Wireless Networks Design in the Era of Deep Learning: Model-Based, AI-Based, or Both?

Alessio Zappone, Marco Di Renzo, Merouane Debbah

▶ **To cite this version:**

Alessio Zappone, Marco Di Renzo, Merouane Debbah. Wireless Networks Design in the Era of Deep Learning: Model-Based, AI-Based, or Both?. IEEE Transactions on Communications, 2019, 10.1109/tcomm.2019.2924010 . hal-02001737

HAL Id: hal-02001737

https://centralesupelec.hal.science/hal-02001737

Submitted on 31 Jan 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Wireless Networks Design in the Era of Deep Learning: Model-Based, AI-Based, or Both?

Alessio Zappone, *Senior Member, IEEE*, Marco Di Renzo, *Senior Member, IEEE*, Mérouane Debbah, *Fellow, IEEE*

(Invited Paper)

***Abstract*—This work addresses the use of emerging data-driven techniques based on deep learning and artificial neural networks in future wireless communication networks. In particular, a key point that will be made and supported throughout the work is that data-driven approaches should not replace traditional design techniques based on mathematical models. On the contrary, despite being seemingly mutually exclusive, there is much to be gained by merging data-driven and model-based approaches.**

To begin with, a detailed presentation is given for the reasons why deep learning based on artificial neural networks will be an indispensable tool for the design and operation of future wireless communications networks, as well as a description of the recent technological advances that make deep learning practically viable for wireless applications. Our vision of how artificial neural networks should be integrated into the architecture of future wireless communication networks is presented, explaining the main areas where deep learning provides a decisive advantage over traditional approaches.

Afterwards, a thorough description of deep learning methodologies is provided, starting with presenting the general machine learning paradigm, followed by a more in-depth discussion about deep learning. Artificial neural networks are introduced as the peculiar feature that makes deep learning different and more performing than other machine learning techniques. The most widely-used artificial neural network architectures and their training methods will be analyzed in detail. Moreover, bridges will be drawn between deep learning and other major learning frameworks such as reinforcement learning and transfer learning.

After introducing the deep learning framework, its application to wireless communication is addressed. This part of the work first provides the state-of-the-art of deep learning for wireless communication networks, and then moves on to address several novel case-studies wherein the use of deep learning proves extremely useful for network design. In particular, the connection between deep learning and model-based approaches is emphasized, proposing several novel techniques for cross-fertilization between these two paradigms. For each case-study, it will be shown how the use of (even approximate) mathematical models can significantly reduce the amount of live data that needs to be acquired/measured to implement data-driven approaches. For each application, the merits of the proposed approaches will be demonstrated by a numerical analysis in which the implementation and training of the artificial neural network used to solve the problem is discussed.

Finally, concluding remarks describe those that in our opinion are the major directions for future research in this field.

A. Zappone and M. Debbah are with the Large Networks and Systems Group, CentraleSupelec, Université Paris-Saclay, 3 rue Joliot-Curie, 91192 Gif-sur-Yvette, France,(alessio.zappone@l2s.centralesupelec.fr, merouane.debbah@l2s.centralesupelec.fr). M. Debbah is also with the Mathematical and Algorithmic Sciences Lab, Huawei France R&D, Paris, France (merouane.debbah@huawei.com)

M. Di Renzo is with the Laboratory of Signals and Systems (CNRS - CentraleSupelec - Univ. Paris-Sud), Université Paris-Saclay, 3 rue Joliot-Curie, 91192 Gif-sur-Yvette, France, (marco.direnzo@l2s.centralesupelec.fr).

I. Introduction and Vision

All past and present generations of wireless communication networks are based on mathematical *models*, that are either derived from theoretical considerations, or from field measurement campaigns. Mathematical models are at the heart of all phases of network design, describing in quantitative terms the effect that each system component has on the overall performance. Mathematical models are used for initial network planning and deployment, for network resource management, as well as for network maintenance and control. Based on underlying models, infrastructure nodes are statically deployed to cover and manage fixed geographical areas, and traditional optimization theory is used to optimize the network performance through the centralized allocation of the available system resources. However, this traditional approach to network design has at least two drawbacks:

1) Depending on the complexity of the scenario, an accurate mathematical model might not be available. Moreover, even if available, every model is inherently an approximation, and a trade-off exists between the accuracy of the model and its complexity. Accurate models can be too complex to handle, whereas simple models can just be not accurate enough.
2) Static infrastructure deployment might not be flexible enough to adapt to heterogeneous service requirements and randomly evolving environments, with unpredictable on-demand connectivity requests.

The relevance of these two issues mostly depends on the complexity of the scenario in which the communication network must operate and on the performance level that must be guaranteed. In other words, the issues above can be ignored if the scenario allows the derivation of a mathematical model which is both accurate and tractable enough to meet the desired performance requirements. This has been the case for the past and present generations of wireless networks, from 1G to 4G systems. However, future networks (5G and beyond) are anticipated to witness an exponential complexity increase, due to the dramatic growth of connected devices as well as to the the rise of innovative vertical services with heterogeneous and stringent performance requirements.

Indeed, our society is undergoing a digitization revolution, with a dramatic increase of both Internet users and connected devices. It is forecasted that by 2020 over one billion people and more than 26 billion devices will be connected to the Internet, raising the number of connected devices by more

than 10 billion compared to 2015. This will exponentially increase global IP traffic, that is expected to grow at 22% Compound Annual Growth Rate (CAGR) in the time frame from 2015 to 2020 [1]. In this context, the lion's share will be for wireless communications, which are forecasted to account for 78% of Internet traffic already by 2020, with smartphones broadband connections representing alone 30% of total IP traffic, through both cellular and WiFi links [1]. However, for this vision to come true, wireless communications will have to surpass the performance of current fixed-line broadband connections, and this requires ubiquitous connectivity, 1000x higher data-rates, 2000x higher bit-per-Joule energy efficiency compared to today's levels [2]. In order to meet these requirements, innovative communication technologies are being developed and will be most likely implemented in future wireless networks, such as infrastructure densification, antenna densification, use of frequency bands in the mmWave range, energy-efficient network management, and so on [3]–[5]. It is widely believed that the joint use of these novel technologies makes 5G capacity and energy targets theoretically achievable, but on the other hand, their integration in future networks adds to the overall network complexity, leading to higher OPEX and CAPEX, which are already a major challenge in present wireless networks [6]. In addition, besides the communication performance requirements mentioned above, another critical challenge for future wireless networks is the heterogeneity of the services to provide. Future wireless networks will have to support many innovative vertical services, each with its own specific requirements [7], e.g.

- 1 ms end-to-end latency and reliability higher than 99.999% for Ultra Reliable Low Latency Communications (URLLC).
- Terminal densities of 1M terminal per square kilometer for massive Internet of Things (mIoT) applications.
- Per-user data-rate larger than 50 Mb/s for mobile broadband (mBB) applications.
- Terminal location accuracy of the order of 0.1 m for Vehicular-to-X (V2X) communications.

The integration of such diverse vertical services into the same network architecture calls for an extremely flexible and adaptive architecture, but this is in sharp contrast with the traditional "one-size-fits-all" approach adopted in present networks, where the control and data plane are bundled inside the network nodes. It is clear that just deploying more performing communication technologies does not ensure the flexibility required to accomodate diverse classes of users with extremely heterogeneous service requirements. Instead, new architectural and management solution are required.

A recently-proposed approach to tackle this issue is the network slicing paradigm, which proposes to logically separate the control and data plane, thus effectively slicing the physical network into multiple virtual networks co-existing over a common shared physical infrastructure. Each network slice constitutes a logically separate virtual network that can be customized to meet the specific requirements of different vertical services, by using techniques like Software Defined Networking (SDN) [8] and Network Function Virtualization (NFV) [9]. Network slicing applies to both the core and access network segments and paves the way for a new generation of programmable and software-oriented wireless networks, able to support flexible and on-demand network resources provisioning. Network slicing allows service providers to tailor resource use to the specific needs of the different classes of services to be provided. Moreover, parallel to software flexibility through slicing and reprogrammability, the physical network infrastructure will also be made flexible by using flying base stations implemented through drones and unmanned aerial vehicles that can be redeployed based on heterogeneous traffic conditions to support on-demand connectivity requests [10].

On the other hand, in order to fully exploit the potential of re-configurable and re-deployable wireless networks, it is necessary to overcome critical challenges related to network management and operation. In order to provide end-users with a perceived seamless and limitless connectivity, the reconfiguration of network resources and/or the re-deployment of network nodes in response to new data demands, as well as to connectivity problems and/or failures of hardware components, must be prompt and timely. To this end, it is necessary to make the network fully self-organizing, automating all management, operation, and maintenance tasks, limiting direct human intervention as much as possible. The concept of Self-Organzing Networks (SON) is not new to wireless networks. It was introduced by the Next Generation Mobile Networks (NGMN) alliance, and then standardized by 3GPP for LTE networks. However, despite having garnered much attention since its inception, SON failed to achieve the expected end-goal of fully automated networks, being used primarily for specific Radio Access Network (RAN) applications, but without providing a true end-to-end solution. In our opinion, the main reason why SON failed to deliver on its expectations is due to the lack of intelligence and cognition in past and present networks. In order to unlock the true advantages of SON and make future networks truly self-organizing, it is necessary to pair the SON paradigm with self-aware network nodes, capable of autonomous and intelligent behavior by sensing the surrounding environment and processing the acquired data. An automated network infrastructure with self-configuration and self-healing capabilities is expected reduce CAPEX and OPEX by a factor 5 relative to 2010 levels [11].

On the other hand, while intelligent software networks appear as the most promising approach for future wireless communication systems, they also pose strict security and privacy concerns, since all management processes are automated and carried out without close human supervision. In legacy mobile communications networks, telecom operators were responsible for user authentication only as far as network access is concerned, while the authentication between user and services was not provided by the network. Instead, in future networks, security and privacy are regarded as additional services to be provided by the network itself, which requires a cooperation between operators and vertical service providers to ensure a proper security management [12].

Summing up, the direction towards which future wireless communications are moving leads to an unprecedented level of complexity, which requires a radical paradigm shift in

the way networks are designed and operated, as far as both transmission technology and architecture are concerned. New transmission technologies and software-oriented architectures can help up address the **resource crunch**, (e.g. capacity, spectrum, energy shortage) of present wireless communications networks, but they come with a **complexity crunch** issue that must be overcome by future wireless communication networks. In other words, we are rapidly reaching the point at which the quality and heterogeneity of the services we demand of communication systems will exceed the capabilities and applicability of present approaches and models.

Our vision to overcome this situation is to resort to a *data-driven* paradigm for network design, in which the best policy to use does not *only* come from the analysis of a mathematical model, but *also* from the study and processing of previous communication data. In other words, based on the performance obtained by given policies during previous communication sessions, one should be able to decide what is the best policy to use at present. A framework that goes in this direction is that of *machine learning* and the focus of this work is on a specific machine learning approach, i.e. *deep learning*, which is *the* most widely-used learning technique in many fields of science, but which has started being brought to the attention of the communication community only very recently. Before introducing and further motivating the use of deep learning for communications, we should stress one important point. Although the main reason for using machine learning tools is to reduce the reliance of network design and operation on mathematical models, our vision is not for data-driven approaches to completely replace mathematical modeling and analysis. On the contrary, we believe that in order to overcome the complexity crunch, a cross-fertilization between model-based and data-drive approaches is necessary. Our vision, that will be supported throughout this work, is for deep learning and mathematical modeling to complement each other.

A. Deep Learning in Communications: Why Now?

As the name suggests, machine learning is a framework that aims at endowing computers with the ability to learn from data instead of being explicitly programmed [13]. Machine learning techniques are not new to communication systems, and indeed several machine learning approaches have been developed and proposed to aid the design and operation of communication systems, e.g. support vector machines, decision-tree learning, Bayesian networks, genetic algorithms, rule-based learning, and inductive logical programming, among others. Detailed surveys and tutorials about machine learning and its applications to wireless networks have appeared in [14]–[18], and its use to enable SON networks has been proposed in [19]. However, the focus of this work is on a specific machine learning technique, *deep learning* [20]–[22], that has started being envisioned for wireless communications only very recently.

Deep learning is a particular machine learning technique that implements the learning process elaborating the data through Artificial Neural Networks (ANNs). In the last few years, deep learning has become the most rapidly advancing component of machine learning, exhibiting far superior performance than other machine learning schemes, and is expected to become soon by far the most widely used machine learning method and the largest driver of revenue in the whole Artificial Intelligence (AI) field. Deep learning has been recognized as the first among the top ten AI technology trends for 2018 [23] and is already the leading machine learning technique in many scientific fields such as image recognition, text recognition, speech recognition, audio and language processing, robotics. Surprisingly, its use in communications systems has been envisioned only very recently [24], and its applications to wireless networks are almost unexplored. In our opinion, this is mainly due to the fact that, unlike other fields of science, communication engineers could always rely on mathematical models for system design. However, as we have described so far, this fundamental postulate is going to be significantly weakened in the near future, which puts forth the need for deep learning in communication system design. Moreover, there are also several recent technological advancements that make deep learning a viable technology for future communication networks:

- In order to gain the most out of deep learning algorithms, it is necessary to process large datasets. At present, exactly the exponential increase of wireless devices results in a corresponding growth of available data that can be used to this end [25]–[27]. The availability of large datasets is also considered an enabler for the implementation of the SON paradigm [25].
- Modern advancements in computing capacity makes it possible to execute larger and more complex algorithms much faster. In particular, Graphics Processing Units (GPUs) can repurposed to execute deep learning algorithms at speeds many times faster than traditional processor chips.
- The rise of the Blockchain technology provides an efficient way of guaranteeing the accuracy of data for a number of AI applications, both for feeding data into AI systems and for recording results from them [28]. Moreover, the Blockchain technology provides a distributed database that can be shared by all parties in a network [29]. This can be used to reduce the complexity of training the neural network, either by deploying dedicated computational centers which perform training tasks and then feedback only the resulting configuration parameters to the users, or by distributing the training complexity over the network nodes [30].

In our opinion the main question is not whether deep learning will be integrated in future wireless networks, but rather how and when this integration will take place. This statement is supported by the positions recently taken by several leading telecommunication companies, e.g. [31], [32], and by the fact that initial steps towards the standardization of intelligent wireless communication systems have already been taken. For example, in February 2017 the European Telecommunications Standards Institute (ETSI) activated an Industry Specification Group named *Experiential Network Intelligence*. Its purpose is to define a *cognitive network management* architecture capable

of using AI techniques and context-aware policies to adjust offered services based on changes in user needs, environmental conditions, and business goals. Such a paradigm is referred to as the *observe-orient-decide-act* control paradigm and represents the first standardization step towards the definition of an *experiential* system, i.e. a system that learns from previous experience to improve its knowledge of how to act in the future. This is anticipated to help operators automate their network configuration and monitoring processes, thereby reducing their operational expenditure and improving the use and maintenance of their networks.

Before concluding this section, we should emphasize that machine learning is a anticipated to be a game-changing technology not only for mainstream wireless communication networks, but also for emerging communication technologies that are being investigated as a way to complement traditional wireless approaches in specific scenarios. Among others, we mention here *optical communications* [33], [34], which promise very high data-rates by communicating over the visible spectrum, and *molecular communications*, which dispense with electromagnetic waves, relying on chemical signals as information carriers, thus enabling communication through media where electromagnetic signals do not propagate well, such as water or inside the walls of buildings [35], [36]. The interest in both technologies is rapidly increasing in recent years, but one main drawback is the difficulty of developing accurate models for such complex scenarios. Clearly, data-driven approaches might provide a decisive contribution to the practical implementation of optical and molecular communication systems. As a example, [37] employs deep learning to solve Schrödinger equations in fiber-optic communications.

B. Novelty and Organization

The vast majority of survey contributions on machine learning focus on different fields than wireless communication networks [13], [16], [20]–[22], [38], [39]). As far as wireless communications are concerned, most previous surveys consider other machine learning techniques [14], [17], [18], [40]–[42]), without discussing deep learning in detail. Only a few very recent overview works focus specifically on deep learning and ANNs for wireless communications [24], [43], [44]. All these three previous contributions envision the use of deep learning in future wireless networks, identifying AI as the key technology of the future and identifying many use-cases and scenarios in which deep learning has the potential of simplifying the design and improving the performance. On the other hand, compared to these previous efforts, our work provides the four major novelties (**N.1**-**N.4**), described below.

(N.1) The contributions in [24], [43], [44] focus on deep learning, but do not address how to use mathematical models to improve deep learning techniques. Instead, our work is the first to analyze the connection between model-based and data-driven methodologies. A systematic framework is developed for cross-fertilization between these two paradigms, by embedding the prior knowledge contained in available mathematical models into deep learning techniques.

(N.2) The contributions in [24], [43], [44] describe many potential and relevant applications of ANNs to wireless communications, but do not provide any quantitative analysis of how to practically use ANNs in these use-cases, and of the resulting performance. Instead, our work is the first to not only describe several wireless scenarios where deep learning proves useful, but also to provide, for each considered application, a mathematical formulation of the considered problem, the ANN architecture to be used and its training algorithm, plus a numerical analysis of the resulting performance.

(N.3) The contributions in [43], [44] provide a comprehensive survey of deep learning approaches to wireless communications, but do not tackle the mathematical theory of deep learning. More contribution in this sense can be found in [24], where a brief overview of ANNs architectures and training methods is presented. Instead, our work provides a solid and self-contained description of the theoretical fundamentals of deep learning, the most relevant ANNs architectures and training methods, and their fine-tuning for improved performance.

(N.4) The contributions in [24], [43], [44] do not describe in detail the relation between deep learning and other major learning frameworks, such as reinforcement learning and transfer learning. Instead, we provide a dedicated analysis of both **deep reinforcement learning** and **deep transfer learning**. Moreover, we also describe the approach of **deep unfolding**, that has been proposed as a way to map iterative algorithms to ANNs architectures.

The rest of this section explains our vision about how deep learning should be integrated into future networks with reference to all major phases of the design of a wireless communication network. Next, the remaining sections are organized as follows:

- Section II discusses in deeper detail the connection between machine learning and deep learning. First, the fundamental paradigms of supervised learning, unsupervised learning, and reinforcement learning are introduced, and then the position of deep learning and ANN in this general framework is explained.
- Section III addresses Novelties **N.3** and **N.4**, providing a theoretical description of deep learning, introducing the basic components of ANNs, the most widely-used ANN architectures and training methods and explaining the connection between deep learning and reinforcement learning and transfer learning, as well as the deep unfolding approach.
- Novelties **N.1** and **N.2** are addressed in Section IV. First, a detailed overview of the applications and research contributions of deep learning for wireless communications is provided. Next, several concrete examples and use-cases are presented, in which the joint use of mathematical models and deep learning methods provide huge gains compared to state-of-the-art approaches. For each use-case, a quantitative analysis is explicitly carried out, describing the design of an ANN to tackle the problem and discussing the resulting performance.

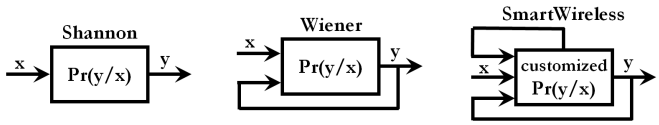

Figure 1. Current networks vs. a smart radio environment (or smart wireless).

- Finally Section V provides concluding remarks, outlining the major challenges to be overcome to fully enable the rise of deep-learning-based wireless communication networks.

C. Deep Learning for Network Deployment and Planning

Future wireless networks will be more than allowing people, mobile devices, and objects to communicate with each other [45]. Future wireless networks will be turned into a distributed intelligent wireless communications, sensing, and computing platform, which, besides communications, will be capable of sensing the environment to realize the vision of smart living in smart cities by providing them with context-awareness capabilities, of locally storing and processing information in order to accommodate the time critical, ultra-reliable, and energy efficient delivery of data, of accurately localizing people and objects in environments and scenarios where the global positioning system is not an option. Future wireless networks will have to fulfill the challenging requirement of interconnecting the physical and digital worlds in a seamless and sustainable manner [46], [47].

To fulfill these challenging requirements, we think that it is not sufficient anymore to rely solely on wireless networks whose *logical* operation is software-controlled and optimized [48]. The *wireless environment* itself needs to be turned into a software-reconfigurable entity [49], whose operation is optimized to enable uninterrupted connectivity. Future wireless networks need a smart radio environment, i.e., a wireless environment that is turned into a reconfigurable space that plays an active role in transferring and processing information. We refer to this emerging wireless future as "smart radio environment".

To better elucidate our notion of reconfigurable and programmable wireless environment, let us consider the block diagram illustrated in Fig. 1. Conceptually, the difference between current wireless networks and a smart radio environment can be summarized as follows. According to Shannon [50], the system model is given and is formulated in terms of transition probabilities (i.e., $\Pr\{y/x\}$). According to Wiener [51], the system model is still given, but its output is feedback to the input, which is optimized by taking the output into account. For example, the channel state is sent from a receiver back to a transmitter for channel-aware beamforming. In a smart radio environment, the environmental objects are capable of sensing the system's response to the radio waves (the physical world) and feed it back to the input (the digital world). Based on the sensed data, the input signal and the response of the environmental objects to the radio waves are jointly optimized and configured through a software controller, respectively. For example, the input signal is steered towards a given environmental object, which reflects, by employing some optimized phase shifts, it towards a given receiver that is, in turn, steered towards it.

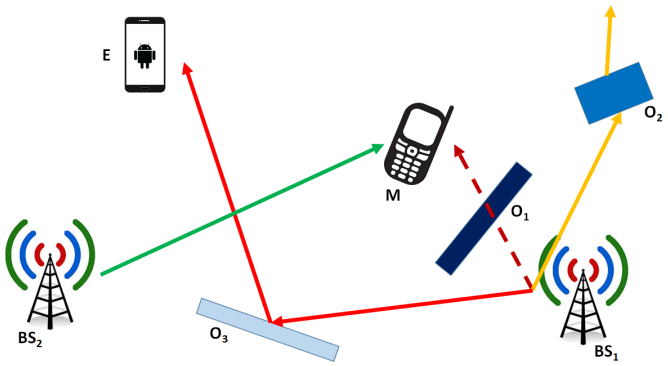

Figure 2. Current cellular networks operation.

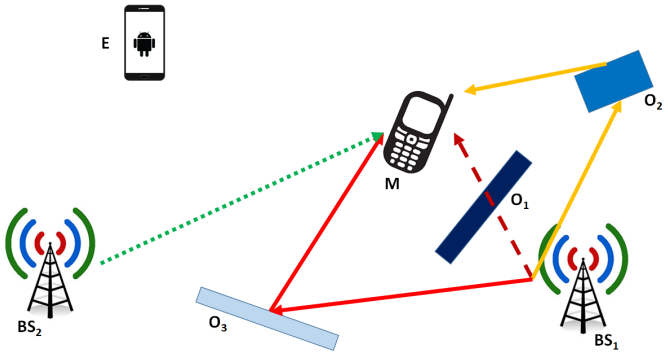

Figure 3. Cellular networks operation in a smart radio environment.

Different solutions towards realizing the vision of smart radio environments are currently emerging [52]- [53]. Among them, the use of reconfigurable metasurfaces constitutes a promising and enabling solution to fulfill the challenging requirements of future wireless networks [54]. Metasurfaces are thin metamaterial layers that are capable of modifying the propagation of the radio waves in fully customizable ways [55], thus owing the potential of making the transfer and processing of information more reliable [56]. Also, they constitute a suitable distributed platform to perform low-energy and low-complexity sensing [57], storage [58], and analog computing [59]. In [53], in particular, the authors have put forth a network scenario where every environmental object is coated with reconfigurable metasurfaces, whose response to the radio waves is programmed in software by capitalizing on the enabling technology and hardware platform currently being developed in [60].

An example of using reconfigurable metasurfaces in a cellular network scenario is sketched in Figs. 2 and 3. In Fig. 2, a mobile terminal (M) wants to connect to the Internet via a cellular network. In the absence of environmental objects

(O1, O2, O3), BS1 is the base station that provides the best signal to M. Due to the blocking object O1, however, the received signal from BS1 is not sufficiently strong, and M connects to the Internet via BS2, while BS1 is kept active to serve other users. Since BS2 is far from M, its received signal is not sufficient enough for high rate transmission. Because of the refractive object O2, the signal emitted by BS1 generates strong interfering signals in other locations. Also, the reflective object O3 generates a strong reflected signal towards a malicious user (E) that can intercept the signal from BS1. In Fig. 3, by contrast, we illustrate the operation of cellular networks in a smart radio environment. The objects O1, O2, O3 are now coated with reconfigurable metasurfaces that modify the radio waves according to the generalized laws of reflection and refraction [55]. Figure 3 shows how the operation of wireless networks changes fundamentally. The link BS1-M is still obstructed by O1. The responses of the reconfigurable metasurfaces on O2 and O3 are, however, appropriately controlled and optimized: O2 refracts the signal from BS1 towards M and avoids interfering other users. O3 reflects the signal towards M and protects BS1 against E. In contrast to Fig. 2, the reflected and refracted signals at M allow it to reliably connect to the Internet. Now, BS2 serves other users at, e.g., a higher speed.

Current research efforts towards realizing the vision of smart radio environments are primarily focused on implementing hardware testbeds, e.g., reflect-arrays and metasurfaces, and on realizing point-to-point experimental tests [52]- [53]. To the best of the authors knowledge, on the other hand, there exist no theoretic and algorithmic methodologies that provide one with the ultimate performance limits of this emerging wireless future, and with the algorithms and protocols for achieving those limits. We argue, in addition, that the design of smart radio environments is unlike to be possible by relying solely on conventional methods. We believe, on the other hand, that deep learning and artificial intelligence will play a major role in this context. In the following two sections, we will first discuss in deeper details the difference and potential advantages of smart radio environments against current wireless network solutions, and then discuss the importance of deep learning in this context.

1) Current Networks vs. Future Smart Radio Environments: To better elucidate the difference and significance of smart radio environments with respect to the most advanced technologies employed in wireless networks at present, let us consider, as an example, a typical cellular network.

The distinguishable feature of cellular networks lies in the users' mobility. The locations of the base stations cannot, in general, be modified according to the user's locations. Some exceptions, however, exist [61], [62], and we will elaborate on this below. The mobility of the users throughout a location-static deployment of base stations renders the user distribution uneven throughout the network, which results in some base stations to be severely overloaded and some others to be under-utilized. This is a well-known issue in cellular networks, and is tackled in different ways, among which load balancing methods [63] and the densification of base stations (ultra-dense networks). Network densification is certainly a promising approach, but has its own limitations [64], [65]. It is known, e.g., that network densification increases the network power consumption as the number of base stations per square kilometer increases. This is exacerbated even more with the advent of the Internet of Things (IoT), where the circuit power consumption increases with the number of users per square kilometer [66], [67]. Ultra-dense network deployments, in addition, enhance the level of interference, which needs to be appropriately controlled in order to achieve good performance. Furthermore, each base station necessitates a backhaul connection, which may not always be available. Other solutions based on massive Multiple-Input-Multiple-Output (MIMO) schemes could be employed, but they usually necessitate a large number of individually controllable radio transmitters and advanced signal processing algorithms [68]. Their performance, in addition, is determined by the existence of favorable propagation conditions, which may not always be fulfilled in cellular networks. Similar comments (i.e., power consumption, hardware complexity, blocking of links, etc.) apply to using millimeter-wave communications [69], [70]. It is worth mentioning that millimeter-wave systems can take advantage of the presence of reconfigurable metasurfaces as a source of controllable reflectors that can overcome non-line-of-sight propagation conditions, and enable the otherwise impossible communication between the devices [71]. Without pretending to be exhaustive, other relevant solutions that are typically employed in wireless encompass retransmission methods that negatively impact the network spectral efficiency, the optimized deployment of specific network elements, e.g., relays, which increase the network power consumption as they are made of active elements (e.g., power amplifiers), and that either reduce the achievable link rate if operate in half-duplex mode or are subject to severe self-interference if operate in full-duplex mode [72]- [73].

Metasurfaces-enabled smart radio environments are fundamentally different. The metasurfaces are made of low-cost passive elements that do not require any active power sources for transmission [47]. Their circuitries can be powered with energy harvesting modules as well [74]. They do not apply any sophisticated signal processing algorithms (coding, decoding, etc.), but primarily rely on the programmability and re-configurability of the metasurfaces and on their capability of modifying the radio waves impinging upon them [75]. They can operate in full-duplex mode without significant or any self-interference, and do not need any backhaul connections to operate. Even more importantly, the metasurfaces are deployed where the issue naturally arises: Where the environmental objects, which, in current wireless networks, reflect, refract, distort, etc. the radio waves in undesirable and uncontrollable ways, are located. Since the input-output response of the metasurfaces is not subject to conventional Snell's laws anymore, the locations of the objects that assist a pair of transmitter and receiver to communicate, and the functions that they apply on the received signals can be chosen to minimize the impact of multi-hop-like signal attenuation. In addition, the phase of the many atomic elements that constitute the metasurfaces can be optimized to coherently focus the waves towards the intended destination, thus obtaining a substantial beamforming

gain without using active elements. These functionalities, in addition, are transparent to the users, as there is no need to change the hardware and software of the devices. Furthermore, the number of environmental objects can potentially exceed the number of antennas at the endpoint radios, which implies that the degrees of freedom for system optimization can potentially exceed that of current wireless network deployments [76]. The freedom of controlling the response of each metasurface and choosing their location via a software-programmable interface makes, in addition, the optimization of wireless networks agnostic to the underlying physics of wireless propagation and metamaterials. Despite the practical challenges of deploying robotic (terrestrial) base stations capable of autonomously moving throughout a given region [61], [62], experimental results conducted in an airport environment, where the base stations were deployed on a rail located in the ceiling of a terminal building [77], showed promising gains. The possibility to deploy mobile reconfigurable metasurfaces is, on the contrary, practically viable. The metasurfaces can be easily attached to and removed from objects (e.g., facades of buildings, indoor walls and ceilings, advertising displays), respectively, thus yielding a high flexibility for their deployment. The position of small-size metasurfaces on large-size objects, e.g., walls, can be adaptively optimized as an additional degree of freedom for system optimization: Thanks to their 2D structure, the metasurfaces can be mechanically displaced, e.g., along a discrete set of possible locations (moving grid) on a given wall. It is apparent, therefore, that the concept of smart radio environment can potentially impact wireless networks immensely.

2) The Need for Deep Learning: In summary, the concept of smart radio environment is a fundamental paradigm shift with respect to the design of current wireless networks. In current wireless networks, broadly speaking, the environment, i.e., the set of physical objects that alter the propagation of radio waves, is not controllable. The environment ignores the underlying process of transferring and processing information, and is perceived, in addition, as an adversary to the communication process, i.e., it has usually a negative effect that needs to be counteracted by the transmitters and receivers [78]. The advent of reconfigurable metasurfaces, reconfigurable reflectarrays, reconfigurable large-intelligent surfaces, etc. challenges this status quo, but asks for new methods for modeling, analyzing, and optimizing wireless networks.

The design of wireless networks, in fact, relies either on analytical models or on system-level simulations. The use of models has the advantage of yielding deep insight into the system behavior and of allowing one to formulate optimization problems that can often be solved in a computationally efficient manner and that often lead to system designs with proved optimality properties. In general, however, analytical models are seldom sufficient for accurate system design and optimization in complex wireless networks. For example, cellular networks are often modeled by relying on the abstraction model based on Poisson point processes [79], [80]. This approach is analytically tractable, but it may not be sufficiently accurate to capture important details in practical cellular network deployments. It is known, on the other hand, that using spatial models different from Poisson leads to complicated utility functions that are often difficult to interpret and to optimize [81], [82]. The use of system-level simulators usually leads, by contrast, to accurate system designs, but it offers limited insight into identifying optimal deployments, and developing optimal algorithms and protocols. The optimization of complex (large-scale) networks based solely on brute-force numerical methods may, in addition, be prohibitive due to the very large number of variables to optimize.

By direct inspection of Fig. 1, smart radio environments are, without any doubts, an emerging network architecture that is much more difficult to optimize than current wireless networks, and whose ultimate performance limits are unknown to date. In a smart radio environment, the operation of each environmental object may be optimized besides the operation of the transmitter and receiver (the end points of the network). Accurately modeling such an emerging network scenario and optimizing it in real time and at a low complexity is an open issue. As far as the modeling is concerned, it is very challenging to devise a model that is sufficiently accurate to account for customizable reflections, refractions, blocking, displacements of the surfaces, etc. As far as the optimization is concerned, event if such a model could be developed, it would be very unlikely amenable for optimization due to the large number of variables to optimize and the complexity of the resulting utility functions. Compared with current network models, in addition, Fig. 1 highlights that smart radio environments need much more context-aware information for configuring and optimizing the operation of all the environmental objects, which results in a larger feedback overhead that has a strong impact in applications with high mobility. This implies that optimizing smart radio environments based only on empirical data, i.e., data-driven approach, may not be, in general, a practically affordable solution.

Motivated by these considerations, we argue that the optimal design of smart radio environments need to be tackled by taking the benefits of both model-based and data-driven (or simulation-driven) approaches, by leveraging the concept of *transfer learning* [83]. In simple terms, our idea consists of first optimizing the network using a mismatched, but simpler for optimization, model, and then refining the result with (fewer) empirical data. We believe that this approach constitutes an enabler for recovering the model mismatch that originates from the difficulty of accurately modeling the system, and, at the same time, the difficulty of collecting a large amount of empirical samples for data-driven optimization because of the associated overhead, e.g., for reporting large amounts of sensed data from the metasurfaces to the network controller in charge of optimizing the network.

In the sequel, we will elaborate on the concept of transfer learning, and we will provide some examples of application towards the design and optimization of wireless networks, which synergistically leverage model-based and data-driven methods taking the best of both worlds.

D. Deep Learning for Network Resource Management

The goal of resource management is to allocate the available network resources in order to maximize one or more

performance metrics. Transmit powers, beamforming vectors, receive filters, frequency chunks, computing power, memory space, etc., can be scheduled among the network terminals based on traffic demands, propagation channel conditions, terminals requirements, so as to optimize the network throughput, the communication latency, the energy efficiency, while at the same time ensuring that all end-users experience the guaranteed quality-of-service (QoS). Formally speaking, denoted by f the performance function to maximize and by $\mathbf{x} \in \mathcal{S}$ the resource to allocate, with $\mathcal{S}$ the set containing the admissible values of $\mathbf{x}$, the resource allocation problem can be cast as the optimization program

$$\max_{\mathbf{x}\in\mathcal{S}} f(\mathbf{x}) \ . \tag{1}$$

Thus, the conventional approach to resource management is based on the use of traditional optimization theory techniques. However, as already mentioned, this approach only works if one is able to come up with a suitable mathematical model of the problem, i.e. with tractable, but accurate, formulas describing the objective f and the feasible set $\mathcal{S}$. Moreover, even assuming that this can be done, the optimal resource allocation will inevitably depend on the system parameters, e.g. the users' positions, the number of connected users, slow-fading or fast fading channel realizations. Anytime one of these parameters changes, which happens quite frequently in mobile environments, the optimization problem needs to be solved anew. This causes a significant complexity overhead, that limits the real-time implementation of available optimization frameworks, especially in large and complex systems like future wireless communication networks.

The use of deep learning techniques based on ANNs provides a novel approach to tackle this issue and enable true online resource management in wireless communication networks. The idea is that the general resource allocation problem in (1) can be regarded as an unknown function mapping from the ensemble of all network parameters of interest, denoted by $\boldsymbol{c} \in \mathbb{R}^N$, with N the number of system parameters of interest, to the corresponding optimal resource allocation $\mathbf{x}^* \in \mathcal{S}$. Formally, we can view Problem (1) as the non-linear map

$$\mathcal{F} : \boldsymbol{c} \in \mathbb{R}^N \rightarrow \mathbf{x}^* \in \mathcal{S} \subseteq \mathbb{R}^N \ . \tag{2}$$

Thus, we have converted Problem (1) into learning the unknown map (2), a task that ANNs are able to tackle. Indeed, as it will be discussed in deeper detail in Section II, ANNs are, under very mild assumptions, universal approximators, i.e., if properly trained, they are able to learn the input-output relation between the system parameters and the desired resource allocation to use, thus emulating the function $\mathcal{F}$ in (64). This means that we can optimize a desired performance function for given system parameters without explicitly having to solve any optimization problem, but rather letting an ANN compute the resource allocation for us. A detailed analysis of this approach will be presented in Section IV.

At this point, the natural question is how to integrate ANN-based resource management into the topology and architecture of a wireless network. Where should we store the data required by the ANN tasked with network resource management, and where should the related computations be executed? Ideally, the optimal approach would be to have a cloud-based approach in which an "artificial brain" placed in a single point oversees all tasks related to resource management across the whole network or at least a network segment. All available data should be stored in this artificial brain which is tasked with executing all required computations and with feeding back the resulting resource allocation to all other network terminals. Unfortunately, such a centralized approach is not compatible with future wireless networks due to at least three major reasons:

1) **Latency.** Some vertical sectors of future wireless networks, e.g. URLLC, require a strict end-to-end communication latency, lower than a millisecond. Thus, for these applications, it is not possible to wait for the cloud to perform the computations and then feed back the results. Instead, the computations should be performed locally by each user equipment.
2) **Privacy.** Unlike previous wireless networks generations, future wireless networks will not be simply about having a faster mobile network or richer functions in smartphones. The integration of innovative vertical services aims at making the vision of the "everything connected world" true, but this comes with critical privacy and security requirements. Accordingly, for some vertical applications it is not desirable to share information with the cloud, which makes cloud-based deep learning not a viable approach.
3) **Connectivity.** Future wireless networks promise ubiquitous service delivery. This means that a user terminal should be able to operate also in areas or times in which a poor connection to the cloud exists. This requirement is not compatible with a pure cloud-based implementation, but instead each user device should have some "local intelligence" to be able to operate in these scenarios, too.

Therefore, in order to make deep learning compatible with future wireless communication networks, the intelligence can not be concentrated only in a centralized network brain. Instead, some intelligence should be distributed across the network mobile devices, implementing a ***Mobile AI*** architecture. It is interesting to observe that this approach resembles the way in which human knowledge is developed: like human societies in which there is a collective intelligence that belongs to everybody, and an individual intelligence, the mobile AI paradigm envisions both a *cloud intelligence*, which every node of the network can access by connecting to the cloud, and a *device intelligence* specific to each network device.

In order to implement this mobile AI paradigm, a first natural approach that we put forth is to regard each device in the network as a rational and independent decision-maker, which acquires its own local dataset and uses it to build its own local ANN model. This techniques does not require any interaction between the network infrastructure and the edge users, as far as data sharing and processing are concerned, and has the potential of enabling the 5G vision of distributed, self-managing networks true. On the other hand, due to limited

storage and processing capabilities, mobile devices might not be able to develop accurate models on their own and the resulting performance gap must be analyzed. Moreover, the self-organizing nature of the devices poses questions about reaching a stable network operating point and about the efficiency of such point. The Noble-prize-winner framework of game theory appears as the natural way to answer at least the last points, as it provides sophisticated mathematical tools to analyze the interactions among independent decision-makers [84]–[86]. Game theory has been already extensively used for resource management in wireless communication networks [18], [87], [88], although never in connection with deep learning.

A second approach that we envision is based on the use of the so-called *federated learning* technique [89], [90]. The main idea of federated learning is to distribute the data and computation tasks among a federation of local devices that are coordinated by a central server. The server owns a global ANN model that is built by integrating the local models from the devices, which are developed based on local datasets. Thus, what is uploaded to the server is only the update to the global model, but not the local datasets themselves. By this approach, the individual intelligence owned by each device contributes to the collective intelligence of the whole federation of devices, which is maintained by the server. As a refinement of this approach, [91] proposes to exchange not the updates to the model, but rather the updates to the algorithm that is used to compute the model. In other words, each local model is computed by processing the local dataset by some algorithm, and what the devices communicate to the server is not the model itself, but instead an update of the parameters of the algorithm that is used to compute the global model.

E. Deep Learning for Network Operation and Maintenance

Maintenance and operation of a wireless network is a broad field that involves many different tasks, such as users' localization, channel estimation, quality-of-service monitoring, fault and anomaly detection, hand-over execution, intrusion detection, etc. Although seemingly quite diverse, operation and maintenance tasks have a common denominator, all involving the acquisition of some measurable data, from which the desired information must be extracted. Formally speaking, all above tasks can be formulated as the task of guessing the realization of some random vector $\mathbf{x}$ based on the observation of another random vector $\boldsymbol{y}$, that is somehow correlated to $\mathbf{x}$, i.e. that was generated from $\mathbf{x}$ through some unknown transformation. Such a problem can be cast into the framework of classical decision and estimation theory, but classical detection and estimation methods require the conditional distribution $f(\mathbf{x}|\boldsymbol{y})$ and the prior distribution $f(\mathbf{x})$, whose availability is strongly related to the availability of a tractable model for the specific problem at hand. Even in present wireless applications, this is an unrealistic assumption for several operation and maintenance tasks. A notable example is that of hand-overs of users moving along the boundary of two cells, a crucial problem in cellular networks. This is typically heuristically handled by comparing the users' signal-to-noise ratio (SNR) towards the neighboring cells over a given time window. However, deriving a statistical model for this scenario that accounts for the users' mobility patterns is quite challenging, and indeed the optimization of the thresholds for hand-overs is an open problem even in present cellular communications. Given the foreseen complexity increase in future wireless communication networks, statistical approaches will become less and less practical.

A suitable way of coping with the lack of models and statistical information as to the random vectors $\mathbf{x}$ and $\boldsymbol{y}$ is represented by machine learning. Indeed, operation and maintenance is probably the field of wireless communications in which machine learning approaches have been used first. Recent surveys on applications of machine learning for maintenance tasks have appeared in [92]–[95], showing how machine learning approaches perform well even without any statistical distribution information. Specifically, they assume that a training set containing examples of correct matches between the realizations of $\mathbf{x}$ and $\boldsymbol{y}$ is available, which can be built off-line by simply observing and storing previous traffic data. By elaborating the training set, machine learning methods are able to devise a rule for learning the value of $\mathbf{x}$ corresponding to yet unobserved values of $\boldsymbol{y}$.

As far as the integration of deep learning for network maintenance into future wireless architectures is concerned, it is our opinion that it could be carried out following a more centralized approach than for the resource management scenario described in Section I-D. Indeed, most operation and maintenance tasks (e.g. fault and anomaly detection, hand-overs, intrusion detection) are inherently centralized in the sense that all computations are executed by network infrastructure nodes and do not require any specific information exchange with edge-users. On the other hand, in case of very large datasets and very demanding computations to perform, we envision the use of a distributed or federated approach, but only among dedicated network nodes. More in detail, a suitable approach appears to share storage and computation tasks among a cluster of fixed infrastructure nodes connected by high-speed links and deployed in different points of the network. Then, each node of the cluster could either be tasked with operating and maintaining only a specific part of the network, or the data and computing power of each cluster node could be jointly exploited in a federated approach.

II. Machine Learning and Deep Learning: What is New?

The term *machine learning* broadly refers to algorithmic techniques able to perform a given task without running a fixed computer program explicitly written and designed for the problem at hand, but instead processing available data and progressively learning from it. Formally speaking, a computer program is said to learn from experience **E** with respect to a task **T** and performance measure **P**, if its performance at task **T**, as measured by **P**, improves with experience **E** [96].

The tasks that can be solved by machine learning are very diverse. In general, machine learning techniques prove extremely useful for all tasks for which no explicit and/or

viable programming approach exists to date, e.g. classification, regressions, pattern recognition, automatic language translation, anomaly detection, etc. As diverse as the task to perform may be, a machine learning algorithm can be mathematically described as the map

$$\mathcal{F} : \mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^n \rightarrow \boldsymbol{y} \in \mathcal{Y} \subseteq \mathbb{R}^m \ , \tag{3}$$

wherein $\mathbf{x}$ is a data vector whose components are the *features* describing the task to be solved, $\boldsymbol{y}$ is the output produced by the machine learning algorithm representing the answer to the problem at hand, $\mathcal{X}$ and $\mathcal{Y}$ are the sets in which $\mathbf{x}$ and $\boldsymbol{y}$ may vary. It is important not to confuse the task performed by a machine learning technique with the action of learning. The former is the final objective of the algorithm, while the latter is the method that is used to carry out the task.

In order to evaluate the ability of a machine learning algorithm to solve the assigned task, i.e. to produce output vectors close to the desired ones, a performance criterion $\mathbf{P}$ must be defined. Several performance measures can be considered and typically the best choice is application-dependent. However, two general performance functions can be identified, namely:

- The mean square error (MMSE) between the output $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$ produced by the algorithm for some input points $\mathbf{x}_1, \ldots, \mathbf{x}_N$, and the corresponding desired output $\boldsymbol{y}_1^\star, \ldots, \boldsymbol{y}_N^\star$.
- The cross entropy between the output produced by the algorithm $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$ for some input points $\mathbf{x}_1, \ldots, \mathbf{x}_N$, and the desired output $\boldsymbol{y}_1^\star, \ldots, \boldsymbol{y}_N^\star$. The justification of this performance function stems from the fact that it is equal, up to the entropy of the desired output, to the Kullback-Leibler divergence between the algorithm output and the desired output.

The last component of a machine learning algorithm to be introduced is the experience $\mathbf{E}$, i.e. the knowledge and data that the algorithm can exploit to carry out the task. Machine learning algorithms typically experience a set of data points $\mathcal{S}_{TR}$, called **training set**. Depending on the information contained in $\mathcal{S}$, machine learning algorithms can be grouped into two main categories:

- **Unsupervised learning**: the experienced data training set $\mathcal{S}_{TR}$ contains only input features, i.e. $\mathcal{S}_{TR} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$. Based on $\mathcal{S}_{TR}$, the machine learning algorithm must be able to extrapolate the statistical structure of the input or any other information needed to carry out the desired task.
- **Supervised learning**: the experienced data training set $\mathcal{S}_{TR}$ contains both input features and the corresponding desired outputs, referred to as *labels* or *targets*, i.e. $\mathcal{S} = \{(\mathbf{x}_1, \boldsymbol{y}_1), \ldots, (\mathbf{x}_N, \boldsymbol{y}_N)\}$. Thus, in supervised learning, the training set provides a series of examples to instruct the algorithm how to behave when some specific inputs are considered.

In both supervised and unsupervised learning, the available dataset is fixed. This models a scenario in which the algorithm does not directly interact with the environment where it operates. Instead, a different machine learning paradigm that does not fall in the categorization above is that of **reinforcement learning** [97]. The approach of reinforcement learning is to enable a feedback loop between the algorithm and the environment, allowing the algorithm to experience a dataset that changes over time as a result of the interaction with the surrounding environment. The focus of this work will be primarily on supervised learning, which is the typical approach in deep learning. Reinforcement learning will also be considered, primarily considering its integration with deep learning tools, which leads to the recently introduced paradigm of **deep reinforcement learning** [98], [99].

Before continuing, it is important to remark that, while the setting described above bears some resemblance to the general problem of classical decision/estimation theory, a fundamental difference exists. Classical decision/estimation theory assumes that the probability distributions of the output vector given the input $p(\boldsymbol{y}|\mathbf{x})$ and that of the input vector $p(\mathbf{x})$ are known. Instead, machine learning does not need this assumption and is able to operate based only on some realizations of the underlying distributions, even though the distributions themselves are not known.

A. Overfitting and Underfitting

We have seen how a machine learning algorithm experiences a training set $\mathcal{S}_{TR}$ which, in any case, contains some input features $\mathbf{x}_1, \ldots, \mathbf{x}_N$. In the supervised scenario, each input feature is also accompanied by the corresponding desired output. While this information is essential to configure the learning scheme, the key problem of any machine learning algorithm is to perform well on *previously unseen* inputs. This means that the algorithm should be able to grasp from $\mathcal{S}_{TR}$ a general rule to produce a suitable output $\boldsymbol{y}$ also when $\tilde{\mathbf{x}} \notin \mathcal{X}$. This is referred to as the algorithm *generalization capability*. During the training phase, the information in the training set is used to set the algorithm parameters in order to minimize any desired performance metric. As it will be seen in the sequel, this basically amounts to solving an optimization problem, but what makes machine learning fundamentally different from optimization theory, is that the ultimate goal is to make the algorithm able to generalize well to new data inputs. In order to evaluate its generalization capability, after the algorithm has been designed as a result of the training phase, its performance is tested over a new set of different inputs $\mathcal{S}_T$, called the **test set**. For any given error measure, the error evaluated over the test set is called *generalization error* or *test error*. Similarly, the error evaluated over the training set is called the *training error*. Clearly, in order for the algorithm to generalize well, the data samples in the training set $\mathcal{S}_{TR}$ and in the test set $\mathcal{S}_T$ need to be drawn from the same distribution, called *data generating distribution*, even though they should be drawn independently with each other. Clearly, the expected generalization error will be larger than the expected training error, and the gap between the two is called the *generalization gap*. Thus, minimizing the training error can be regarded as a necessary but not sufficient condition to obtain also a low generalization error. A machine learning algorithm is said to be:

- **Underfitting** if it is not able to make the error over the training set small.

- **Overfitting** if it is not able to make the gap between the training and test error small.

The factor that controls whether overfitting or underfitting occurs is the **capacity** of the algorithm, i.e. the ability of the algorithm to properly fit the training set. Intuitively, the capacity of the algorithm is related to the degrees of freedom or parameters that can be chosen when designing the algorithm. Clearly, if the algorithm does not have enough free parameters, it will not have enough degrees of freedom to capture the structure of the training set and the algorithm will underfit. Instead, the overfitting scenario is more subtle. One may think that increasing the number of free parameters will always lead to better performance, and that an upper limit is represented only by the computational complexity that we are prepared to sustain. Actually, this is not the case. If the algorithm has too many degrees of freedom, it will learn the structure of the training set too well, memorizing specific properties that are peculiar only to the training set, but that do not hold in general. As a result, there is an optimal capacity that a machine algorithm should have to minimize the generalization gap.

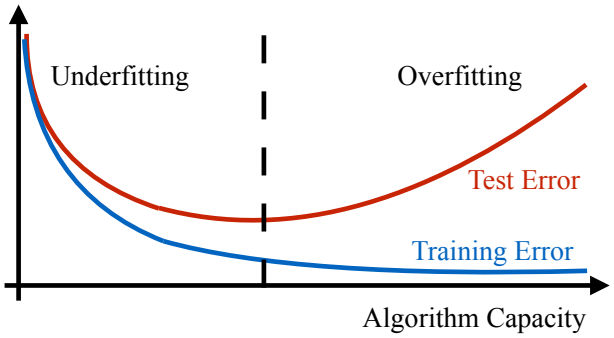

Figure 4. Typical behaviors of the training and test errors.

As shown in Fig. 4, the training error decreases with the algorithm capacity, asymptotically reaching its minimum value. Instead, the test error has a U-shaped behavior, following the training error up to a capacity value, and then increasing, thereby originating the generalization gap. Fundamental results from statistical learning theory have established that the generalization gap is bounded from above, with the upper bound increasing for larger model capacity, and decreasing for larger training sets [100]–[103]. On the other hand, a lower-bound to both the training and test error is given by the well-known Bayes error, i.e. the error obtained by an oracle with access to the true underlying distribution sampling from which the training and test set are obtained.

Another way to interpret the phenomenon of overfitting is to observe that any finite training set will also contain atypical realizations of the underlying distribution, that should be overlooked or given little importance when adjusting the algorithm parameters. However, having too many parameters to design, the algorithm will try to perfectly fit the complete training set, thus originating the overfitting phenomenon. This concept is illustrated in the example shown in Fig. 5, where it is assumed that a machine learning classifier must output a decision boundary to separate objects belonging to two different classes. It can be seen how a linear decision boundary is not able to properly separate the samples in the training set, thus causing underfitting. On the other hand, having enough degrees of freedom, one can design a complex boundary to perfectly separate the samples in the training set, even those samples that happen to be surrounded by samples of the other class. However, this leads to including in both decision regions areas that are likely to contain samples from the wrong class, thus causing overfitting. Instead, the curved, but more regular, decision region in the middle better captures the structure of the underlying distribution.

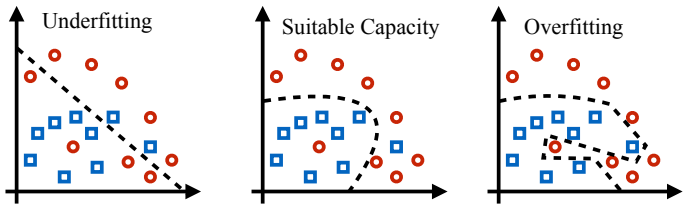

Figure 5. Three possible decision boundaries for a classification problem. The left and right figures show the underfitting and overfitting scenarios. The middle figure shows classifier with the proper capacity.

It is interesting to observe that choosing the decision boundary in the middle illustration of Fig. 5 is in agreement with the Occam's razor principle, stating that among different and equally motivated explanations of a phenomenon, one should choose the simplest one. Of course one should also be careful not to oversimplify the model, ending up in the underfitting regime.

As mentioned above, one of the fundamental features that distinguishes machine learning theory from classical decision theory is the fact that the distribution underlying the task to perform is not known. This could lead to the belief that machine learning algorithms are universal, in the sense that the attainable performance depend only on how the parameters of the algorithm are set and on the size of the training set, but not on the properties of the underlying distribution, and, thus, on the task to perform. Unfortunately, this belief is disproved by a fundamental result of machine learning, known as the *no free lunch theorem*, basically stating that the test error of any machine learning algorithm is the same when averaged over all possible underlying distributions. This means that there exists no machine learning algorithm that outperforms any other algorithm at every possible task. Instead, different algorithms will achieve different performance when applied to tackle different tasks, i.e. when the underlying distribution varies.

B. Hyperparameters and Validation Set

Parallel to the parameters that are to be optimized by the training procedure, machine learning algorithms also have hyperparameters, i.e. parameters that are not directly set during the training phase, either because they are difficult to optimize, or because they should not be learnt from the training set. The latter is the case with all parameters that directly affect the capacity of the model. Indeed, if a parameter that affects the model capacity should be tuned based only on the training set,

the result will be that the parameter should be set in order to minimize the training error as much as possible. However, we have seen how this would lead to a poor generalization error, due to overfitting.

To be more specific, anticipating some notions about ANN to be discussed in deeper detail in the next section, an ANN is composed of several nodes whose input-output relationship is defined by some weights and bias terms, which are the parameters to be tuned during the training phase. On the other hand, the total number of nodes in the network and the way in which the nodes are interconnected are hyperparameters that are considered fixed while the training algorithm is executed. Besides the difficulty to optimize these discrete parameters, a critical to problem is that the number of nodes in an ANN is directly related to the capacity of the network, since more nodes imply more degrees of freedom. Therefore, if we set this parameter based on the training set, the answer will be to use as many nodes as physically possible, thus causing overfitting.

On the other hand, it is also not possible to use the test set to perform hyperparameter tuning, because all choices pertaining to the algorithm to be designed must be independent of the data set that is used to assess the algorithm performance. Otherwise, the estimation of the generalization error will be biased. This means that we need a third data set for the purpose of hyperparameter tuning, the **validation set**. The validation set is typically obtained by partitioning the training data into the training set and the validation set. The training procedure fixes some values of the hyperparameters and optimizes the network parameters based only on the training set. Afterwards, an estimate of the generalization error obtained with the considered hyperparameter configuration is obtained through the validation set. This procedure is repeated for different hyperparameter configurations to identify the best model to use. After both the parameters and hyperparameters have been set, the true generalization error is computed using the test set. The main steps of the whole procedure are summarized in Algorithm 1.

Algorithm 1 Hyperparameter and parameters tuning

while Error on validation set not satisfactory **do**
  Choose a set of hyperparameters;
  Given the chosen hyperparameters run the learning procedure for parameter optimization using the training set;
  Evaluate the error on the validation set;
end while

While Algorithm 1 provides a systematic way for training a machine learning algorithm, it does not address how to update the hyperparameter configuration in each loop. In general, there is no simple, algorithmic way to do this, and indeed hyperparameter tuning is more an art than a science. In particular, manual hyperparameter tuning is specific to the task to carry out and some guidelines will be discussed for the specific case of deep learning in Section III-C2. Nevertheless, three systematic approaches for automated hyperparameter selection, that are general enough for many machine learning techniques, can be identified as follows:

- If the complexity of running the training procedure for a given hyperparameter configuration allows it, hyperparameters can be learnt by means of a grid search.
- As a variation of the grid search, a random search has been shown to provide good performance, while at the same time significantly reducing the overall complexity [104].
- A nested learning procedure can be used, in which a second machine learning algorithm is wrapped around the algorithm to be trained, with the task of learning the best hyperparameters for the inner algorithm.

C. Beyond classical machine learning

So far, the general principles at the basis of machine learning have been introduced, and some well-established machine learning algorithms have been mentioned. The rest of this section discusses their inherent limitations, motivating why a different approach is needed, especially when the complexity of the task increases.

As discussed, the main challenge of machine learning is to learn how to generalize to previously unseen inputs. One could think that in order to reduce the generalization error one just needs to train the algorithm over a larger amount of data. In fact, increasing the size of the training set is surely helpful, but clearly there is a limit, both in terms of computation and storage capacity, to the amount of data we can process. Therefore, an essential feature to be concerned with, is the rate at which the performance of our machine learning algorithm improves as the training set size grows. The concept of deep learning will be formally introduced in the coming section, but Fig. 6 already anticipates how deep learning is able to improve the performance at a much faster rate than other machine learning techniques, as the dimension of the training data increases.

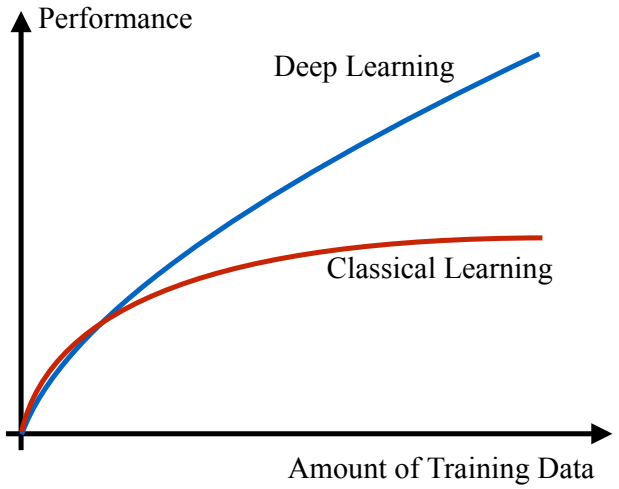

Figure 6. Classical and Deep learning vs. training set size.

It should also be stressed that, instead, for small-to-medium training set sizes, the relation among deep learning and other

machine learning techniques is not well-defined, and in many cases it turns out that classical machine learning can slightly outperform deep learning.

How can we explain the behavior of Fig. 6? The key phenomenon to consider is the so-called *curse of dimensionality*, which refers to the fact that the number of distinct configurations of a set increases exponentially with the number of variables describing each element of the set. Recalling the formal description of a learning algorithm as the map in (3), it is to be remarked that the dimensionality here does not directly refer to the dimension of the training set, but instead to the number of features n describing each element $\mathbf{x}$ in the set of possible inputs $\mathcal{X}$. Nevertheless, it is clear that as n increases, we need more training samples to successfully learn the structure of $\mathcal{X}$, thus devising a map $\mathcal{F}$ that is able to achieve low generalization error. The approach of other machine learning methods to cope with the curse of dimensionality is based on two main points:

- Assuming prior beliefs about the structure that a good function $\mathcal{F}$ should have, such as the smoothness prior, i.e. assuming that the function $\mathcal{F}$ will not change drastically when evaluated for two neighboring points $\mathbf{x}_1$ and $\mathbf{x}_2$. However, in high-dimensional spaces even a very smooth function can vary at a different scale along different dimensions. Moreover, even assuming that all the derivatives of the function are similar in the different directions, the smoothness assumption is reasonable only when the points $\mathbf{x}_1$ and $\mathbf{x}_2$ are sufficiently close to each other. Depending on the magnitude of the derivatives this might require an unfeasible amount of training data.
- Incorporating task-specific assumptions to perform manual feature selection, i.e. deciding which components of $\mathbf{x}$ are relevant to the specific problem at hand and performing a customized processing of these features. However, this process requires the analysis of a realistic mathematical model for the problem at hand, which might not be available. Moreover, the settings used for one task are not general in the sense that they may not apply to other problems.

Deep learning adopts quite a different approach. It assumes that the data has been generated by a composition of factors with a hierarchical order and develops a learning method that is able to automatically understand the structure of the underlying distribution, extracting directly from the data the features which are important to devise a good map $\mathcal{F}$. In other words, deep learning assumes that some correlations exist among the behavior of $\mathcal{F}$ over different regions of space, as a result of the structure of the underlying distribution of the data. This is clearly a more general assumption than the smoothness prior, which constraints the local behavior of $\mathcal{F}$ in the neighborhood of each point. This has been shown to enable deep learning to generalize non-locally [105]. Moreover, deep learning is able to understand the structure of the underlying distribution, without requiring task-specific assumptions, thus enabling more general-purpose algorithms. These improvements are possible thanks to the use of ANNs, which represent the tool used by deep learning techniques to implement the learning process, as discussed in detail next.

III. Deep learning by artificial neural networks

As anticipated at the end of the previous section, ANNs are the enablers of deep learning [39], [106], thanks to their ability to learn complex input-output relationships and statistical structures directly from the observed data. Following the deep learning approach, ANNs are organized hierarchically in layers of elementary processing units, called *neurons*. More in detail, a ANN is characterized by:

- An input layer, which forwards the input data to the rest of the network.
- One or more hidden layers, which process the input data.
- An output layer which applies a final processing to the data before outputting it.
- Weights and bias terms that model the strength of the connections among neurons.

If the network has only one hidden layer, it is referred to as a *shallow network*, whereas if it has more than one hidden layer, it is referred to as a *deep network*, hence the name deep learning. As discussed in Section III-A, deep networks are preferred, since they usually require a lower number of neurons to achieve a given accuracy. It is probably the use of deep architectures in which multiple neurons process the information and propagate the result that has motivated the analogy between ANNs and natural neural networks, i.e. the human brain, which is also composed of a network of elementary processing units, the neurons, that elaborate information and then propagate the results to other neurons.

A first broad classification of ANNs is based on how the information flows from input to output. Specifically:

- **Feed-forward Neural Networks** (FNN) are neural networks in which each neuron is connected only to the neurons in the following layer and thus the input data can only propagate forward, from the input layer to the output layer, without the possibility of any feedback loop.
- **Recurrent Neural Networks** (RNN) are neural networks in which feedback loops are allowed, and the output of a neuron can become again an input of the same neuron, as well as of other neurons in the same or in a previous layer.

It should be mentioned that several neural networks architectures exist within each of the two main categories introduced above. A notable example is that of Convolutional Neural Networks (CNNs), to be described in Section III-A1, and that have been extensively used for image processing and pattern recognition [107]. Nevertheless, in this work, we have decided to adopt the broad classification above, because the distinctions among other neural networks architectures are somewhat blurry, since different kinds of layers can co-exist in the same neural network. Instead, a more specific classification can be made by considering the type of layers composing the ANN. The most common kinds of layers are the following:

- **Fully-connected layer.** It is the typical layer to be found in FFNs and is characterized by the fact that each neuron of the layer receives an input from all neurons of the preceding layer, and is connected to all neurons of the

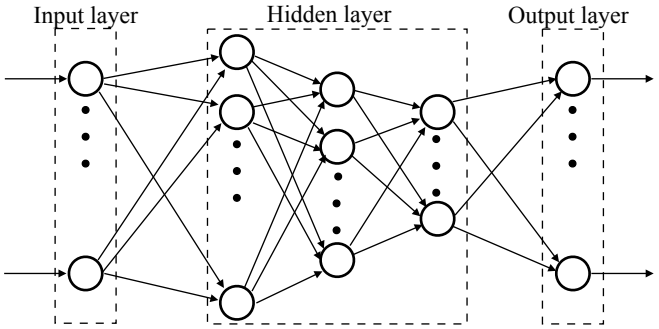

Figure 7. Scheme of a deep ANN with L hidden layers and N_ℓ units in layer ℓ, for all $\ell = 1, \ldots, L$.

following layer. The input data is linearly processed, passed through a non-linearity, and then propagated to the following layer.

- **Convolutional layer.** It is another kind of layer used in FFNs, and more precisely in CNNs. Similarly to a fully-connected layer, it filters the input by a linear operation, namely a convolution, applies a non-linearity, and then forwards the result. However, one neuron need not be connected to all neurons in the preceding and following layer.
- **Pooling layer.** It is a layer normally used in CNNs which operates by dividing the input data into blocks, and then selecting either the maximum element of each block, or computing the average of the elements within each block.
- **Recurrent layer** It is the typical layer of RNNs. After performing again an affine combination of the input and passing it through a non-linearity, the output is not just propagated forward, but a feedback loop is also present.

More details on the operation of the different kinds of layers are provided in the rest of this section.

A. Feedforward Neural Networks

Here the focus will be on FFNs with fully-connected layers, which is the quintessential ANN architecture. Instead, convolutional layers will be discussed in detail in Section III-A1.

The general structure of a FFN is depicted in Fig. 7. An N_0-dimensional input vector $\mathbf{x}_0$ is fed to the network through the N_0 neurons of the input layer. Afterwards, it passes through L so-called hidden layers, with Layer ℓ having N_ℓ neurons. Finally, the (N_L+1)-dimensional output is retrieved from the N_L+1 neurons of the output layer. To elaborate, let us denote by $\mathbf{x}_{\ell-1}$ the input to the ℓ-th layer of the network. Then, for all $\ell = 1, \ldots, L+1$ and $n = 1, \ldots, N_\ell$, the output $\mathbf{x}_\ell(n)$ of neuron n in layer ℓ is obtained as:

$$\mathbf{x}_\ell(n) = f_{n,\ell}(z_{n,\ell}) \ , \ z_{n,\ell} = \boldsymbol{w}_{n,\ell}^T \mathbf{x}_{\ell-1} + b_{n,\ell} \ , \qquad (4)$$

wherein $\boldsymbol{w}_{n,\ell} \in \mathbb{R}^{N_{\ell-1}}$ with $w_{n,\ell}(k)$ being the weight of the link between the k-th neuron in layer $\ell-1$ and the n-th neuron in layer ℓ, $b_{n,\ell} \in \mathbb{R}$ is the bias term of neuron n in layer ℓ, while $f_{n,\ell}$ is the so-called **activation function** of neuron n in layer ℓ. Thus, the processing done by each neuron can be seen as a two-step procedure in which first an affine combination of the inputs is computed with weights $\boldsymbol{w}_{n,\ell}$ and bias term $b_{n,\ell}$, yielding the intermediate term $z_{n,\ell}$. Then, the final output is obtained by applying the activation function $f_{n,\ell}$ to $z_{n,\ell}$.

As for the choice of the activation functions, over the years several functions have been considered. The first choice was to use sigmoidal functions

$$\sigma(z_{n,\ell}) = \frac{1}{1+e^{-z_{n,\ell}}} \ , \qquad (5)$$

or hyperbolic tangent functions

$$\tanh(z_{n,\ell}) = \frac{e^{z_{n,\ell}} - e^{-z_{n,\ell}}}{e^{z_{n,\ell}} + e^{-z_{n,\ell}}} \ . \qquad (6)$$

The sigmoid function is able to produce feasible probability values, being limited between zero and one, and for this reason nowadays it is typically used as activation function of the output layer for applications that require to estimate a probability. However, its use for the hidden layers is no longer recommended, due to the fact that it saturates for a significant portion of its domain, thus having derivatives very close to zero when the argument is large in modulus. This causes the so-called *vanishing gradient* problem, which slows down the convergence of gradient-based training algorithms. Another way of looking at the problem is to say that sigmoid activation functions are able to learn only when the input is around zero, i.e. in their (approximately) linear region, where the output of the sigmoid function is sensitive to variations of the input. Instead, in other regions of its domain the sigmoid function saturates and the output tends to be approximately constant even in response to significant changes in the input, which does not provide much useful learning information. Similar considerations also hold for the hyperbolic tangent function, which is linked to the sigmoid function by the relation: $\tanh(z_{n,\ell}) = 2\sigma(2z_{n,\ell}) - 1$.

Nowadays, the most widely-used choice for the activation function of the hidden layers is the Rectified Linear Unit (ReLU) function [108]–[110], defined as:

$$\mathrm{ReLU}(z_{n,\ell}) = \max(0, z_{n,\ell}) \ . \qquad (7)$$

ReLU functions are linear whenever the neuron is active, which makes them easier to optimize. Whenever the neuron produces a non-zero output, the gradient of the activation function is constantly equal to one, and no second-order effects are present. The drawback is that ReLU units do not provide any useful learning information when their input is negative. To counter this issue, some refinements of the ReLU function have introduced a non-zero slope also for negative inputs, considering the function:

$$f_{n,\ell}(z_{n,\ell}) = \max(0, z_{n,\ell}) + c\min(0, z_{n,\ell}) \ . \qquad (8)$$

The Leaky ReLU function sets $c = 0.01$ as proposed in [111]; the *absolute value rectification* approach proposed in [108] considers $c = -1$, while the parametric ReLU approach proposed in [112] treats c as a parameter to be optimized during the training process.

A further generalization of the ReLU is the exponential ReLU, which behaves like the ReLU for positive inputs, but outputs

$$f_{n,\ell}(z_{n,\ell}) = \alpha(e^{z_{n,\ell}} - 1) \ , \qquad (9)$$

when the input x is negative, with α a scalar typically set to 1, [113].

The discussion about the ReLU function, its generalizations and how they try to preserve the properties of linear functions, seems to lead to the conclusion that the best activation functions are linear functions. In fact, linear activation functions can be considered as far as the output layer is concerned, to perform specific operations such as computing arithmetic averages. However, their use in the hidden layers is not encouraged, as they might prevent the network from learning non-linear maps. For example, in the extreme case in which all activation functions were linear, the input-output relation of the FNN would reduce to being always linear, when instead one of the strengths of neural networks lies in their ability to combine multiple non-linearities to emulate virtually any input-output map. This fact was formally established in [114], stating that any deterministic continuous function over a compact set can be approximated arbitrarily well by a single-layer FNN with enough neurons and sigmoidal activation functions[1]. This fundamental result is known as the **universal approximation theorem** of FNNs and was later extended to a broader class of activation functions, including the ReLU function and its generalizations, in [115]. Nevertheless, despite its high theoretical importance the universal approximation theorem is not constructive, because:

- it does not establish how many neurons are required in order to obtain the desired level of approximation accuracy.
- it does not establish whether it is more convenient to use a shallow or deep architecture in order to improve the approximation accuracy or reduce the number of required neurons.
- it does not establish how to configure the FNN in order to obtain the desired approximation accuracy.

An answer to the first question was provided in [116], which provides bounds for the number of nodes in a shallow network to obtain a given approximation accuracy. Unfortunately, the bounds show that in general an exponential number of nodes is required.

As for the second issue, deep architectures seem to require a lower number of nodes, even though a formal proof of this result in a general setting is still an open problem. Nevertheless, some available results prove that certain classes of functions can be represented more efficiently by increasing the network depth. For example in [117] it is shown that the number of regions of a piece-wise linear function that can be reliably represented scales exponentially with the number of layers L. Moreover, many empirical results have shown that deep architectures provide lower generalization error than shallow architectures [20, Sec. 6.4.1].

Finally, the third issue is perhaps the most problematic. Although the universal approximation theorem ensures that there exists a FNN able to learn the desired map, it provides no indication as to how to configure the weights $\boldsymbol{w}_{n,\ell} \in \mathbb{R}^{N_{\ell-1}}$ and bias $b_{n,\ell} \in \mathbb{R}$ of each node to this end. This shows how configuring the parameters of an ANN represents the most critical step when employing deep learning. The training process of ANNs will be addressed in detail in Section III-C.

[1]The result is proved assuming *squashing activation functions*, which include sigmoid as special cases.

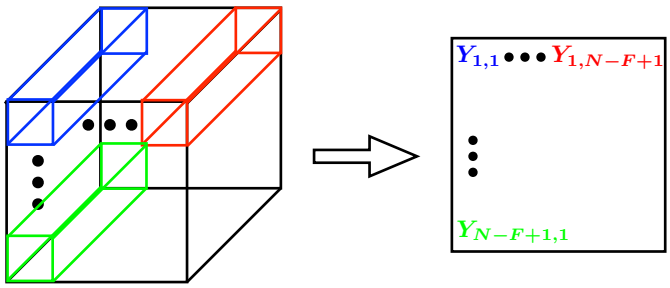

Figure 8. 3D-Convolution in convolutional neural networks. The input data is arranged in an $N \times N \times N_c$ matrix, which is filtered by a sliding $F \times F \times N_c$ matrix, yielding a $N - F + 1 \times N - F + 1$ output matrix.

1) Convolutional neural networks: CNNs are a kind of FFNs that have been established as the main tool for image processing, and in general for processing data with a spatial structure. The main ingredient of CNNs is the 3D-convolution operation, which amounts to a particular linear processing of the input data. For this reason, CNNs can be considered as a sub-category of FFNs.

When using a CNN, the input data is assumed to be organized in a multi-dimensional matrix $\boldsymbol{X}$ with dimensions $N \times N \times N_c$, where the parameter N_c is called the number of channels and is typically equal either to $N_c = 3$ when color images are processed, or to $N_c = 1$ when black-and-white images are processed. Each node of a convolutional layer is also represented as a multi-dimensional matrix $\boldsymbol{W}$ with dimensions $F \times F \times N_c$ (with $F \leq N$) containing the weights of the node. Then, the 3D-convolution operation outputs a bi-dimensional matrix $\boldsymbol{Y}$, with dimensions $N-F+1 \times N-F+1$, obtained by sliding the weight matrix over the input matrix, computing each time the cross-correlation between the weight matrix and the corresponding chunk of the input matrix, as depicted in Fig. 8. Mathematically, the $(\ell - m)$-th element of the output matrix $\boldsymbol{Y}$ is expressed as:

$$\boldsymbol{Y}_{\ell,m} = \sum_{i=1}^{F} \sum_{j=1}^{F} \sum_{k=1}^{N_c} \boldsymbol{W}_{i,j,k} \boldsymbol{X}_{i+\ell,j+m,k} \ , \qquad (10)$$

It can be seen that, as already mentioned, each element of the output matrix is obtained through a cross-correlation rather than a convolution, even though the term convolution is universally used in the ANN jargon to refer to the operation in (10). In the following, we will embrace this terminology. After computing (10) for all ℓ and m, the output of the node is obtained by first summing a scalar bias term b and then applying an activation function to each component of $\boldsymbol{Y}$, like in a traditional fully-connected layer. Finally, the bi-dimensional output of each node in the layer are stacked together to form a new matrix with dimensions $N - F + 1 \times N - F + 1 \times N_F$, with N_F the number of nodes in the convolutional layer, that becomes the input of the next layer of the CNN.

It is interesting to observe that (10) can be rewritten as a scalar product just like in a fully-connected layer, upon vectorizing the input and weight matrices. For example, denoting by $\mathbf{x}$ and $\boldsymbol{w}$ the $N^2N_c \times 1$ and $F^2N_c \times 1$ vectors obtained by vectorizing $\boldsymbol{X}$ and $\boldsymbol{W}$, the output element $\boldsymbol{Y}_{1,1}$ can be obtained as

$$\boldsymbol{Y}_{1,1} = \mathbf{x}^T \widetilde{\boldsymbol{w}} \,, \tag{11}$$

wherein $\widetilde{\boldsymbol{w}} = [\boldsymbol{w}\ \mathbf{0}_{(N^2-F^2)N_c}]$. All other elements of $\boldsymbol{Y}$ can be obtained in a similarly way, upon considering suitably zero-padded version of $\boldsymbol{w}$. As a result, each node of a convolutional layer is equivalent to $(N - F + 1)^2$ nodes of a fully-connected layer, in which the weights of many connections are permanently set to zero. This sparsity of the connections is one of the major strengths of CNNs, since it enables to process very large data using a relatively small number of parameters, which helps avoid overfitting. On the other hand, the underlying assumption that justifies the use of CNNs is the presence of strong spatial correlations in the input. Only if this is fulfilled, as is typically the case in image processing, it is possible to apply the same filter to different parts of the input matrix, thus avoiding unnecessary connections among the neurons.

The operation defined in (10) is the normal convolution employed in CNNs. In some cases, it can be slightly modified by applying what are called **padding** and **stride**:

- **Padding.** When computing (10), the components at the border of the input matrix $\boldsymbol{X}$ are used less frequently than the components in the middle. In order to avoid this, it is possible to apply (10) to a zero-padded version of $\boldsymbol{X}$, in which P rows and columns of zeros are appended to $\boldsymbol{X}$. Then, the zero-padded input matrix will have dimensions $N + 2P \times N + 2P$, and the output matrix will have dimensions $(N+2P-F+1) \times (N+2P-F+1)$. Then, if F is odd, choosing

$$P = (F-1)/2 \,, \tag{12}$$

  yields an output with the same dimensions as the input.
- **Stride.** The convolution operation described in (10) slides the weight matrix $\boldsymbol{W}$ over the input matrix moving by one position at each step. This can be generalized by sliding the weight matrix by S positions at each step, where S is called the stride parameter. In this case, assuming also a padding P is used, the output matrix will have dimensions:

$$\left\lfloor \frac{N+2P-F}{S} + 1 \right\rfloor \times \left\lfloor \frac{N+2P-F}{S} + 1 \right\rfloor \,. \tag{13}$$

While the convolution operation is the defining feature of CNNs, another widely used operation in a CNN is the **Pooling**. Unlike the convolution, which is individually performed by each node of layer before the different bi-dimensional matrices are combined together, the pooling is performed at the layer level and operates separately on each channel of the input matrix $\boldsymbol{X}$. Two types of pooling are commonly used:

- **Max Pooling.** For each channel of the input matrix $\boldsymbol{X}$, say $\boldsymbol{X}_{n_c} = \boldsymbol{X}(:,:,n_c)$, a Max Pooling layer with parameter F selects the maximum element out of each $F \times F$ sub-matrix of $\boldsymbol{X}_{n_c}$.
- **Average Pooling.** For each channel of the input matrix $\boldsymbol{X}$, say $\boldsymbol{X}_{n_c} = \boldsymbol{X}(:,:,n_c)$, an average Pooling layer with parameter F computes the arithmetic average of each $F \times F$ sub-matrix of $\boldsymbol{X}_{n_c}$.

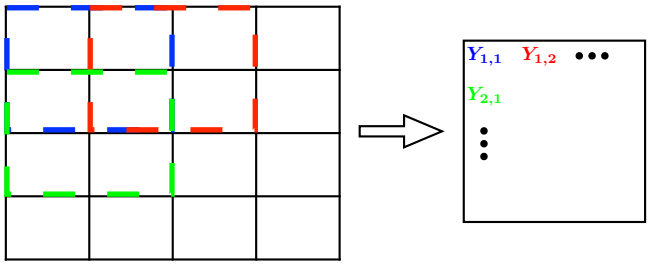

Figure 9. Pooling with $S = 1$ of a single channel of a $4 \times 4 \times N_c$ input by a 2×2 filter. From each 2×2 sub-matrix of the input, either the maximum element or the average are computed.

In both cases, a stride S can also be used, meaning that the sliding window over which the maximum or average are computed moves by S positions each time. An example of pooling with $S = 1$ is shown in Fig. 9.

As a final remark before concluding this section, it should be mentioned that practical FFNs are composed by a mixture of convolutional, pooling, and fully-connected layers, normally performing convolutions and pooling in the first layers, thus decreasing the size of the data, and employing fully-connected layers at the end once the dimension of the data has become more manageable.

B. Recurrent neural networks

If CNNs are meant to process data exhibiting spatial correlations, RNNs are designed to work with temporal sequences of data with correlated samples. As already anticipated, the main difference compared to FFNs is that the information does not only propagate forward, but loops are allowed. More in detail, each layer of a RNN may receive as input also its own activation value. To elaborate, using a similar notation as in Section III-A, the output $\mathbf{x}_\ell^{[t]}(n)$ of neuron n in layer ℓ at time t is obtained as:

$$\boldsymbol{a}_\ell^{[t]}(n) = f_{n,\ell}(\boldsymbol{w}_{n,\ell}^T \mathbf{x}_{\ell-1}^{[t]} + \tilde{\boldsymbol{w}}_{n,\ell}^T \boldsymbol{a}_\ell^{[t-1]} + b_{n,\ell}) \tag{14}$$

$$\mathbf{x}_{n,\ell}^{[t]} = g_{n,\ell}(\bar{\boldsymbol{w}}_{n,\ell}^T \boldsymbol{a}_\ell^{[t]} + \bar{b}_{n,\ell}) \,, \tag{15}$$

wherein $f_{n,\ell}$ and $g_{n,\ell}$ are node-dependent activation functions. Thus, it can be seen that each node in a recurrent layer combines by different weights not only the current input, but also the intermediate vector $\boldsymbol{a}_\ell$ computed in the previous step. This introduces a correlation among the different computations that is beneficial to exploit the correlation in the input sequence. Moreover, a recurrent layer has two activation functions, f and g. Popular choices here are to use the hyperbolic tangent or the ReLU for f and the sigmoid function for g.

The architecture described above is the general architecture of recurrent layers. Several variants exist that are commonly used in real-world RNNs. In addition, we stress that a deep

RNN typically has just a few recurrent layers, and it is possible to have hybrid architectures composed of some initial recurrent layers, followed by feed-forward layers. Nevertheless, it is not the main purpose of this work to provide a detailed account of RNNs architectures, which can be found in specialized references on ANNs, like [20].

C. Training Neural Networks

Training a neural network is the process that tunes the parameters[2] $\boldsymbol{w}_{n,\ell} \in \mathbb{R}^{N_{\ell-1}}$ and $b_{n,\ell} \in \mathbb{R}$ in a supervised learning fashion in order for the FNN to learn the desired input-output relation. To elaborate, let us consider a training set composed of N_{TR} input samples with the corresponding desired output, namely

$$\mathcal{S}_{TR} = \left\{ \left(\mathbf{x}_0^{(1)}, \mathbf{x}_{L+1}^{(1)} \right), \ldots, \left(\mathbf{x}_0^{(N_{TR})}, \mathbf{x}_{L+1}^{(N_{TR})} \right) \right\} . \quad (16)$$

Moreover, for each layer $\ell = 1, \ldots, L+1$, let us stack the weight vectors into the $N_{\ell-1} \times N_\ell$ matrix $\boldsymbol{W}_\ell$ and the bias terms into the $N_\ell \times 1$ vector $\boldsymbol{b}_\ell$, respectively defined as

$$\boldsymbol{W}_\ell = [\boldsymbol{w}_{1,\ell}, \ldots, \boldsymbol{w}_{N_\ell,\ell}] , \quad (17)$$

$$\boldsymbol{b}_\ell = [b_{1,\ell}, \ldots, b_{N_\ell,\ell}]^T . \quad (18)$$

The *actual* output of the FNN when the input is the nt-th training sample $\mathbf{x}_0^{(nt)}$ will depend on the network weights and bias terms, and is denoted as:

$$\widehat{\mathbf{x}}_{L+1}^{(nt)} \left(\{\boldsymbol{W}_\ell, \boldsymbol{b}_\ell\}_{\ell=1}^L \right) , \forall\, nt = 1, \ldots, N_{TR} . \quad (19)$$

The goal of the training algorithm is to optimize the network weights and bias terms in order to minimize the loss incurred between the actual output $\widehat{\mathbf{x}}_{L+1}^{(nt)}$ in (19), and the desired output $\mathbf{x}_{L+1}^{(nt)}$ defined by the training set in (16), for all $nt = 1, \ldots, N_{TR}$, as quantified by the **loss function**

$$L\Big(\{\boldsymbol{W}_\ell, \boldsymbol{b}_\ell\}_{\ell=1}^L\Big) = \frac{1}{N_{TR}} \sum_{nt=1}^{N_{TR}} \mathcal{L}\left(\mathbf{x}_{L+1}^{(nt)}, \widehat{\mathbf{x}}_{L+1}^{(nt)} \Big(\{\boldsymbol{W}_\ell, \boldsymbol{b}_\ell\}_{\ell=1}^L \Big) \right), \quad (20)$$

wherein $\mathcal{L}(\mathbf{x}_{L+1}^{(nt)}, \widehat{\mathbf{x}}_{L+1}^{(nt)})$ is an individual loss function that models the loss between $\widehat{\mathbf{x}}_{L+1}^{(nt)}$ and the desired output $\mathbf{x}_{L+1}^{(nt)}$. As for the expression of the individual loss function, a natural choice is the mean square error, namely:

$$\mathcal{L}(\mathbf{x}, \widehat{\mathbf{x}}) = \text{MSE}(\mathbf{x}, \widehat{\mathbf{x}}) = \sum_{i=1}^{N_{\ell+1}} (\mathbf{x}(i) - \widehat{\mathbf{x}}(i))^2 . \quad (21)$$

The MSE has the advantage of being applicable to virtually any scenario, and enables a simple computation of its derivatives. However, in some cases it can slow down the learning algorithm. Instead, faster convergence of the learning algorithm is typically observed by using the cross-entropy loss function, defined as

$$\mathcal{L}(\mathbf{x}, \widehat{\mathbf{x}}) = H(\mathbf{x}, \widehat{\mathbf{x}}) = -\sum_{i=1}^{N_{\ell+1}} \mathbf{x}(i) \log(\widehat{\mathbf{x}}(i)) + (1-\mathbf{x}(i)) \log(1-\widehat{\mathbf{x}}(i)) . \quad (22)$$

However, the applicability of (22) is not so wide as that of the MSE function. Indeed, clearly (22) applies only to those cases in which both the desired and actual output data belong to the interval $[0, 1]$, and thus can be interpreted as distributions of random variables. A notable case in which this holds true is when sigmoid activation functions are used in the output layer, aiming at estimating a probability distribution. Assuming that both $\mathbf{x}$ and $\widehat{\mathbf{x}}$ have entries in $[0, 1]$, the cross entropy in (22) represents a measure of the divergence between $\mathbf{x}$ and $\widehat{\mathbf{x}}$, since the cross entropy of two distributions p and q is equal to the Kullbach-Leibler divergence between p and q plus the entropy of p [118]. Applying this result, (22) can be rewritten as

$$\begin{aligned} H(\mathbf{x}, \widehat{\mathbf{x}}) &= -\sum_{i=1}^{N_{\ell+1}} \mathbf{x}(i) \log(\widehat{\mathbf{x}}(i)) + (1-\mathbf{x}(i)) \log(1-\widehat{\mathbf{x}}(i)) \\ &= -\sum_{i=1}^{N_{\ell+1}} \mathbf{x}(i) \log\left(\frac{\widehat{\mathbf{x}}(i)}{\mathbf{x}(i)} \right) + (1-\mathbf{x}(i)) \log\left(\frac{1-\widehat{\mathbf{x}}(i)}{1-\mathbf{x}(i)} \right) \\ &= -\sum_{i=1}^{N_{\ell+1}} \mathbf{x}(i) \log(\mathbf{x}(i)) + (1-\mathbf{x}(i)) \log(1-\mathbf{x}(i)) \\ &= \sum_{i=1}^{N_{\ell+1}} KL(\mathbf{x}(i), \widehat{\mathbf{x}}(i)) + H_b(\mathbf{x}(i)) , \end{aligned} \quad (23)$$

with $KL(\cdot, \cdot)$ and $H_b(\cdot)$ denoting Kullbach-Leibler divergence and binary entropy, respectively. Then, since $H_b(\mathbf{x})$ does not depend on the network parameters, minimizing the cross-entropy in (22) is equivalent to minimizing the Kullbach-Leibler divergence between the desired and actual output.

In any case, regardless of the chosen performance metric, the training process mathematically amounts to solving the optimization problem[3]

$$\min \ \frac{1}{N_{TR}} \sum_{nt=1}^{N_{TR}} \mathcal{L}\left(\mathbf{x}_{L+1}^{(nt)}, \widehat{\mathbf{x}}_{L+1}^{(nt)}(\boldsymbol{W}, \boldsymbol{b}) \right) \quad (24a)$$

$$\text{s.t. } \boldsymbol{W}_\ell \in \mathbb{R}^{N_{\ell-1} \times N_\ell} , \ \forall\, \ell = 1, \ldots, L+1 \quad (24b)$$

$$\boldsymbol{b}_\ell \in \mathbb{R}^{N_\ell \times 1} , \ \forall\, \ell = 1, \ldots, L+1 , \quad (24c)$$

wherein $\boldsymbol{W} = \{\boldsymbol{W}_\ell\}_{\ell=1}^L$, $\boldsymbol{b} = \{\boldsymbol{b}_\ell\}_{\ell=1}^L$. However, it is also to be stressed that the end-goal is not so much to minimize the cost function of (24), i.e. the training error, but rather to ensure a low generalization gap. Tuning the parameters of the network to achieve a low training error is a prerequisite to achieving a low test error, but an equally important task is that of tuning the network hyperparameters, (e.g. the number of layers L, the number of neurons per layer N_ℓ, the size of the training set N_{TR}), to fit the training data, avoiding both underfitting and overfitting. The coming Subsection III-C1 discusses the design of suitable algorithms to tackle (24) in an efficient and effective way, while Subsection III-C2 provides some guidelines for hyperparameter tuning in FNNs.

*1) **Parameter tuning - Tackling** (24):* Traditionally, in optimization theory the critical property that marks the watershed between problems that can be solved with affordable complexity, and problems that in general require an unfeasible

[2]For ease of notation, and without loss of generality, this section focuses on FFNs with fully connected layers. Results directly apply to CNNs and can be extended to RNNs with minor modifications.

[3]In case of RRNs, an additional sum over the time dimension is present to account for the loss over time of each training sample.

complexity, is convexity. A convex problem, defined as a problem whose objective and constraint functions are convex in the optimization variables [119]–[121], enjoys several useful properties, among which the following two have played a critical role in enabling the development of a consolidated theory of convex optimization, featuring practical algorithms able to provide theoretical optimality guarantees:

- **[P.1]:** Every stationary point of a convex function is a global minimum, i.e. the minimization of a convex function can be performed by simply looking for a point where the gradient of the function vanishes. This property establishes that first-order optimality conditions are necessary and sufficient for convex functions.
- **[P.2]:** For any $\varepsilon > 0$, the complexity required to find an ε-optimal solution of a generic convex problem with n variables scales in the worst case as the fourth power of n and as $\log\left(\frac{1}{\varepsilon}\right)$ [121, Section 5]. This property establishes that convex problems can be solved with polynomial complexity in the number of variables.

Unfortunately, neither of the two properties above holds for Problem (24) because the objective function is not convex with respect to the optimization variables, due to the presence of multiple layers combining several non-linear activation functions. This implies that the cost function of Problem (24) might have stationary points that are either local minima, or local maxima, or saddle points, a circumstance that becomes more and more likely as the dimensionality of the problem increases. In fact, it is quite typical for fairly deep model to have a very large number of points where the gradient vanishes, but that are not global minima. Moreover, the complexity required in order to find the global solution of Problem (24) is not guaranteed to be polynomial, scaling in general exponentially with the number of variables, which is equal to $\sum_{\ell=1}^{L+1} N_\ell(N_{\ell-1}+1)$. As a result, finding the global solution of Problem (24) appears a very challenging task, especially considering the in realistic FNNs, the number of nodes and layers can be fairly large.

Having said this, it might seem hopeless to perform an effective and efficient training of any reasonably-sized FNNs. Fortunately, this is not the case and several efficient algorithms exist to effectively train FNNs. To understand why the non-convexity of (24) does not pose a fundamental problem, one must recall that, although the training process amounts to solving an optimization problem, machine learning differs from pure optimization theory, in that the ultimate goal is not so much to minimize the training error, but rather to minimize the generalization error. As discussed in Section III, the training error lower bounds the generalization error, but there is no guarantee that a lower training error will also result in a lower generalization error. Actually, aiming for a very low training error typically causes overfitting. Therefore, when tackling Problem (24), it is surely desirable to find a configuration of parameters that yield a low training error, so as to avoid underfitting, but it is also not necessary to pursue the global minimization of the training error, which would most likely lead to overfitting. Any training algorithm will aim at progressively reducing the training error, stopping as soon as the generalization error evaluated over the validation set is below a desired threshold, regardless of the value of the training error. It is not uncommon that a training algorithm stops when the training error is relatively large compared to its global minimum.

As a result, the presence of stationary points of the cost function of Problem (24) would be a major issue only if the training algorithm were likely to converge to a suboptimal point yielding a too high training error, thus causing underfitting. A definitive formal proof that this is not the case is still an open research problem, but extensive experimental evidence has shown that, for sufficiently large neural networks, most local minima lead to a satisfactory training error [122]–[125]. In addition, especially in higher-dimensional spaces, local minima and local maxima of random functions are much less frequent compared to saddle points [123]. This phenomenon has been proved for some specific shallow neural networks in [126], while some theoretical arguments as well as experimental evidence that a similar behavior holds also in deep networks is provided in [122], [123], [125]. Therefore, the main issue related to the non-convexity of Problem (24) is related not so much to local minima, but rather to the presence of saddle-points. In this respect, empirical evidence provided in [124] has shown how first-order methods based on gradient descent are able to escape saddle points. This behavior can be theoretically justified by observing that gradient-based methods are not explicitly designed to find point with zero gradient. Rather, they are designed to reduce the cost function moving in the direction of maximum decrease which is pointed by the gradient. Of course, this implies that the algorithm stops if a point with rigorously zero gradient is reached, but it makes the algorithm capable of moving away from the neighborhood of a saddle point even for relatively small step-sizes. On the other hand, second-order method like Newton's method do not share this property, having a higher probability of being stuck around saddle points. A training algorithm based on an approximate Newton's method with a regularization strategy is the Levenberg-Marquardt method [127], [128], which has been found to have good performance as long as the negative eigenvalues of the Hessian of the cost function are relatively close to zero. Instead, a recent modification of Newton's method, designed to be more robust to the saddle-point problem in FNNs, has been introduced in [123]. However, despite enjoying stronger convergence properties in the convex case, at present the use of second-order method to tackle the non-convex Problem (24) is not so well-established as the use of first-order methods based on gradient descent. For this reason, the rest of this section will focus on presenting the main first-order training methods for FNNs.

Backpropagation algorithm. The first problem that we encounter towards the implementation of a gradient-based training algorithm for FNNs is the complexity related to the computation of the gradient. In large neural networks with many neurons and large training sets, direct computation of the derivatives of the training error in (24a) with respect to all network weights and bias terms would require an unmanageable complexity. Luckily, a fast algorithm to compute the gradient of the training error was developed in [129]. It makes

a clever use of the chain rule from multivariable calculus, and was called backpropagation algorithm, for reasons that will become clear after describing its operation.

To begin with, let us observe that the derivative of (24a) is written as the average of the derivatives of the individual cost function $\mathcal{L}(\mathbf{x}_{L+1}, \hat{\mathbf{x}}_{L+1}(\boldsymbol{W}, \boldsymbol{b}))$ over the training set. In fact, the backpropagation algorithm provides a way of computing the derivatives of $\mathcal{L}(\mathbf{x}_{L+1}, \hat{\mathbf{x}}_{L+1}(\boldsymbol{W}, \boldsymbol{b}))$. Specifically, given a training input sample $\mathbf{x}_0$, the first step of the backpropagation algorithm is to compute the corresponding actual output $\hat{\mathbf{x}}_{L+1}(\boldsymbol{W}, \boldsymbol{b})$. This step is referred to as *forward propagation* because it propagates the input forward through the network, computing (4) for all n and ℓ.

After completing the forward propagation, the derivative of the cost function with respect to $z_{n,L+1}$ can be computed as

$$\frac{\partial \mathcal{L}}{\partial z_{n,L+1}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{L+1}(n)} f'_{n,L+1}(z_{n,L+1}) \ , \ \forall \ n = 1, \ldots, N_{L+1} \tag{25}$$

The following step computes the derivatives of the cost function with respect to $z_{n,\ell}$, for all $\ell = L, L-1, \ldots, 1$, in a recursive way. This is the step that gives the name to the algorithm, since the derivatives are computed backwards, proceeding from the last to the first layer. Specifically, it holds[4]

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial z_{n,\ell}} &= \sum_{k=1}^{N_\ell+1} \frac{\partial \mathcal{L}}{\partial z_{k,\ell+1}} \frac{\partial z_{k,\ell+1}}{\partial z_{n,\ell}} \\ &= \sum_{k=1}^{N_\ell+1} \frac{\partial \mathcal{L}}{\partial z_{k,\ell+1}} w_{k,\ell+1}(n) f'(z_{n,\ell}) \ , \end{aligned} \tag{26}$$

which can be easily computed based on the derivatives with respect to $z_{k,\ell+1}$, $k = 1, \ldots, N_{\ell+1}$ obtained from Layer $\ell+1$. Finally, based on (26) and recalling (4), the derivatives with respect to the weights and bias terms are readily obtained as:

$$\frac{\partial \mathcal{L}}{\partial w_{n,\ell}(k)} = \frac{\partial \mathcal{L}}{\partial z_{n,\ell}} \mathbf{x}_{\ell-1}(k) \ , \tag{27}$$

$$\frac{\partial \mathcal{L}}{\partial b_{n,\ell}} = \frac{\partial \mathcal{L}}{\partial z_{n,\ell}} \ . \tag{28}$$

Thus, the backpropagation procedure can be stated as in Algorithm 2. Its strength lies in exploiting the recursive structure of the derivatives to compute, which enables to obtain them by simply computing a forward pass through the network, plus the corresponding backward pass, that has a similar complexity as the forward pass. Instead, any approach that tried a direct computation would require evaluating the cost function for each derivative to compute, thus having to perform a number of forward passes equal to the number of weights and bias in the network, which, for large networks, leads to an unfeasible computational complexity.

Stochastic Gradient Descent. While the backpropagation algorithm is computationally more convenient compared to direct derivative computation, its complexity still scales with the size of the training set. In order to implement Algorithm 2, one must forward-propagate and backward-propagate all N_{TR} samples in the training set. This poses a complexity issue since typically large training sets are used by ANNs. More in general, any algorithm that tried to compute the *true* gradient of the cost function of Problem (24), i.e.

$$\nabla L(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{N_{TR}} \sum_{nt=1}^{N_{TR}} \nabla \mathcal{L}\left(\mathbf{x}_{L+1}^{(nt)}, \hat{\mathbf{x}}_{L+1}^{(nt)}(\boldsymbol{W}, \boldsymbol{b})\right) \ , \tag{29}$$

would have a complexity proportional to N_{TR}. To address this issue, state-of-the-art training algorithms for FNNs employ a variant of the gradient descent algorithm known as Stochastic Gradient Descent (SGD) [130]. While the standard (or deterministic) implementation of the gradient descent requires computing (29), the stochastic variant of the gradient descent algorithm computes an estimates of (29) based on a randomly-selected subset of the entire training set, called mini-batch. In more details, denoting by $\mathcal{S}_{SGD}$ the set of indexes associated to the selected mini-batch, and by N_S the cardinality of $\mathcal{S}_{SGD}$, an estimate of the gradient is given by:

$$\widehat{\nabla L}(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{N_S} \sum_{nt \in \mathcal{S}_{SGD}} \nabla \mathcal{L}\left(\mathbf{x}_{L+1}^{(nt)}, \hat{\mathbf{x}}_{L+1}^{(nt)}(\boldsymbol{W}, \boldsymbol{b})\right) \ . \tag{30}$$

Each time a gradient descent step is taken, the estimated gradient (30) is evaluated based on a new, randomly selected set $\mathcal{S}_{SGD}$, and used in place of the true gradient. The overall procedure is provided in Algorithm 3.

Algorithm 2 Backpropagation Algorithm.

for $nt = 1 \rightarrow N_{TR}$ **do**
  `Training input` $\mathbf{x}_0^{(nt)}$ `with desired output` $\mathbf{x}_{L+1}^{(nt)}$;
  Forward Propagation: `Compute the actual output` $\hat{\mathbf{x}}_{L+1}^{(nt)}$ `by (4) for all` $\ell = 1, \ldots, L+1$;
  Backward Propagation: `Compute` $\frac{\partial \mathcal{L}}{\partial z_{n,\ell}}$ `by (25) and (26) for all` $\ell = L+1, \ldots, 1$;
  `Compute (27) and (28) for every weight` $w_{n,\ell}(k)$ `and bias term` $b_{n,\ell}$;
end for
$\nabla_{\boldsymbol{W}} L(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{N_{TR}} \sum_{nt=1}^{N_{TR}} \nabla_{\boldsymbol{W}} \mathcal{L}(\mathbf{x}_{L+1}^{(nt)}, \hat{\mathbf{x}}_{L+1}^{(nt)}(\boldsymbol{W}, \boldsymbol{b}))$;
$\nabla_{\boldsymbol{b}} L(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{N_{TR}} \sum_{nt=1}^{N_{TR}} \nabla_{\boldsymbol{b}} \mathcal{L}(\mathbf{x}_{L+1}^{(nt)}, \hat{\mathbf{x}}_{L+1}^{(nt)}(\boldsymbol{W}, \boldsymbol{b}))$;

Algorithm 3 Stochastic Gradient Descent for FNNs training.

`Set` $\varepsilon > 0$, $\boldsymbol{W}, \boldsymbol{b}$;
while Validation Error larger than ε **do**
  `Sample a random mini-batch` $\mathcal{S}_{SGD}$;
  `Compute (30) by Algorithm 2`;
  $\boldsymbol{W} = \boldsymbol{W} - \alpha \widehat{\nabla L}(\boldsymbol{W}, \boldsymbol{b})$;
  $\boldsymbol{b} = \boldsymbol{b} - \alpha \widehat{\nabla L}(\boldsymbol{W}, \boldsymbol{b})$;
end while

In Algorithm 3 it is not explicitly stated how to set the step-size α. In the context of machine learning, α is usually referred to as the **learning rate** as it controls how fast the algorithm reduces the cost function, and thus learns. The learning rate is a key parameter of the SGD algorithm and must be carefully selected. While the traditional gradient descent can use a fixed α and converges as long as α is not too large, the SGD uses

[4]Recall that the derivative with respect to x of the function $g(\boldsymbol{y}(x))$, with $\boldsymbol{y}(x) = [y_1(x), \ldots, y_I(x)]$, is given by $\sum_{i=1}^{I} (\nabla_y g)^T J_x \boldsymbol{y}$, where J_x denotes the Jacobian operator with respect to x.

a variable α_k to be used in iteration k, due to the inherent deviation of (30) from the true gradient. More formally, a sufficient condition for the convergence of Algorithm 3 is:

$$\sum_{k=1}^{\infty} \alpha_k = \infty \ , \ \sum_{k=1}^{\infty} \alpha_k^2 < \infty \tag{31}$$

A common approach is to update α_k for the first t iterations according to the formulas:

$$\alpha_k = \left(1 - \frac{k}{t}\right) \alpha_0 + \frac{k}{t} \alpha_t \ , \text{ for } k \leq t \ , \tag{32}$$

while keeping α constant after the t-th iteration. Typically, α_t should be roughly one hundredth of α_0, but in practice the parameters t, α_t, and α_0 are typically chosen by trial and error monitoring the error obtained over the validation set for different configurations of parameters.

Two more remarks concerning Algorithm 3 are in order.

Remark 1: The computational complexity of SGD depends on the size N_S of the mini-batches. If $N_S = N_{TR}$ the algorithm reduces to standard gradient descent, also called deterministic or batch gradient descent. Instead, if $N_S = 1$, the algorithm is also called online gradient descent. Typically, SGD uses $1 < N_S < N_{TR}$ and the choice is also dictated by the particular hardware where the algorithm runs, since too low values of N_S may underutilize modern multi-core architectures. Also, some architectures, e.g. GPUs are more efficient when $N_S = 2^n$, with n usually chosen in the range from 16 to 256.

Remark 2: Since the SGD operates based only on an estimate of the true gradient, it typically requires more iterations than its deterministic counterpart to converge. However, each iteration is computationally much faster and the total number of computations required to reach convergence is much lower than that of the deterministic gradient descent method. In particular, SGD has a complexity per update that does not scale with the total size of the training set N_{TR}, since it might converge also without having to pass through the entire training set. On the other hand, typically several passes through the training set, called *epochs*, are required to achieve satisfactory training results.

Momentum for Stochastic Gradient Descent. One drawback of SGD is that learning can be sometimes slow due to the fact that only an estimate of the gradient is computed in each iteration. The method of momentum is a general strategy in optimization theory [131], that can be used to accelerate the learning process. The basic idea of the momentum algorithm is to perform the gradient update by an exponentially decaying moving average, as stated in Algorithm 4.

Algorithm 4 introduces the new parameter $\boldsymbol{v}$, which is called *velocity*, in analogy with the fact that it controls the velocity with which the updates move through the parameter space. Due to the presence of the velocity term and to the exponential average of multiple gradient points, the magnitude of the step depends on the magnitude of the sequence of gradients, and also on how aligned these gradients are. This tends to smooth out the oscillations of the standard SGD algorithm. The velocity $\boldsymbol{v}$ represents the cumulative effect of the past gradients, while the term δ weighs the relative importance of the current gradient with respect to the cumulated gradient. The larger $\delta \in [0, 1)$ is with respect to α, the more the past gradients affect the direction of the update. If all the gradients of the sequence were equal to $\widehat{\nabla \bar{L}}$, the updates will accelerate in the direction of the common negative gradient until reaching a limit velocity

$$\boldsymbol{v}_\infty = \frac{\varepsilon \|\widehat{\nabla \bar{L}}\|}{1 - \delta} \ . \tag{33}$$

Thus, the parameter δ determines the relative speed of the updates compared to the SGD method without momentum. Common values of δ are 0.5, 0.9, and 0.99, and it is also desirable to adapt δ as well as α iteration after iteration, similarly to what is done for the basic SGD method.

Algorithm 4 Stochastic Gradient Descent with Momentum for FNNs training.

```
Set ε > 0, v ⪰ 0, W, b;
while Validation Error larger than ε do
    Sample a random mini-batch S_SGD;
    Compute (30) by Algorithm 2;
    v = δv − α∇̂L(W, b);
    W = W + v;
    b = b + v;
end while
```

Nesterov Momentum for Stochastic Gradient Descent. A variant of the momentum for SGD appeared in [132]. Following the approach of Nesterov's gradient method [133], the idea is to compute the estimate of the gradient taking into account the velocity term, as shown in Algorithm 5.

Algorithm 5 Stochastic Gradient Descent with Nesterov's Momentum for FNNs training.

```
Set ε > 0, v ⪰ 0, W, b;
while Validation Error larger than ε do
    Sample a random mini-batch S_SGD;
    Compute (30) evaluated at W + δv and b + δv by Algorithm 2;
    v = δv − α∇̂L(W + δv, b + δv);
    W = W + v;
    b = b + v;
end while
```

Nesterov's momentum enjoys several pleasant properties in a convex scenario, such as a quadratic convergence rate. However, these advantages are not guaranteed to hold in non-convex scenarios as those faced when training FNNs.

AdaGrad algorithm The AdaGrad algorithm belongs to the class of gradient-descent algorithms that adapt the learning rate based on the cumulated gradient evaluated over multiple mini-batches. Specifically, the AdaGrad scales the learning rate by a factor that is inversely proportional to the sum of the gradients for all used mini-batches [134]. The effect of this strategy is that parameters with larger partial derivatives of the cost function decrease more rapidly than the parameters with a smaller partial derivatives. The AdaGrad algorithm is reported in Algorithm 6, with the parameter δ being a small number

(typically of the order of 10^{-7}), introduced to avoid a division by zero when updating the parameters.

Algorithm 6 AdaGrad algorithm for FNNs training.

Set $\varepsilon > 0$, $\beta > 0$, $\boldsymbol{r} = \boldsymbol{0}$, $\boldsymbol{W}, \boldsymbol{b}$;
while Validation Error larger than ε **do**
  Sample a random mini-batch $\mathcal{S}_{SGD}$;
  Compute (30) by Algorithm 2;
  $\boldsymbol{r} = \boldsymbol{r} + \widehat{\nabla L}(\boldsymbol{W}, \boldsymbol{b}) \odot \widehat{\nabla L}(\boldsymbol{W}, \boldsymbol{b})$;
  $\boldsymbol{W} = \boldsymbol{W} - \frac{\alpha}{\beta + \sqrt{\boldsymbol{r}}} \widehat{\nabla L}(\boldsymbol{W} + \delta \boldsymbol{W}, \boldsymbol{b} + \delta \boldsymbol{b})$;
  $\boldsymbol{b} = \boldsymbol{b} - \frac{\alpha}{\beta + \sqrt{\boldsymbol{r}}} \widehat{\nabla L}(\boldsymbol{W} + \delta \boldsymbol{W}, \boldsymbol{b} + \delta \boldsymbol{b})$;
end while

RMSProp algorithm. AdaGrad algorithm enjoys several pleasant properties in the convex case. However, when dealing with non-convex problems, it has been empirically observed that summing over all squared gradients used in the training process can cause a premature and excessive decrease of the learning rate. As a consequence, the learning rate might have become already too small when the algorithm finally finds a region around a (local) minimum of the cost function. The RMSProp algorithm aims at improving this drawback of AdaGrad, by introducing a moving weighted average of the gradients to reduce the relevance of gradients observed many iterations before. The formal procedure is reported in Algorithm 7 and can be readily modified to include the use of Nesterov's momentum to accelerate convergence.

Algorithm 7 RMSProp Algorithm for FNNs training.

Set $\varepsilon > 0$, $\beta > 0$, $\rho \in (0, 1)$, $\boldsymbol{r} = \boldsymbol{0}$, $\boldsymbol{W}, \boldsymbol{b}$;
while Validation Error larger than ε **do**
  Sample a random mini-batch $\mathcal{S}_{SGD}$;
  Compute (30) by Algorithm 2;
  $\boldsymbol{r} = \rho \boldsymbol{r} + (1 - \rho) \widehat{\nabla L}(\boldsymbol{W}, \boldsymbol{b}) \odot \widehat{\nabla L}(\boldsymbol{W}, \boldsymbol{b})$;
  $\boldsymbol{W} = \boldsymbol{W} - \frac{\alpha}{\beta + \sqrt{\boldsymbol{r}}} \widehat{\nabla L}(\boldsymbol{W} + \delta \boldsymbol{W}, \boldsymbol{b} + \delta \boldsymbol{b})$;
  $\boldsymbol{b} = \boldsymbol{b} - \frac{\alpha}{\beta + \sqrt{\boldsymbol{r}}} \widehat{\nabla L}(\boldsymbol{W} + \delta \boldsymbol{W}, \boldsymbol{b} + \delta \boldsymbol{b})$;
end while

Adam algorithm. The Adam algorithm was introduced in [135], and is based on the application of momentum to the RMSProp method. However, the momentum technique is used with a different flavor from the conventional momentum. Specifically, the Adam algorithm employs both the first and second moment of the gradient estimated in each mini-batch. Moreover, Adam applies a correction term to both first and second moments, scaling them by a factor approaching one as the algorithm progresses. The procedure is formally stated in Algorithm 8.

As far as Adam Algorithm is concerned, the suggested value for β is 10^{-8}, whereas the two weighting parameters ρ_1 and ρ_2 are suggested to be initialized to 0.9 and 0.999. Nevertheless, although Adam is usually quite robust to the choice of the hyperparameters, sometimes the default values need to be adjusted to obtain good convergence properties.

Parameters initialization. One critical point of any training algorithm is the initialization of the parameters, and in particular of the weights[5] $\boldsymbol{W}$. Given the non-convexity of the problem, the training algorithm will converge to some suboptimal point, and thus a suitable initialization point can make the difference between converging to an efficient or inefficient suboptimal point. Unfortunately, the design of improved initialization strategies for neural networks is a little understood topic. Consolidated approaches from pure optimization theory should be applied with caution, since they focus on obtaining a low cost function, i.e. a low training error, but there is no guarantee that this will also translate into a low generalization error.

Algorithm 8 Adam Algorithm for FNNs training.

Set $\varepsilon > 0$, $\beta > 0$, $\rho_1, \rho_2 \in (0, 1)$, $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$, $t = 0$, $\boldsymbol{W}, \boldsymbol{b}$;
while Validation Error larger than ε **do**
  Sample a random mini-batch $\mathcal{S}_{SGD}$;
  Compute (30) by Algorithm 2;
  $t = t + 1$;
  $\boldsymbol{s} = \rho_1 \boldsymbol{s} + (1 - \rho_1) \widehat{\nabla L}(\boldsymbol{W}, \boldsymbol{b})$;
  $\boldsymbol{r} = \rho_2 \boldsymbol{r} + (1 - \rho_2) \widehat{\nabla L}(\boldsymbol{W}, \boldsymbol{b}) \odot \widehat{\nabla L}(\boldsymbol{W}, \boldsymbol{b})$;
  $\widehat{\boldsymbol{s}} = \frac{\boldsymbol{s}}{1 - \rho_1^t}$;
  $\widehat{\boldsymbol{r}} = \frac{\boldsymbol{r}}{1 - \rho_2^t}$;
  $\boldsymbol{W} = \boldsymbol{W} - \frac{\alpha \widehat{\boldsymbol{s}}}{\beta + \sqrt{\widehat{\boldsymbol{r}}}} \widehat{\nabla L}(\boldsymbol{W} + \delta \boldsymbol{W}, \boldsymbol{b} + \delta \boldsymbol{b})$;
  $\boldsymbol{b} = \boldsymbol{b} - \frac{\alpha \widehat{\boldsymbol{s}}}{\beta + \sqrt{\widehat{\boldsymbol{r}}}} \widehat{\nabla L}(\boldsymbol{W} + \delta \boldsymbol{W}, \boldsymbol{b} + \delta \boldsymbol{b})$;
end while

Presently, two general rules are known concerning the initialization of the network parameters:

- Two hidden nodes connected to the same input and with the same activation function should have different initial parameters. This is needed to avoid any redundancy, since otherwise any deterministic algorithm would update the parameters of these two nodes in the same way.
- All matrices $\boldsymbol{W}_\ell$ should be initialized to full-rank matrices, since otherwise some patterns might be lost in the parameters null-space.

These two guidelines motivate a random initialization of the parameters. Accordingly, initialization values are typically chosen as independent random variables, following either the Gaussian or uniform distribution, but a critical issue is how to choose the parameters of these distributions. These choices affect the initial scale of the parameters, which can have a significant impact on the generalization error. Larger initial weights are able to suppress redundancy more effectively, but on the other hand might cause vanishing gradients due to the saturation of sigmoidal activation functions, as well as other numerical problems. In [136] it is proposed to initialize the weights of Layer ℓ following a uniform distribution in $[-\frac{-6}{N_\ell + N_{\ell-1}}, \frac{-6}{N_\ell + N_{\ell-1}}]$. Instead, [122] recommends initializing the weights to random orthogonal matrices, that are scaled by a specific gain factor depending on the particular non-linearity used in each layer. In [137] it is shown that by properly choosing the gain factor, the orthogonality assumption of the

[5]The initialization of the bias terms $\boldsymbol{b}$ has been found to have a more limited impact on the final performance.

weight matrices can be relaxed. In [138] a sparse initialization strategy is proposed in which each unit is initialized to have a pre-defined number of non-zero weights.

Regularization. When training an FNN it should always be kept in mind that the ultimate goal is to minimize the test error, rather than the training error. To this end, an essential technique is to perturb the training process so as to reduce the capacity of the neural network, thus avoiding overfitting. Any strategy aimed at reducing the test error at the expense of the training error is a regularization strategy. It has been found that it can be more effective in reducing the test error to apply a regularization strategy to an ANN with a high capacity, rather than trying to directly tune the size of the model in terms of number of layers and nodes. Over the years, several regularization methods have been proposed, and the most widely used ones are discussed here.

a) L^p regularization. One main regularization approach is to add to the cost function of Problem (24) a perturbation term proportional to the p-th power of the L^p norm of the weights. This leads to modifying the cost in (24) into

$$L_r(\boldsymbol{W}, \boldsymbol{b}) = L(\boldsymbol{W}, \boldsymbol{b}) + \phi \|\boldsymbol{W}\|_p^p \,, \tag{34}$$

wherein $\phi \in [0, \infty)$ is a hyperparameter that weighs the relative contribution of the norm penalty term relative to the standard cost function. It should be stressed that the regularization term depends only on the weights and not also on the bias terms. This is because the weights have a more significant impact on the test error, as they directly link the input and output of a node, whereas the bias terms only directly affect the output. Thus, regularizing the weights is expected to be more important than regularizing the bias terms, which would only add to the complexity of the training process without bringing much improvement. This intuition has been experimentally confirmed in many research works over the years and motivates the current practice in neural networks to perform only weights regularization.

Among the different norms that can be considered in (34), the most widely used is the L^2 norm. This type of regularization is also called *weight decay* because it can be seen to reduce the magnitude of the weights, especially for larger ϕ. This results in limiting the impact of many network connections on the final output, thereby reducing the network capacity. Moreover, reducing the magnitude of the weights causes sigmoidal or hyperbolic tangent activation functions to operate in their linear regions, thus retaining the advantages of a linear model.

Another widely used regularization norm is the L^1 norm. In comparison to L^2 regularization, L^1 regularization tends to produce a more sparse weight matrix $\boldsymbol{W}$, in which many connections in every layer are effectively turned off. Besides reducing the network capacity, this also reduces the memory required to store the model.

b) Early stopping. Perhaps the simplest form of regularization is represented by the early stopping technique. All training algorithms are designed to minimize the training error in (24) iteration after iteration. However, recalling also Fig. 4, the validation error initially decreases together with the training error, but at some point tends to increase again. Thus, the idea of early stopping is to stop the training phase when the validation error reaches its minimum value. In practice, the network parameters are saved after each gradient update and when the validation error has not improved for a pre-specified number of iterations, the training algorithm stops and the parameters corresponding to the lowest observed validation error are returned. It is observed in [139] and [140] that limiting the number of training iterations t reduces the volume of parameter space reachable from the initial parameters, thereby reducing the capacity of the ANN and acting as a regularizer.

c) Dropout. The idea of dropout is to introduce a perturbation by randomly changing the topology of the neural network every time a new data sample is used [141]. Specifically, for each data sample, each neuron in the ANN has a probability p of being included in the network and if it is not included the corresponding weights are not updated in that particular iteration of the algorithm. Dropout is an effective regularizer due to two main reasons:

- By randomly removing a subset of connections each time, dropout is actively weakening the coupling among neighboring neurons. This reduces the possibility of performing too complex operations, which could cause overfitting.
- Each time a subset of neurons is randomly disconnected, a different reduced network is being trained. As a result, using dropout effectively trains a large number of different, random ANNs, and then averages the results, which tends to reduce the net effect of overfitting.

Batch Normalization. One issue when working with gradient-based methods, is the different scale that the features in the input vector, as well as the activation values of each layer, might have. In the presence of vectors with components that have very different scales with one another, numerical problems can arise and gradient descent can be slow. In order to avoid this issue, [142] has proposed to normalize the input data and/or the activation values of each layer in the network.

Formally speaking, let us consider the training data points $\mathbf{x}_0^{(1)}, \ldots, \mathbf{x}_0^{(N_{TR})}$. Then, batch normalization modifies the operation performed by the input layer, which will not simply forward the input vector, but will apply the transformation:

$$\tilde{\mathbf{x}}_0 = \frac{\mathbf{x}_0^{(nt)} - \boldsymbol{\mu}_0}{\boldsymbol{\Psi} + \boldsymbol{\sigma}_0} \,, \; \forall \; nt = 1, \ldots, N_{TR} \,, \tag{35}$$

wherein the division is meant component-wise, $\boldsymbol{\Psi}$ is vector with positive components of the order of 10^{-8}, whose purpose is to avoid dividing by zero, while $\boldsymbol{\mu}_0$ and $\boldsymbol{\sigma}_0$ are mean and standard deviation vectors defined as

$$\boldsymbol{\mu}_0 = \frac{1}{N_{TR}} \sum_{nt=1}^{N_{TR}} \mathbf{x}_0^{(nt)} \tag{36}$$

$$\boldsymbol{\sigma}_0 = \sqrt{\frac{1}{N_{TR}} \sum_{nt=1}^{N_{TR}} (\mathbf{x}_0^{(nt)} - \boldsymbol{\mu}_0) \odot (\mathbf{x}_0^{(nt)} - \boldsymbol{\mu}_0)} \,, \tag{37}$$

where the square root operation is meant component-wise.

Denoting by $\boldsymbol{z}_\ell^{(nt)}$ the N_ℓ-dimensional vector of activation values of layer ℓ when $\mathbf{x}_0^{(nt)}$ is the input of the network, a similar normalization technique can be applied to the vectors

$\{\boldsymbol{z}_\ell^{(1)}, \ldots, \boldsymbol{z}_\ell^{(N_S)}\}$ in each mini-batch, thus changing the arguments of the activation functions of the ℓ-th layer to be:

$$\tilde{\boldsymbol{z}}_\ell = \frac{\boldsymbol{z}_\ell^{(nt)} - \boldsymbol{\mu}_\ell}{\boldsymbol{\Psi} + \boldsymbol{\sigma}_\ell} \ , \ \forall \ nt = 1, \ldots, N_S \ , \tag{38}$$

with $\boldsymbol{\mu}_\ell$ and $\boldsymbol{\sigma}_\ell$ having similar definitions as in (36) and (37). In addition, when applied to a hidden layer, it is common to further modify the input to the activation functions in (38) as:

$$\tilde{\boldsymbol{z}}_\ell = \boldsymbol{\gamma}_\ell \odot \tilde{\boldsymbol{z}}_\ell + \boldsymbol{\beta}_\ell \ , \ \forall \ nt = 1, \ldots, N_S \ , \tag{39}$$

with $\boldsymbol{\gamma}_\ell$ and $\boldsymbol{\beta}_\ell$ being N_ℓ-dimensional parameters to be learnt during the training phase. The operation in (39) is aimed at preserving the representational power of the network, which would be significantly diminished by constraining each layer to have zero-mean and unit-variance activation inputs. However, this approach might seem counterintuitive, since it seems to defeat the purpose of applying the normalization step in (38) in the first place. The advantage of using (39) lies in the fact that now $\boldsymbol{\gamma}_\ell$ and $\boldsymbol{\beta}_\ell$ are parameters to be learnt based on the normalized values in $\tilde{\boldsymbol{z}}_\ell$, which are more conveniently handled by gradient descent. Moreover, while batch normalization increases the number of parameters to optimize during the training phase, it can be seen that applying (39) makes the bias terms in each node useless. In other words, when using batch normalization, it should be set $\boldsymbol{b}_\ell = \boldsymbol{0}$ for any normalized layer, since the role of $\boldsymbol{b}_\ell$ is now played by $\boldsymbol{\beta}_\ell$. As consequence, the only new parameters to be trained are the vectors $\boldsymbol{\gamma}_\ell$ for the layers where normalization is applied.

It is also important to mention that batch normalization has a regularization effect, too, due to at least two main reasons:

- Since $\boldsymbol{\mu}_\ell$ and $\boldsymbol{\sigma}_\ell$ are computed on each mini-batch, they will be slightly different for each mini-batch. This introduces a slight perturbation that has a regularizing effect on the overall network, similarly to what is done by the dropout technique.
- The fact that batch normalization reduces the variability of the input data to each layer weakens the coupling among different layers, again having a similar effect as the dropout technique.

So far, batch normalization has been described as a technique to aid the training process. However, since it modifies the structure and operation of the network, it also affects the network use at test time. In other words, if a FNN has been trained using batch normalization, at test time we should compute (39) in each layer, using the trained parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$. However, the problem in doing this is that at test time there might not be a large enough dataset at our disposal to compute reliable estimates of mean an variance for each activation input. This problem is typically solved computing an exponentially-weighted average including the means and variances computed during the training phase on each mini-batch, plus the new data sample at test time.

*2) **Hyperparameter tuning - Fitting the data**:* So far, many techniques have been presented to tune the parameters of a FNN in order to achieve a low generalization error. However, the performance of all algorithms that have been presented depends on several hyperparameters, that are not directly tuned during the training phase. Examples of hyper-parameters are the number of network layers and neurons per layer, the size of the training set and of each mini-batch, the learning rate, the regularization coefficient, etc. Moreover, other choices that have a significant impact on the overall performance are related to which training algorithm is used, what initialization point is adopted, what regularization strategy to use, whether or not to use batch normalization, etc.

As discussed in Section II-B, hyperparameter tuning can be performed either manually or in an automated way. The three automated methods introduced in Section II-B, grid-search, random search, hyperparameter optimization, are general enough to apply not only to the deep learning context, but rather to generic machine learning methods. However, grid search and hyperparameter optimization are rarely used for deep learning applications. The former is deemed practical only when three or fewer hyperparameters must be tuned, in which case a logarithmic search scale is used to span a wider range of values. The latter is problematic due to the lack of an expression of the cost function with respect to some hyperparameters and also because, as a general fact, a hyperparameter optimization algorithm in turn has its own hyperparameters to set, even though they are typically less problematic to tune. Instead, random search appears a more feasible solution, which has been shown to reduce the validation error to acceptable values much faster than grid search [104].

Parallel to these automated methods, manual hyperparameter setting represents an effective way to achieve the desired performance with affordable complexity. Nevertheless, compared to automated approaches, manual tuning of the hyperparameters requires a higher degree of experience, and is typically carried out by monitoring both training and validation error during the training phase, thereby determining whether the network is underfitting or overfitting, and modifying the hyperparameters to adjust the network capacity accordingly. To this end, in general a trial and error procedure is required, since it is very challenging to know in advance the optimal configuration of hyperparameters for the specific problem at hand. Nonetheless, some general guidelines can be identified, recalling that the capacity of an ANN depends on three main factors: 1) the ability of the network to represent the problem at hand; 2) the ability of the learning algorithm to successfully minimize the cost function during the training phase; 3) the degree to which the training procedure regularizes the model, thus avoiding overfitting.

As shown in Fig. 10, when configuring a neural network, the first issue to take care of is to make sure that the network does not underfit. If the performance on the training set is poor, it means that the network can not fit the available training data and thus it is generally useless to gather more data. Instead, the focus should be on improving the optimization algorithm and the most important hyperparameter in this sense is the learning rate. Unfortunately, each task has its own optimal learning rate, and trial and error should be used to find a learning rate that yields a low enough training error for the task at hand.

Apart from the learning rate, other strategies to increase the network capacity are to tune the other hyperparameters

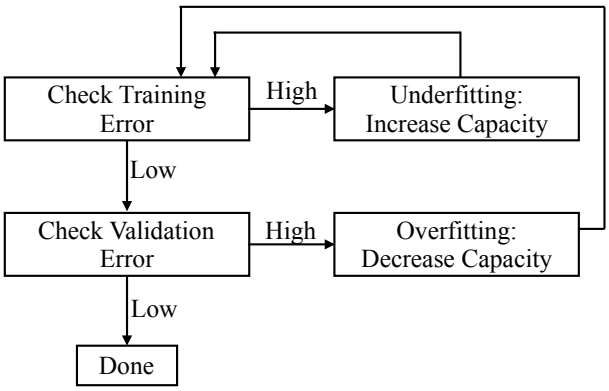

Figure 10. Scheme for manual hyperparameter setting in ANNs.

of the algorithm in use or to consider more sophisticated optimization algorithms. Widely-used choices are SGD with momentum, RMSProp, or Adam, possibly coupled with Nesterov's momentum technique. Moreover, batch normalization should be included if the training error does not decrease as desired. If these strategies are not effective, the problem could be in the size of each mini-batch, which might be too small to provide a reliable estimate of the gradient. Finally, another conceptually simple way to increase the network capacity is to use more neurons and layers. This is a very powerful way to avoid underfitting, but comes at the expense of a larger complexity and the feasibility of this approach should be evaluated given the available computational resources. If none of these strategies work, the problem might just be in the quality of the training data, which might be too noisy and/or might not include the right features to describe the problem at hand. In this case, it might be worth starting over, collecting different training data.

Once a low enough training error has been achieved, the validation error should be checked. If it is unsatisfactory, then it is likely that the problem lies in overfitting. In this case, the most effective strategy is to just gather more data. However, gathering more data can be costly and requires higher storage and processing capabilities. A simpler way of reducing the network capacity is to employ a regularization technique. Actually, it is advisable to use early stopping from the start, while other regularizing techniques could be included in the training procedure if needed. Finally, a third approach consists of manually reducing the model size, limiting the number of neurons and layers. If these approaches do not work even after a careful tuning of their hyperparameters, then gathering more data appears as the only way to avoid overfitting.

Finally, it should be stressed that the validation error is simply an estimate of the test error and the discussion above assumes that such an estimate is reliable. If it happens that test error is high despite the validation error being low, then the most effective approach is to increase the size of the validation set. However, if this does not solve the problem then it is likely that the cost function used for training and validation is not appropriate for the task to perform and thus a different performance function could be considered.

D. Deep Reinforcement Learning

This section presents the framework of deep reinforcement learning, which merges deep learning with reinforcement learning. It is to be stressed that the framework of reinforcement learning is not directly related to deep learning, but rather it is a different machine learning approach that implements the learning procedure in an adaptive way, namely by interacting with the environment by taking actions and receiving feedback on the result of the actions that have been taken. However, recently, the idea has been put forth to merge deep learning and reinforcement learning, motivated by the consideration that deep learning provides an efficient way to implement reinforcement learning techniques. More in detail, in a reinforcement learning scenario, depending on the information available on the environment, several techniques can be used to determine the optimal sequence of actions to take in order to maximize the long-term utility. In this context, ANNs can facilitate the computation of the optimal policy to follow when, as it is often the case, the statistics and parameters of the environment are not fully known. The rest of this section first provides a short introduction to reinforcement learning, whose purpose is solely to define basic terminology and briefly describe mathematically the scenario where deep learning can be exploited. For a dedicated and comprehensive treatment of the reinforcement learning framework, we refer the reader to [97].

Reinforcement learning apply to scenarios that can be mathematically described by a Markov Decision Process (MDP). An MDP is defined by the following quantities:

- $\mathcal{S}$, the set of possible states.
- $\mathcal{A}$, the set of possible actions that the agent can take.
- $\mathcal{P}$, the set of transition probabilities, with $P(s_t, s_{t+1}, a_t)$ the probability of moving from state s_t to state s_{t+1} by taking action a_t.
- $\mathcal{R}$, the set of rewards, with $\mathcal{R}(s_t, a_t) = \mathbb{E}\left[R_{t+1}|s_t, a_t\right]$, and R_{t+1} the reward obtained at step $t+1$.
- $\gamma \in [0, 1]$, a discount factor adjusting the weight of more recent actions.

Based on this notation, it is possible to define the long-term reward as

$$G_t = \sum_{k=0}^{+\infty} \gamma^k R_{t+k+1} \ , \tag{40}$$

and a (stationary) policy as the probability of taking action a at time t, when being in state s, namely:

$$\pi(a, s) = P(A_t = a|S_t = s) \ , \tag{41}$$

where the word stationary refers to the fact that the probability of taking action a when in state s does not depend on time.

A key concept when analyzing an MDP is that of **action-value function**, measuring the value, in terms of expected reward, of being in state s and taking action a, following policy π, namely:

$$Q_\pi(S_t = s, A_t = a) = \mathbb{E}_\pi\left[G_t|S_t = s, A_t = a\right] \tag{42}$$

The action-value function can be also rewritten as the sum of the reward at step $t+1$, plus the long-term reward from $t+1$ to ∞, namely:

$$\begin{aligned} &Q_\pi(S_t = s, A_t = a) = \\ &\mathbb{E}_\pi \left[R_{t+1} + \sum_{k=1}^{+\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right] = \\ &\mathbb{E}_\pi \left[R_{t+1} + \gamma \sum_{k=0}^{+\infty} R_{t+k+2} | S_t = s, A_t = a \right] = \\ &\mathcal{R}(S_t, A_t) + \gamma Q_\pi(S_{t+1}, A_{t+1}) \end{aligned} \tag{43}$$

At this point, we can also define the optimal action-value function as

$$Q^*(s,a) = \max_\pi \; Q_\pi(s,a) \; . \tag{44}$$

Solving (44) for each pair (s,a) provides a full characterization of the MDP problem, and allows determining the best policy to follow for each possible state and action. To this end, several methods are available, depending on the information available on the MDP. An optimality condition for Problem (44) is the so-called Bellman's optimality equation, which however requires full knowledge of the MDP model and parameters to be solved.

However, in practical scenarios, assuming a complete knowledge of the MDP model is quite unrealistic. Typically, only the response from the environment is observable, but no information is available as to the statistics regulating the MDP process, such as the transition probabilities, which makes it impossible to compute the value of the Q function for any pair (s,a). In these cases, one possible approach is to obtain the values of the Q function from experience, i.e. by starting the process from each possible (s,a) pair, and then following different policies, observing the rewards returned by the environment at each step. However, this approach has the clear drawback of requiring a too high computational complexity, especially when the number of possible (s,a) pairs is very large. A similar drawback is suffered by all other alternative methods aimed at building a table collecting the possible values $Q(s,a)$, for all possible $s \in \mathcal{S}$ and $a \in \mathcal{A}$.

In scenarios with a very large (possibly even infinite) number of (s,a) pairs, the state-of-the-art approach is that of Q-learning. As the name implies, this approach is based on learning the values of the Q function. More specifically, Q-learning algorithms assume a functional form for the function $Q(s,a)$, namely:

$$Q(s,a) \approx \widehat{Q}(s,a,\boldsymbol{w}) \; , \tag{45}$$

with $\widehat{Q}$ a known function, and $\boldsymbol{w}$ a set of parameters to be determined by any machine learning method, with the goal of improving the accuracy of the approximation. More specifically, Q-learning methods assume that some points of the Q function, say $\{Q(s_i,a_i)\}_{i=1}^{N_T}$, have been already determined, for example by trying some actions and observing the response of the environment. Then, the parameters in the vector $\boldsymbol{w}$ are determined so as to minimize the mean squared error between the ground truth values $\{Q(s_i,a_i)\}_{i=1}^{N_T}$ and the model (45).

Now, traditional Q-learning approaches typically employ a linear model for $\widehat{Q}$, but more recently it has been proposed to adopt an ANN with weights $\boldsymbol{w}$, that takes as input a pair (s,a) and outputs the corresponding value $Q(s,a)$. The parameters $\boldsymbol{w}$ are trained by using the samples $\{Q(s_i,a_i)\}_{i=1}^{N_T}$ as the training set. This is the main idea of deep reinforcement learning [98], [99], which can be considered an algorithm belonging to the family of Q-learning methods, with the peculiarity that the approximate function $\widehat{Q}(s,a,\boldsymbol{w})$ is specified through an ANN. Thus, compared to other Q-learning methods, deep reinforcement learning has the significant advantage of not specifying a-priori the functional form of $\widehat{Q}$, leaving to the ANN the task of determining the best functional form to use. Indeed, being universal function approximators, ANNs will be able to approximate the true function $Q(s,a)$ within any desired tolerance, provided a proper training phase is performed.

E. Deep unfolding

As discussed, one of the issues with ANNs is how to determine the number of neurons and layers to use. However, in some cases it is possible to match the iterations of some iterative algorithm to the layers of an ANN by a technique called deep unfolding [143]. This provides a way to determine the hyperparameters of an ANN to efficiently implement a given number of iterations of a recursive algorithm.

To elaborate, the idea of deep unfolding applies to all algorithms that take as input a vector $\mathbf{x} = [x_1, \ldots, x_N]$ and produce as output a vector $\boldsymbol{y} = [y_1, \ldots, y_M]$ expressed by

$$y_i = g_i(\mathbf{x}, \boldsymbol{\phi}, \boldsymbol{\theta}) \; , \; \forall \; i = 1, \ldots, M \; , \tag{46}$$

wherein $\boldsymbol{\theta}$ is a vector containing all the parameters of the algorithm, while $\boldsymbol{\phi} = [\phi_1, \ldots, \phi_N]$ is iteratively updated according to the formula

$$\phi_i^{(k)} = f_i(\mathbf{x}, \boldsymbol{\phi}^{(k-1)}, \boldsymbol{\theta}) \; , \tag{47}$$

with k the iteration index and $\boldsymbol{\phi}^{(0)}$ the initial value. This formalism applies to detection tasks [144], as well as to the computation of posterior probabilities by the belief propagation method, or to inference techniques aimed at estimating a distribution by minimizing its divergence from an approximate distribution [143].

The main idea of deep unfolding lies in the observation that (46) can be regarded as the input-output relationship of a deep neural network, with (47) being the input-output relationship of Layer k, and $\boldsymbol{\theta}$ representing the parameters of the neural network, i.e. all weights and bias of each layer. Then, the iterative algorithm can be *unfolded* by mapping each iteration to one layer of an ANN, which takes as inputs $\mathbf{x}$ and $\boldsymbol{\phi}_0$, compute $\boldsymbol{\phi}^{(k)}$ at the output of the k-th hidden layer, and finally produce $\boldsymbol{y}$ as output, as displayed in Fig. 11.

Two main points are to be highlighted here:

- Unlike what typically happens with ANNs, in the case of deep unfolding, the operation of the ANN in terms of number of nodes and layers is determined by the particular algorithm that is unfolded. Specifically, the number of layers is fixed by the amount of iterations of

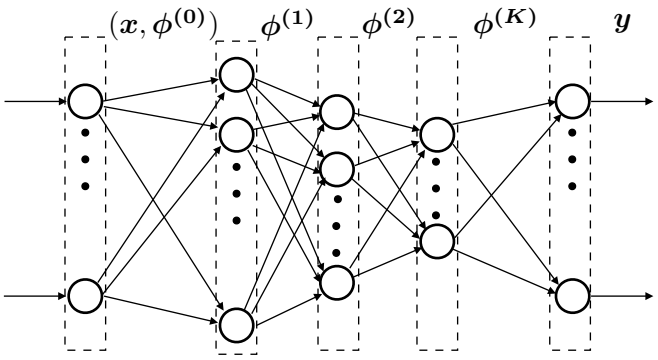

Figure 11. $K+1$ iterations of an iterative algorithm are unfolded onto the K hidden layers and onto the output layer of an ANN. The output of each layer is equal to the output produced by one iteration of the algorithm, and the output of the last layer is equal to the output of the last iteration of the algorithm.

the algorithm, while the number of nodes in each layer is fixed by the sizes of the vectors $\mathbf{x}$, $\boldsymbol{\phi}$, and $\boldsymbol{y}$.

- The advantage of unfolding an algorithm onto an ANN rather than implementing it directly, lies in the fact that the parameters $\boldsymbol{\theta}$ of the algorithm are determined by an ANN, instead of being set by more conventional methods. Moreover, once the parameters are determined, the ANN can be directly used as a fast implementation of the iterative algorithm to compute $\boldsymbol{y}$ based on the chosen parameters $\boldsymbol{\theta}$.

F. Deep Transfer learning

Deep transfer learning is yet another recent framework that combines deep learning with another machine learning framework, namely transfer learning. In the broadest sense, transfer learning studies how to transfer the knowledge that is used in a given context to execute a given task, into a different, but related context, to execute another task. Formally speaking, four fundamental components can be identified in a transfer learning problem:

- A source task, $\mathcal{T}_S$, i.e. the original task for whose execution the knowledge to be transferred was developed.
- A source domain, $\mathcal{D}_S$, i.e. the context in which the task $\mathcal{T}_S$ was executed.
- A target task, $\mathcal{T}_T$, i.e. the new task to be executed thanks to the knowledge transfer.
- A target domain, $\mathcal{D}_T$, i.e. the new context in which the task $\mathcal{T}_T$ must be executed.

Clearly, such a problem formulation is very general, and need not be related to any deep learning problem. However, transfer learning can be successfully used to facilitate the implementation of deep learning algorithms, especially by reducing the amount of data to be acquired for training and validation purposes. Indeed, as discussed, the availability of large quantities of data is a prerequisite for deep learning to outperform other machine learning methods, but in the context of wireless communication networks the acquisition of large amount of data can be too expensive and/or not practical. In these cases, transfer learning techniques can be used by transferring knowledge from other related scenarios in which data acquisition has been already performed. For example, datasets for similar communication systems can be used, and/or datasets generated according to (possibly inaccurate) mathematical models describing the task to be executed.

Despite being a relatively recent approach, many techniques for deep transfer learning have already appeared in the literature and it is difficult to provide a general taxonomy. Here, following the taxonomy by the recent tutorial [145], we categorize transfer learning techniques into four main classes.

*1) **Instance-based transfer learning**:* This approach assumes to have data from both the source domain $\mathcal{D}_S$ and target domain $\mathcal{D}_T$. Then, the idea is to exploit both datasets to carry out the target task $\mathcal{T}_T$, by assigning a different weight to each instance of the source and target data. Otherwise stated, data from the source domain is used to augment the data from the target domain, but it must be weighted differently to ensure that instances that are specific to the source domain are given less or no importance during the training process. After this re-weighting step, the augmented data set is used as training set for the target task by any traditional training algorithm, with the re-weighting factors acting as hyperparameters to be adjusted during the validation process.

In principle, this method does not require having labeled data, in the sense that, once the new dataset has been built, it can be used in conjunction with any machine learning method. However, as far as training a neural network is concerned, it is required that the training set be labelled in order to implement available training algorithms. Recently, instance-based transfer learning has proved effective when employed in conjunction with the AdaBoost training algorithm, addressing both classification and regression problems [146], [147].

*2) **Mapping-based transfer learning**:* Mapping-based transfer learning redefines the training cost function in order to account for the presence of data from both the source and target domains. Specifically, the cost function used during the training phase is defined as:

$$\mathcal{L}(\boldsymbol{W}, \boldsymbol{b}) = \mathcal{L}_S(\boldsymbol{W}, \boldsymbol{b}) + \lambda \mathcal{L}_T(\boldsymbol{W}, \boldsymbol{b}) + R^2(\boldsymbol{W}, \boldsymbol{b}) \ , \quad (48)$$

wherein $\mathcal{L}_S$ is the cost function for the source task, taking as input training samples from the source domain, $\mathcal{L}_T$ is the cost function for the target task, taking as input training samples from the target domain, λ is a non-negative term weighting the relative importance of the two cost functions, and R is a regularization function that accounts for the differences between source and target domains. More in detail, the regularizer R is typically chosen as the *maximum mean discrepancy* function between the source and target domains, with respect to a generic representation $\phi(\cdot)$, namely [148]

$$\text{MMD} = \left\| \frac{1}{|\mathcal{X}_S|} \sum_{x \in \mathcal{X}_S} \phi(x) - \frac{1}{|\mathcal{X}_T|} \sum_{x \in \mathcal{X}_T} \phi(x) \right\| \ , \quad (49)$$

wherein $\mathcal{X}_S$ and $\mathcal{X}_T$ denote the source and target available datasets. Thus, this approach requires having labelled data from both the source and target domains. Based on (48), any standard training algorithm can be executed, exploiting all available labeled data.

Recent studies on mapping-based transfer learning have focused on analyzing the performance when other regularizers are used. In [149] it is proposed to use a multiple kernel variant of the MMD (MK-MMD), while in [150] it is proposed to use the joint maximum mean discrepancy as regularizer. Finally, we mention [151], where Wasserstein's distance is used as regularizer and is shown to achieve better performance than the MDD in some cases.

*3) **Network-based transfer learning**:* Network-based deep transfer learning implements the transfer of knowledge by first training an ANN to execute the source task $\mathcal{T}_S$ in the source domain $\mathcal{D}_S$, and then reusing and/or refining the obtained network configuration to execute the target task $\mathcal{T}_T$ in the target domain $\mathcal{D}_T$. This general concept can be applied in several different ways. For example, it is possible to identify a part of the ANN that extracts general features that describe both the source and target tasks. Then, after training the ANN in the source domain, the part of the ANN that applies to both source and target tasks need not be trained again. This approach is taken in [152], where a language processing application is considered, and it is proposed to divide the ANN in two parts. The former extracts language-independent features, that can be reused for all languages, while the latter is language-specific and needs to be trained for each new language.

Nevertheless, a more common approach is to perform a two-step training, in which the ANN is first trained to execute the source task, yielding a tentative configuration of network parameters. Next, a second training phase is performed in the target domain, which uses the configuration of the weights and bias from the first phase as the initialization point for the training algorithm. This approach is very useful in all situations in which a lot of training data is available in the source domain, whereas the target domain provides only a few labeled training samples. As to be describes in more detail in Section IV, this is the typical scenario that arises in wireless communications applications, and indeed Section IV will present several case-studies wherein this particular transfer learning method proves extremely useful. Techniques inspired to network-based transfer learning have been recently proposed for resource allocation in wireless communications in [153], [154].

*4) **Adversarial-based transfer learning**:* The main idea of adversarial transfer learning is to identify the common feature between source and target task through the use of an another deep neural network, called generative adversarial network (GAN) [155]. The first step of the approach is to divide the neural network that implements the source task into two segments, one that extracts the salient features of the source domain, and one that exploits these features to carry out the source task. Then, the output of the first segment of the neural network is also fed to another neural network, the GAN, which has the task of discriminating whether the input comes form the source domain or from the target domain. The two networks are trained together as if they were a single neural network, even though they have competing goals: the adversarial network aims at minimizing the error in the discrimination between target and source inputs, while the main network aims at minimizing the error on the source task, while at the same time aiming at *maximizing* the error that the adversarial network makes in discriminating between data coming from the source or target domain. Indeed, if the adversarial network is not able to distinguish between source and target domain, then the first segment of the main network has found a representation of the source domain that is virtually indistinguishable from the target domain, and thus the main network can be used to execute both the source and target tasks. The contrasting goals during the training process are modeled by defining the overall training cost function as:

$$\mathcal{L}(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{V}, \boldsymbol{c}) = \mathcal{L}_m(\boldsymbol{W}, \boldsymbol{b}) - \lambda \mathcal{L}_a(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{V}, \boldsymbol{c}) \ , \quad (50)$$

wherein $\mathcal{L}_m$ is the error on the source task, $\mathcal{L}_a$ is the error in discriminating between source and target inputs, λ is a factor weighting the relative importance of these two errors, $\boldsymbol{W}$ and $\boldsymbol{b}$ are the weights and bias terms of the main network, while $\boldsymbol{V}$ and $\boldsymbol{c}$ are the weights and bias of the adversarial network, and the overall cost function needs to be minimized with respect to $\boldsymbol{W}, \boldsymbol{b}$, and maximized with respect to $\boldsymbol{V}, \boldsymbol{c}$. By minimizing (50) with respect to $\boldsymbol{W}, \boldsymbol{b}$, the primary network is minimizing $\mathcal{L}_m$ while at the same time maximizing $\mathcal{L}_a$. Instead, by maximizing (50) with respect to $\boldsymbol{V}$ and $\boldsymbol{c}$ the adversarial network is minimizing $\mathcal{L}_a$. As a result, unlike typical training procedures that aim at minimizing the training cost function, the goal here is to determine a saddle point of (50), which can be accomplished by several saddle-point algorithms based again on stochastic gradient descent techniques, as in regular training procedures [156], [157]. It is to be stressed that, in order to find a saddle point of (50), it is not required to know the desired output for each training sample. Indeed, each training sample must simply carry a label discriminating whether the sample comes from the source or target domain, but the desired output is required only if the sample comes from the source domain. This means that adversarial training can be used for neural network training also when the available target data is not labeled.

IV. Applications to wireless communications

After presenting the main concepts and tools of the deep learning framework, this section describes practical applications to the design of wireless communication systems. First, a literature survey is performed, reviewing available contributions about the application of deep learning to the physical layer of wireless communication systems, and then several novel applications are described in detail.

A. State-of-the art Review

Deep learning as a tool to perform various tasks at the physical layer of wireless communication systems has started attracting research attention only very recently, mostly in the last couple of years. For this reason, fewer contributions have appeared than in other areas of wireless communications. Nevertheless, two main research directions can be identified:

- Deep learning to operate the physical layer, simplifying the execution of tasks such as data detection, decoding, channel estimation, localization, etc.

- Deep learning to manage the physical layer, simplifying radio resource allocation tasks.

1) Operation of the physical layer: The first area of application of deep learning at the physical layer of wireless networks has been the use of ANN to simplify the implementation of detection and/or estimation operations such as information decoding, channel estimation, localization, etc. [158]–[184].

In [158], the authors use deep FFNs to emulate the transmitter and the receiver of point-to-point communication systems, while assuming the communication channel is known. The end-to-end system is modeled as deep neural network composed of the cascade of a neural network implementing the data transmission process, one layer implementing the known channel (whose parameters are fixed and not trainable), and another neural network implementing the reception process. The overall network receives as input the information symbol and provides as output the corresponding symbol estimate. This architecture is referred to as an *auto-encoder*, since the goal of the network is to reproduce at the output, the input data. It is shown that, without having any information about the implementation of the transmitter/receiver chains, the auto-encoder is able to outperform traditional approaches that instead design the system based on (approximate) mathematical models of the transmitter/receiver chains. The work in [158] paved the way for many subsequent studies that exploited ANNs at the physical layer of wireless devices. In [159] it is proposed to use an auto-encoder to jointly minimize the system BER and PAPR, and again an improvement over traditional methods is obtained. Deep learning is used for data detection in MIMO systems in [160], [161], in decode-and-forward relay channels [162], and for equalization and synchronization in OFDM systems in [163].

In all of these works, perfect knowledge about the communication channel is assumed. Several subsequent works have tried to relax this assumption. In [164] a two-stage approach is taken. At first, a synthetic channel model is used to provide a first training of the neural network. Next, this initial training is refined at the receiver based the true channel characteristics. Generative Adversarial Networks are used in [165]–[167], exploiting a surrogate channel for training purposes. A combination of supervised training and reinforcement learning is used in [168] to remove the need of channel knowledge. In [169], the auto-encoder approach is further extended to the case in which no channel state information is available by exploiting a stochastic perturbation approach. A similar scenario is considered in [170], where the auto-encoder approach has been used for data detection without any channel knowledge, considering molecular communications as a main application scenario. The use of fully connected ANNs for molecular communications is also investigated in [171].

In [172] it is shown that a deep neural network can reliably learn the MMSE channel estimator, while in [173] convolutional neural networks are successfully used to implement a fingerprinting-based scheme for users localization. Channel estimation through neural networks is successfully demonstrated in [174] and also in [175], where an FDD massive MIMO system is considered, and the channels are assumed to be representable by a finite-size dictionary. Experiments showing the performance of deep learning methods for users localization in outdoor environments are provided in [176], showing that even simple ANNs architectures can achieve satisfactory performance. In [177] it is shown that deep learning can be successfully used to implement error correction tasks, while [178] shows that machine learning is able to provide reliable channel estimation from compressed measurements. Channel estimation in rapidly time-varying environments is discussed in [179], and it is shown that deep architectures are able to cope with this more challenging setup, while [180] proposes a deep learning approach for joint equalization and decoding in wireless networks. Surveys on the use of ANNs to implement encoding/decoding information as well as channel estimation tasks with limited side information have appeared in [181], [182], whereas an information-theoretic study on the mutual information between input and output of a shallow neural network is provided in [183]. Channel estimation and signal detection are also performed through deep learning in [184], showing that similar performance as traditional methods can be achieved, but with a much lower computational complexity.

2) Management of the physical layer: More recently, a second emerging application area of deep learning has been the use of deep learning to perform radio resource allocation at the physical layer, with minimum complexity and/or side-information requirements [153], [185]–[195].

The works [185], [186] put forward the idea of using ANNs for network resource management, providing an overview of potential applications of AI for network resource management in future 5G wireless networks, discussing pros and cons of supervised, unsupervised, and reinforcement learning. In [189], a fully connected FNN is used for sum-rate maximization in interference networks, by learning the input-output map given by each iteration of the iterative weighted MMSE power control algorithm [196]. This approach is able to mimic the performance of the weighted MMSE resource allocation algorithm, while at the same time significantly reducing the computational complexity. In [153], [190], the problem of energy efficiency maximization in wireless interference networks by a fully connected FNN is tackled. Unlike [189], in [153], [190] the FNN is directly trained based on the optimal energy-efficient power allocation, that can be computed offline by the method appeared in [197]. The results indicate that the globally optimal performance can be approached with limited complexity, thus enabling an online implementation. A similar approach is proposed in [191], [192] for the cases of power control in massive MIMO systems and for user-cell association in multi-cell systems. Instead, a different approach is taken in [194], where a fully connected ANN is trained to solve the sum-rate maximization problem subject to maximum power and minimum rate constraints. In order to reduce the complexity of building the training set, the authors propose to train the neural network using directly the system sum-rate as training cost function. The result show a gain compared with previous low-complexity optimization methods, even though a performance loss compared to the global solution is expected.

In [188] a cloud-RAN system with caching capabilities is considered. Echo-state neural networks, an instance of RNNs, are used to enable base stations to predict the content

request distribution and mobility pattern of each user, thus determining which content to cache. It is shown that the use of deep learning increases the network sum effective capacity of around 30% compared with baseline approaches based on random caching. In [187] deep reinforcement learning is used to develop a power control algorithm for a cognitive radio systems in which a primary and secondary user share the spectrum. It is shown that both users can meet their QoS requirements despite the fact that the secondary user has no information about the primary user's transmit power. The use of deep reinforcement learning is also considered in [193], where it is used to develop a power control algorithm for weighted sum-rate maximization in interference channels subject to maximum power constraints. The proposed algorithm exhibits fast convergence and satisfactory performance. A decentralized robust precoding scheme in a network MIMO system is developed in [195] by ANNs. The proposed method is shown to outperform state-of-the-art approaches.

B. Learning to optimize

The rest of this section will describe in detail several applications, primarily focusing on the most recent area of ANN-based physical layer resource allocation. In this context, a promising approach is to develop methodologies to embed available prior knowledge about the problem to solve, into deep learning methods, that instead are purely data-driven. The motivation for this approach lies in the consideration that purely data-driven approaches become unfeasibly complex for large-scale applications, due to the huge amount of required data, and to the related processing complexity. This is an emerging topic even in fields of science where purely data-driven deep learning techniques are a consolidated reality. For example, in [198] image processing for object position detection in robotics applications is considered, and it is observed that augmenting a small training set of real images with a large dataset of synthetic images significantly improves the estimation accuracy with respect to processing only the small dataset of real images. Similar results have been obtained in [199] with reference to speech recognition applications.

In the context of wireless communications merging purely data-driven techniques based on deep learning, with expert knowledge coming from (even approximate) theoretical models holds an even greater potential. Indeed, despite their possible inaccuracy or cumbersomeness, theoretical wireless models still provide much deeper prior information compared to what is available in other fields of science. In our opinion, this clear advantage of wireless communications should not be wasted. More specifically, when performing resource allocation, depending on the system complexity, one is faced with one of the four cases shown in Tab. I:

While, it is clear that Cases C.1 and C.4 should be handled by traditional system design approaches, and fully data-driven techniques, respectively, the most appropriate way of tackling Cases C.2 and C.3 is an open issue. Indeed, Cases C.2 and C.3 offer the possibility of a cross-fertilization between model-aided and data-driven approaches, due to the fact that a model is available, even though it is inaccurate or cumbersome

C1: An accurate and tractable theoretical model is available (e.g. point-to-point channel capacity and bit-error rates.
C2: An accurate, but intractable theoretical model is available (e.g. achievable sum-rate in interference-limited systems).
C3: A tractable, but inaccurate model is available (e.g. dense networks deployment, energy consumption, hardware impairments).
C4: Only inaccurate and intractable models are available (e.g. molecular communications, end-to-end wireless communications).

Table I
SCENARIOS FOR RESOURCE MANAGEMENT IN WIRELESS NETWORKS

to optimize. Moreover, C.2 and C.3 are the typical cases in wireless communications, where models and optimization algorithms are usually available, despite being the result of some approximations and simplifications.

In order to tackle Cases C.2 and C.3, we propose the following two methodological approaches, to be further detailed through the case-studies presented in the rest of this section:

- **Optimizing a model.** When in Case C.2, an analytical expression of the performance metric to optimize is available. Then, an ANN can be trained to learn the map between the system parameters and the corresponding optimal resource allocation, namely following the technique anticipated in Section I-D. This approach is depicted in Fig. 12.
- **Refining a model.** When in Case C.3, a two-step approach can be exploited. In the first step, an ANN is trained based on synthetic data generated from the approximate model. Next, a second training phase based on true, measured data will refine the ANN configuration. This approach is depicted in Fig. 13.

As it will become clear from the applications to be presented next, the two main advantages of the proposed approaches are that:

- They drastically reduce the complexity compared to purely model-based methods, thus enabling real-time resource allocation with near-optimal performance.
- They drastically reduce the amount of required data compared to purely data-driven methods, thus dispensing with expensive and unpractical measurements campaigns.

With the exception of one case-study related to the auto-encoder approach, all applications to be described in the following will address resource allocation problems by one of the two methodologies described above.

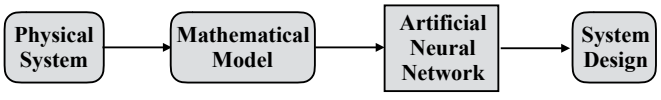

Figure 12. **Optimizing a model.** An ANN is trained based on data generated from the theoretical models. No measurement campaign is needed.

1) Physical layer design: Optimizing the receiver of a molecular communication system: In this section, we consider

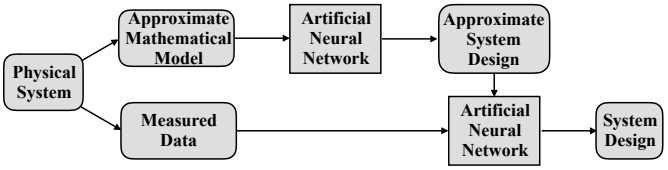

Figure 13. **Refining a model.** An ANN is first trained based on data generated from the approximate theoretical models, and then refined based on a small dataset of measured data.

the typical case study of optimizing the receiver of a communication system. As an example, we focus our attention on a molecular communication system, where chemical signals instead of electromagnetic signals are used to convey information [200]. The motivation of this choice is the complexity of modeling molecular communication systems, and the possibility of leveraging data-driven methods in this context [201]. A similar approach can be used to design and optimize the receivers of different communication systems. The objective is to prove that, by assuming that the system model is accurate, model-based and data-driven methods yield the same optimal receiver designs if they are both appropriately designed.

As a practical case study, we consider a molecular communication system where diffusion is employed for allowing information particles propagate from a transmitter to a receiver. Due to the intrinsic characteristics of diffusion, the resulting transmission channel is usually affected by a non-negligible Inter-Symbol Interference (ISI), which, if not taken into account for system optimization, may severely degrade the system performance. For this reason, we focus our attention on optimizing the receiver operation in the presence of ISI. In particular, we consider a threshold-based demodulator and denote by τ the demodulation threshold. Let $\bar{s}_i$ be the estimate of symbol s_i at time-slot i, a threshold-based demodulator operates as follows:

$$\bar{s}_i = \begin{cases} 0, & r_i \leq \tau \\ 1, & r_i > \tau \end{cases} \tag{51}$$

where r_i is the number of molecular received at time-slot i.

Under the typical operating conditions discussed in detail in [202] for a binary modulation scheme, the error probability as a function of τ can be formulated as follows:

$$P_e(\tau) = \frac{1}{2^L} \sum_{\boldsymbol{s}_{i-1}} P_e(\boldsymbol{s}_{i-1}, \tau) \tag{52}$$

where:

$$\begin{aligned} P_e(\boldsymbol{s}_{i-1}, \tau) = \frac{1}{2}[Q(\lambda_0 T + \sum_{j=1}^{L} s_{i-j} C_j, \lceil \tau \rceil) \\ + 1 - Q(\lambda_0 T + \sum_{j=1}^{L} s_{i-j} C_j + C_0, \lceil \tau \rceil)] \end{aligned} \tag{53}$$

and $Q(\lambda, n) = \sum_{k=n}^{\infty} \frac{e^{-\lambda} \lambda^k}{k!}$ is the incomplete Gamma function, L is the memory of the chemical channel, i.e., the length of the ISI, λ_0 is the background noise power per unit time, T is the duration of the time-slot, and C_j is the average number of received information particles at the jth time-slot.

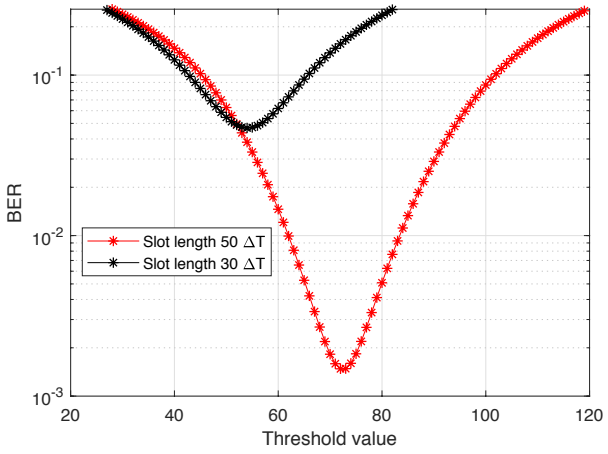

Figure 14. Error probability as a function of τ (the signal-to-noise-ratio is equal to 30 dB) for two different durations of the time-slot (amount of ISI).

In order to obtain appropriate performance and, thus, reduce the error probability, the detection threshold, τ, needs to be appropriately chosen and optimized. In Fig. 14, we depict the error probability as a function of τ for a typical system setup. We observe that an optimal value of τ exists that minimizes the error probability and that depends on the time slot duration T, i.e., the amount of ISI for a given channel.

In mathematical terms, the optimal threshold that minimizes the error probability can be formulated as follows:

$$\tau^* = \arg\min_{\tau} \quad P_e(\tau) \tag{54}$$

Due to the analytical complexity of (52), it is not possible to compute τ^* explicitly, but it can be obtained numerically at an affordable complexity.

An alternative approach is to employ a data-driven approach that does not rely on any model but uses only empirical data, e.g., a large set of values for r_j. More precisely, we consider an ANN whose aim is to demodulate the transmitted data by minimizing the error probability. An ANN-based demodulator is a system whose input is the number of received information particles, r_i at the ith time-slot, and the outputs are the probabilities that the transmitted bit is 0 or 1, i.e., $P_i(s_i = 0|r_i)$ and $P_i(s_i = 1|r_i)$, respectively. Since, $P_i(s_i = 1|r_i) + P_i(s_i = 0|r_i) = 1$, only one of the two probabilities is needed. We use the notation $P_i = P_i(s_i = 1|r_i)$. Based on the outputs, the ANN demodulate the received bits as follows:

$$\bar{s}_i = \begin{cases} 0, & P_i \leq 0.5 \\ 1, & P_i > 0.5 \end{cases} \tag{55}$$

where the threshold 0.5 accounts for the fact that the bits are equiprobable.

In order to train the ANN, we consider a supervised learning approach, i.e., we compute the parameters (e.g., the bias factors and the weights) of the ANN by using a known sequence of transmitted bits. In particular, we use the

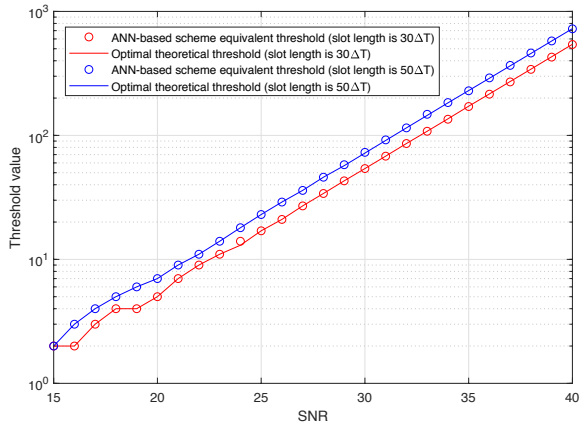

Figure 15. Demodulation thresholds: Model-based vs. data-driven for two different durations of the time-slot (amount of ISI).

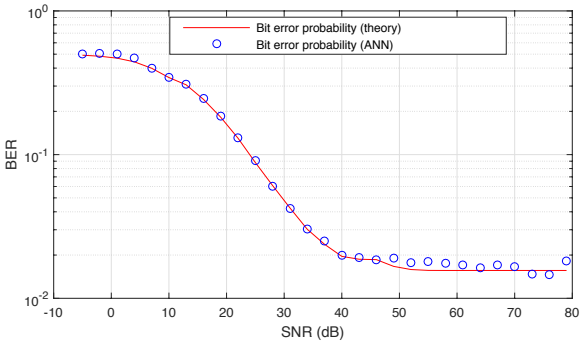

Figure 16. Bit error probability of the ANN-based demodulator vs. the analytical framework - $T = 30\Delta T$.

Bayesian regularization back propagation technique (trainbr), which updates the weights and biases by using the Levenberg-Marquardt optimization algorithm. The set of parameters to train and operate the ANN are as follows: The number of layers is 10, the learning rate is 0.01, the training epoch is 200, the number of training bits is 1000, the number of validation bits is 100000, and the replication time is 50. In particular, the training is performed in a batch mode, and the replication time denotes the number of batches each of which is 1000-bit long.

In Fig. 15, we compare the optimal threshold computed numerically from (54) as a function of the signal-to-noise-ratio, and the demodulation threshold that is learned by the ANN-based demodulator. In the latter case, the threshold is obtained, after completing the training of the ANN, and identifying the input, i.e., the number of information particles, for which the output probability is equal to 0.5. We observe that the ANN-based implementation is capable of learning the demodulation threshold in a very accurate manner.

In Fig. 16 and Fig. 17, we compare the bit error probability of the ANN-based demodulator against the bit error probability in (52) by considering a short symbol time (small ISI) and

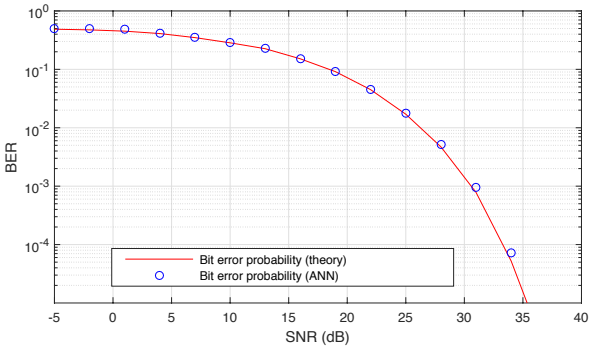

Figure 17. Bit error probability of the ANN-based demodulator vs. the analytical framework - $T = 50\Delta T$.

a long symbol time (large ISI), respectively. As for the analytical model, the optimal threshold is estimated from (54) for each value of the signal-to-noise-ratio. We note a very good agreement even with only 10 layers.

In summary, this section shows that an optimal receiver design can be obtained by relying solely on data-driven methods that that the resulting ANN can be used for system optimization, e.g., to optimize the demodulation threshold.

2) Optimizing a model: power control in wireless networks: This application focuses on the maximization of the bit-per-Joule energy efficiency in generic wireless interference networks. The consideration of the energy efficiency as a performance metric communication systems has emerged as key aspect of future wireless networks, motivated by the need to provide 1000x higher data rates compared to present systems, while at the same time halving the energy consumption. As a result, in 5G wireless networks, bit-per-Joule energy efficiency will have to increase by a factor 2000 compared to present wireless networks [2], [4].

Traditional approaches for energy efficiency maximization in wireless networks are based on the theory of fractional programming, the branch of optimization theory that focuses on the optimization of fractional functions. A tutorial on fractional programming methods for energy efficiency maximization in wireless communication is available in [5]. There, it is observed that achieving the global maximum of the energy efficiency metric requires in general exponential complexity whenever the communication system is interference-limited. Here, we will show how the global maximum of the energy efficiency can be approached with limited complexity through the use of ANNs.

To elaborate, let us consider an interference networks in which K single-antenna transmitters communicate with M receivers, each equipped with N antennas. Denote by $\boldsymbol{h}_{k,m}$ the $N \times 1$ channel from transmitter k to receiver m, by p_k the transmit power of transmitter k, by $\boldsymbol{c}_k$ the $N \times 1$ receive vector used by the receiver associated to transmitter k, and by σ_m^2 the received noise power at receiver m. Then, the signal to interference plus noise ratio (SINR) enjoyed by transmitter

k at its associated receiver m_k is expressed as:

$$\gamma_k = \frac{p_k |\boldsymbol{c}_k^H \boldsymbol{h}_{k,m_k}|^2}{\sigma^2 + \sum_{j \neq k} p_j |\boldsymbol{c}_k^H \boldsymbol{h}_{j,m_k}|^2} = \frac{p_k d_{k,k}}{\sigma^2 + \sum_{j \neq k} p_j d_{k,j}} \ , \tag{56}$$

with $d_{k,j} = |\boldsymbol{c}_k^H \boldsymbol{h}_{j,m_k}|^2$, for all k and j.

Based on (56), the network global energy efficiency (GEE) is given by

$$\text{GEE} = \frac{B \sum_{k=1}^{K} \log_2(1+\gamma_k)}{P_c + \sum_{k=1}^{K} \mu_k p_k} \quad \text{[bit/Joule]} \ , \tag{57}$$

wherein B is the communication bandwidth, P_c is the hardware static power consumed in the whole system, and μ_k the inverse of the efficiency of the power amplifier used by transmitter k. It is important to stress that P_c will depend on system parameters such as the number of antennas and the efficiency of the system hardware components. However, it is assumed that P_c does not depend on the transmit powers, and therefore the specific model expressing P_c as a function of the system hardware components is inessential as far as resource allocation is concerned.

Given this setup, the considered energy efficiency maximization problem is stated as the maximization of the GEE subject to power constraints, namely

$$\max_{\{p_k\}_{k=1}^K} \ \text{GEE}(p_1, \ldots, p_K) \tag{58a}$$

$$\text{s.t. } P_{min,k} \leq p_k \leq P_{max,k} \ , \forall \ k = 1, \ldots, K \tag{58b}$$

with $P_{max,k}$ and $P_{min,k}$ being the maximum feasible and minimum acceptable transmit powers for user k. The challenge in tackling (58) lies in the fact that the numerator of (58a) is not a concave function of $\mathbf{p} = \{p_k\}_{k=1}^K$, as a consequence of the fact that multi-user interference is present in (58). In this case, global optimization methods are required to find the optimal power allocation, while more practical approaches guarantee first-order optimality with a polynomial complexity [197]. Moreover, Problem (58) needs to be solved anew whenever the channel realizations $\{\boldsymbol{h}_{\ell,m_k}\}_{k,\ell}$ change. This represents a critical drawback, especially considering that the resource allocation process must be completed well before the end of the channel coherence time in order for the optimized power vector to be practically useful. This observation makes it difficult to employ even polynomial-complexity algorithm to perform resource allocation in real-time, i.e. following the small-scale variations of the channel coefficients.

In oder to address this issue and enable real-time power control, it is possible to resort to deep ANNs paired with the use of energy efficiency models and traditional optimization approaches. Specifically, in this scenario we are in Case C.2 of Table I, since a model is available and has allowed us to formulate Problem (58). However, the model is too complex (for practical implementations) to be optimized by directly using traditional optimization methods. On the other hand, we can exploit the model by using it to train an ANN to learn the map between the system parameters, and the corresponding optimal power allocation. To elaborate, let us observe that Problem (58) can be regarded as an unknown function mapping from the coefficients $\{d_{k,\ell}\}_{k,\ell}$ and maximum/minimum transmit powers P_{max} and P_{min}, to the optimal power allocation vector $\mathbf{p}^*$, namely

$$\mathcal{F} : \boldsymbol{d} = \{d_{k,\ell}, P_{min,k}, P_{max,k}\}_{k,\ell} \in \mathbb{R}^{K(M+2)} \rightarrow \mathbf{p}^* \in \mathbb{R}^K \tag{59}$$

Then, based on the result that ANNs are universal function approximators [114], it is possible to train an ANN so that its input-output relationship reproduces the unknown map (64). This leads to considering an ANN with $K(M+2)$ input nodes and K output nodes, to be trained so that it outputs the optimal $K \times 1$ power vector $\mathbf{p}^*$ corresponding to a given $K(M+2) \times 1$ input of system parameters $\boldsymbol{d}$. This enables to update the resource allocation without having to solve any optimization problem every time the system parameters change, but by simply feeding the new vector $\boldsymbol{d}$ to the ANN, and obtaining the corresponding power allocation as the output of the ANN.

It is important to emphasize that this entails a negligible computational complexity compared to using sophisticated numerical optimization algorithms. Indeed, once all the parameters and hyperparameters of the ANN are fixed, the ANN basically provides a closed-form expression of its input-output relationship, whose complexity is related to computing $\sum_{\ell=1}^{L+1} N_{\ell-1} N_\ell$ real multiplications[6] and evaluating $\sum_{\ell=1}^{L+1} N_\ell$ activation functions, with N_ℓ denoting the number of neurons in Layer ℓ in accordance with the notation of Section III-A.

Instead, a higher complexity is required to generate a suitable training set, because this requires to consider many different system parameters realizations $\{\boldsymbol{d}_{nt}\}_{nt=1}^{N_T}$, and to compute the corresponding desired power allocation vector $\{\mathbf{p}_{nt}^*\}_{nt=1}^{N_T}$ by actually solving (58) N_T times, by leveraging existing optimization framework, such as monotonic fractional programming or sequential fractional programming [197]. At a first sight, this might seem to cause a complexity overhead that defeats the purpose of using ANNs to reduce the computational complexity of resource allocation. However, this is not the case for at least two major reasons that make the generation of the training set fundamentally different from solving Problem (58) in real-time:

- The training set can be generated and used *offline* to train the ANN. Thus, a higher complexity can be afforded and real-time constraints do not apply.
- The training set needs to be updated at a much longer time-scale than that with which the network parameters change.

In other words, the training process needs not be executed each time a system parameter changes, and the solution needs not be obtained within the channel coherence time. Thus, the use of traditional optimization theory to generate the training set does not defeat the practicality of the proposed ANN-based approach. On the contrary, the use of mathematical models to formulate the optimization problem and the use of traditional optimization techniques to build the training set, represent the expert knowledge that is exploited to facilitate the use of ANNs for real-time power control in wireless networks.

Numerical performance analysis. We considered the uplink of a MIMO system in which $N_c = 4$ base stations serve

[6]The complexity related to additions is negligible compared to that related to multiplications

$K = 10$ users deployed in a square area with edge 2000 m. The base stations are deployed at coordinates $(500, 500)$ m, $(500, -500)$ m, $(-500, -500)$ m, $(-500, 500)$ m, and are all equipped with $N_R = 2$ antennas. Instead, the mobile users are randomly deployed in the square area and are equipped with a single antenna. The path-loss has been modeled following [203], with power decay factor equal to 4.5, while fast fading terms have been modeled as realizations of zero-mean, unit-variance complex Gaussian random variables. The circuit power consumption term is equal to $P_c = 0.01$ W for all users and maximum ratio combining is adopted at all base stations. The noise power at the base station has been generated as $\sigma^2 = F\mathcal{N}_0 B$, wherein $F = 3$ dB is the receive noise figure, $B = 180$ kHz is the communication bandwidth, and $\mathcal{N}_0 = -174$ dBm/Hz is the noise spectral density.

In this context, the optimal system global energy efficiency can be computed using the monotonic fractional programming framework developed in [197], which however requires exponential complexity. Nevertheless, we can use the monotonic fractional programming method to produce a training set for an ANN that will learn the optimal power allocation policy. Specifically, we have considered $N_{TR} = 10^5$ instances of the MIMO systems, that were independently generated as far as users' locations and channel realizations are concerned, optimizing the transmit powers for each instance, which yields the training set $\{\boldsymbol{d}_i, \mathbf{p}_i^*\}_{i=1}^{N_{TR}}$. Out of the N_{TR} data samples, 10^4 have been used for validation purposes, while the others have been used to implement Adam training algorithm with Nesterov's momentum over a FFN with $L = 6$ fully-connected hidden layers, having 512, 256, 128, 64, 32, 16 neurons, respectively, plus an output layer with $K = 10$ neurons providing the power allocation to use. All hidden layers except the first one have ReLU activation functions, while an exponential linear unit activation has been used for the first layer. Finally, the output layer employs a linear activation function, motivated by the consideration that enforcing the power constraints directly in the activation function of the last layer might mislead the FFN. Indeed, enforcing the power constraint directly through the activation function might provide low MSEs simply due to the use of cut-off levels in the activation function, instead of being the result of proper adjustment of the hidden layer parameters. In this case, the FFN would not be able to realize that the current MSE level is acceptable only because the desired power level is close to either P_{max} or P_{min}, and the clipping at the output layer provides by construction such a power level, regardless of the configuration adopted in the hidden layers.

As loss function, for each training sample, the mean squared error has been used, defined as:

$$\text{MSE} = \frac{1}{K}\sum_{k=1}^{K}\left(p_k - p_k^*\right)^2 \ . \tag{60}$$

After the training and validation phase, the ANN has been tested over $N_{Test} = 10^4$ independently generated channel realizations. Here, the power constraint has been enforced a-posteriori, by post-processing the ANN output, manually setting to P_{max} all output powers exceeding the maximum power constraint and to P_{min} all outputs below the minimum power constraint.

Table IV-B2 reports the MSE in the power estimation and the relative MSE in the corresponding GEE value over the test set, respectively defined as

$$\text{MSE}_{\mathbf{p}} = \frac{1}{KN_{test}}\sum_{i=1}^{N_{test}} \|\mathbf{p}_{opt}^{(i)} - \mathbf{p}_{ANN}^{(i)}\|^2$$

$$\text{MSE}_{r,\text{GEE}} = \frac{1}{N_{test}}\sum_{i=1}^{N_{test}} \left(\frac{\text{GEE}_{opt}^{(i)} - \text{GEE}_{ANN}^{(i)}}{\text{GEE}_{opt}^{(i)}}\right)^2$$

wherein $\mathbf{p}_{opt}^{(i)}$ and $\text{GEE}_{opt}^{(i)}$ are the optimal power vector and corresponding GEE value for the i-th test sample, while $\mathbf{p}_{ANN}^{(i)}$ and $\text{GEE}_{ANN}^{(i)}$ are the power vector and corresponding GEE value output by the ANN for the i-th test sample. Different values of P_{max}, assumed equal for all users, are shown.

Table II
MEAN SQUARED ERROR BETWEEN THE OPTIMAL POWER ALLOCATION AND GEE VALUE AND THOSE OUTPUT BY THE TRAINED ANN, FOR DIFFERENT VALUES OF P_{max}.

	-20 dBW	**-10 dBW**	**0 dBW**	**10 dBW**
$\mathbf{MSE}_{\mathbf{p}}$	$5.02{*}10^{-4}$	$8.11{*}10^{-4}$	$9.44{*}10^{-4}$	$8.72{*}10^{-4}$
$\mathbf{MSE}_{r,\text{GEE}}$	0.020	0.013	0.012	0.013

The results evidence a remarkable ability of the ANN to approach the optimal power allocation profile, for different values of P_{max}, which corroborates the use of ANNs for practical power allocation in wireless networks.

This result is further confirmed by Fig. 18, that shows the optimal GEE value achieved over the test set versus P_{max}, comparing it with the GEE value predicted by the ANN in the saturation region (i.e. $P_{max} \geq -10$ dbW), that is the region of interest for system operation, since it provides the best GEE value. It is seen that the ANN achieves near-optimal performance, attaining around 98% of the optimal GEE value and largely outperforming the benchmark scheme in which full power allocation is used, i.e. when all terminals transmit with the maximum feasible power P_{max}. This further confirms the usefulness of ANN-based power control as a power allocation method striking a better performance-complexity trade-off than existing alternatives.

3) Optimizing a model: user-cell association in wireless networks: This application has a similar flavor as that in Section IV-B2, with the difference that instead of allocating the users' transmit powers, the problem is that of deciding the assignment between transmitters and receivers. This means that, while the case-study in Section IV-B2 tackles a continuous resource allocation problem, and thus can be regarded as a regression problem, here the focus is on a discrete resource allocation problem, which can be seen as a classification problem.

To proceed further, let us consider a similar system set-up and notation as in Section IV-B2, with the difference that now the receiver associated to transmitter k has not been fixed,

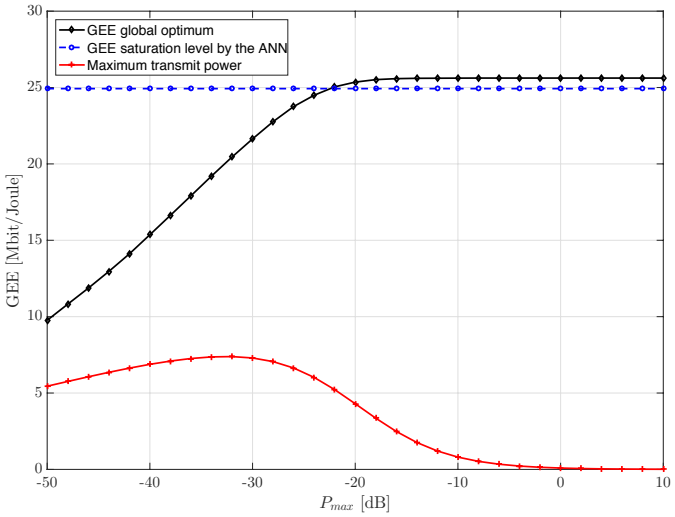

Figure 18. GEE versus P_{max} by: (a) Global optimum; (b) ANN-based power allocation; (c) Full power transmission.

yet. Then, the SINR enjoyed by transmitter k if associated to receiver m is written as:

$$\gamma_{k,m} = \frac{p_k |\boldsymbol{c}_{k,m}^H \boldsymbol{h}_{k,m}|^2}{\sigma_k^2 + \sum\limits_{\ell \neq k} p_\ell |\boldsymbol{c}_{k,m}^H \boldsymbol{h}_{\ell,m}|^2} \ , \tag{61}$$

Then, denoting by $\rho_{k,m}$ the binary variable taking value 1 when transmitter k is served by receiver m and zero otherwise, the system sum-rate is expressed as

$$\text{SR} = B \sum_{k=1}^{K} \sum_{m=1}^{M} \rho_{k,m} \log_2(1 + \gamma_{k,m}) \ , \tag{62}$$

with B denoting the communication bandwidth.

Defining for notational convenience $d_{k,m} = \log_2(1+\gamma_{k,m})$, for all k and m, and $\boldsymbol{\rho} = \{\rho_{k,m}\}_{k,m}$, the sum-rate maximization problem is cast as:

$$\max_{\boldsymbol{\rho}} \sum_{k=1}^{K} \sum_{m=1}^{M} \rho_{k,m} d_{k,m} \tag{63a}$$

$$\text{s.t.} \sum_{m=1}^{M} \rho_{k,m} \leq 1 \ , \ \forall \ k = 1, \ldots, K \tag{63b}$$

$$\sum_{k=1}^{K} \rho_{k,m} \leq a_m \ , \ \forall \ m = 1, \ldots, M \tag{63c}$$

$$\sum_{m=1}^{M} \rho_{k,m} d_{k,m} \geq R_{min,k} \ , \ \forall \ k = 1, \ldots, K \tag{63d}$$

$$\rho_{k,m} \in \{0,1\} \ , \ \forall \ m = 1, \ldots, M \ , \ \forall \ k = 1, \ldots, K \ , \tag{63e}$$

wherein Constraints (63b) and (63c) ensure that each transmitter can be associated to only one receiver and that each receiver can serve at most a_m transmitters, while Constraint (63d) guarantees minimum QoS for each transmitter, and Constraint (63e) is due to the integrality of the association variables.

Typical approaches to solve linear programs such as (63) resort to branch-and-cut techniques, which require solving a series of continuous relaxations of (63). In some special cases, i.e. when $R_{min,k}$ is integer for all k, the constraint matrix of Problem (63) can be shown to be totally uni-modular, which enables to solve (63) through just one continuos relaxation. Nevertheless, this still requires to employ numerical optimization algorithms, whose complexity might still be quite high, especially in large networks. Moreover, as in the power control case from Section IV-B2, the optimal association rule needs to be computed in real-time, thus implying that Problem (63) needs to be solved anew each time any of the coefficients $\{\boldsymbol{d}_{k,m}\}_{k,m}$ changes. Moreover, in order to be useful, the solution needs to be obtained well before the coefficients $\{\boldsymbol{d}_{k,m}\}_{k,m}$ change again.

In order to reduce the complexity of the resource allocation process, we observe that the considered problem is again an instance of Case C.2 of Table I, since a model is available and has allowed us to formulate Problem (63). Then, following a similar approach as in Section IV-B2, the optimization program in (63) can be seen as the problem of determining the unknown map:

$$\mathcal{F} : \boldsymbol{d} = \{d_{k,m}, R_{min,k}, a_m\}_{k,m} \in \mathbb{R}^{KM+K+M} \rightarrow \boldsymbol{\rho}^* \in \{0,1\}^{KM} \tag{64}$$

which can be tackled by resorting again to a fully-connected FFNs, taking $(KM + K + M)$-dimensional inputs and producing KM-dimensional outputs, with similar implementation and complexity considerations as those in Section IV-B2.

Numerical performance analysis. We considered the uplink of a MIMO system in which $N_c = 4$ base stations serve $K = 10$ users deployed in a square area with edge 2000 m. The base stations are deployed at coordinates $(500, 500)$ m, $(500, -500)$ m, $(-500, -500)$ m, $(-500, 500)$ m, and are all equipped with $N_R = 2$ antennas. Instead, the mobile users are randomly deployed in the square area and are equipped with a single antenna. The path-loss has been modeled following [203], with power decay factor equal to 4.5, while fast fading terms have been modeled as realizations of zero-mean, unit-variance complex Gaussian random variables. The noise power at the base station has been generated as $\sigma^2 = F\mathcal{N}_0 B$, wherein $F = 3$ dB is the receive noise figure, $B = 180$ kHz is the communication bandwidth, and $\mathcal{N}_0 = -174$ dBm/Hz is the noise spectral density. All users are assumed to transmit with power P_{max}, assumed equal across the uplink terminals.

For the numerical results, no rate constraints have been enforced in Problem (63). As a results, as already mentioned, it can be shown that the constraint matrix of Problem (63) becomes totally unimodular, which implies that Problem (63) can be globally solved by solving its continuous relaxation in which $\rho_{k,m} \in [0,1]$, for all k and m. In other words, if no rate constraints are enforced, relaxing the association variables in Problem (63) to continuous values in $[0,1]$ leads to a continuous problem whose optimal solution is such that $\rho_{k,m} \in \{0,1\}$, and thus is feasible for Problem (63), too. On the other hand, if also rate constraints, are enforced, the optimal solution of Problem (63) can be obtained by means of branch-and-cut techniques.

In our simulation, we have built a training set with size $N_{TR} = 10^5$, considering independent instances of the MIMO system with respect to users' locations and channel realizations, and solving (63) for each system scenario, which yields the training set $\{\boldsymbol{d}_i, \boldsymbol{\rho}_i^*\}_{i=1}^{N_{TR}}$. In addition, 10^4 more data samples have been used for validation purposes. Adam training algorithm with Nesterov's momentum has been used to train a fully-connected ANN with $L = 6$ hidden layers, having 1024, 512, 256, 256, 128, 64 neurons, respectively, while the output layer has $KM = 40$ neurons yielding the association between users and base stations. All hidden layers have ReLU activation functions, whereas the output layer has a sigmoidal activation function. The mean squared error has been used as loss function during the training phase, for each training sample, namely:

$$\text{MSE} = \sum_{i=1}^{N_{TR}} \left\| \boldsymbol{\rho}^{(i)} - \boldsymbol{\rho}^{*(i)} \right\|^2 . \tag{65}$$

After the training and validation phase, the ANN has been tested over $N_{Test} = 10^4$ independently generated channel realizations. For each data sample, denoting by $\boldsymbol{\rho} = \{\rho_{k,m}\}_{k,m}$ the ANN output, user k has been associated to base station $\bar{m}$ if

$$\bar{m} = \arg\max_{m} \ \rho_{k,m} \tag{66}$$

The results in terms of accuracy of the power allocation estimation are reported in Table IV-B3 for different values of P_{max}, assumed equal for all users. Specifically, Table IV-B3 reports, for each considered value of P_{max}, the average MSE, over the test set, between the optimal assignment and the assignment predicted by the ANN, as well as the MSE in the corresponding sum-rate values, namely:

$$\text{MSE}_{\boldsymbol{\rho}} = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \|\boldsymbol{\rho}_{opt}^{(i)} - \boldsymbol{\rho}_{ANN}^{(i)}\|^2$$

$$\text{MSE}_{r,\text{SR}} = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \left(\frac{\text{SR}_{opt}^{(i)} - \text{SR}_{ANN}^{(i)}}{\text{SR}_{opt}^{(i)}} \right)^2 ,$$

wherein $\boldsymbol{\rho}_{opt}^{(i)}$ and $\text{SR}_{opt}^{(i)}$ are the optimal assignment vector and corresponding sum-rate value, while $\boldsymbol{\rho}_{ANN}^{(i)}$ and $\text{SR}_{ANN}^{(i)}$ are the assignment vector and corresponding sum-rate value output by the ANN.

Table III
MEAN SQUARED ERROR BETWEEN THE OPTIMAL TRANSMITTER-RECEIVER ASSOCIATION AND THE RESULTING SUM-RATE VALUE, AND THOSE OUTPUT BY THE TRAINED ANN, FOR DIFFERENT VALUES OF P_{max}.

	-20 dBW	**-10 dBW**	**0 dBW**	**10 dBW**
MSE$_{\boldsymbol{\rho}}$	0.176	0.169	0.170	0.168
MSE$_{r,\text{SR}}$	$2.75*10^{-4}$	$3.02*10^{-4}$	$2.32*10^{-4}$	$3.11*10^{-4}$

The results indicate that the ANN is able to associate transmitters and receivers in such a way to yield a near-optimal system sum-rate, for different values of P_{max}.

This is further confirmed in Fig. 19, which shows the optimal sum-rate value achieved over the test set versus P_{max}, comparing it with the sum-rate value predicted by the ANN for $P_{max} = 10$ dBW (i.e. in the region of interest for typical communication system operation). It is seen that the ANN achieves near-optimal performance, attaining more than 99% of the optimal sum-rate value. Thus, ANN-based resource allocation proves useful also to solve classification problems, in which the variables to optimize take on discrete values, rather than continuous ones.

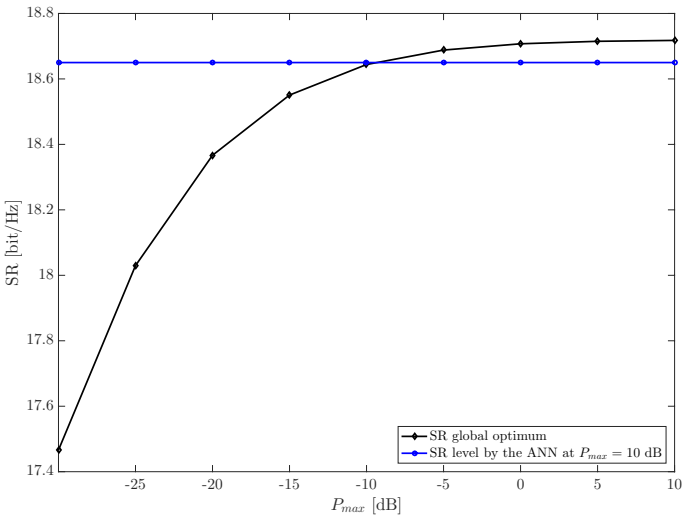

Figure 19. Comparison between the maximum sum-rate obtained by standard optimization methods and the sum-rate obtained by the ANN at $P_{max} = 10$ dB.

4) Refining a model by deep learning - Cellular networks beyond the Poisson point process: In this section, we consider the case study where an analytical model exists and is analytical tractable, but it is not considered to be sufficiently accurate for system optimization. We assume, in addition, that more accurate network models are difficult to develop and/or are not suitable for system optimization. As a practical example, we consider the optimization of the Energy Efficiency (EE) [66] in non-Poisson cellular networks [81], which is known to be an intractable optimization problem because of the analytical complexity of the utility function to optimize.

As discussed in Section I-C, we propose to solve this issue by relying on deep transfer learning. Our proposed idea consists of jointly exploiting model-based and data-driven optimization. The approach consists of first optimizing the network using a mismatched, but simpler for optimization, model, and then refining the result with (few) empirical data. Let us assume, as a practical example, that the mismatched (approximated) model is the Poisson model. More precisely, we assume that the only inaccuracy of the system model is the spatial distribution of the cellular base stations, while all the other parameters and modeling assumptions as considered to be accurate. More general system setups can be considered, and another example is studied in the next section. In detail, the approximated model is assumed to be Poisson point process model, while the "exact" point process model is assumed to be the square grid model [79]. This is a simple example that

is chosen in order to shed light on our proposed approach, and that is also easy to simulate and reproduce.

From [66], we know that the EE in Poisson cellular networks is available in closed-form and is amenable for optimization. Thus, a large dataset of optimal values for the EE as a function of any system parameters can be readily obtained. This dataset is used to train a (mismatched) ANN with the desired accuracy. The issue, as mentioned, is that the original network model is non-Poisson. We assume, however, that the considered cellular network deployment is equipped with a sensing platform, e.g., by using the metasurfaces discussed in Section I-C, that can sense and report some contextual data about the network, which is used to obtain a dataset of just a few empirical but optimal values of the EE, which account for the actual non-Poisson spatial model. This dataset is used to tune the ANN and to correct the mismatch. The intuition behind this proposed approach is that, despite mismatched, the initial ANN embeds the most important features of the cellular network already, and thus less data is needed compared with the case study where no pre-training is performed. The objective of this section is to study the amount of empirical samples that the proposed approach based on transfer learning, which jointly combines model and data, requires to achieve similar performance as a pure data-driven method. If the amount of empirical data is not that large, the proposed approach will be successful and will also reduce the amount of overhead, to collect the empirical samples, that is needed to network optimization.

In the rest of this section, we discuss both pure model-based and data-driven approaches, and then combine them together based on transfer learning principles, and, more precisely on network-based transfer learning.

Model-based optimization. From [66], the EE in Poisson cellular networks can be formulated as follows:

$$\mathrm{EE}\left(\lambda_{\mathrm{BS}}\right)=\frac{\mathrm{SE}\left(\lambda_{\mathrm{BS}}\right)}{\mathrm{P}_{\mathrm{grid}}\left(\lambda_{\mathrm{BS}}\right)} \tag{67}$$

where

$$\begin{aligned}\mathrm{SE}\left(\lambda_{\mathrm{BS}}\right)=&\mathrm{B_W}\log_2\left(1+\gamma_{\mathrm{D}}\right)\frac{\lambda_{\mathrm{BS}}\mathrm{L}\left(\lambda_{\mathrm{MT}}/\lambda_{\mathrm{BS}}\right)}{1+\Upsilon\mathrm{L}\left(\lambda_{\mathrm{MT}}/\lambda_{\mathrm{BS}}\right)}\\&\times\mathrm{Q}\left(\lambda_{\mathrm{BS}},\mathrm{P}_{\mathrm{tx}},\lambda_{\mathrm{MT}}/\lambda_{\mathrm{BS}}\right)\end{aligned} \tag{68}$$

$$\begin{aligned}\mathrm{P}_{\mathrm{grid}}\left(\lambda_{\mathrm{BS}}\right)=&\lambda_{\mathrm{BS}}\mathrm{P}_{\mathrm{tx}}\mathrm{L}\left(\lambda_{\mathrm{MT}}/\lambda_{\mathrm{BS}}\right)\\&+\lambda_{\mathrm{MT}}\mathrm{P}_{\mathrm{circ}}+\lambda_{\mathrm{BS}}\mathrm{P}_{\mathrm{idle}}\left(1-\mathrm{L}\left(\lambda_{\mathrm{MT}}/\lambda_{\mathrm{BS}}\right)\right)\end{aligned} \tag{69}$$

are the spectral efficiency and the power consumption of the cellular network, respectively.

Equations (68) and (69) depend on many parameters, which are all defined in [66]. As far as the present paper is concerned, we are interested in four main parameters: λ_{BS}, which is the deployment density of the base stations, P_{tx}, which is the transmit power of the base stations, $\mathrm{P}_{\mathrm{circ}}$, which is the circuit power consumption of the base stations, and $\mathrm{P}_{\mathrm{idle}}$, which is the idle power consumption of the base stations. In this section $\mathrm{P}_{\mathrm{circ}}$ and $\mathrm{P}_{\mathrm{idle}}$ are assumed to be fixed, and they are further analyzed in the next section. The objective is to identify the optimal deployment density of the base stations, λ_{BS}, given some values of the transmit power P_{tx}. In [66], it is proved that this optimization problem has a unique solution, which is formulated as the unique root of a non-linear equation. Therefore, the optimal density of the base stations that maximizes the EE can be computed efficiently, for any given values of the transmit power. By solving this optimization problem, we can easily obtain the optimal pairs $\left(\mathrm{P}_{\mathrm{tx}},\lambda_{\mathrm{BS}}^{(\mathrm{opt})}\right)$, where $\lambda_{\mathrm{BS}}^{(\mathrm{opt})}=\arg\max_{\lambda_{\mathrm{BS}}}\left\{\mathrm{EE}\left(\lambda_{\mathrm{BS}}\right)\right\}$. These pairs can then be used as the input, P_{tx}, and the output, $\lambda_{\mathrm{BS}}^{(\mathrm{opt})}$, for training an ANN that yields the optimal deployment density of the base stations as a function of the transmit power that they employ.

Data-driven optimization. Let us assume now that we cannot rely on any analytical models and that the EE needs to be estimated by collecting empirical samples from the cellular network, from which the optimal cellular network deployment needs to be inferred. In particular, the spectral efficiency and the power consumption can be computed, respectively, as follows:

$$\begin{aligned}\mathrm{PSE}\left(\bullet\right)=&\frac{1}{\mathrm{AreaNet}}\sum_{\mathrm{Cell}(1)\in\mathrm{Net}}\sum_{\mathrm{N_{MT}}\in\mathrm{Cell}(1)}\Bigg\{\\&\frac{\mathrm{B_W}}{\mathrm{N_{MT}}}\log_2\left(1+\gamma_{\mathrm{D}}\right)\mathbf{1}\left(\mathrm{SIR}\geq\gamma_{\mathrm{D}},\overline{\mathrm{SNR}}\geq\gamma_{\mathrm{A}}\right)\Bigg\}\end{aligned} \tag{70}$$

$$\begin{aligned}\mathrm{P}_{\mathrm{grid}}\left(\bullet\right)=&\frac{1}{\mathrm{AreaNet}}\Bigg(\sum_{\mathrm{Cell}(0)\in\mathrm{Net}}\mathrm{P}_{\mathrm{idle}}\\&+\sum_{\mathrm{Cell}(1)\in\mathrm{Net}}\left(\mathrm{P}_{\mathrm{tx}}+\mathrm{P}_{\mathrm{circ}}\sum_{\mathrm{N_{MT}}\in\mathrm{Cell}(1)}\mathrm{N_{MT}}\right)\Bigg)\end{aligned} \tag{71}$$

These two formulas can be interpreted as follows. Let us consider the spectral efficiency as an example. Each mobile terminal in the cellular network determines, based on the received signal, whether it is in coverage. This is performed by measuring the average signal-to-noise-ratio during the cell association phase and the signal-to-interference-ratio during data transmission (if the first phase was successful). This condition corresponds to the term $\mathbf{1}\left(\mathrm{SIR}\geq\gamma_{\mathrm{D}},\overline{\mathrm{SNR}}\geq\gamma_{\mathrm{A}}\right)$, where $\mathbf{1}\left(\cdot\right)$ is the indicator function. Each mobile terminal, reports whether it is in coverage or not to a network controller (one bit of information). Based on the number of mobile terminals that are in coverage on a given cell (say $\mathrm{N_{MT}}$), the corresponding base station equally allocates the available spectrum (say $\mathrm{B_W}$) among them, and transmit data with a fixed rate $\frac{\mathrm{B_W}}{\mathrm{N_{MT}}}\log_2\left(1+\gamma_{\mathrm{D}}\right)$. Based on the information gathered by all the mobile terminals, it is possible to identify the base stations that have at least one mobile terminal in their corresponding cells (say $\mathrm{Cell}\left(1\right)$) and to compute the number of mobile terminals that lie in each of them for each network realization. The spectral efficiency can then be estimated by summing the rates all of active base stations and by normalizing by the area of the network under analysis. It is worth mentioning that in order to identify, e.g., the optimal

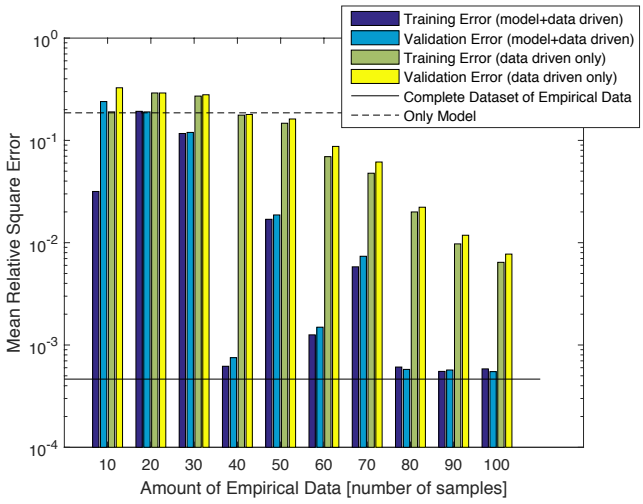

Figure 20. Comparison between model-based, data-driven, and transfer learning based optimization - Error performance of the actual and estimated average cell radius.

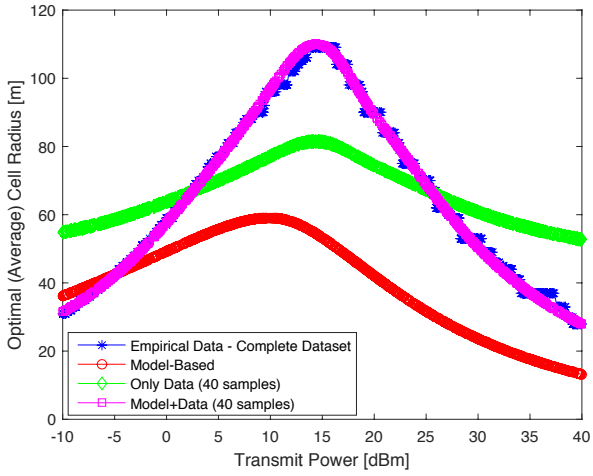

Figure 21. Comparison between model-based, data-driven, and transfer learning based optimization - Optimal deployment.

deployment density of the base stations, we need to repeat this procedure by considering all possible combinations of base station patterns, given the number of base stations actually deployed. If the optimization variable is the transmit power of the base stations, all possible values of transmit power should be tested and the value corresponding to the optimal EE should be recorded and used to train an ANN, similar to the approach discussed for model-based optimization. Based on this simple description, we can readily understand that the amount of empirical data that is necessary to train an ANN may not be negligible, and, in any case, may strongly affect the overhead for network optimization.

Network-based transfer learning optimization. Network-based transfer learning is a solution to overcome the limitations of model-based and data-driven approaches, since it is apparent that both have advantages and limitations. As mentioned already, the idea is to first train and optimize an ANN by using a model-based approach, and then refining the obtained ANN by using some empirical data (data-driven approach). Once the first model-based ANN is obtained, in particular, we consider that its configuration, i.e., the number of layers, neurons, weights, and biases, constitute the initial configuration of the second ANN that is refined based on empirical data. In our case study, we assume that, during the refinement phase, the number of layers and neurons are is modified, while the weights and biases are finely-tuned in order to account for the empirical data and to capture those features of the actual network setup that the assumed model, in order to keep its complexity at a low level, is not capable of doing.

In Figs. 20 and 21, we illustrate some numerical examples that compare the performance of the three proposed approaches. As far as the architecture of the ANN is concerned, we have considered a simple setup that is made of two layers and four neurons. As for training the ANN, we have considered 100 iterations and the Bayesian regularization back propagation algorithm. The complete dataset of empirical data is made of 10,000 samples, and the validation dataset is made of 1,000 samples. Each case study based on the transfer learning method is obtained by considering 10,000 samples, which are split among model-based and data-driven steps as described in the figures: If x empirical samples are used to refine the second ANN, then $10,000 - x$ samples from the analytical model as used to train the first ANN.

Figures 20 and 21, in particular, show that the proposed approach based on transfer learning is capable of providing performance that is very close to the best performance bound that is obtained by using the complete dataset of 10,000 samples. In Fig. 20, in particular, we observe that by using a data-driven approach and by increasing the amount of empirical data, the error performance decreases, as we would expect from a correct implementation of deep learning. We observe, in addition, that using only a model yields estimates of the optimal density of the base stations, or equivalently the average cell radius, that is significantly different from the actual optimal value that is obtained by considering the actual spatial distribution of the cellular network. By using a transfer learning based approach that uses only 40 empirical samples, we obtain almost perfect estimates of the actual deployment density of the base stations. This is confirmed in Fig. 21, where the optimal density of base stations as a function of their transmit power is depicted, by considering model-based, data-driven, and the proposed joint approach. Notably, we observe that a data-driven approach based on only 40 empirical samples yield inaccurate estimates of the optimal deployment density of the base stations: This highlights the relevance of performing the model-based pre-training before employing actual measurements for system optimization.

In summary, based on the results reported in Figs. 20 and 21, we conclude that the proposed approach based on transfer learning constitutes a suitable approach to take the best of both model-based and data-driven methods. Based on our

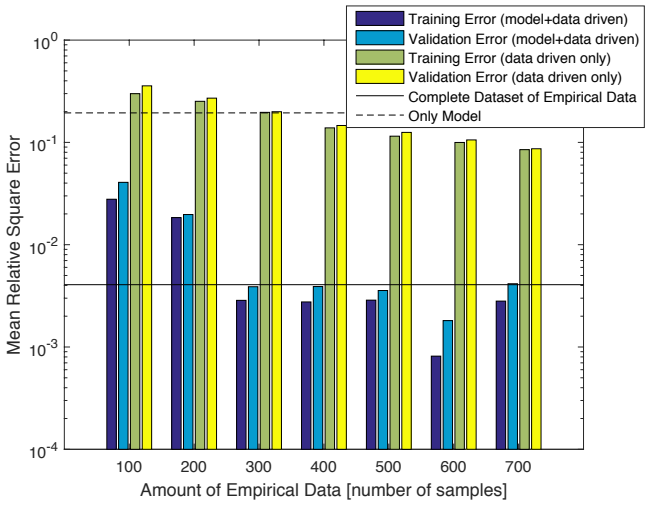

Figure 22. Comparison between model-based, data-driven, and transfer learning based optimization - Error performance of the actual and estimated average cell radius.

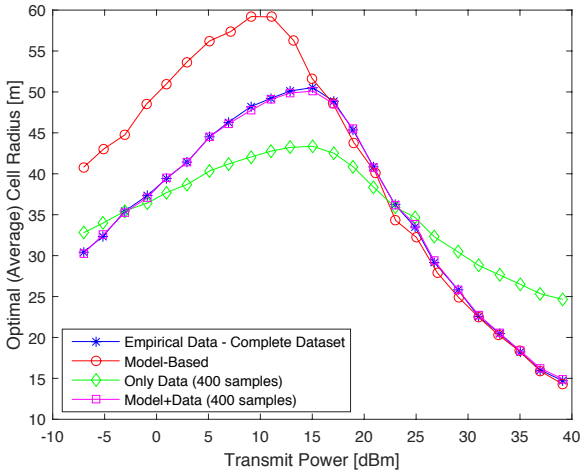

Figure 23. Comparison between model-based, data-driven, and transfer learning based optimization - Optimal deployment.

trials, however, we have observed that optimizing the proposed approach may not be trivial. In many cases, in fact, we have noticed the so-called *negative transfer*, i.e., the refined ANN yields performance that is worst than the pre-trained ANN. In spite of the promising results illustrated in Figs. 20 and 21, we think that the application of transfer learning to the design and optimization of wireless networks is a non-trivial research issue.

5) Refining a model by deep learning - Cellular networks with inaccurate power consumption models: In this section, we consider a similar optimization problem as in the previous section. Rather than focusing on the impact of the spatial distribution of the cellular base stations, we focus our attention on the power consumption model of the base stations. More precisely, we assume that the Poisson point process is sufficiently accurate to account for the distribution of the cellular base stations. As far as the power consumption model of the cellular base stations is concerned, on the other hand, we assume a model based on a uniform distribution for $\mathrm{P_{circ}}$ and $\mathrm{P_{idle}}$, while the empirical model is assumed to be based on the Gaussian distribution. The optimization problem that we are interested in is still concerned with identifying the optimal deployment density of the base stations, but as a function of three variables: $\mathrm{P_{tx}}$, $\mathrm{P_{circ}}$, and $\mathrm{P_{idle}}$. The model-based, the data-driven, and the transfer learning based approach follow the same approach as that described in the previous section. As far as the architecture of the ANN is concerned, on the other hand, we consider a different network setup that is made of six layers and four neurons. The ANN model is, therefore, more complicated because three input parameters instead of one are considered in this case study.

The results are illustrated in Figs. 20 and 21. The obtained performance trends are similar to those obtained in the previous case study. We note, however, that in this case a large number of empirical sample is needed in order to obtain adequate performance, especially compared against the benchmark constituted by data-driven optimization that leverages the complete dataset of 10,000 empirical samples. By using only 400 empirical samples, the proposed approach based on transfer learning is capable of obtaining good performance, and of providing sufficiently accurate performance predictions of the optimal network deployment. Based on these results, we conclude that the proposed network-based transfer learning approach is a promising alternative to bridge the critical tension between modeling accuracy, optimization complexity, and sensing overhead for network optimization.

V. Conclusions and Future Research Directions

The complexity of future wireless communication networks makes deep learning an indispensable design tool. Moreover, recent technological advancements in the area of computer processing units and distributed data storage make the use of deep learning now more practical than ever. Nevertheless, research in this field has just started, and a great deal of open problems must be solved before true ANN-based wireless communication networks can be operated.

The first challenge to be overcome is represented by the huge quantity of data that ANN need in order to ensure satisfactory performance. As remarked in Section II, deep learning outperforms other machine learning techniques in the large data regime. However, while this might not be an obstacle in other fields of science, the acquisition of large datasets in wireless networks requires measurement campaigns that could be too expensive and/or not practical. As shown in this work, the most promising approach to overcome this challenge is the joint use of data-driven approaches and model-based techniques. The transfer learning methodologies developed in Section IV demonstrate how even approximate mathematical models contain useful prior information that, if successfully embedded into deep learning techniques, can significantly reduce the amount of data required to achieve the desired performance. Nevertheless, this represents only

the tip of the iceberg, and many open issues remain to be investigated. As far deep transfer learning is concerned, it is not clear how to perform hyperparameter setting (e.g. amount of model-based data, number of ANN layers and neurons, etc.) to prevent a negative transfer. Moreover, other transfer learning techniques remain to be explored, as well as other ways of embedding expert knowledge into ANNs, based for example on the deep unfolding and deep reinforcement learning methods. As an example, embedding some prior information into a deep reinforcement learning algorithm could potentially speed up its convergence. In addition, a research direction that could provide guidance to achieve a cross-fertilization between mathematical models and deep learning is that aimed at deriving a theoretical explanation of how ANNs work and how to configure them to perform a certain task. Opening the *black box* of ANNs to understand the information-theoretic principles that regulate their behavior is surely a major topic for future investigation. A recent contribution in this direction is [204], which employs the so-called information bottleneck approach.

The second challenge to be overcome is the integration of ANN into future wireless network architectures. As motivated in this work, deep learning should be implemented in a distributed fashion. However, this poses several issues that need to be overcome in the next years. Integrating AI technologies into distributed wireless networks will not only affect the transmission technologies, but it will also significantly impact the way the network should be controlled through feedback signals to avoid instability and malfunctioning. A distributed network in which each node has its own ANN, that is trained based on a dataset acquired from local measurement and experience, inevitably leads to different nodes having different learning capabilities. Each distributed dataset might differ in both size, since different nodes might have different measurement and storage capabilities, as well as quality, since different nodes might experience different data perturbations due to the non-ideality of the measurement sensors. This could potentially lead to instabilities and, in the worst case, cause the wireless network to crash. Moreover, another issue to be addressed in distributed setups is the possibility for each node to optimize its own performance, rather than the system-wide utility, which might cause a device to learn how to cheat for individual gain. Thus, security mechanisms must be put in place to ensure the correct evolution of a distributed, ANN-based wireless communication network.

A third challenge to be overcome is to make deep learning techniques robust against corrupted data. Indeed, due to inevitable errors over feedback channels or in the storage process of data into memory banks, the datasets used to train ANNs might be corrupted and possibly lead to undesirable training results. Techniques that are able to make the training process robust to these events are warranted, especially in light of the distributed implementation of ANN-based wireless networks, which makes the overall network highly prone to inconsistencies and failures.

REFERENCES

[1] Cisco, "2020 cisco highlights," *http://www.telecompetitor.com/3-4-device-connections-per-person-worldwide-2020-cisco-highlights-11th-visual-networking-index/*, 2017.

[2] "NGMN alliance 5G white paper," *https://www.ngmn.org/5g-white-paper/5g-white-paper.html*, 2015.

[3] J. Andrews, S. Buzzi, W. Choi, S. 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.

[4] S. Buzzi, C.-L. I, T. E. Klein, H. V. Poor, C. Yang, and A. Zappone, "A survey of energy-efficient techniques for 5G networks and challenges ahead," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 5, 2016.

[5] A. Zappone and E. Jorswieck, "Energy efficiency in wireless networks via fractional programming theory," *Foundations and Trends® in Communications and Information Theory*, vol. 11, no. 3-4, pp. 185–396, 2015.

[6] C. G. Aliu *et al.*, "A survey of self organisation in future cellular networks," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 1, pp. 336–361, 2013.

[7] 5G-PPP, "5G empowering vertical industries," *Euro-5G Project Brochure*, February 2016.

[8] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[9] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, "Network function virtualization in 5G," *IEEE Communications Magazine*, vol. 54, no. 4, pp. 84–91, 2016.

[10] M. Alzenad, A. El-Keyi, F. Lagum, and H. Yanikomeroglu, "3-D placement of an unmanned aerial vehicle base station for energy-efficient maximal coverage," *IEEE Wireless Communications Letters*, vol. 6, no. 4, pp. 434—437, August 2017.

[11] Telus and Huawei, "Next generation SON for 5G," *White Paper*, 2016.

[12] H. W. Paper, "5G security: Forward thinking," 2015.

[13] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[14] O. Simeone, "A brief introduction to machine learning for engineers," *Foundations and Trends® in Communications and Information Theory*, pp. 1—191, 2017.

[15] ——, "A very brief introduction to machine learning with applications to communication systems," *https://arxiv.org/pdf/1808.02342.pdf*, 2018.

[16] S. Shalev-Shwartz, "Online learning and online convex optimization," *Foundations and Trends® in Communications and Information Theory*, vol. 4, no. 2, pp. 107–194, 2012.

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

[18] S. Lasaulce and H. Tembine, *Game Theory and Learning for Wireless Networks*. Elsevier, 2011.

[19] J. Moysen and L. Giupponi, "From 4G to 5G: Self-organized network management meets machine learning," *https://arxiv.org/pdf/1707.09300.pdf*, 2018.

[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[21] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[22] L. Deng and D. Yu, "Deep learning methods and applications," *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[23] A. Rao, J. Voyles, and P. Ramchandani, "Top 10 artificial intelligence technology trends for 2018," *http://usblogs.pwc.com/emerging-technology/top-10-ai-tech-trends-for-2018/*, 2017.

[24] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, "Machine learning for wireless networks with artificial intelligence: A tutorial on neural networks," *https://arxiv.org/pdf/1710.02913.pdf*, 2017.

[25] A. Imran, A. Zoha, and A. Abu-Dayya, "Challenges in 5G: how to empower SON with big data for enabling 5G," *IEEE Network*, vol. 28, no. 6, pp. 27–33, 2014.

[26] 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, October 2015.

[27] X. Cheng, L. Fang, L. Yang, and S. Cui, "Mobile big data: The fuel for data-driven wireless," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1489–1516, October 2017.

[28] P. Harris, "Analysis: What blockchain technology means for artificial intelligence," *http://www.nasdaq.com/article/analysis-what-blockchain-technology-means-for-artificial-intelligence-cm888540*, December 2017.

[29] F. Corea, "The convergence of AI and blockchain: what's the deal?" *https://medium.com/@Francesco_AI/the-convergence-of-ai-and-blockchain-whats-the-deal-60c618e3accc*, December 2017.

[30] T. McConaghy, "How blockchains could transform artificial intelligence," *http://dataconomy.com/2016/12/blockchains-for-artificial-intelligence/*, December 2016.

[31] 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.

[32] S. Kovach, "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.

[33] P. H. Pathak, X. Feng, P. Hu, and P. Mohapatra, "Visible light communication, networking, and sensing: A survey, potential and challenges," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 4, pp. 2047–2077, 2015.

[34] D. Karunatilaka, F. Zafar, V. Kalavally, and R. Parthiban, "LED based indoor visible light communications: State of the art," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 3, pp. 1649–1678, 2015.

[35] T. Nakano, M. J. Moore, F. Wei, A. V. Vasilakos, and J. Shuai, "Molecular communication and networking: Opportunities and challenges," *IEEE Transactions on NanoBioscience*, vol. 11, no. 2, pp. 135–148, 2012.

[36] N. Farsad, H. B. Yilmaz, A. Eckford, C.-B. Chae, and W. Guo, "A comprehensive survey of recent advancements in molecular communication," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 3, pp. 1887–1919, 2016.

[37] C. Häger and H. D. Pfister, "Deep learning of the nonlinear schrödinger equation in fiber-optic communications," *https://export.arxiv.org/pdf/1804.02799*, 2018.

[38] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436—444, 2015.

[39] J. Schmidhuber, "Deep learning in neural networks: An overview," *https://arxiv.org/abs/1404.7828*, 2014.

[40] 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.

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

[42] P. V. Klaine, M. A. Imran, O. Onireti, and R. D. Souza, "A survey of machine learning techniques applied to self organizing cellular networks," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2392–2431, 2017.

[43] P. Kasnesis, C. Patrikakis, and I. Venieris, "Changing the game of mobile data analysis with deep learning," *IT Professional*, vol. PP, no. 99, 2017.

[44] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *https://arxiv.org/abs/1803.04311*, 2018.

[45] "https://www.comsoc.org/ctn/what-will-6g-be."

[46] P. Hu, P. Zhang, M. Rostami, and D. Ganesan, "An integrated active-passive radio for mobile devices with asymmetric energy budgets," in *ACM SIGCOMM*, 2016.

[47] C. Liaskos, S. Nie, A. Tsioliaridou, A. Pitsillides, S. Ioannidis, and I. F. Akyildiz, "Realizing wireless communication through software-defined hypersurface environments," in *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, 2018.

[48] "5GPPP vision on software networks and 5g sn wg, jan. 2017."

[49] C. Liaskos, S. Nie, A. Tsioliaridou, A. Pitsillides, S. Ioannidis, and I. F. Akyildiz, "A new wireless communication paradigm through software-controlled metasurfacesâ, ieee communications magazine, vol. 56, no. 9, pp. 162-169, sep. 2018." *IEEE Communications Magazine*, vol. 56, no. 9, pp. 162–169, September 2018.

[50] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[51] N. Wiener, *Cybernetics, or Control and Communication in the Animal and the Machine*. MIT Press, 1948.

[52] L. Subrt and P. Pechac, "Controlling propagation environments using intelligent walls," in *European Conference on Antennas and Propagation*, 2012.

[53] C. Liaskos, A. Tsioliaridou, A. Pitsillides, S. Ioannidis, and I. F. Akyildiz, "Using any surface to realize a new paradigm for wireless communications," *Communications of the ACM*, vol. 61, no. 11, pp. 30–33, November 2018.

[54] A. Tsioliaridou, C. Liaskos, and S. Ioannidis, "Towards a circular economy via intelligent metamaterials," in *IEEE International Conference on Computer-Aided Modeling Analysis and Design of Communication Links and Networks*, 2018.

[55] N. Yu, P. Genevet, M. A. Kats, F. Aieta, J.-P. Tetienne, F. Capasso, and Z. Gaburro, "Light propagation with phase discontinuities: Generalized laws of reflection and refraction," *Science*, vol. 334, no. 6504, pp. 333–337, 2011.

[56] C. L. Holloway, E. F. Kuester, J. A. Gordon, J. O'Hara, J. Booth, and D. R. Smith, "An overview of the theory and applications of metasurfaces: The two-dimensional equivalents of metamaterials," *IEEE Antennas and Propagation Magazine*, vol. 54, no. 2, pp. 10–35, April 2012.

[57] L. Spada, "Metamaterials for advanced sensing platforms," *Research Journal on Optical Photonics*, vol. 1, no. 1, October 2017.

[58] T. Nakanishi, T. Otani, Y. Tamayama, and M. Kitano, "Storage of electromagnetic waves in a metamaterial that mimics electromagnetically induced transparency," *Physical Review B*, vol. 87, no. 161110, 2013.

[59] A. Silva, F. Monticone, G. Castaldi, V. Galdi, A. Alu, and N. Engheta, "Performing mathematical operations with metamaterials," *Science*, vol. 343, no. 6167, pp. 160–163, 2014.

[60] "A hardware platform for software-driven functional metasurfaces," *H2020 VISORSURF project*.

[61] H. Claussen, L. T. W. Ho, H. R. Karimi, F. J. Mullany, and L. G. Samuel, "Base station: Cognisant robots and future wireless access networks," in *IEEE Consumer Communications and Networking Conference*, 2006.

[62] H. Claussen, "Autonomous self-deployment of wireless access networks," *Bell Labs Technical Journal*, vol. 14, no. 1, pp. 55–71, 2009.

[63] S. Singh, H. S. Dhillon, and J. G. Andrews, "Offloading in heterogeneous networks: Modeling, analysis, offloading in heterogeneous networks: Modeling, analysis, and design insights," *IEEE Transactions on Wireless Communications*, vol. 12, no. 5, pp. 2484–2497, May 2013.

[64] J. G. Andrews, X. Zhang, G. D. Durgin, and A. K. Gupta, "Are we approaching the fundamental limits of wireless network densification?" *IEEE Communications Magazine*, vol. 54, no. 10, pp. 184–190, October 2016.

[65] M. D. Renzo, W. Lu, and P. Guan, "The intensity matching approach: A tractable stochastic geometry approximation to system-level analysis of cellular networks," *IEEE Transactions on Wireless Communications*, vol. 15, no. 9, pp. 5963–5983, September 2016.

[66] M. D. Renzo, A. Zappone, T. T. Lam, and M. Debbah, "System-level modeling and optimization of the energy efficiency in cellular networks—a stochastic geometry framework," *IEEE Transactions on Wireless Communications*, 2018.

[67] ——, "Spectral-energy efficiency pareto front in cellular networks: A stochastic geometry frameworks," *IEEE Wireless Communications Letters*, 2018.

[68] C. Mollen, "High-end performance with low-end hardware: Analysis of massive mimo base station transceivers," *Doctoral Thesis, Linkoping University, Sweden*, 2017.

[69] R. W. Heath, N. Gonzalez-Prelcic, S. Rangan, W. Roh, and A. Sayeed, "An overview of signal processing techniques for millimeter wave MIMO systems," *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 3, April 2016.

[70] J. G. Andrews, T. Bai, M. N. Kulkarni, A. Alkhateeb, A. K. Gupta, and R. W. Heath, "Modeling and analyzing millimeter wave cellular systems," *IEEE Transactions on Communications*, vol. 65, no. 1, pp. 403–430, 2017.

[71] O. Abari, D. Bharadia, A. Duffield, and D. Katabi, "Enabling high-quality untethered virtual reality," in *USENIX Symposium on Networked Systems Design and Implementation*, 2017.

[72] W. Lu and M. D. Renzo, "Stochastic geometry modeling and system-level analysis and optimization of relay-aided downlink cellular networks," *IEEE Transactions on Communications*, vol. 63, no. 11, pp. 4063–4085, November 2015.

[73] A. Shojaeifard, K.-K. Wong, M. D. Renzo, G. Zheng, K. A. Hamdi, and J. Tang, "Massive MIMO-enabled full-duplex cellular networks," *IEEE Transactions on Communications*, vol. 65, no. 11, pp. 4734–4750, 2 2017.

[74] S. Abadal *et al.*, "Computing and communications for the software-defined metamaterial paradigm: A context analysis," *IEEE Access*, vol. 5, pp. 6225–6235, 2017.

[75] F. Liu *et al.*, "Programmable metasurfaces: State of the art and prospects," in *IEEE International Symposium on Circuits and Systems*, 2018.

[76] A. Welkie, L. Shangguan, J. Gummeson, W. Hu, and K. Jamieson, "Programmable radio environments for smart spaces," in *ACM Workshop on Hot Topics in Networks*, 2017.

[77] H. Claussen, "Autonomous self-deployment of wireless access networks in an airport environment," *Autonomic Communication, Lecture Notes in Computer Science, Springer*, vol. 3854, pp. 86–98, 2006.

[78] R. Chandra and K. Winstein, "Programmable radio environments for smart spaces," in *HotNets-XVI Dialogue, ACM Workshop on Hot Topics in Networks*, 2017.

[79] J. G. Andrews, F. Baccelli, and R. K. Ganti, "A tractable approach to coverage and rate in cellular networks," *IEEE Transactions on Communications*, vol. 59, no. 11, pp. 3122–3134, November 2011.

[80] M. D. Renzo, A. Guidotti, and G. E. Corazza, "Average rate of downlink heterogeneous cellular networks over generalized fading channels: A stochastic geometry approach," *IEEE Transactions on Communications*, vol. 61, no. 7, pp. 3050–3071, July 2013.

[81] M. D. Renzo, S. Wang, and X. Xi, "Modeling and analysis of cellular networks by using inhomogeneous poisson point processes," *IEEE Transactions on Wireless Communications*, vol. 17, no. 8, pp. 5162–5182, August 2018.

[82] M. D. Renzo, T. T. Lam, A. Zappone, and M. Debbah, "A tractable closed-form expression of the coverage probability in poisson cellular networks," *IEEE Wireless Communications Letters*, 2018.

[83] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, October 2010.

[84] D. Fudenberg and J. Tirole, *Game Theory*. MIT Press, 1993.

[85] T. Basar and G. J. Olsder, *Dynamic Noncooperative Game Theory*, 2nd ed., ser. Classics In Applied Mathematics. SIAM, 1999.

[86] R. B. Myerson, *Game theory: analysis of conflict*. Harvard University Press, 1997.

[87] Z. Han, D. Niyato, W. Saad, T. Basar, and A. Hjorungnes, *Game Theory in Wireless and Communication Networks: Theory, Models, and Applications*. Cambridge University Press, 2011.

[88] A. MacKenzie and L. DaSilva, "Game theory for wireless engineers," *Synthesis Lectures on Communications*, vol. 1, no. 1, pp. 1–86, 2006.

[89] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *20 th International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 54, 2017.

[90] J. Konečný, H. B. McMahan, F. X. Yu, A. T. Suresh, D. Bacon, and P. Richtárik, "Federated learning: Strategies for improving communication efficiency," *https://arxiv.org/abs/1610.05492*, 2017.

[91] F. Chen, Z. Dong, Z. Li, and X. He, "Federated meta-learning for recommendation," *https://arxiv.org/abs/1802.07876*, 2018.

[92] X. Wei, Q. Wang, T. Wang, and J. Fan, "Jammer localization in multi-hop wireless network: A comprehensive survey," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 2, pp. 765–799, 2017.

[93] G. Han, J. Jiang, C. Zhang, T. Q. Duong, M. Guizani, and G. K. Karagiannidis, "A survey on mobile anchor node assisted localization in wireless sensor networks," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 3, pp. 2220–2243, 2016.

[94] Y. Zhang, N. Meratnia, and P. Havinga, "Outlier detection techniques for wireless sensor networks: A survey," *IEEE Communications Surveys and Tutorials*, vol. 12, no. 2, pp. 1–12, 2010.

[95] J. Granjal, E. Monteiro, and J. Sá Silva, "Security for the internet of things: A survey of existing protocols and open research issues," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 3, pp. 1294–1312, 2015.

[96] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[97] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction, 2nd Edition Draft*. MIT Press, 2017.

[98] Y. Li, "Deep reinforcement learning: An overview," *https://arxiv.org/abs/1701.07274*, 2017.

[99] K. Arulkumaran, M. P. Deisenroth, and M. B. A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, November 2017.

[100] V. N. Vapnik and A. Y. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," *Theory of Probability and Its Applications*, vol. 16, pp. 264–280, 1971.

[101] A. Blumer, A. Ehrenfeucht, and D. H. M. K. Warmuth, "Learnability and the Vapnik-Chervonenkis dimension," *Journal of the ACM*, vol. 36, no. 4, pp. 865–929, 1989.

[102] V. N. Vapnik, *Estimation of Dependences Based on Empirical Data*. Springer-Verlag, 1982.

[103] ——, *The nature of Statistical Learning Theory*. Springer, 1995.

[104] J. Bergstra and Y. Bengio, "Random search for hyperparameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[105] Y. Bengio, H. Larochelle, and P. Vincent, "Non-local manifold parzen windows," *NIPS, MIT Press*, 2005.

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

[107] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012.

[108] K. Jarret, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recogntion?" in *International Conference on Computer Vision*, 2009.

[109] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *International Conference on Machine Learning*, 2010.

[110] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *International Conference on International Conference of Artificial Intelligence and Statistics*, 2011.

[111] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier non linearities improve neural network acoustic models," in *International Conference on Machine Learning*, 2013.

[112] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: surpassing human-level performance on ImageNet classification," *https://arxiv.org/abs/1502.01852*, 2015.

[113] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," *https://arxiv.org/abs/1511.07289*, 2015.

[114] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[115] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks 6*, vol. 6, pp. 861–867, 1993.

[116] A. E. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 930–945, 1993.

[117] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," in *Neural Information Processing Systems*, 2014.

[118] T. Cover and J. Thomas, *Elements of information theory*. Wiley, 2006.

[119] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2004.

[120] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, 1999.

[121] A. Ben-Tal and A. Nemirovski, *Lectures on Modern Convex Optimization: Analysis, Algorithms, Engineering Applications*. MPS-SIAM Series on Optimization, 2001.

[122] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," in *International Conference on Learning Representation*, 2013.

[123] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," in *Neural Information Processing Systems*, 2014.

[124] I. J. Goodfellow, O. Vinyals, and A. M. Saxe, "Qualitatively characterizing neural network optimization problems," in *International Conference on Learning Representations*, 2015.

[125] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The loss surface of multilayer networks," in *Artificial Intelligence and Statistics*, 2015.

[126] P. Baldi and K. Hornik, "Neural networks and principal component analysis: Learning from examples without local minima," *Neural Networks*, vol. 2, pp. 53–58, 1989.

[127] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Journal of Applied Mathematics, Second Quarter*, no. 2, pp. 164–168, 1944.

[128] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the Society of Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

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

[130] L. Bottou, *Online algorithms and stochastic approximations*, D. Saad, Ed. Cambridge University Press, Cambridge, UK, 1998.

[131] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964.

[132] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International Conference on Machine Learning 2013*, 2013.

[133] Y. Nesterov, *Introductory lectures on convex optimization : a basic course*, ser. Applied optimization. Boston, Dordrecht, London: Kluwer Academic Publisher, 2004.

[134] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, 2011.

[135] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representation*, 2015.

[136] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS'2010*, 2010.

[137] D. Sussillo and L. F. Abbott, "Random walks: Training very deep nonlinear feed-forward networks with smart initialization," *https://arxiv.org/abs/1412.6558v3*, 2015.

[138] J. Martens, "Deep learning via hessian-free optimization," in *Twenty-seventh International Conference on Machine Learning*, 2010.

[139] C. M. Bishop, "Regularization and complexity control in feed-forward networks," in *International Conference on Artificial Neural Networks*, 1995.

[140] J. Sjöberg and L. Ljung, "Overtraining, regularization and searching for a minimum, with application to neural networks," *International Journal of Control*, vol. 62, no. 6, pp. 1391–1407, 1995.

[141] N. Srivastava, G. Hinton, A. K. I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[142] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *32nd International Conference on Machine Learning*, vol. 37, 2015.

[143] J. R. Hershey, J. Le Ru, and F. Weninger, "Deep unfolding: Model-based inspiration of novel deep architectures," *https://arxiv.org/pdf/1409.2574.pdf*, 2014.

[144] H. He, S. Jin, C.-K. Wen, F. Gao, G. Y. Li, and Z. Xu, "Model-driven deep learning for physical layer communications," *https://arxiv.org/pdf/1809.06059.pdf*, 2018.

[145] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," *https://arxiv.org/abs/1808.01974*, 2018.

[146] Y. Yao and G. Doretto, "Boosting for transfer learning with multiple sources," in *2010 IEEE conference on Computer vision and pattern recognition (CVPR)*, 2010.

[147] D. Pardoe and P. Stone, "Boosting for regression transfer," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, 2010.

[148] E. Tzeng *et al.*, "Deep domain confusion: Maximizing for domain invariance," *https://arxiv.org/pdf/1412.3474.pdf*, 2014.

[149] M. Long, Y. Cao, J. Wang, and M. Jordan, "Learning transferable features with deep adaptation networks," *International Conference on Machine Learning*, pp. 97–105, 2015.

[150] M. Long, H. Zhu, J. Wang, and M. I. Jordan, "Deep transfer learning with joint adaptation networks," *https://arxiv.org/abs/1605.06636*, 2017.

[151] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *https://arxiv.org/abs/1701.07875*, 2017.

[152] J. T. Huang, J. Li, D. Yu, L. Deng, and Y. Gong, "Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing Conference (ICASSP)*, 2013.

[153] A. Zappone, M. Di Renzo, M. Debbah, T. T. Lam, and X. Qian, "Model-aided wireless artificial intelligence: Embedding expert knowledge in deep neural networks towards wireless systems optimization," *https://arxiv.org/abs/1808.01672*, 2018.

[154] Y. Shen, Y. Shi, J. Zhang, and K. B. Letaief, "Transfer learning for mixed-integer resource allocation problems in wireless networks," *https://arxiv.org/abs/1811.07107*, 2018.

[155] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, pp. 2672—2680, 2014.

[156] Y. Ganin *et al.*, "Domain-adversarial training of neural networks," *Journal of Machine Learning Research*, vol. 17, pp. 1–35, 2016.

[157] Z. Cao, M. Long, J. Wang, and M. I. Jordan, "Partial transfer learning with selective adversarial networks," *https://arxiv.org/pdf/1707.07901.pdf*, 2017.

[158] T. O'Shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, December 2017.

[159] M. Kim, W. Lee, and D.-H. Cho, "A novel PAPR reduction scheme for OFDM system based on deep learning," *IEEE Communications Letters*, vol. 22, no. 3, pp. 510–513, 2018.

[160] N. Samuel, T. Diskin, and A. Wiesel, "Learning to detect," *https://arxiv.org/pdf/1805.07631.pdf*, 2018.

[161] S. Xue, Y. Ma, N. Yi, and R. Tafazolli, "Unsupervised deep learning for MU-SIMO joint transmitter and noncoherent receiver design," *IEEE Wireless Communications Letters*, 2018.

[162] X. Jin and H.-N. Kim, "Deep learning detection networks in MIMO decode-forward relay channels," *https://arxiv.org/abs/1807.09571*, 2018.

[163] A. Felix, S. Cammerer, S. Dörner, J. Hoydis, and S. t. Brink, "OFDM-autoencoder for end-to-end learning of communications systems," *https://arxiv.org/pdf/1803.05815.pdf*, 2018.

[164] S. Dörner, S. Cammerer, J. Hoydis, and S. ten Brink, "Deep learning-based communication over the air," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 132–143, 2018.

[165] T. J. O'Shea, T. Roy, N. West, and B. C. Hilburn, "Physical layer communications system design over-the-air using adversarial networks," *https://arxiv.org/abs/1803.03145v1*, 2018.

[166] T. J. O'Shea, T. Roy, and N. West, "Approximating the void: Learning stochastic channel models from observation with variational generative adversarial networks," *https://arxiv.org/abs/1805.06350*, 2018.

[167] H. Ye, G. Y. Li, B. H. F. Juang, and K. Sivanesan, "Channel agnostic end-to-end learning based communication systems with conditional gan," *https://arxiv.org/abs/1807.00447*, 2018.

[168] F. A. Aoudia and J. Hoydis, "End-to-end learning of communications systems without a channel model," *https://arxiv.org/pdf/1804.02276.pdf*, 2018.

[169] V. Raj and S. Kalyani, "Backpropagating through the air: Deep learning at physical layer without channel models," *IEEE Communications Letters*, 2018.

[170] N. Farsad and A. Goldsmith, "Detection algorithms for communication systems using deep learning," *https://arxiv.org/abs/1705.08044*, 2017.

[171] X. Qian and M. Di Renzo, "Receiver design in molecular communications: An approach based on artificial neural networks," in *IEEE International Symposium on Wireless Communication Systems (ISWCS)*, 2018.

[172] D. Neumann, T. Wiese, and W. Utschick, "Learning the MMSE channel estimator," *https://arxiv.org/abs/1707.05674v2*, 2017.

[173] J. Vieira, E. Leitinger, M. Sarajlic, X. Li, and F. Tufvesson, "Deep convolutional neural networks for massive MIMO fingerprint-based positioning," *https://arxiv.org/pdf/1708.06235.pdf*, 2017.

[174] S. Navabi, C. Wang, O. Y. Bursalioglu, and H. Papadopoulos, "Predicting wireless channel features using neural networks," *https://arxiv.org/abs/1802.00107*, 2018.

[175] Y. Ding and B. D. Rao, "Dictionary learning based sparse channel representation and estimation for FDD massive MIMO systems," *https://arxiv.org/abs/1612.06553*, 2018.

[176] A. Decurninge *et al.*, "CSI-based outdoor localization for massive MIMO: Experiments with a learning approach," in *IEEE International Symposium on Wireless Communication Systems (ISWCS)*, 2018.

[177] S. Schibisch *et al.*, "Online label recovery for deep learning-based communication through error correcting codes," in *IEEE International Symposium on Wireless Communication Systems (ISWCS)*, 2018.

[178] M. Koller *et al.*, "Machine learning for channel estimation from compressed measurements," in *IEEE International Symposium on Wireless Communication Systems (ISWCS)*, 2018.

[179] X. Ma, H. Ye, and G. Y. Li, "Learning assisted estimation for time-varying channels," in *IEEE International Symposium on Wireless Communication Systems (ISWCS)*, 2018.

[180] W. Xu *et al.*, "Joint neural network equalizer and decoder," in *IEEE International Symposium on Wireless Communication Systems (ISWCS)*, 2018.

[181] T. Wang, C.-K. Wen, H. Wang, F. Gao, T. Jiang, and S. Jin, "Deep learning for wireless physical layer: Opportunities and challenges," *https://arxiv.org/pdf/1710.05312.pdf*, 2017.

[182] Z. Qin, H. Ye, G. Y. Li, and B.-H. F. Juang, "Deep learning in physical layer communications," *https://arxiv.org/abs/1807.11713*, 2018.

[183] A. Javid, S. Chatterjee, and M. Skoglund, "Mutual information preserving analysis of a single layer feedforward network," in *IEEE International Symposium on Wireless Communication Systems (ISWCS)*, 2018.

[184] H. Ye, G. Y. Li, and B.-H. Juang, "Power of deep learning for channel estimation and signal detection in OFDM systems," *IEEE Wireless Communications Letters*, vol. 7, no. 1, pp. 114–117, 2018.

[185] R. Li, Z. Zhao, X. Zhou, G. Ding, Y. Chen, Z. Wang, and H. Zhang, "Intelligent 5G: When cellular networks meet artificial intelligence," *IEEE Wireless Communications*, vol. 24, no. 5, pp. 175–183, October 2017.

[186] F. D. Calabrese, L. Wang, E. Ghadimi, G. Peters, and P. Soldati, "Learning radio resource management in 5G networks: Framework, opportunities and challenges," *https://arxiv.org/abs/1611.10253*, 2017.

[187] J. Fang, X. Li, W. Cheng, Z. Chen, and H. Li, "Intelligent power control for spectrum sharing: A deep reinforcement learning approach," *https://arxiv.org/pdf/1712.07365.pdf*, 2018.

[188] 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.

[189] 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," *https://arxiv.org/pdf/1705.09412.pdf*, 2017.

[190] A. Zappone, M. Debbah, and Z. Alltman, "Online energy-efficient power control in wireless networks by deep neural networks," in *IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications*, 2018.

[191] A. Zappone, L. Sanguinetti, and M. Debbah, "User association and load balacing for massive MIMO through deep learning," in *Asilomar Conference on Signals, Systems, and Computers*, 2018.

[192] L. Sanguinetti, A. Zappone, and M. Debbah, "A deep-learning framework for energy-efficient resource allocation in massive MIMO systems," in *Asilomar Conference on Signals, Systems, and Computers*, 2018.

[193] Y. S. Nasir and D. Guo, "Deep reinforcement learning for distributed dynamic power allocation in wireless networks," *https://arxiv.org/abs/1808.00490*, 2018.

[194] F. Liang, C. Shen, W. Yu, and F. Wu, "Towards optimal power control via ensembling deep neural networks," *https://arxiv.org/abs/1807.10025*, 2018.

[195] P. De Kerret and D. Gesbert, "Robust decentralized joint precoding using team deep neural network," in *IEEE International Symposium on Wireless Communication Systems (ISWCS)*, 2018.

[196] Q. Shi, M. Razaviyayn, Z. Q. Luo, and C. He, "An Iteratively Weighted MMSE Approach to Distributed Sum-Utility Maximization for a MIMO Interfering Broadcast Channel," *IEEE Transactions on Signal Processing*, vol. 59, no. 9, pp. 4331–4340, September 2011.

[197] A. Zappone, E. Björnson, L. Sanguinetti, and E. Jorswieck, "Globally optimal energy-efficient power control and receiver design in wireless networks," *IEEE Transactions on Signal Processing*, vol. 65, no. 11, pp. 2844–2859, June 2017.

[198] T. Inoue, S. Chaudhury, G. D. Magistris, and S. Dasgupta, "Transfer learning from synthetic to real images using variational autoencoders for robotic applications," *https://arxiv.org/pdf/1709.06762.pdf*, 2017.

[199] C. Kim, E. Variani, A. Narayanan, and M. Bacchiani, "Efficient implementation of the room simulator for training deep neural network acoustic models," *https://arxiv.org/pdf/1712.03439.pdf*, 2017.

[200] N. Farsad, H. B. Yilmaz, A. Eckford, C.-B. Chae, and W. Guo, "A comprehensive survey of recent advancements in molecular communication," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 3, pp. 1887–1919, 2016.

[201] N. Farsad and A. Goldsmith, "Sliding bidirectional recurrent neural networks for sequence detection in communication systems," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2018.

[202] X. Qian and M. D. Renzo, "Receiver design in molecular communications: An approach based on artificial neural networks," in *IEEE International Symposium on Wireless Communications Systems*, 2018.

[203] G. Calcev, D. Chizhik, B. Goransson, S. Howard, H. Huanga, A. Kogiantis, A. Molisch, A. Moustakas, D. Reed, and H. Xu, "A wideband spatial channel model for system-wide simulations," *IEEE Transactions on Vehicular Technology*, vol. 56, no. 2, March 2007.

[204] N. Wolchover, "New theory cracks open the black box of deep learning," *Quanta Magazine*, September 2017.