

[58] R. M. Neal, *Bayesian learning for neural networks*, vol. 118, Springer Science & Business Media, 2012.

[59] R. Rojas, *Neural networks: a systematic introduction*, Springer Science & Business Media, 2013.

[60] H. B. Demuth, M. H. Beale, O. De Jess, and M. T. Hagan, *Neural network design*, Martin Hagan, 2014.

[61] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, Jan. 2015.

[62] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell, "An overview of machine learning," in *Machine learning*, pp. 3–23. 1983.

[63] R. E. Schapire, "The boosting approach to machine learning: An overview," in *Nonlinear Estimation and Classification*, pp. 149–171. Springer, 2003.

[64] E. Basegmez, "The next generation neural networks: Deep learning and spiking neural networks," in *Advanced Seminar in Technical University of Munich*, 2014, pp. 1–40.

[65] "Machine learning: What it is and why it matters," https://www.sas.com/en_us/insights/analytics/machine-learning.html.

[66] E. Alpaydin, *Introduction to machine learning*, MIT press, 2014.

[67] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[68] J. A. Anderson, *An introduction to neural networks*, MIT Press, 1995.

[69] J. W. Kalat, *Biological psychology*, Nelson Education, 2015.

[70] J. Wang, "Analysis and design of a k-winners-take-all model with a single state variable and the Heaviside step activation function," *IEEE Transactions on Neural Networks*, vol. 21, no. 9, pp. 1496–1506, Sept. 2010.

[71] B. Müller, J. Reinhardt, and M. T. Strickland, *Neural networks: An introduction*, Springer Science & Business Media, 2012.

[72] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, Aug. 2009.

[73] D. CireşAn, U. Meier, J. Masci, and J. Schmidhuber, "Multi-column deep neural network for traffic sign classification," *Neural Networks*, vol. 32, pp. 333–338, 2012.

[74] Y. Chauvin and D. E. Rumelhart, *Backpropagation: Theory, architectures, and applications*, Psychology Press, 1995.

[75] P. Baldi, "Gradient descent learning algorithm overview: A general dynamical systems perspective," *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 182–195, Jan. 1995.

[76] M. Mazur, "A step by step backpropagation example," https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/.

[77] S. Klein, J. P. W. Pluim, M. Staring, and M. A. Viergever, "Adaptive stochastic gradient descent optimisation for image registration," *International Journal of Computer Vision*, vol. 81, no. 3, pp. 227–239, Aug. 2009.

[78] S. Ruder, "An overview of gradient descent optimization algorithms," *available online: arXiv:1609.04747*, Sep. 2016.

[79] K. Greff, R. Srivastava, J. Koutnik, B. Steunebrink, and J. Schmidhuber, "LSTM: a search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, Oct. 2016.

[80] Y. Nesterov, "A method of solving a convex programming problem with convergence rate o (1/k2)," in *Proc. of Soviet Mathematics Doklady*, 1983, vol. 27.

[81] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, July 2011.

[82] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *available online: arXiv:1212.5701*, Dec. 2012.

[83] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *available online arXiv:1510.00149*, Feb. 2016.

[84] D. P. Mandic and J. A. Chambers, *Recurrent neural networks for prediction: learning algorithms, architectures and stability*, Wiley Online Library, 2001.

[85] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[86] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.

[87] J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Delyon, P. Y. Glorennec, H. Hjalmarsson, and A. Juditsky, "Nonlinear black-box modeling in system identification: A unified overview," *Automatica*, vol. 31, no. 12, pp. 1691–1724, 1995.

[88] R. J. Williams and D. Zipser, "Experimental analysis of the real-time recurrent learning algorithm," *Connection Science*, vol. 1, no. 1, pp. 87–111, Jan 1989.

[89] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, pp. 65–85, June 1994.

[90] Peter J. M. Van L. and E. HL Aarts, "Simulated annealing," in *Simulated annealing: Theory and applications*, pp. 7–15. Springer, 1987.

[91] J. Kennedy, "Particle swarm optimization," in *Proc. of Encyclopedia of Machine Learning*, pp. 760–766. 2011.

[92] M. Lukoševicius, *A Practical Guide to Applying Echo State Networks*, Springer Berlin Heidelberg, 2012.

[93] H. Jaeger, "Short term memory in echo state networks," in *GMD Report*, 2001.

[94] R. Ali and T. Peter, "Minimum complexity echo state network," *IEEE Transactions on Neural Networks*, vol. 22, no. 1, pp. 131–144, November 2011.

[95] G. Holzmann, "Echo state networks with filter neurons & a delay and sum readout with applications in audio signal processing," *Unpublished master thesis, Graz University of Technology*, 2008.

[96] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, no. 5667, pp. 78–80, April 2004.

[97] R. Sacchi, M. C. Ozturk, J. C. Principe, A. AFM Carneiro, and I. N. Da Silva, "Water inflow forecasting using the echo state network: A brazilian case study," in *Proc. of International Joint Conference on Neural Networks*, Orlando, FL, USA, October 2007.

[98] M. Chen, W. Saad, C. Yin, and M. Debbah, "Echo state networks for proactive caching in cloud-based radio access networks with mobile users," *IEEE Transactions on Wireless Communications*, vol. 16, no. 6, pp. 3520–3535, June 2017.

[99] I. Szita, V. Gyenes, and A. Lőrincz, "Reinforcement learning with echo state networks," in *Proc. of International Conference on Artificial Neural Networks*, Berlin, Heidelberg, Sept. 2006.

[100] C. Gallicchio and A. Micheli, "Deep reservoir computing: A critical analysis," in *Pro. of European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, Bruges, Belgium, April 2016.

[101] C. M. Bishop, "Training with noise is equivalent to tikhonov regularization," *Training*, vol. 7, no. 1, pp. 108–116, 2008.

[102] B. Farhang-Boroujeny, *Adaptive filters: theory and applications*, John Wiley & Sons, 2013.

[103] Herbert Jaeger, "Adaptive nonlinear system identification with echo state networks," in *Proc. of Advances in Neural Information Processing Systems*, Cambridge, MA, USA, 2002.

[104] J. J. Steil, "Backpropagation-decorrelation: online recurrent learning with o (n) complexity," in *Proc. of IEEE International Joint Conference on Neural Networks*, Budapest, Hungary, July 2004.

[105] D. Sussillo and L. F. Abbott, "Generating coherent patterns of activity from chaotic neural networks," *Neuron*, vol. 63, no. 4, pp. 544–557, Aug. 2009.

[106] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of Physiology*, vol. 117, no. 4, pp. 500–544, Aug. 1952.

[107] E. Orhan, "The leaky integrate-and-fire neuron model," http://corevision.cns.nyu.edu/~eorhan/notes/lif-neuron.pdf, 2012.

[108] Hélene Paugam-Moisy and Sander Bohte, "Computing with spiking neuron networks," in *Handbook of Natural Computing*, pp. 335–376. 2012.

[109] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, Nov. 2003.

[110] W. M. Kistler, W. Gerstner, and J. L. van Hemmen, "Reduction of the hodgkin-huxley equations to a single-variable threshold model," *Neural Computation*, vol. 9, no. 5, pp. 1015–1045, May 1997.

[111] A. M. Andrew, "Spiking neuron models: Single neurons, populations, plasticity," *Kybernetes*, vol. 32, no. 7, 2003.

[112] J. E. Matser, "Structured liquids in liquid state machines," M.S. thesis, 2011.

[113] R. Kempter, W. Gerstner, and J. L. Van Hemmen, "Hebbian learning and spiking neurons," *Physical Review E*, vol. 59, no. 4, April 1999.

[114] G. Bi and M. Poo, "Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type," *Journal of neuroscience*, vol. 18, no. 24, pp. 10464–10472, Dec. 1998.

[115] G. M. Wittenberg and S. S. H. Wang, "Malleability of spike-timing-dependent plasticity at the CA3–CA1 synapse," *Journal of Neuroscience*, vol. 26, no. 24, pp. 6610–6617, June 2006.

[116] E. Fino, J. M. Deniau, and L. Venance, "Cell-specific spike-timing-dependent plasticity in GABAergic and cholinergic interneurons in corticostriatal rat brain slices," *The Journal of Physiology*, vol. 586, no. 1, pp. 265–282, Jan. 2008.

[117] S. McKennoch, T. Voegtlin, and L. Bushnell, "Spike-timing error backpropagation in theta neuron networks," *Neural Computation*, vol. 21, no. 1, pp. 9–45, Jan. 2009.

[118] L. S. Smith and A. Hamilton, *Neuromorphic systems: Engineering silicon from neurobiology*, vol. 10, 1998.

[119] S. M. Bohte, H. La Poutré, and J. N. Kok, "Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer RBF networks," *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 426–435, Mar. 2002.

[120] M. Chen, W. Saad, and C. Yin, "Liquid state machine learning for resource allocation in a network of cache-enabled lte-u uavs," in *Proc. of the IEEE Global Communications Conference (GLOBECOM)*, Singapore, Dec. 2017.

[121] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, Nov. 2002.

[122] D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout, "Isolated word recognition with the liquid state machine: A case study," *Information Processing Letters*, vol. 95, no. 6, pp. 521–528, Sept. 2005.

[123] Stefan Kok, *Liquid state machine optimization*, Ph.D. thesis, Master Thesis, Utrecht University, 2007.

[124] W. Maass, "Liquid state machines: motivation, theory, and applications," *Computability in Context: Computation and Logic in the Real World*, pp. 275–296, 2010.

[125] J. J. Moré, "The levenberg-marquardt algorithm: implementation and theory," in *Numerical analysis*, pp. 105–116. 1978.

[126] A. Courville I. Goodfellow, Y. Bengio, *Deep Learning*, The MIT press, 2016.

[127] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier networks," in *Proc. of Artificial Intelligence and Statistics (AISTATS)*, Fort Lauderdale, FL, USA, June 2011.

[128] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. of the International Conference on Neural Information Processing Systems (NIPS)*. Lake Tahoe, Nevada, Dec. 2012.

[129] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, chapter Chapter 7: Regularization for Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org.

[130] T. Tieleman and G. Hinton, "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude," Technical report, 2012.

[131] J. Schmidhuber, "Learning complex, extended sequences using the principle of history compression," *Neural Computation*, vol. 4, pp. 234–242, Mar. 1992.

[132] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, June 2016.

[133] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. of International Conference on Learning Representations (ICLR)*. Banff, Canada, April 2014.

[134] Y. LeCun, L. Bottou, G. Orr, and K. Muller, *Neural Networks: Tricks of the Trade*, chapter Efficient backprop, pp. 9–50, Springer, 1998.

[135] Y. Bengio, *Neural Networks: Tricks of the Trade*, chapter Practical Recommendations for Gradient-Based Training of Deep Architectures, Springer, 2013.

[136] "A beginner guide to recurrent networks and LSTMs," https://deeplearning4j.org/lstm.html.

[137] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM networks," in *Proc. of IEEE International Joint Conference on Neural Networks (IJCNN)*. Montreal, Canada, Aug. 2005.

[138] I. Sutskever, O. Vinyals, and Q. Le, "Sequence to sequence learning with neural networks," in *Proc. of International Conference on Neural Information Processing Systems (NIPS)*. Montreal, Canada, Dec. 2014.

[139] F. Gers, N. Schraudolph, and J. Schmidhuber, "Learning precise timing with LSTM recurrent networks," *Journal of Machine Learning Research*, vol. 3, pp. 115–143, Aug. 2002.

[140] K. Cho, B. Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proc. of Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, Oct. 2014.

[141] K. Greff, R. Srivastava, J. Koutnik, B. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017.

[142] T. OShea, J. Corgan, and T. Clancy, "Convolutional radio modulation recognition networks," 2016.

[143] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. of Advances in neural information processing systems*, 2012, pp. 1097–1105.

[144] M. Soh, "Learning CNN-LSTM architectures for image caption generation," http://cs224d.stanford.edu/reports/msoh.pdf, 2016.

[145] W. MAAss and H. Markram, "On the computational power of circuits of spiking neurons," *J. Comput. Syst. Sci.*, vol. 69, no. 4, pp. 593–616, Dec. 2004.

[146] J. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Front. Neurosci.*, vol. 10, Nov. 2016.

[147] T. Kohonen, "The self-organizing map," *Neurocomputing*, vol. 21, no. 1-3, pp. 1–6, Nov. 1998.

[148] M. J. L. Orr, "Introduction to radial basis function networks," 1996.

[149] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Computation*, vol. 3, no. 2, pp. 246–257, 1991.

[150] S. Chen, B. Mulgrew, and P. M. Grant, "A clustering technique for digital communications channel equalization using radial basis function networks," *IEEE Transactions on Neural Networks*, vol. 4, no. 4, pp. 570–590, July 1993.

[151] M. J. Er, S. Wu, J. Lu, and H. L. Toh, "Face recognition with radial basis function (RBF) neural networks," *IEEE Transactions on Neural Networks*, vol. 13, no. 3, pp. 697–710, May 2002.

[152] L. Yingwei, N. Sundararajan, and P. Saratchandran, "A sequential learning scheme for function approximation using minimal radial basis function neural networks," *Neural Computation*, vol. 9, no. 2, pp. 461–478, Feb. 1997.

[153] W. Shen, X. Guo, C. Wu, and D. Wu, "Forecasting stock indices using radial basis function neural networks optimized by artificial fish swarm algorithm," *Knowledge-Based Systems*, vol. 24, no. 3, pp. 378–385, April 2011.

[154] G. Auda and M. Kamel, "Modular neural network classifiers: A comparative study," *Journal of Intelligent & Robotic Systems*, vol. 21, no. 2, pp. 117–129, Feb. 1998.

[155] B. Zhang and R. S. Govindaraju, "Prediction of watershed runoff using bayesian concepts and modular neural networks," *Water Resources Research*, vol. 36, no. 3, pp. 753–762, Mar. 2000.

[156] J. M. J. Murre, *Learning and categorization in modular neural networks*, Psychology Press, 2014.

[157] G. Baldassarre, "A modular neural-network model of the basal ganglia s role in learning and selecting motor behaviours," *Cognitive Systems Research*, vol. 3, no. 1, pp. 5–13, Mar. 2002.

[158] P. Melin, *Modular neural networks and type-2 fuzzy systems for pattern recognition*, Springer, 2012.

[159] A. Nugent, "Physical neural network design incorporating nanotechnology," May 3 2005, US Patent 6,889,216.

[160] S. Haykin, *Neural networks: a comprehensive foundation*, Prentice Hall PTR, 1994.

[161] A. J. Maren, C. T. Harston, and R. M. Pap, *Handbook of neural computing applications*, Academic Press, 2014.

[162] E. Bastug, M. Bennis, and M. Debbah, "Living on the edge: The role of proactive caching in 5G wireless networks," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 82–89, Aug. 2014.

[163] L. J. Lin, "Reinforcement learning for robots using neural networks," Tech. Rep., Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[164] Y. Zeng, R. Zhang, and T. J. Lim, "Wireless communications with unmanned aerial vehicles: opportunities and challenges," *IEEE Communications Magazine*, vol. 54, no. 5, pp. 36–42, May 2016.

[165] U. Challita and W. Saad, "Network formation in the sky: Unmanned aerial vehicles for multi-hop wireless backhauling," in *Proc. of the IEEE Global Communications Conference (GLOBECOM)*, Singapore, Dec. 2017.

[166] M. Mozaffari, W. Saad, M. Bennis, and M. Debbah, "Wireless communication using unmanned aerial vehicles (UAVs): Optimal transport theory for hover time optimization," *IEEE Transactions on Wireless Communications*, to appear, 2017.

[167] D. Nodland, H. Zargarzadeh, and S. Jagannathan, "Neural network-based optimal adaptive output feedback control of a helicopter UAV," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 7, pp. 1061–1073, July 2013.

[168] X. Wang, V. Yadav, and S. N. Balakrishnan, "Cooperative UAV formation flying with obstacle/collision avoidance," *IEEE Transactions on Control Systems Technology*, vol. 15, no. 4, pp. 672–679, July 2007.

[169] J. R. G. Braga, H. F. C. Velho, G. Conte, P. Doherty, and Élcio H. S., "An image matching system for autonomous UAV navigation based on neural network," in *Proc. of International Conference on Control, Automation, Robotics and Vision (ICARCV)*, Phuket, Thailand, Nov. 2016.

[170] A. Zeggada and F. Melgani, "Multilabel classification of UAV images with convolutional neural networks," in *Proc. of IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, Beijing, China, July 2016.

[171] M. Chen, M. Mozaffari, W. Saad, C. Yin, M. Debbah, and C. S. Hong, "Caching in the sky: Proactive deployment of cache-enabled unmanned aerial vehicles for optimized quality-of-experience," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 5, pp. 1046–1061, May 2017.

[172] H. Jaeger, "Controlling recurrent neural networks by conceptors," *available online: arxiv.org/abs/1403.3369*, 2014.

[173] M. Chen, W. Saad, and C. Yin, "Echo state networks for self-organizing resource allocation in LTE-U with uplink-downlink decoupling," *IEEE Transactions on Wireless Communications*, vol. 16, no. 1, pp. 3–16, Jan. 2017.

[174] U. Challita, Z. Dawy, G. Turkiyyah, and J. Naoum-Sawaya, "A chance constrained approach for LTE cellular network planning under uncertainty," *Computer Communications*, vol. 73, pp. 34–45, Jan. 2016.

[175] HTC, "HTC vive," https://www.vive.com/us/.

[176] J. Weissmann and R. Salomon, "Gesture recognition for virtual reality applications using data gloves and neural networks," in *Proc. of International Joint Conference onNeural Networks (IJCNN)*, Washington, DC, USA, July 1999.

[177] Deyou Xu, "A neural network approach for hand gesture recognition in virtual reality driving training system of SPG," in *International Conference on Pattern Recognition, (ICPR)*, Hong Kong, China, Aug. 2006.

[178] R. Hambli, A. Chamekh, and H. B. H. Salah, "Real-time deformation of structure using finite element and neural networks in virtual reality applications," *Finite Elements in Analysis and Design*, vol. 42, no. 11, pp. 985–991, July 2006.

[179] J. Xie, R. Girshick, and A. Farhadi, "Deep3d: Fully automatic 2D-to-3D video conversion with deep convolutional neural networks," in *European Conference on Computer Vision*, Amsterdam, The Netherlands, Oct. 2006.

[180] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan, "Optimizing 360 video delivery over cellular networks," in *Proc. of ACM Workshop on All Things Cellular: Operations, Applications and Challenges*, NY, USA, Oct. 2016.

[181] G. A. Koulieris, G. Drettakis, D. Cunningham, and K. Mania, "Gaze prediction using machine learning for dynamic stereo manipulation in games," in *Proc. of IEEE Virtual Reality (VR)*, Greenville, SC, USA, March 2016.

[182] M. Chen, W. Saad, and C. Yin, "Virtual reality over wireless networks: Quality-of-service model and learning-based resource management," *available online: arxiv.org/abs/1703.04209*, Mar. 2017.

[183] M. Chen, W. Saad, and C. Yin, "Echo state transfer learning for data correlation aware resource allocation in wireless virtual reality," in *Proc. of Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, USA, Oct. 2017.

[184] A. E. Abbas, "Constructing multiattribute utility functions for decision analysis," *INFORMS Tutorials in Operations Research*, pp. 62–98, Oct. 2010.

[185] M. Abrash, "What VR could, should, and almost certainly will be within two years," https://media.steampowered.com/apps/abrashblog/Abrash%20Dev%20Days%202014.pdf.

[186] Cisco, "Global mobile data traffic forecast update, 2013–2018," *White Paper*, 2014.

[187] C. K. Toh, "Maximum battery life routing to support ubiquitous mobile computing in wireless ad hoc networks," *IEEE Communications Magazine*, vol. 39, no. 6, pp. 138–147, June 2001.

[188] Y. Fadlallah, A. M. Tulino, D. Barone, G. Vettigli, J. Llorca, and J. M. Gorce, "Coding for caching in 5G networks," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 106–113, Feb. 2017.

[189] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, to appear, 2017.

[190] M. Peng, Y. Sun, X. Li, Z. Mao, and C. Wang, "Recent advances in cloud radio access networks: System architectures, key techniques, and open issues," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 3, pp. 2282–2308, Mar. 2016.

[191] M. S. Elbamby, M. Bennis, and W. Saad, "Proactive edge computing in latency-constrained fog networks," *available online arXiv:1704.06749*, 2017.

[192] E. Zeydan, E. Bastug, M. Bennis, M. A. Kader, I. A. Karatepe, A. S. Er, and M. Debbah, "Big data caching for networking: Moving from cloud to edge," *IEEE Communications Magazine*, vol. 54, no. 9, pp. 36–42, Sept. 2016.

[193] J. Cobb and H. ElAarag, "Web proxy cache replacement scheme based on back-propagation neural network," *Journal of Systems and Software*, vol. 81, no. 9, pp. 1539–1558, Sept. 2008.

[194] S. Romano and H. ElAarag, "A neural network proxy cache replacement strategy and its implementation in the squid proxy server," *Neural Computing and Applications*, vol. 20, no. 1, pp. 59–78, Feb. 2011.

[195] W. Tian, B. Choi, and V. Phoha, "An adaptive web cache access predictor using neural network," *Developments in Applied Artificial Intelligence*, pp. 113–117, 2002.

[196] E. Bastug, M. Bennis, E. Zeydan, M. A. Kader, I. A. Karatepe, A. S. Er, and M. Debbah, "Big data meets telcos: A proactive caching perspective," *Journal of Communications and Networks*, vol. 17, no. 6, pp. 549–557, December 2015.

[197] D. Wu and R. Negi, "Effective capacity-based quality of service measures for wireless networks," *Mobile Networks and Applications*, vol. 11, no. 1, pp. 91–99, February 2006.

[198] D. Wang and Z. Han, *Sublinear Algorithms for Big Data Applications*, Springer Berlin Heidelberg, 2015.

[199] Y. Gu, W. Saad, M. Bennis, M. Debbah, and Z. Han, "Matching theory for future wireless networks: fundamentals and applications," *IEEE Communications Magazine, Special Issue on Emerging Applications, Services, and Engineering for Cellular Cognitive Systems*, vol. 53, no. 15, pp. 52–59, May 2015.

[200] 5GPPP, "The 5G infrastructure public private partnership: the next generation of communication networks and services.," Feb. 2015.

[201] J. Andrews, "Seven ways that HetNets are a cellular paradigm shift," *IEEE Communications Magazine*, vol. 51, no. 3, pp. 136–144, Mar. 2013.

[202] Q. Li, H. Niu, A. Papathanassiou, and G. Wu, "5G network capacity: Key elements and technologies," *IEEE Vehicular Technology Magazine*, vol. 9, no. 1, pp. 71–78, Mar. 2014.

[203] O. Semiari, W. Saad, M. Bennis, and M. Debbah, "Performance analysis of integrated sub-6 ghz-millimeter wave wireless local area networks," in *Proc. of IEEE Global Communications Conference (GLOBECOM)*, Washington DC, USA, Dec. 2017.

[204] O. Semiari, W. Saad, M. Bennis, and M. Debbah, "Joint millimeter wave and microwave resources allocation in cellular networks with dual-mode base stations," *IEEE Transactions on Wireless Communications*, vol. 16, no. 7, pp. 4802–4816, July 2017.

[205] ETSI EN 301 893 V2.1.1, "5 GHz RLAN; Harmonised Standard covering the essential requirements of article 3.2 of Directive 2014/53/EU ," May 2017.

[206] U. Challita and M. K. Marina, "Holistic small cell traffic balancing across licensed and unlicensed bands," in *Proc. of ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, Malta, Nov. 2016.

[207] U. Challita, L. Dong, and W. Saad, "Deep learning for proactive resource allocation in LTE-U networks," in *Proc. of European Wireless Conference*. Dresden, Germany, May 2017.

[208] S. Ha, S. Sen, C. Joe-Wong, Y. Im, and M. Chiang, "TUBE: time dependent pricing for mobile data," in *Proc. of Special Interest Group on Data Communication (ACM SIGCOMM)*. Helsinki, Finland, Aug. 2012.

[209] O. Semiari, W. Saad, M. Bennis, and Z. Dawy, "Inter-operator resource management for millimeter wave, multi-hop backhaul networks," *IEEE Transactions on Wireless Communications*, vol. 16, no. 8, pp. 5258–5272, Aug. 2017.

[210] N. Burlutskiy, M. Petridis, A. Fish, A. Chernov, and N. Ali, *An Investigation on Online Versus Batch Learning in Predicting User Behaviour*, Research and Development in Intelligent Systems XXXIII. Springer, 2016.

[211] L. Giupponi, R. Agusti, J. Perez-Romero, and O. Sallent, "Joint radio resource management algorithm for multi-RAT networks," in *Pro. of IEEE Global Telecommunications Conference (GLOBECOM)*. St. Louis, MO, Nov. 2005.

[212] M. Bennis, M. Simsek, A. Czylwik, W. Saad, S. Valentin, and M. Debbah, "When cellular meets WiFi in wireless small cell networks," *IEEE Communication Magazine*, vol. 51, no. 6, pp. 44–50, June 2013.

[213] M. El Helou, M. Ibrahim, S. Lahoud, D. Mezher, and B. Cousin, "A network-assisted approach for RAT selection in heterogeneous cellular networks," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 6, pp. 1055–1067, March 2015.

[214] S. Baban, D. Denkoviski, O. Holland, L. Gavrilovska, and H. Aghvami, "Radio access technology classification for cognitive radio networks," in *Pro. of IEEE International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. London, UK, Sept. 2013.

[215] Y. Cui, Y. Xu, R. Xu, and X. Sha, "A multi-radio packet scheduling algorithm for real-time traffic in a heterogeneous wireless network environment," *Information Technology Journal*, vol. 10, pp. 182–188, Oct. 2010.

[216] J. Perez-Romero, O. Sallent, R. Ferrus, and R. Agusti, "A robustness analysis of learning-based coexistence mechanisms for LTE-U operation in non-stationary conditions," in *Proc. of IEEE Vehicular Technology Conference (VTC Fall)*. Glasgow, Scotland, Sept. 2015.

[217] U. Challita, L. Dong, and W. Saad, "Proactive resource management in LTE-U systems: A deep learning perspective," *arXiv:1702.07031*, 2017.

[218] R. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, May 1992.

[219] R. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in Neural Information Processing Systems*, vol. 12, pp. 1057–1063, 2000.

[220] M. Agiwal, A. Roy, and N. Saxena, "Next generation 5G wireless networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 1617–1655, Feb. 2016.

[221] N. Kaminski, I. Macaluso, E. Di Pascale, A. Nag, J. Brady, M. Kelly, K. Nolan, W. Guibene, and L. Doyle, "A neural-network-based realization of in-network computation for the Internet of Things," in *Proc. of IEEE International Conference on Communications*, Paris, France, May 2017.

[222] S. R. Naidu, E. Zafiriou, and T. J. McAvoy, "Use of neural networks for sensor failure detection in a control system," *IEEE Control Systems Magazine*, vol. 10, no. 3, pp. 49–55, April 1990.

[223] H. Ning and Z. Wang, "Future internet of things architecture: Like mankind neural system or social organization framework?," *IEEE Communications Letters*, vol. 15, no. 4, pp. 461–463, March 2011.

[224] F. Alam, R. Mehmood, I. Katib, and A. Albeshri, "Analysis of eight data mining algorithms for smarter internet of things (IoT)," *Procedia Computer Science*, vol. 98, pp. 437–442, Dec. 2016.

[225] X. Luo, Y. Lv, M. Zhou, W. Wang, and W. Zhao, "A laguerre neural network-based ADP learning scheme with its application to tracking control in the internet of things," *Personal and Ubiquitous Computing*, vol. 20, no. 3, pp. 361–372, June 2016.

[226] L. Du, Y. Du, Y. Li, and M. C. F. Chang, "A reconfigurable streaming deep convolutional neural network accelerator for internet of things," *available online: arXiv:1707.02973*, July 2017.

[227] G. Karagiannis, O. Altintas, E. Ekici, G. Heijenk, B. Jarupan, K. Lin, and T. Weil, "Vehicular networking: A survey and tutorial on requirements, architectures, challenges, standards and solutions," *IEEE Communications Surveys & Tutorials*, vol. 13, no. 4, pp. 584–616, July 2011.

[228] J. Park, Z. Chen, L. Kiliaris, M. L. Kuang, M. A. Masrur, A. M. Phillips, and Y. L. Murphey, "Intelligent vehicle power control based on machine learning of optimal control parameters and prediction of road type and traffic congestion,"

IEEE Transactions on Vehicular Technology, vol. 58, no. 9, pp. 4741–4756, Nov. 2009.

[229] J. Morton, T. A. Wheeler, and M. J. Kochenderfer, "Analysis of recurrent neural networks for probabilistic modeling of driver behavior," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 5, pp. 1289–1298, Sept. 2017.

[230] N. Alsharif, K. Aldubaikhy, and X. S. Shen, "Link duration estimation using neural networks based mobility prediction in vehicular networks," in *Proc. of IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, Vancouver, BC, Canada, May 2016.

[231] M. Fathollahi and R. Kasturi, "Autonomous driving challenge: To infer the property of a dynamic object based on its motion pattern using recurrent neural network," *available online arXiv:1609.00361*, Sept. 2016.

[232] T. Dreossi, S. Ghosh, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Systematic testing of convolutional neural networks for autonomous driving," *available online arXiv:1708.03309*, Aug. 2017.

[233] V. Rausch, A. Hansen, E. Solowjow, C. Liu, E. Kreuzer, and J. Hedrick, "Learning a deep neural net policy for end-to-end control of autonomous vehicles," in *Proc. of American Control Conference (ACC)*, Seattle, WA, USA, May 2017.

[234] A. Gharaibeh, M. A. Salahuddin, S. J. Hussini, A. Khreishah, I. Khalil, M. Guizani, and A. Al-Fuqaha, "Smart cities: A survey on data management, security and enabling technologies," *IEEE Communications Surveys & Tutorials*, to appear 2017.

[235] A. Ferdowsi, A. Sanjab, W. Saad, and N. B. Mandayam, "Game theory for secure critical interdependent gas-power-water infrastructure," *available online arXiv:1707.04589*, 2017.

[236] F. Moretti, S. Pizzuti, S. Panzieri, and M. Annunziato, "Urban traffic flow forecasting through statistical and neural network bagging ensemble hybrid modeling," *Neurocomputing*, vol. 167, pp. 3–7, Nov. 2015.

[237] S. Campisi-Pinto, J. Adamowski, and G. Oron, "Forecasting urban water demand via wavelet-denoising and neural network models. case study: city of syracuse, italy," *Water Resources Management*, vol. 26, no. 12, pp. 3539–3558, July 2012.

[238] Y. Zheng, S. Rajasegarar, and C. Leckie, "Parking availability prediction for sensor-enabled car parks in smart cities," in *Proc. of IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, Singapore, April 2015.