1 Machine Learning and Communications: An Introduction

Deniz Gündüz, Yonina C. Eldar, Andrea Goldsmith, and H. Vincent Poor

1.1 Introduction

Wireless communications is one of the most prominent and impactful technologies of the last few decades. Its transformative impact on our lives and society is immeasurable. The fifth generation (5G) of mobile technology has recently been standardised and is now being rolled out. In addition to its reduced latency and higher data rates compared to previous generations, one of the promises of 5G is to connect billions of heterogeneous devices to the network, supporting new applications and verticals under the banner of the Internet of Things (IoT). The number of connected IoT devices is expected to increase exponentially in the coming years, enabling new applications including mobile healthcare, virtual and augmented reality, and self-driving cars as well as smart buildings, factories, and infrastructures.

In parallel with recent advances in wireless communications, modern machine learning (ML) techniques have led to significant breakthroughs in all areas of science and technology. New ML techniques continue to emerge, paving the way for new research directions and applications, ranging from autonomous driving and finance to marketing and healthcare, just to name a few. It is only natural to expect that a tool as powerful as ML should have a transformative impact on wireless communication systems, similarly to other technology areas. On the other hand, wireless communication system design has traditionally followed a model-based approach, with great success. Indeed, in communication systems, we typically have a good understanding of the channel and network models, which follow fundamental physical laws, and some of the current systems and solutions already approach fundamental information theoretical limits. Moreover, communication devices typically follow a highly standardized set of rules; that is, standardization dictates what type of signals to transmit, as well as when and how to transmit them, with tight coordination across devices and even different networks. As a result, existing solutions are products of research and engineering design efforts that have been optimized over many decades and generations of standards, which are based upon theoretical foundations and extensive experiments and measurements. From this perspective, compared to other areas in which ML has made significant advances in recent years, communication systems can be considered more amenable to model-based solutions rather than generic ML approaches. Therefore, one can question what potential benefits ML can provide to wireless communications [1–3].

The chapters in the first part of this book provide many answers to this question. First, despite the impressive advances in coding and communication techniques over the last few decades, we are still far from approaching the theoretical limits in complex networks of heterogeneous devices, or even understanding and characterizing such limits. Current design approaches are built upon engineering heuristics: we try to avoid or minimize interference through various access technologies, which allows us to reduce a complex interconnected network into many parallel point-to-point links. We then divide the design of the point-to-point communication systems into many different blocks, such as channel estimation, feedback, equalization, modulation, and coding, which are easier to model and may even lend themselves to optimal solutions. Although this modular design is highly attractive from an engineering perspective, it is generally suboptimal.

With 5G, we aim to connect billions of IoT devices, which have significantly different requirements in terms of latency and energy efficiency compared to conventional mobile handsets. The networks will have to sustain a much more diverse set of devices and traffic types, each with different requirements and constraints. This will require significantly more flexibility and more adaptive and efficient use of the available resources. Therefore, the performance loss due to the aforementioned modular design and separate optimization of the individual blocks is becoming increasingly limiting. Cross-layer design as an alternative has been studied extensively, but this type of design still depends on a model-based approach that is often significantly more complex than the modular design. This complexity has led to ad-hoc solutions for each separate cross-layer design problem and scenario, with performance gains that were often limited or nonexistent relative to more traditional modular design. Data-driven ML techniques can provide an alternative solution, with potentially larger gains in performance and reasonable complexity in implementation.

Second, in current systems, even with the modular approach, the optimal solution may be known theoretically, yet remain prohibitive to attain computationally. For example, we can write down the maximum likelihood detection rule in a multiuser scenario, but the complexity of its solution grows exponentially with the number of users [4]. Similarly, in a communication network with a multi-antenna transmitter and multiple single-antenna receivers, the optimal beamforming vectors can be written as the solution of a well-defined optimization problem, albeit with no known polynomial complexity solution algorithm [5]. As a result, we either provide low-complexity yet suboptimal solution techniques to the problem [6] or resort to simple solutions that can be solved in closed-form [7]. Recent results have shown that machine learning techniques can provide more attractive complexity-performance trade-offs [8, 9].

There has already been a significant amount of research proposing data-driven ML solutions for different components of a communication system, such as channel estimation [10–12], channel state information compression and feedback [13, 14], channel equalization [15], channel decoding [16], and more, many of which will be discussed in detail in the later chapters of this book. These solutions are shown to outperform conventional designs in many scenarios, particularly when we do not have a simple model of the communication system (e.g., when the channel coefficients do

not follow a distribution with a known covariance matrix). However, another potential benefit of the ML approaches is to go beyond the aforementioned modular design and learn the best communication scheme in an end-to-end fashion rather than targeting each module separately [17, 18]. This end-to-end ML approach facilitates attacking much more complex problems with solutions that were deemed elusive with conventional approaches. A good example is the joint source-channel coding problem, which is typically divided into the compression and channel coding subproblems. Despite the known suboptimality of this approach, and decades of efforts in designing joint source-channel coding schemes, these often resulted in ad-hoc solutions for different source and channel combinations with formidable complexity. However, recent results have shown that ML can provide significant improvements in the end-to-end performance of joint source-channel coding [19–22].

An additional benefit of ML is that is does not require knowledge of channel parameters or statistics. In particular, many known communication methods rely on knowledge of the channel parameters or statistics or the ability to estimate these statistics with minimal resources. In modern wireless networks, this is no longer the case. Thus, ML can offer efficient techniques to compensate for unknown and varying channel knowledge [23, 24]. In addition, hardware limitations, such as the use of low bit rate quantizers or nonlinear power amplifiers, can significantly increase the complexity of the underlying channel model. This renders a design based on exact channel knowledge difficult to implement [25].

Another dimension of the potential synergies between communications and ML is the design of communication networks to enable ML applications [26]. While a significant amount of information is collected and consumed by mobile devices, most of the learning is still carried out centrally at cloud servers. Such a centralized approach to learning at the edge has limitations. First, offloading all the data collected by edge devices to the cloud for processing is not scalable. For example, an autonomous car is expected to collect terrabytes of data every day. As the number of vehicles increases, the cellular network infrastructure cannot support such a huge rise in data traffic solely from autonomous cars. A similar surge in traffic is expected from other connected devices. The sheer volume of the collected data makes such centralized approaches highly unsustainable. This is particularly problematic when the collected data has large dimensions and has low *information density*, which refers to the information within the data relevant for the underlying task. For example, a surveillance camera may collect and offload hours of recording of a still background, which has little use for the task of detecting intruders. Therefore, it is essential to enable edge devices with some level of intelligence to extract and convey only the relevant information for the underlying learning task [27–29].

Having all the intelligence centralized also causes significant privacy and security concerns [30–32]. Most of the data collected from edge devices, such as smart meters [33, 34], autonomous cars [35], health sensors [36], or entertainment devices [37], collect highly sensitive information that reveals significant personal information about our lives and daily habits. Sharing this data in bulk with other parties is a growing concern for consumers, which can potentially hinder the adoption of some of

these services. The alternative can be to carry out local learning at each individual device with the available data; however, the locally available data can be limited in quantity and variety, resulting in overfitting.

Fully centralized intelligence also introduces latency. In many applications that involve edge devices, inference and action need to be fast. For example, autonomous vehicles must detect and avoid pedestrians or other obstacles rapidly. Even though some of the inference tasks can be carried out on board, devices may not have the necessary processing capability, and some of the inference tasks may require fusing information distributed across multiple edge devices. For example, an autonomous car may benefit from camera or LIDAR data from other nearby cars or terrain information available at a nearby base station to make more accurate decisions about its trajectory and speed. In such scenarios, it is essential to enable distributed inference algorithms that can rapidly gather the most relevant information and make the most accurate decisions [28].

In light of these observations, an important objective in merging communications and ML is to bring intelligence to the network edge, rather than offloading the data to the cloud and relying on centralized ML algorithms. The two core ingredients behind the recent success of ML techniques are massive datasets and tremendous memory and processing power to efficiently and rapidly process the available data to train huge models, such as deep neural networks (DNNs). Both of these ingredients are plentiful at the network edge, yet in a highly distributed manner. The second part of this book is dedicated to distributed ML algorithms, particularly focusing on learning across wireless networks, addressing these challenges and highlighting some of the important research achievements and remaining challenges.

Before revieweing the individual chapters in the book, we provide a brief overview of basic ML techniques, particularly focusing on their potential applications in communication problems.

1.2 Taxonomy of Machine Learning Problems

The type of problems to which machine learning algorithms are applied can be grouped into three categories: *supervised learning*, *unsupervised learning*, and *reinforcement learning (RL)*. All three types of problems have found applications in wireless communications. Here we provide a general overview of these different categories and what types of problems they are used to solve and then highlight their applications in wireless communications.

1.2.1 Supervised Learning

In supervised learning the goal is to teach an algorithm the input-output relation of a function. This can be applied to a wide variety of functions and input/output types. Consider, for example, the input vector denoted by $\mathbf{x} \in \mathcal{X}$ and the associated vector of target variables $\mathbf{c} \in \mathcal{C}$. Supervised learning problems are classified into two

groups: if the input is mapped to one of a finite number of discrete classes (i.e., $\mathcal{C}$ is a discrete set), then the learning task is called *classification*. If, instead, the output can take continuous values (i.e., $\mathcal{C}$ is a continuous set), then the learning task is called *regression*. The often cited toy problem of classifying images into dog and cat labels is a classical example of a classification problem. Most classification tasks have more than two classes, and other common examples that are often used to compare and benchmark supervised learning algorithms are classification of handwritten digits and spam detection.

In communications, the design of a receiver for a fixed transmission scheme is a classification task. For example, consider a simple modulation scheme mapping input bits to constellation points. The receiver trying to map each received noisy symbol to one of the constellation points can be considered a classification problem. This can also be extended to decoding of coded messages transmitted over a noisy channel, where the decoder function tries to map a vector of received symbols to a codeword [16, 38].

Another application of classification in communications is the detection of the type of wireless signals in the air, which is often required for military applications, or for cognitive radios [39]. This includes the detection of the transmitting device by identifying the particular hardware impairments of each individual transmitter [40], detection of the modulation type used by the device [41–43], or detection of the wireless technology employed by the transmitter [44]. A dataset of synthetic simulated channel effects on wireless modulated signals of 11 different modulation types has been released in [45], which has significantly helped the research community to test and compare proposed techniques on a benchmark dataset.

A well-known example of a regression problem encountered in wireless communication systems is channel estimation [10, 11, 46], where the goal is to estimate the channel coefficients from noisy received versions of known pilot signals. While traditional channel estimation methods assume a known channel model, and try to estimate the parameters of this model through least squared or minimum mean-squared error estimation techniques, a data-driven channel estimator does not make any assumptions on the channel model. It instead relies on training data generated from the underlying channel. In the context of wireless communications, while a data-driven approach is attractive when an accurate channel model is not available, the requirement of a large training dataset can mean that training needs to be done off-line.

Supervised learning can also be used as an alternative method to rapidly obtain reasonable suboptimal solutions to complex optimization problems. In wireless networks, we often face highly complex nonconvex or combinatorial optimization problems, for example, in scheduling or deciding transmit powers in a network of interfering transmitters. Many of these problems do not have low-complexity optimal solutions; however, in practice, we need a fast solution to be implemented within the coherence time of the channels. Therefore, we typically resort to some low-complexity suboptimal solution. An alternative would be to train a neural network using the optimal solution for supervision. This can result in a reasonable performance that can be rapidly obtained once the network is trained. Such methods have been extensively

applied to wireless network optimization problems with promising results in terms of the performance-complexity trade-off [8, 47].

1.2.2 Unsupervised Learning

In unsupervised learning problems, we have training data without any output values, and the goal is to learn functions that describe the input data. Such functions may be useful for more efficient supervised learning, as they can be seen as a method for feature extraction. They may also be used to make the input data more amenable to human understanding and interpretation. Unsupervised learning is not as well-defined as supervised learning, since it is not clear what type of description of the data we are looking for. Moreover, it is often the case that the measure to use to compare different descriptions is not obvious and may highly depend on the type of data and application we have in mind. Common unsupervised learning problems are *clustering*, *dimension reduction*, and *density estimation*, all of which have been used extensively in communication systems.

Indeed, clustering is nothing but source compression or quantization, where the goal is to identify a small number of representatives that can adequately represent all possible input vectors. In density estimation, the goal is to determine the distribution of data as accurately as possible from a limited number of samples. Parameterized density estimation is often used in wireless communications when estimating the channel from a limited number of pilot signals. Dimensionality reduction is similar to clustering in the sense that we want a more efficient representation of data, but rather than limiting this representation to a finite number of clusters, we limit its dimension. Projecting a large-dimensional input data to two or three dimensions is used for visualization. A common technique for dimensionality reduction is principle component analysis (PCA), which is also known as the Karhunen-Loeve transform and is used for lossy data compression. More recently neural networks in the form of autoencoders have been employed for dimensionality reduction. Autoencoders play an important role in the data-driven design of compression and communication schemes.

In machine learning, some approaches first try to learn the distribution of data (or the data as well as the output in the context of supervised learning). These are called *generative models*, because once the underlying distribution is learned, one can generate new samples from this distribution. With the advances in deep learning, DNNs have also been used in *deep generative models*, which have shown remarkable performance in modeling complex distributions. Two popular architectures for deep generative modeling are variational autoencoders (VAEs) [48] and generative adversarial networks (GANs) [49]. Generative models have recently been employed in [50, 51] to model a communication channel from data; such models can be used for training other communication components when no channel model is available.

1.2.3 Reinforcement Learning (RL)

RL is another class of machine learning problems, where the goal is to learn how to interact in a random unknown environment based on feedback received in the form

of costs or rewards following each action. In contrast to supervised learning, where the outputs corresponding to a dataset of input samples are given, in RL these need to be learned through interactions with the environment. The environment has a state, and the agent interacts with the environment by taking actions. The goal is to learn the right action to take at each state in order to maximize (minimize) the long-term reward (cost). The cost/reward acquired depends both on the state and the action taken and is typically random with a stationary distribution. Notable examples of RL algorithms are game playing agents that can beat human masters in chess and Go, or more recently in Atari games or more advanced multiplayer video games.

A fundamental aspect of RL is the trade-off between *exploration* and *exploitation*. Exploration refers to taking new unexplored actions to gather more information about the environment to potentially discover actions with higher rewards (lower costs). Exploitation, on the other hand, refers to exploiting the actions that are more likely to provide higher rewards (lower costs) based on past observations. A conservative agent that is more likely to exploit its current knowledge risks losing out on high reward actions that it has never tried. On the other hand, an agent that keeps exploring without considering its past experiences ends up with a low reward. Hence, the goal is to find the right balance between exploration and exploitation.

RL has found applications in wireless networks as early as in the 1990s [52, 53], including power optimization in the physical layer for energy efficient operation [54]. Similarly to other machine learning tools, RL algorithms can be used for two types of problems requiring interactions with an environment, which can model the wireless network environment a device is operating in: the device might have an accurate model of the environment, but the solution of an optimal operation policy for the device may be elusive. In such a case, RL techniques can be used as a numerical solution technique to characterize the optimal (or near optimal) strategy for the device. Note that this is similar to the application of RL in games such as chess or Go, which have well-understood rules but are still highly complex to study methodically. This type of problem typically appears in networking, where multiple devices are scheduled to share the limited spectral resources.

Alternatively, RL methods can be used in wireless networking problems for which the environment is known to be stationary, yet we do not have a good model to characterize its statistical behaviour. This might be the case, for example, when operating over unlicensed bands, where it is difficult or impossible to model the statistics of device activations, spectrum activity, traffic arrivals, and channel variations. In such scenarios, RL tools can be used to learn the best operation strategy through direct interactions with the environment in an online fashion. An example of the application of RL techniques to such a problem is channel access for cognitive users, where the probability of availability of each channel is unknown to the user. This problem is formulated in [55] as a multiarmed bandit (MAB) problem, which is a special case of RL with a single state. The name MAB is motivated by bandit machines in casinos: a gambler wants to play the arm that has the highest chance of winning, but this is unknown in advance. Hence, the gambler has to try as many different arms as possible to discover the best one, while also trying to exploit the estimated best arm as much as possible, since trying each arm has a cost. In [56], scheduling of multiple

energy-harvesting wireless transmitters is considered. Due to the presence of finite-capacity batteries, this problem is formulated as a restless MAB, which refers to the fact that the rewards of the arms are governed by an underlying Markov process, and the state of the arms may change even if they are not played.

1.3 ML Tools Used in Communication System Design

In the previous section we classified ML problems into three main groups. There are many different formulations and approaches within each of these categories. Moreover, many different techniques are available to solve the same problem, with each method providing a different complexity-performance trade-off depending on the implementation constraints, availability of data, and uncertainty in the system parameters. While providing background on all existing tools is beyond the scope of this chapter, we briefly review here three fundamental tools that are commonly used in the design of communication networks (and many other engineering problems), which also appear in most of the later chapters of the book. In particular, we provide a brief introduction to neural networks and how they are trained, particularly for supervised learning tasks, then present the concept of an autoencoder for unsupervised learning, and finally discuss the main challenges of RL along with common solution approaches.

1.3.1 Deep Neural Networks (DNNs)

An artificial neural network is a popular ML model consisting of several layers of "neurons," which are processing units loosely inspired by biological neurons. In a *feedforward neural network*, neurons are organized into layers, and there are no feedback connections that would feed the output of any layer back into the model itself. Neural networks that incorporate feedback connections are called *recurrent neural networks*. Please see Fig. 1.1 for an illustration of a neural network architecture.

The first layer takes as input an affine function of the values of the input signal, while the following layers take as input affine functions of the outputs of the neurons in the previous layer. The left-most layer is typically considered the *input layer* and the right-most layer is the *output layer*, while the remaining layers in between are called the *hidden layers*; see Fig. 1.1. The number of hidden layers provides the *depth* of the network, which led to the terminology of "deep learning." Each hidden layer typically consists of many neurons, the number of which determines the *width* of the network. Let d_k denote the width of the kth layer, where d_0 corresponds to the input dimension (e.g., the number of pixels in the input image). If we denote the output of neuron i in layer k by $x_i^{(k)}$, and its input by $y_i^{(k)}$, we have

$$y_i^{(k)} = w_{0,i}^{(k)} + \sum_{n=1}^{d_{k-1}} w_{n,i}^{(k)} x_n^{(k-1)}, \tag{1.1}$$

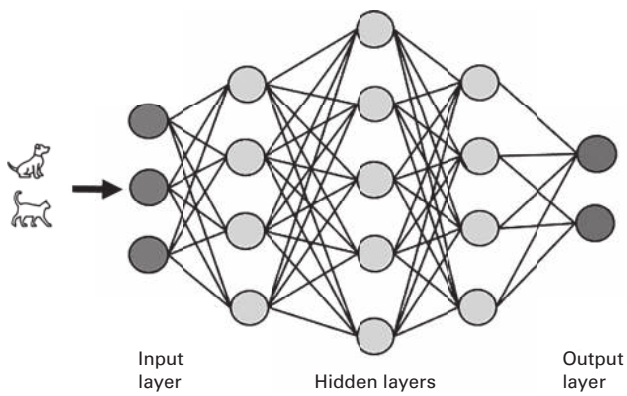

Figure 1.1 Neural network architecture.

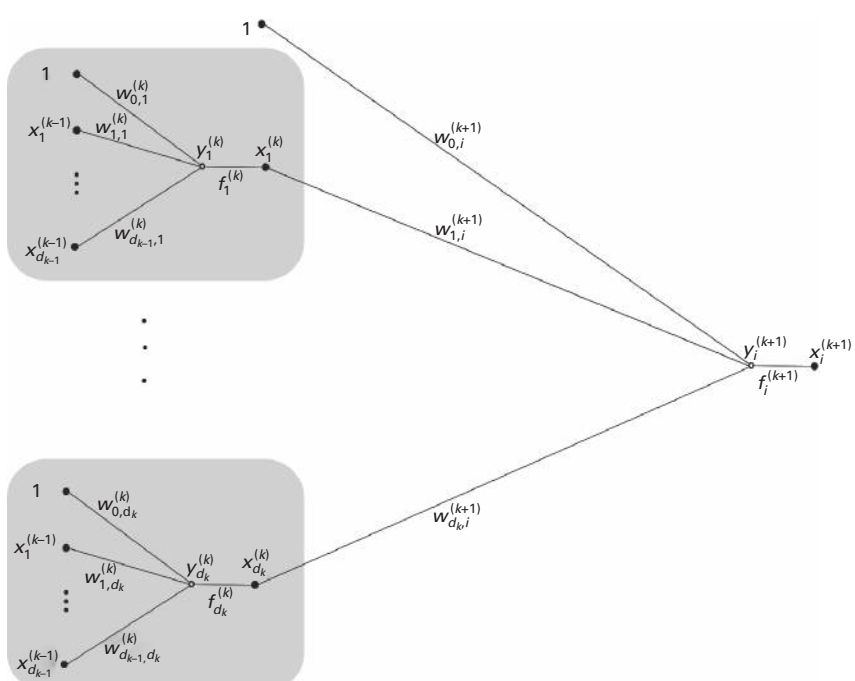

Figure 1.2 One layer of a neural network, and the forward propagation of the neuron activations.

where $w_{n,i}^{(k)}$ denotes the weight of the connection from the output of the nth neuron in layer $k-1$ to the input of the ith neuron in layer k. Here, $w_{0,i}^{(k)}$ corresponds to the bias term for the ith neuron at layer k. Each neuron then applies a nonlinear "activation function" on its input, denoted by $f_i^{(k)}$ for the ith neuron at layer k, resulting in

$$x_i^{(k)} = f_i^{(k)}(y_i^{(k)}), \tag{1.2}$$

as illustrated in Fig. 1.2.

The evaluation of the output values of the neurons in a neural network by recursively computing Eqs. (1.1) and (1.2) is called *forward propagation*, as the information flows from the input layer toward the output layer.

Different activation functions can be used in Eq. (1.2). The most common activation functions are the identity function, step function, a rectified linear unit (ReLU), sigmoid function, hyperbolic tangent, and softmax. Softmax is a generalization of the sigmoid function, which represents a binary variable. Note that the model in Eq. (1.2) allows for employing a different activation function for each neuron in the network. Softmax is often used as the activation function in the output layer of a classifier network to represent a probability distribution over n classes. On the other hand, ReLU is typically used as the activation function for hidden layers, although it is not possible to say which activation function will perform the best. This is often determined through trial and error.

Another aspect of a neural network that we must design is its *architecture*, which refers to the depth of the network and the width of each layer. Although a network with a single layer would be sufficient to fit the training dataset, deeper networks have been shown to generalize better to the test set. Deeper networks may also use less parameters in total; however, they are harder to train. Therefore, the network architecture is often determined through trial and error, taking into account any constraints on the training time and complexity.

Neural networks without an activation function, or an identity activation function, can only learn linear models; nonlinear activation functions are required to learn more complex nonlinear functions from the inputs to the outputs. Even though the choice of the activation function and the network architecture are important design parameters in practice, a network with a single hidden layer with a sigmoid activation function is sufficient to approximate (with arbitrary precision) any continuous function between any two Euclidean spaces [57, 58]. Note, however, that this is an existence result, and we still need to identify the right parameters that would provide the desired approximation. Identification of these parameters corresponds to the *training* stage of the neural network.

For a given neural network architecture, the output corresponding to each input data sample is determined by the weights, $w_{n,i}^{(k)}$. Therefore, the goal of the training process is to determine the weight values that will result in the best performance. Here, the performance measure will depend on the underlying problem. It can be the accuracy of detecting the correct label in a classification problem or determining the correct value in a regression problem. For example, in a multiclass classification problem with C classes, the width of the output layer is chosen as C, and the normalized values of the output layer are interpreted as the likelihoods of the corresponding classes. Let $\mathbf{s}_1, \ldots, \mathbf{s}_N$ denote the data points in the training dataset, with corresponding labels $\mathbf{l}_1, \ldots, \mathbf{l}_N$ represented as one-hot encoded vectors of size C, whose elements are all 0 except for the entry corresponding to the correct class, which is set to 1. That is, if data sample $\mathbf{s}_n$ in the training dataset belongs to class c, then we have $l_{n,c} = 1$, and $l_{n,m} = 0$ for $m \neq c$. In this case, the loss function will be given by