DOI:
10.1109/TCCN.2018.2881442

Document Version
Peer reviewed version

A Very Brief Introduction to Machine Learning With Applications to Communication Systems

Osvaldo Simeone, *Fellow, IEEE*

***Abstract*—Given the unprecedented availability of data and computing resources, there is widespread renewed interest in applying data-driven machine learning methods to problems for which the development of conventional engineering solutions is challenged by modelling or algorithmic deficiencies. This tutorial-style paper starts by addressing the questions of why and when such techniques can be useful. It then provides a high-level introduction to the basics of supervised and unsupervised learning. For both supervised and unsupervised learning, exemplifying applications to communication networks are discussed by distinguishing tasks carried out at the edge and at the cloud segments of the network at different layers of the protocol stack, with an emphasis on the physical layer.**

I. Introduction

After the "AI winter" of the 80s and the 90s, interest in the application of data-driven Artificial Intelligence (AI) techniques has been steadily increasing in a number of engineering fields, including speech and image analysis [1] and communications [2]. Unlike the logic-based expert systems that were dominant in the earlier work on AI (see, e.g., [3]), the renewed confidence in data-driven methods is motivated by the successes of pattern recognition tools based on machine learning. These tools rely on decades-old algorithms, such as backpropagation [4], the Expectation Maximization (EM) algorithm [5], and Q-learning [6], with a number of modern algorithmic advances, including novel regularization techniques and adaptive learning rate schedules (see review in [7]). Their success is built on the unprecedented availability of data and computing resources in many engineering domains.

While the new wave of promises and breakthroughs around machine learning arguably falls short, at least for now, of the requirements that drove early AI research [3], [8], learning algorithms have proven to be useful in a number of important applications – and more is certainly on the way.

King's College London, United Kingdom (email: osvaldo.simeone@kcl.ac.uk). This work has received funding from the European Research Council (ERC) under the European Union Horizon 2020 research and innovation program (grant agreement 725731).

This paper provides a very brief introduction to key concepts in machine learning and to the literature on machine learning for communication systems. Unlike other review papers such as [9]–[11], the presentation aims at highlighting conditions under which the use of machine learning is justified in engineering problems, as well as specific classes of learning algorithms that are suitable for their solution. The presentation is organized around the description of general technical concepts, for which an overview of applications to communication networks is subsequently provided. These applications are chosen to exemplify general design criteria and tools and not to offer a comprehensive review of the state of the art and of the historical progression of advances on the topic.

We proceed in this section by addressing the question "What is machine learning?", by providing a taxonomy of machine learning methods, and by finally considering the question "When to use machine learning?".

A. What is Machine Learning?

In order to fix the ideas, it is useful to introduce the machine learning methodology as an alternative to the conventional engineering approach for the design of an algorithmic solution. As illustrated in Fig. 1(a), the conventional engineering design flow starts with the *acquisition of domain knowledge*: The problem of interest is studied in detail, producing a *mathematical model* that capture the *physics* of the set-up under study. Based on the model, an *optimized algorithm* is produced that offers *performance guarantees* under the assumption that the given physics-based model is an accurate representation of reality.

As an example, designing a decoding algorithm for a wireless fading channel under the conventional engineering approach would require the development, or the selection, of a physical model for the channel connecting transmitter and receiver. The solution would be obtained by tackling an optimization problem, and it would yield optimality guarantees under the given channel model. Typical example of channel models include Gaussian and fading channels (see, e.g., [12]).

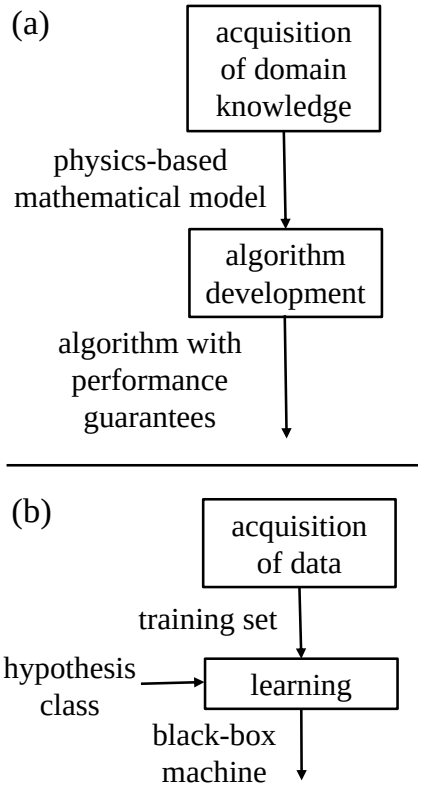

Fig. 1. (a) Conventional engineering design flow; and (b) baseline machine learning methodology.

In contrast, in its most basic form, the machine learning approach substitutes the step of acquiring domain knowledge with the potentially easier task of collecting a sufficiently large number of examples of desired behaviour for the algorithm of interest. These examples constitute the *training set*. As seen in Fig. 1(b), the examples in the training set are fed to a learning algorithm to produce a trained "machine" that carries out the desired task. Learning is made possible by the choice of a set of possible "machines", also known as the *hypothesis class*, from which the learning algorithm makes a selection during training. An example of an hypothesis class is given by a neural network architecture with learnable synaptic weights. Learning algorithms are generally based on the optimization of a performance criterion that measures how well the selected "machine" matches the available data.

For the problem of designing a channel decoder, a machine learning approach can hence operate even in the absence of a well-established channel model. It is in fact enough to have a sufficiently large number of examples of received signals – the inputs to the decoding machine – and transmitted messages – the desired outputs of the decoding machine – to be used for the training of a given class of decoding functions [13].

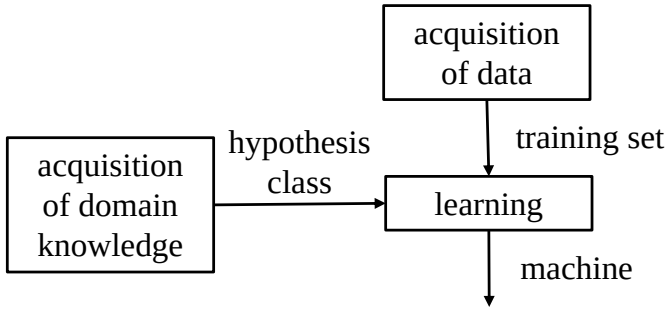

Fig. 2. Machine learning methodology that integrates domain knowledge during model selection.

Moving beyond the basic formulation described above, machine learning tools can *integrate available domain knowledge* in the learning process. This is indeed the key to the success of machine learning tools in a number of applications. A notable example is image processing, whereby knowledge of the translational invariance of visual features is reflected in the adoption of convolutional neural networks as the hypothesis class to be trained. More generally, as illustrated in Fig. 2, domain knowledge can dictate the choice of a specific hypothesis class for use in the training process. Examples of applications of this idea to communication systems, including to the problem of decoding, will be discussed later in the paper.

B. Taxonomy of Machine Learning Methods

There are three main classes of machine learning techniques, as discussed next.

- *Supervised learning*: In supervised learning, the training set consists of pairs of input and desired output, and the goal is that of learning a mapping between input and output spaces. As an illustration, in Fig. 3(a), the inputs are points in the two-dimensional plane, the outputs are the labels assigned to each input (circles or crosses), and the goal is to learn a binary classifier. Applications include the channel decoder discussed above, as well as email spam classification on the basis of examples of spam/ non-spam emails.
- *Unsupervised learning*: In unsupervised learning, the training set consists of unlabelled inputs, that is, of inputs without any assigned desired output. For instance, in Fig. 3(b), the inputs are again points in the two-dimensional plane, but no indication is provided by the data about the corresponding desired output. Unsupervised learning generally aims at discovering properties of the mechanism generating the data. In the example of Fig. 3(b), the goal of unsupervised learning is to cluster together

input points that are close to each other, hence assigning a label – the cluster index – to each input point (clusters are delimited by dashed lines). Applications include clustering of documents with similar topics. It is emphasized that clustering is only one of the learning tasks that fall under the category of unsupervised learning (see Sec. V).

- *Reinforcement learning*: Reinforcement learning lies, in a sense, between supervised and unsupervised learning. Unlike unsupervised learning, some form of supervision exists, but this does not come in the form of the specification of a desired output for every input in the data. Instead, a reinforcement learning algorithm receives feedback from the environment only after selecting an output for a given input or observation. The feedback indicates the degree to which the output, known as action in reinforcement learning, fulfils the goals of the learner. Reinforcement learning applies to sequential decision making problems in which the learner interacts with an environment by sequentially taking actions – the outputs – on the basis of its observations – its inputs – while receiving feedback regarding each selected action.

Most current machine learning applications fall in the supervised learning category, and hence aim at learning an existing pattern between inputs and outputs. Supervised learning is relatively well-understood at a theoretical level [14], [15], and it benefits from well-established algorithmic tools. Unsupervised learning has so far defied a unified theoretical treatment [16]. Nevertheless, it arguably poses a more fundamental practical problem in that it directly tackles the challenge of learning by direct observation without any form of explicit feedback. Reinforcement learning has found extensive applications in problems that are characterized by clear feedback signals, such as win/lose outcomes in games, and that entail searches over large trees of possible action-observation histories [17], [18].

This paper only covers supervised and unsupervised learning. Reinforcement learning requires a different analytical framework grounded in Markov Decision Processes and will not be discussed here (see [17]). For a broader discussion on the technical aspects of supervised and unsupervised learning, we point to [19] and references therein.

C. When to Use Machine Learning?

Based on the discussion in Sec. I-A, the use of a machine learning approach in lieu of a more conventional engineering design should be justified on a case-by-case basis on the basis of its suitability and potential advantages. The following criteria, inspired by [20], offer useful guidelines on the type of engineering tasks that can benefit from the use of machine learning tools.

1. *The traditional engineering flow is not applicable or is undesirable due to a model deficit or to an algorithm deficit* [21].

- With a *model deficit*, no physics-based mathematical models exist for the problem due to insufficient domain knowledge. As a result, a conventional model-based design is inapplicable.
- With an *algorithm deficit*, a well-established mathematical model is available, but existing algorithms optimized on the basis of such model are too complex to be implemented for the given application. In this case, the use of hypothesis classes including efficient "machines", such as neural network of limited size or with tailored hardware implementations (see, e.g., [22], [23] and references therein), can yield lower-complexity solutions.

2. *A sufficiently large training data sets exist or can be created*.

3. *The task does not require the application of logic, common sense, or explicit reasoning based on background knowledge*.

4. *The task does not require detailed explanations for how the decision was made*. The trained machine is by and large a black box that maps inputs to outputs. As such, it does not provide direct means to ascertain why a given output has been produced in response to an input, although recent research has made some progress on this front [24]. This contrasts with engineered optimal solutions, which can be typically interpreted on the basis of physical performance criteria. For instance, a maximum likelihood decoder chooses a given output because it minimizes the probability of error under the assumed model.

5. *The phenomenon or function being learned is stationary for a sufficiently long period of time.* This is in order to enable data collection and learning.

6. *The task has either loose requirement constraints, or, in the case of an algorithm deficit, the required performance guarantees can be provided via numerical simulations*. With the conventional engineering approach, theoretical performance guarantees can be obtained that are backed by a physics-based mathematical model. These guarantees can be relied upon insofar as the model is trusted to be an accurate representation of reality. If a machine learning approach is used to address an algorithm deficit and a physics-based model is available, then numerical results may be sufficient in order to compute satisfactory performance measures. In

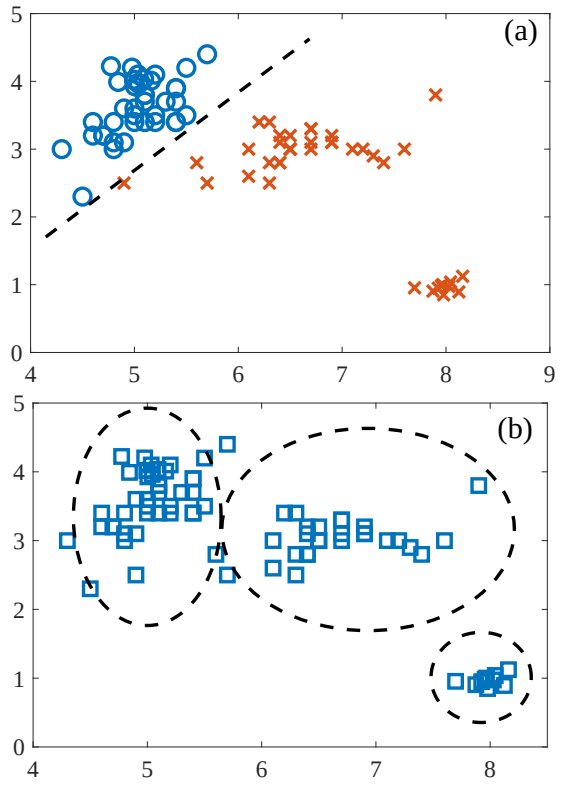

Fig. 3. Illustration of (a) supervised learning and (b) unsupervised learning.

contrast, weaker guarantees can be offered by machine learning in the absence of a physics-based model. In this case, one can provide performance bounds only under the assumptions that the hypothesis class is sufficiently general to include "machines" that can perform well on the problem and that the data is representative of the actual data distribution to be encountered at runtime (see, e.g., [19][Ch. 5]). The selection of a biased hypothesis class or the use of an unrepresentative data set may hence yield strongly suboptimal performance.

We will return to these criteria when discussing applications to communication systems.

II. Machine Learning for Communication Networks

In order to exemplify applications of supervised and unsupervised learning, we will offer annotated pointers to the literature on machine learning for communication systems. Rather than striving for a comprehensive, and historically minded, review, the applications and references have been selected with the goal of illustrating key aspects regarding the use of machine learning in engineering problems.

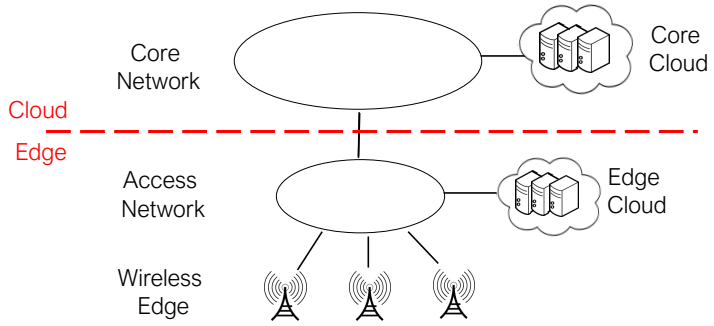

Fig. 4. A generic cellular wireless network architecture that distinguishes between edge segment, with base stations, access points, and associated computing resources, and cloud segment, consisting of core network and associated cloud computing platforms.

Throughout, we focus on tasks carried out at the network side, rather than at the users, and organize the applications along two axes. On one, with reference to Fig. 4, we distinguish tasks that are carried out at the *edge* of the network, that is, at the base stations or access points and at the associated computing platforms, from tasks that are instead responsibility of a centralized *cloud* processor connected to the core network (see, e.g., [25]). The edge operates on the basis of timely local information collected at different layers of the protocol stack, which may include all layers from the physical up to the application layer. In contrast, the centralized cloud processes longer-term and global information collected from multiple nodes in the edge network, which typically encompasses only the higher layers of the protocol stack, namely networking and application layers. Examples of data that may be available at the cloud and at the edge can be found in Table I and Table II, respectively.

As a preliminary discussion, it is useful to ask which tasks of a communication network, if any, may benefit from machine learning through the lens of the criteria reviewed in Sec. I-C. First, as seen, there should be either a model deficit or an algorithm deficit that prevents the use of a conventional model-based engineering design. As an example of model deficit, proactive resource allocation that is based on predictions of human behaviour, e.g., for caching popular contents, may not benefit from well-established and reliable models, making a data-driven approach desirable (see, e.g., [26], [27]). For an instance of algorithm deficit, consider the problem of channel decoding for channels with known and accurate models based on which the maximum likelihood decoder entails an excessive complexity.

Assuming that the problem at hand is characterized by model or algorithm deficits, one should then consider the rest of the criteria discussed in Sec. I-C. Most are

TABLE I
EXAMPLES OF DATA AVAILABLE AT THE EDGE SEGMENT OF A COMMUNICATION NETWORK

Layer	Data
Physical	Baseband signals, channel state information
Medium Access Control/ Link	Throughput, FER, random access load and latency
Network	Location, traffic loads across services, users' device types, battery levels
Application	Users' preferences, content demands, computing loads, QoS metrics

TABLE II
EXAMPLES OF DATA AVAILABLE AT THE CLOUD SEGMENT OF A COMMUNICATION NETWORK

Layer	Data
Network	Mobility patterns, network-wide traffic statistics, outage rates
Application	User's behaviour patterns, subscription information, service usage statistics, TCP/IP traffic statistics

typically satisfied by communication problems. Indeed, for most tasks in communication networks, it is possible to collect or generate training data sets and there is no need to apply common sense or to provide detailed explanations for how a decision was made.

The remaining two criteria need to be checked on a case-by-case basis. First, the phenomenon or function being learned should not change too rapidly over time. For example, designing a channel decoder based on samples obtained from a limited number of realizations of a given propagation channel requires the channel is stationary over a sufficiently long period of time (see [28]).

Second, in the case of a model deficit, the task should have some tolerance for error in the sense of not requiring provable performance guarantees. For instance, the performance of a decoder trained on a channel lacking a well-established channel model, such as a biological communication link, can only be relied upon insofar as one trusts the available data to be representative of the complete set of possible realizations of the problem under study. Alternatively, under an algorithm deficit, a physics-based model, if available, can be possibly used to carry out computer simulations and obtain numerical performance guarantees.

In Sec. IV and Sec. VI, we will provide some pointers to specific applications to supervised and unsupervised learning, respectively.

III. SUPERVISED LEARNING

As introduced in Sec. I, supervised learning aims at discovering patterns that relate inputs to outputs on the basis of a training set of input-output examples. We can distinguish two classes of supervised learning problems depending on whether the outputs are continuous or discrete variables. In the former case, we have a *regression* problem, while in the latter we have a *classification* problem. We discuss the respective goals of the two problems next. This is followed by a formal definition of classification and regression, and by a discussion of the methodology and of the main steps involved in tackling the two classes of problems.

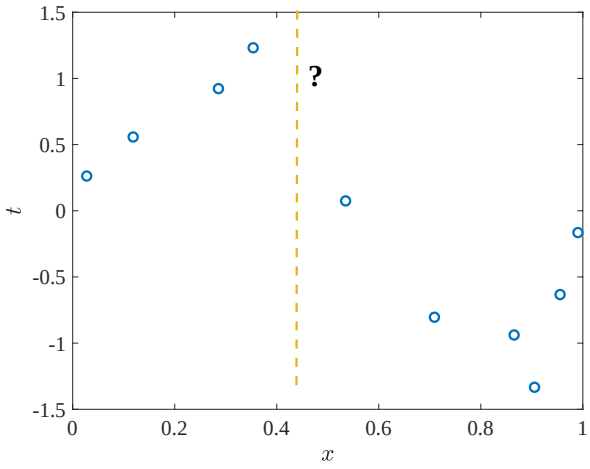

Fig. 5. Illustration of the supervised learning problem of regression: Given input-output training examples (x_n, t_n), with $n = 1, ..., N$, how should we predict the output t for an unobserved value of the input x?

A. Goals

As illustrated in Fig. 5, in a regression problem, we are given a training set $\mathcal{D}$ of N training points (x_n, t_n), with $n = 1, ..., N$, where the variables x_n are the inputs, also known as covariates, domain points, or explanatory variables; while the variables t_n are the outputs, also known as dependent variables, labels, or responses. In regression, the outputs are continuous variables. The problem is to predict the output t for a new, that is, as of yet unobserved, input x.

As illustrated in Fig. 6, classification is similarly defined with the only caveat that the outputs t are discrete

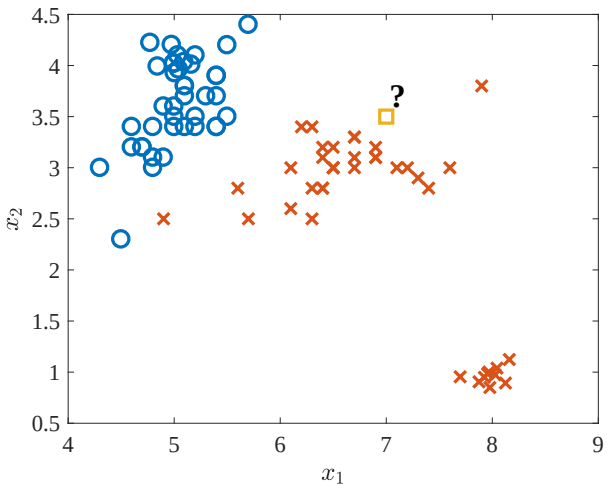

Fig. 6. Illustration of the supervised learning problem of classification: Given input-output training examples (x_n, t_n), with $n = 1, ..., N$, how should we predict the output t for an unobserved value of the input x?

variables that take a finite number of possible values. The value of the output t for a given input x indicates the class to which x belongs. For instance, the label t is a binary variable as in Fig. 6 for a binary classification problem. Based on the training set $\mathcal{D}$, the goal is to predict the label, or the class, t for a new, as of yet unobserved, input x.

To sum up, the goal of both regression and classification is to derive from the training data set $\mathcal{D}$ a predictor $\hat{t}(x)$ that generalizes the input-output mapping in $\mathcal{D}$ to inputs x that are not present in $\mathcal{D}$. As such, learning is markedly distinct from *memorizing*: while memorizing would require producing a value t_n for some recorded input x_n in the training set, learning is about *generalization* from the data set to the rest of the relevant input space.

The problem of extrapolating a predictor from the training set is evidently impossible unless one is willing to make some assumption about the underlying input-output mapping. In fact, the output t may well equal any value for an unobserved x if nothing else is specified about the problem. This impossibility is formalized by the *no free-lunch theorem*: without making assumptions about the relationship between input and output, it is not possible to generalize the available observations outside the training set [14]. The set of assumptions made in order to enable learning are known as *inductive bias*. As an example, for the regression problem in Fig. 5, a possible inductive bias is to postulate that the input-output mapping is a polynomial function of some order.

B. Defining Supervised Learning

Having introduced the goal of supervised learning, we now provide a more formal definition of the problem. Throughout, we use Roman font to denote random variables and the corresponding letter in regular font for realizations.

As a starting point, we assume that the training set $\mathcal{D}$ is generated as

$$(\mathrm{x}_n, \mathrm{t}_n) \underset{\text{i.i.d.}}{\sim} p(x,t), \ n = 1, ..., N, \tag{1}$$

that is, each training sample pair $(\mathrm{x}_n, \mathrm{t}_n)$ is generated from the same true joint distribution $p(x,t)$ and the sample pairs are independent identically distributed (i.i.d.). As discussed, based on the training set $\mathcal{D}$, we wish to obtain a predictor $\hat{t}(x)$ that performs well on any possible relevant input x. This requirement is formalized by imposing that the predictor is accurate for any *test pair* $(\mathrm{x}, \mathrm{t}) \sim p(x,t)$, which is generated independently of all the pairs in the training set $\mathcal{D}$.

The quality of the prediction $\hat{t}(x)$ for a test pair (x,t) is measured by a given loss function $\ell(t, \hat{t})$ as $\ell(t, \hat{t}(x))$. Typical examples of loss functions include the quadratic loss $\ell(t, \hat{t}) = (t - \hat{t})^2$ for regression problems; and the error rate $\ell(t, \hat{t}) = 1(t \neq \hat{t})$, which equals 1 when the prediction is incorrect, i.e., $t \neq \hat{t}$, and 0 otherwise, for classification problems.

The formal goal of learning is that of minimizing the average loss on the test pair, which is referred to as the *generalization loss*. For a given predictor $\hat{t}$, this is defined as

$$L_p(\hat{t}) = \mathrm{E}_{(\mathrm{x},\mathrm{t})\sim p(x,t)}[\ell(\mathrm{t}, \hat{t}(\mathrm{x}))]. \tag{2}$$

The generalization loss (2) is averaged over the distribution of the test pair (x,t).

Before moving on to the solution of the problem of minimizing the generalization loss, we mention that the formulation provided here is only one, albeit arguably the most popular, of a number of alternative formulations of supervised learning. The frequentist framework described above is in fact complemented by other viewpoints, including Bayesian and Minimum Description Length (MDL) (see [19] and references therein).

C. When The True Distribution $p(x,t)$ is Known: Inference

Consider first the case in which the true joint distribution $p(x,t)$ relating input and output is known. This scenario can be considered as an idealization of the situation resulting from the conventional engineering design flow when the available physics-based model is accurate (see Sec. I). Under this assumption, the data set

$\mathcal{D}$ is not necessary, since the mapping between input and output is fully described by the distribution $p(x,t)$.

If the true distribution $p(x,t)$ is known, the problem of minimizing the generalization loss reduces to a standard *inference* problem, i.e., an estimation problem in a regression set-up, in which the outputs are continuous variables, or a detection problem in a classification set-up, in which the outputs are finite discrete variables.

In an inference problem, the optimal predictor $\hat{t}$ can be directly computed from the *posterior* distribution

$$p(t|x) = \frac{p(x,t)}{p(x)}, \tag{3}$$

where $p(x)$ is the marginal distribution of the input x. The latter can be computed from the joint distribution $p(x,t)$ by summing or integrating out all the values of t. In fact, given a loss function $\ell(t,\hat{t})$, the optimal predictor for any input x is obtained as

$$\hat{t}^*(x) = \arg\min_{\hat{t}} \mathrm{E}_{\mathrm{t}\sim p(t|x)}[\ell(\mathrm{t},\hat{t})|x]. \tag{4}$$

In words, the optimal predictor $\hat{t}^*(x)$ is obtained by identifying the value (or values) of t that minimizes the average loss, where the average is taken with respect to the posterior distribution $p(t|x)$ of the output given the input. Given that the posterior $p(t|x)$ yields the optimal predictor, it is also known as the *true predictive distribution*.

The optimal predictor (4) can be explicitly evaluated for given loss functions. For instance, for the quadratic loss, which is typical for regression, the optimal predictor is given by the mean of the predictive distribution, or the posterior mean, i.e.,

$$\hat{t}^*(x) = \mathrm{E}_{\mathrm{t}\sim p(t|x)}[\mathrm{t}|x], \tag{5}$$

while, with the error rate loss, which is typical for classification, problems, the optimal predictor is given by the maximum of the predictive distribution, or the maximum a posteriori (MAP) estimate, i.e.,

$$\hat{t}^*(x) = \arg\max_{t} p(t|x). \tag{6}$$

For a numerical example, consider binary inputs and outputs and the joint distribution $p(x,t)$ such that $p(0,0) = 0.05$, $p(0,1) = 0.45$, $p(1,0) = 0.4$ and $p(1,1) = 0.1$. The predictive distribution for input $x = 0$ is then given as $p(t = 1|x = 0) = 0.9$, and hence we have the optimal predictor given by the average $\hat{t}^*(x = 0) = 0.9 \times 1 + 0.1 \times 0 = 0.9$ for the quadratic loss, and by the MAP solution $\hat{t}^*(x = 0) = 1$ for the error rate loss.

D. When the True Distribution $p(x,t)$ is Not Known: Machine Learning

Consider now the case of interest in which domain knowledge is not available and hence the true joint distribution is unknown. In such a scenario, we have a learning problem and we need to use the examples in the training set $\mathcal{D}$ in order to obtain a meaningful predictor that approximately minimizes the generalization loss. At a high level, the methodology applied by machine learning follows three main steps, which are described next.

1. *Model selection (inductive bias)*: As a first step, one needs to commit to a specific class of *hypotheses* that the learning algorithm may choose from. The hypothesis class is also referred to as model. The selection of the hypothesis class characterizes the inductive bias mentioned above as a pre-requisite for learning. In a probabilistic framework, the hypothesis class, or model, is defined by a family of probability distributions parameterized by a vector θ. Specifically, there are two main ways of specifying a parametric family of distributions as a model for supervised learning:

- *Generative model*: Generative models specify a family of joint distributions $p(x,t|\theta)$;
- *Discriminative model*: Discriminative models parameterize directly the predictive distribution as $p(t|x,\theta)$.

Broadly speaking, discriminative models do not make any assumptions about the distribution of the inputs x and hence may be less prone to bias caused by a misspecification of the hypothesis class. On the flip side, generative models may be able to capture more of the structure present in the data and consequently improve the performance of the predictor [29]. For both types of models, the hypothesis class is typically selected from a common set of probability distributions that lead to efficient learning algorithms in Step 2. Furthermore, any available basic domain knowledge can be in principle incorporated in the selection of the model (see also Sec. VII).

2. *Learning*: Given data $\mathcal{D}$, in the learning step, a learning criterion is optimized in order to obtain the parameter vector θ and identify a distribution $p(x,t|\theta)$ or $p(t|x,\theta)$, depending on whether a generative or discriminative model was selected at Step 1.

3. *Inference:* In the inference step, the learned model is used to obtain the predictor $\hat{t}(x)$ by using (4) with the learned model in lieu of the true distribution. Note that generative models require the calculation of the predictive distribution $p(t|x)$ via marginalization, while discriminative models provide directly the predictive

distribution. As mentioned, the predictor should be evaluated on test data that is different from the training set $\mathcal{D}$. As we will discuss, the design cycle typically entails a loop between validation of the predictor at Step 3 and model selection at Step 1.

The next examples illustrate the three steps introduced above for a binary classification problem.

Example 1: Consider a binary classification problem in which the input is a generic D-dimensional vector $x = [x_1, ..., x_D]^T$ and the output is binary, i.e., $t \in \{0, 1\}$. The superscript "T" represents transposition. In Step 1, we select a model, that is, a parameterized family of distributions. A common choice is given by logistic regression[1], which is a discriminative model whereby the predictive distribution $p(t|x, \theta)$ is parameterized as illustrated in Fig. 7. The model first computes D' fixed features $\phi(x) = [\phi_1(x) \cdots \phi_{D'}(x)]^T$ of the input, where a feature is a function of the data. Then, it computes the predictive probability as

$$p(t = 1|x, w) = \sigma(w^T \phi(x)), \tag{7}$$

where w is the set of learnable weights – i.e., the parameter θ defined above – and $\sigma(a) = (1 + \exp(-a))^{-1}$ is the sigmoid function.

Under logistic regression, the probability that the label is $t = 1$ increases as the linear combination of features becomes more positive, and we have $p(t = 1|x, w) > 0.5$ for $w^T \phi(x) > 0$. Conversely, the probability that the label is $t = 0$ increases as the linear combination of features becomes more negative, with $p(t = 0|x, w) > 0.5$ for $w^T \phi(x) < 0$. As a specific instance of this problem, if we wish to classify emails between spam and non-spam ones, possible useful features may count the number of times that certain suspicious words appear in the text.

Step 2 amounts to the identification of the weight vector w on the basis of the training set $\mathcal{D}$ with the ideal goal of minimizing the generalization loss (2). This step will be further discussed in the next subsection. Finally, in Step 3, the optimal predictor is obtained by assuming that the learned model $p(t|x, w)$ is the true predictive distribution. Assuming an error rate loss function, following the discussion in Sec. III-C, the optimal predictor is given by the MAP choice $\hat{t}^*(x) = 1$ if $w^T \phi(x) > 0$ and $\hat{t}^*(x) = 0$ otherwise. It is noted that the linear combination $w^T \phi(x)$ is also known as *logit or log-likelihood ratio (LLR)*. This rule can be seen to correspond to a *linear classifier* [19]. The performance of the predictor should be tested on new, test, input-output pairs, e.g., new emails in the spam classification example. □

[1]The term "regression" may be confusing, since the model applies to classification.

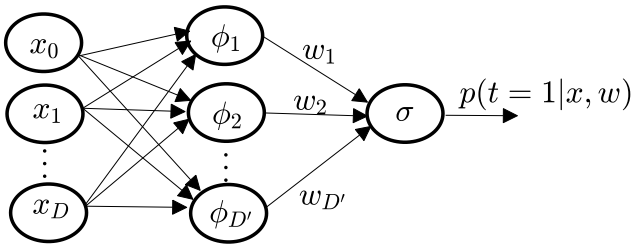

Fig. 7. An illustration of the hypothesis class $p(t|x, w)$ assumed by logistic regression using a neural network representation: functions ϕ_i, with $i = 1, ..., D'$, are fixed and compute features of the input vector $x = [x_1, ..., x_D]$. The learnable parameter vector θ here corresponds to the weights w used to linearly combine the features in (7).

Example 2: Logistic regression requires to specify a suitable vector of features $\phi(x)$. As seen in the email spam classification example, this entails the availability of some domain knowledge to be able to ascertain which functions of the input x may be more relevant for the classification task at hand. As discussed in Sec. I, this knowledge may not be available due to, e.g., cost or time constraints. *Multi-layer neural networks* provide an alternative model choice at Step 1 that obviates the need for hand-crafted features. The model is illustrated in Fig. 8. Unlike linear regression, in a multi-layer neural network, the feature vector $\phi(x)$ used by the last layer to compute the logit, or LLR, that determines the predictive probability (7) is not fixed a priori. Rather, the feature vector is computed by the previous layers. To this end, each neuron, represented as a circle in Fig. 8, computes a fixed non-linear function, e.g., sigmoid, of a linear combination of the values obtained from the previous layer. The weights of these linear combinations are part of the learnable parameters θ, along with the weights of the last layer. By allowing the weights at all layers of the model to be trained simultaneously, multi-layer neural networks enable the joint learning of the last-layer linear classifier and of the features $\phi(x)$ the classifier operates on. As a notable example, deep neural networks are characterized by a large number of intermediate layers that tend to learn increasingly abstract features of the input [7]. □

In the rest of this section, we first provide some technical details about Step 2, i.e., learning, and then we return to Step 1, i.e., model selection. As it will be seen, this order is dictated by the fact that model selection requires some understanding of the learning process.

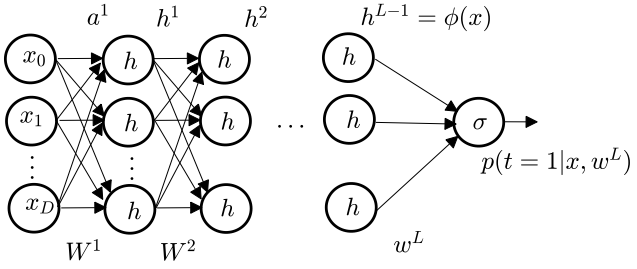

Fig. 8. An illustration of the hypothesis class $p(t|x, w)$ assumed by multi-layer neural networks. The learnable parameter vector θ here corresponds to the weights w^L used at the last layer to linearly combine the features $\phi(x)$ and the weight matrices $W^1, ..., W^{L-1}$ used at the preceding layers in order to compute the feature vector.

E. Learning

Ideally, a learning rule should obtain a predictor that minimizes the generalization error (2). However, as discussed in Sec. III-C, this task is out of reach without knowledge of the true joint distribution $p(x, t)$. Therefore, alternative learning criteria need to be considered that rely on the training set $\mathcal{D}$ rather than on the true distribution.

In the context of probabilistic models, the most basic learning criterion is Maximum Likelihood (ML). ML selects a value of θ in the parameterized family of models $p(x, t|\theta)$ or $p(t|x, \theta)$ that is the most likely to have generated the observed training set $\mathcal{D}$. Mathematically, ML solves the problem of maximizing the log-likelihood function

$$\text{maximize} \;\; \ln p(\mathcal{D}|\theta) \tag{8}$$

over θ, where $p(\mathcal{D}|\theta)$ is the probability of the data set $\mathcal{D}$ for a given value of θ. Given the assumption of i.i.d. data points in $\mathcal{D}$ (see Sec. III-B), the log-likelihood can be written as

$$\ln p(\mathcal{D}|\theta) = \sum_{n=1}^{N} \ln p(t_n|x_n, \theta), \tag{9}$$

where we have used as an example the case of discriminative models. Note that most learning criteria used in practice can be interpreted as ML problems, including the least squares criterion – ML for Gaussian models – and cross-entropy – ML for categorical models.

The ML problem (8) rarely has analytical solutions and is typically addressed by Stochastic Gradient Descent (SGD). Accordingly, at each iteration, subsets of examples, also known as *mini-batches*, are selected from the training set, and the parameter vector is updated in the direction of gradient of the log-likelihood function as evaluated on these examples. The resulting learning rule can be written as

$$\theta^{\text{new}} \leftarrow \theta^{\text{old}} + \gamma \nabla_\theta \ln p(t_n|x_n, \theta)|_{\theta=\theta^{\text{old}}}, \tag{10}$$

where we have defined as $\gamma > 0$ the learning rate, and, for simplicity of notation, we have considered a mini-batch given by a single example (x_n, t_n). It is noted that, with multi-layer neural networks, the computation of the gradient $\nabla_\theta \ln p(t_n|x_n, \theta)$ yields the standard backpropagation algorithm [7], [19]. The learning rate is an example of *hyperparameters* that define the learning algorithm. Many variations of SGD have been proposed that aim at improving convergence (see, e.g., [7], [19]).

ML has evident drawbacks as an indirect means of minimizing the generalization error. In fact, ML only considers the fit of the probabilistic model on the training set without any consideration for the performance on unobserved input-output pairs. This weakness can be somewhat mitigated by *regularization* [7], [19] during learning and by a proper selection of the model via validation, as discussed in the next subsection. Regularization adds a penalty term to the log-likelihood that depends on the model parameters θ. The goal is to prevent the learned model parameters θ to assume values that are a priori too unlikely and that are hence possible symptoms of overfitting. As an example, for logistic regression, one can add a penalty that is proportional to the norm $||w||^2$ of the weight vector w in order to prevent the weights to assume excessively high values when fitting the data in the learning step.

F. Model Selection

We now discuss the first, key, step of model selection, which defines the inductive bias adopted in the learning process. In order to illustrate the main ideas, here we study a particular aspect of model selection, namely that of *model order selection*. To this end, we consider a hierarchical set of models of increasing complexity and we address the problem of selecting (in Step 1) the order, or the complexity, of the specific model to be posited for learning (in Step 2). As an example of model order selection, one may fix a set of models including multi-layer networks of varying number of intermediate layers and focus on determining the number of layers. It is emphasized that the scope of model selection goes much beyond model order selection, including the possible incorporation of domain knowledge and the tuning of the hyperparameters of the learning algorithm.

For concreteness, we focus on the regression problem illustrated in Fig. 5 and assume a set of discriminative models $p(t|x, w)$ under which the output t is distributed as

$$\sum_{m=0}^{M} w_m x^m + \mathcal{N}(0, 1). \tag{11}$$

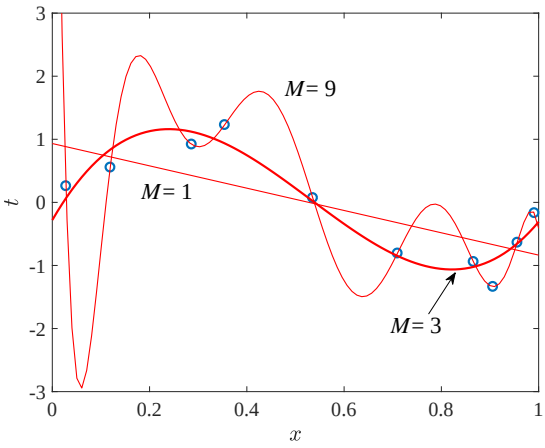

Fig. 9. Training set in Fig. 5, along with a predictor trained by using the discriminative model (11) and ML for different values of the model order M.

In words, the output t is given by a polynomial function of order M of the input x plus zero-mean Gaussian noise of power equal to one. The learnable parameter vector θ is given by the weights $w = [w_0, ..., w_{M-1}]^T$. Model selection, to be carried out in Step 1, amounts to the choice of the model order M.

Having chosen M in Step 1, the weights w can be learned in Step 2 using ML, and then the optimal predictor can be obtained for inference in Step 3. Assuming the quadratic loss, the optimal predictor is given by the posterior mean $\hat{t}(x) = \sum_{m=0}^{M} w_m x^m$ for the learned parameters w. This predictor is plotted in Fig. 9 for different values of M, along with the training set of Fig. 5.

With $M = 1$, the predictor $\hat{t}(x)$ is seen to *underfit* the training data. This is in the sense that the model is not rich enough to capture the variations present in the training data, and, as a result, we obtain a large *training loss*

$$L_{\mathcal{D}}(w) = \frac{1}{N}\sum_{n=1}^{N}(t_n - \hat{t}(x_n))^2. \tag{12}$$

The training loss measures the quality of the predictor defined by weights w on the points in the training set. In contrast, with $M = 9$, the predictor fits well the training data – so much so that it appears to *overfit* it. In other words, the model is too rich and, in order to account for the observations in the training set, it *appears* to yield inaccurate predictions outside it. As a compromise between underfitting and overfitting, the selection $M = 3$ seems to be preferable.

As implied by the discussion above, underfitting can be detected by observing solely the training data $\mathcal{D}$ via the evaluation of the training loss (12). In contrast, overfitting cannot be ascertained on the basis of the training data as it refers to the performance of the predictor outside $\mathcal{D}$. It follows that model selection cannot be carried out by observing only the training set. Rather, some information must be available about the generalization performance of the predictor. This is typically obtained by means of *validation*. In its simplest instantiation, validation partitions the available data into two sets, a training set $\mathcal{D}$ and a *validation set*. The training set is used for learning as discussed in Sec. III-E, while the validation set is used to estimate the generalization loss. This is done by computing the average in (12) only over the validation set. More sophisticated forms of validation exist, including cross-validation [7].

Keeping some data aside for validation, one can obtain a plot as in Fig. 10, where the training loss (12) is compared with the generalization loss (2) estimated via validation. The figure allows us to conclude that, when M is large enough, the generalization loss starts increasing, indicating overfitting. Note, in contrast, that underfitting is detectable by observing the training loss. A figure such as Fig. 10 can be used to choose a value of M that approximately minimizes the generalization loss.

More generally, validation allows for model selection, as well as for the selection of the parameters used by learning the algorithm, such as the learning rate γ in (10). To this end, one compares the generalization loss, estimated via validation, for a number of models and then chooses the one with the smallest estimated generalization loss.

Finally, it is important to remark that the performance of the model selected via validation should be estimated on the basis of a separate data set, typically called the *test set*. This is because the generalization loss estimated using validation is a biased estimate of the true generalization loss (2) due to the process of model selection. In particular, the loss on the validation set will tend to be small, since the model was selected during validation with the aim of minimizing it. Importantly, the test set should never be used during the three steps that make up the machine learning methodology and should ideally only be used once to test the trained predictor.

IV. Applications of Supervised Learning to Communication Systems

In this section, we provide some pointers to existing applications of supervised learning to communication networks. The discussion is organized by following the approach described in Sec. II. Accordingly, we distinguish between tasks carried out at edge and cloud (see

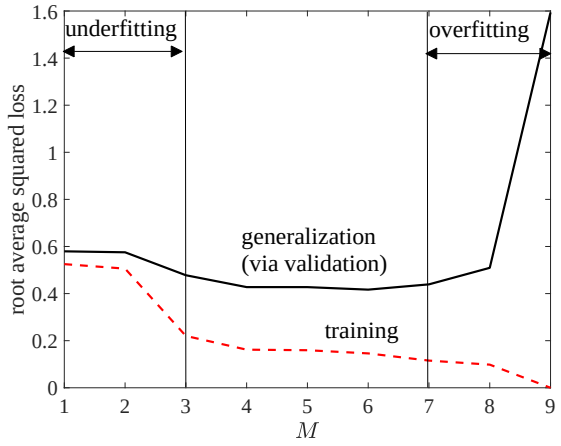

Fig. 10. Training loss and generalization loss, estimated via validation, as a function of the model order M for the example in Fig. 9.

Fig. 4), as well as at different layers of the protocol stack. We refer to Table I and Table II for examples of data types that may be available at the edge and cloud segments.

A. At the Edge

Consider first tasks to be carried out at the edge, i.e., at the base stations or at the associated edge computing platform.

1) Physical Layer: For the physical layer, we focus first on the receiver side and then on the transmitter. At the *receiver*, a central task that can potentially benefit from machine learning is *channel detection and decoding*. This amounts to a multi-class classification problem, in which the input x is given by the received baseband signal and the output is the label of the correct transmitted message (e.g., the transmitted bits) [13], [30]. When can machine learning help? Recalling the discussion in Sec. II, we should first ask whether a modelling or algorithmic deficit exists. A model deficit may occur when operating over channels that do not have well-established mathematical models, such as for molecular communications [31]. Algorithm deficit is more common, given that optimal decoders over a number of well-established channel models tend to be computationally complex. This is the case for channels with strong non-linearities, as recognized as early as the nineties in the context of satellite communication [2], [32] and more recently for optical communications [33]; or for modulation schemes such as continuous phase modulation [34] – another work from the nineties – or in multi–user networks [35].

Assuming that the problem at hand is characterized by a modelling or algorithmic deficit, then one should also check the remaining criteria listed in Sec. II, particularly those regarding the rate of change of the phenomenon under study and the requirements in terms of performance guarantees. For channel decoding, the presence of fast-varying channels may make the first criterion hard to be satisfied in practice (unless channel estimation is made part of the learning process); while stringent reliability requirements may preclude the use of machine learning in the presence of a model deficit.

As mentioned, a generally beneficial idea in the use of data-aided methods is that of *incorporating domain knowledge in the definition of the hypothesis class*. As notable examples related to channel decoding, in [36], [37], knowledge of the near-optimality of message passing methods for the decoding of sparse graphical codes is used to set up a parameterized model that borrows the message passing structure and that is trained to decode more general codes. A related approach is investigated in [38] for polar codes.

Another useful idea is that of directly integrating algorithms designed using the standard engineering flow with trained machines. Instances of this idea include [39] in which a conventional channel decoder is deployed in tandem with a channel equalizer at its input that is trained to compensate for hardware impairments. A related approach is proposed in [40], whereby a conventional decoder is implemented within a turbo-like iterative loop with a machine learning-based regressor that has the role of estimating the channel noise.

Other tasks that can potentially benefit from machine learning at the receiver's side include modulation classification, which is a classification problem justified by the complexity of optimal solutions (algorithm deficit) [41]; localization, which is a regression problem, typically motivated by the lack of tractable channels for complex propagation environments (model deficit) [42]; and channel state information-based authentication, a classification problem made difficult by the absence of well-established models relating channel features with devices' identities (model deficit) [43].

Turning to the *transmitter* side, most emerging applications tackle the algorithmic deficit related to the complexity of the non-convex programs that typically underlie power control and precoding optimization for the downlink. Notably, in [44], a training set is obtained by running a non-convex solver to produce an optimized output power vector for given input channels. Note that the approach does not directly optimize the performance criterion of interest, such as the sum-rate. Rather, it relies on the assumption that similar inputs – the channel coefficients – generally yield similar optimal solutions – the power allocation vector. if the analytical

model available based on domain knowledge is only a coarse approximation of the physical model, the resulting training set can be used to augment the data in order to carry out a preliminary training of a machine learning model [45][2].

For an application at a full-duplex transceiver, we refer to [47], which learns how to cancel self-interference in order to overcome the lack of well-established models for the transmitter-receiver chain of non-linearities.

2) Link and Medium Access Control Layers: At the medium access control layer, we highlight some applications of machine learning that tackle the lack of mathematical models for complex access protocols and communication environments. In [48], a mechanism is proposed to predict whether a channel decoder will succeed on the basis of the outputs of the first few iterations of the iterative decoding process. This binary predictor is useful in order to request an early retransmission at the link layer using Automatic Retransmission Request (ARQ) or Hybrid ARQ (HARQ) in order to reduce latency. At the medium access control layer, data-aided methods can instead be used to predict the availability of spectrum in the presence of interfering incumbent devices with complex activation patterns for cognitive radio applications [49] (see also [50]). An approach that leverages depth images to detect the availability of mmwave channels is proposed in [51].

3) Network and Application Layers: A task that is particularly well-suited for machine learning is the caching of popular contents for reduced latency and network congestion [52]. Caching may take place at the edge and, more traditionally, within the core network segment. Caching at the edge has the advantage of catering directly to the preference of the local population of users, but it generally suffers from a reduced hit rate due to the smaller available storage capacity. Optimizing the selection of contents to be stored at the edge can be formulated as a classification problem that can benefit from a data-driven approach in order to adapt to the specific features of the local traffic [52].

B. At the Cloud

We now turn to some relevant tasks to be carried out at the cloud at both network and application layers.

1) Network: The main task of the network layer is routing (see [53] for further discussion). Considering a software-defined networking implementation, routing requires the availability at a network controller of information regarding the quality of individual communication links in the core network, as well as regarding the status of the queues at the network routers. In the presence of wireless or optical communications, the quality of a link may not be available at the network controller, but it may be predicted using available historical data [33], [54] in the absence of agreed-upon dynamic availability models. In a similar manner, predicting congestion can be framed as a data-aided classification problem [55].

2) Application: Finally, a relevant supervised learning task is that of traffic classification, whereby data streams are classified on the basis of some extracted features, such as packet sizes and inter-arrival times, in terms of their applications, e.g., Voice over IP. [56]

V. Unsupervised Learning

As introduced in Sec. I, unlike supervised learning, unsupervised learning tasks operate over unlabelled data sets consisting solely of the inputs x_n, with $n = 1, ..., N$, and the general goal is that of discovering properties of the data. We start this section by reviewing some of the typical specific unsupervised learning tasks. We then cover methodology, models, and learning, including advanced methods such as Generative Adversarial Networks (GANs) [7].

A. Goals and Definitions

In unsupervised learning, taking a frequentist formulation (see Sec. III-A), we are given a training set $\mathcal{D}$ consisting of N i.i.d. samples $\mathrm{x}_n \sim p(x)$ with $n = 1, ..., N$ generated from an unknown true distribution $p(x)$. The high-level goal is that of learning some useful properties of the distribution $p(x)$. More specifically, we can identify the following tasks.

- *Density estimation*: Density estimation aims at estimating directly the distribution $p(x)$. This may be useful, for example, for use in plug-in estimators of information-theoretic quantities, for the design of compression algorithms, or to detect outliers;
- *Clustering*: Clustering aims at partitioning all points in the data set $\mathcal{D}$ in groups of similar objects, where the notion of similarity is domain-dependent;
- *Dimensionality reduction*, *representation*, *and feature extraction*: These three related tasks represent each data point x_n in a different space, typically of lower dimensionality, in order to highlight independent explanatory factors and/or to ease visualization, interpretation, or the implementation of successive tasks, e.g., classification;
- *Generation of new samples*: Given the data set $\mathcal{D}$, we wish to learn a machine that produces samples that are approximately distributed according

[2]This can be thought of as an example of experience learning as part of small-sample learning techniques [46].

to $p(x)$. As an example, if the data set contains images of celebrities, the idea is to produce plausible images of non-existent celebrities. This can be useful, e.g., to produce artificial scenes for video parameterizes or films.

As suggested by the variety of tasks listed above, unsupervised learning does not have a formal unified formulation as supervised learning. Nevertheless, the general methodology follows three main steps in a manner similar to supervised learning (see Sec. III-D). In Step 1 (model selection), a model, or a hypothesis class, is selected, defining the inductive bias of the learning process. This is done by positing a family of probability distributions $p(x|\theta)$ parameterized by a vector θ. In Step 2 (learning), the data $\mathcal{D}$ is used to optimize a learning criterion with the aim of choosing a value for the parameter vector θ. Finally, in Step 3, the trained model is leveraged in order to carry out the task of interest, e.g., clustering or sample generation.

In the following, we discuss Step 1 (model selection) and Step 2 (learning). For the formulation of specific tasks to be carried out at Step 3, we refer to, e.g., [7], [19], [57].

B. Models

Unsupervised learning models, selected at Step 1 of the machine learning process, typically involve a *hidden or latent* (vector of) variables z_n for each data point x_n. For example, in a clustering problem, the latent variable z_n represents the cluster index of x_n. Latent variables are hidden or unobserved in the sense that they do not appear for any of the data points x_n in $\mathcal{D}$.[3] The relationship between latent variables z_n and observable variables x_n can be modelled in different ways, giving rise to a number of different types of models for unsupervised learning. These are illustrated in Fig. 11 and discussed next.

By way of a short round-up of types of models, with reference to Fig. 11, *directed generative models*, illustrated by Fig. 11(a), posit that there exist hidden causes z yielding the observation x. *Undirected generative models*, represented in Fig. 11(b) model the mutual correlation between x and z. *Discriminative models*, illustrated by Fig. 11(c) model the extraction of the latent representation z from x. Finally, *autoencoders*, represented in Fig. 11(d) assume that x is encoded into a latent representation z in such as way that x can then be approximately recovered from z. In the following, we provide some additional details about directed generative models and autoencoders, and we point to [19] and references therein for a discussion about the remaining models.

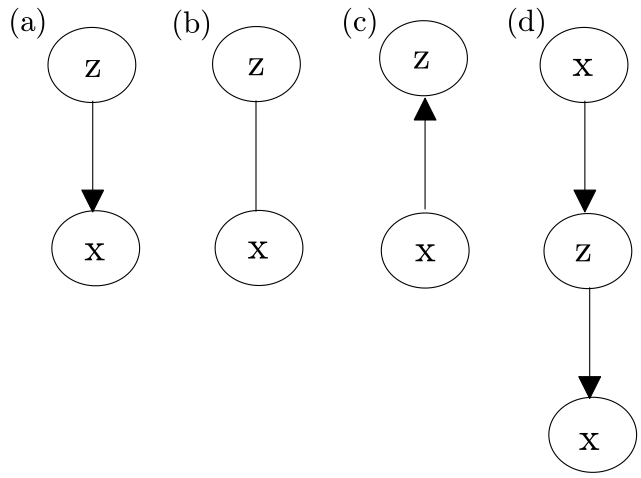

Fig. 11. Illustration of typical unsupervised learning models: (a) directed generative models; (b) undirected generative models; (c) discriminative models; and (d) autoencoders.

As illustrated in Fig. 11(a), directed generative models assume that each data point x is caused[4] by a hidden variable z. This is in the sense that the joint distribution $p(x, z|\theta)$ is parameterized as $p(x, z|\theta) = p(z|\theta)p(x|z, \theta)$, where $p(z|\theta)$ is the distribution of the hidden cause and $p(x|z, \theta)$ is the conditional distribution of the data x given the cause z. As a result, under a directed generative model, the distribution of an observation $\mathrm{x} = x$ can be written as

$$p(x|\theta) = \sum_z p(z|\theta)p(x|z, \theta) = \mathrm{E}_{\mathrm{z}\sim p(z|\theta)}[\ln p(x|\mathrm{z}, \theta)], \tag{13}$$

where the sum in the second term should be replaced by an integration for continuous hidden variables, and the last equality expresses the marginalization over z as an expectation.

As an example, for the problem of document clustering, variable x represents a document in the training set and z is interpreted as a latent topic that "causes" the generation of the document. Model selection requires the specification of a parameterized distribution $p(z|\theta)$ over the topics, e.g., a categorical distribution with parameters equals to the probability of each possible value, and the distribution $p(x|z, \theta)$ of the document given a topic. Basic representatives of directed generative models include mixture of Gaussians and likelihood-free models [19], [58].

[3] Problems in which some of the inputs in $\mathcal{D}$ are labelled by a value z_n are filed under the rubric of semi-supervised learning [29].

[4] The use of the term "cause" is meant to be taken in an intuitive, rather than formal, way. For a discussion on the study of causality, we refer to [8].

As represented in Fig. 11(d), autoencoders model encoding from data x to hidden variables z, as well as decoding from hidden variables back to data. Accordingly, model selection for autoencoders requires the specification of a parameterized family of encoders $p(z|x,\theta)$ and decoders $p(x|z,\theta)$. As an example, autoencoders can be used to learn how to compress an input signal x into a representation z in a smaller space so as to ensure that x can be recovered from z within an admissible level of distortion. Representatives of autoencoders, which correspond to specific choices for the encoder and decoder families of distributions, include Principal Component Analysis (PCA), dictionary learning, and neural network-based autoencoders [19], [57], [58].

C. Learning

We now discuss learning, to be carried out as Step 2. For brevity, we focus on directed generative models and refer to [19] and references therein for a treatment of learning for the other models in Fig. 11. In this regard, we note that the problem of training autoencoders is akin to supervised learning in the sense that autoencoders specify the desired output for each input in the training set.

As for supervised learning, the most basic learning criterion for probabilistic models is ML. Following the discussion in Sec. III-E, ML tackles the problem of maximizing the log-likelihood of the data, i.e.,

$$\underset{\theta}{\text{maximize}} \ \ln p(x|\theta) = \ln \mathrm{E}_{\mathrm{z}\sim p(z|\theta)}[\ln p(x|\mathrm{z},\theta)]. \quad (14)$$

Note that problem (14) considers only one data point x in the data set for the purpose of simplifying the notation, but in practice the log-likelihood needs to be summed over the N examples in $\mathcal{D}$.

Unlike the corresponding problem for supervised learning (8), the likelihood in (14) requires an average over the hidden variables. This is because the value of the hidden variables z is not known, and hence the probability of the observation x needs to account for all possible values of z weighted by their probabilities $p(z|\theta)$. This creates a number of technical challenges. First, the objective in (14) is generally more complex to optimize, since the average over z destroys the typical structure of the model $p(x|z,\theta)$, whose logarithm is often selected as a tractable function (see, e.g., logistic regression). Second, the average in (14) cannot be directly approximated using Monte Carlo methods if the goal is to optimize over the model parameters θ, given that the distribution $p(z|\theta)$ generally depends on θ itself.

To tackle these issues, a standard approach is based on the introduction of a *variational distribution* $q(z)$ over the hidden variables and on the optimization of a tractable lower bound on the log-likelihood known as the *Evidence Lower BOund (ELBO)*. To elaborate, for any fixed value x and any distribution $q(z)$ on the latent variables z (possibly dependent on x), the ELBO $\mathcal{L}(q,\theta)$ is defined as

$$\mathcal{L}(q,\theta) = \mathrm{E}_{\mathrm{z}\sim q(z)}[\ln p(x|\mathrm{z},\theta)] - \mathrm{KL}(q(\mathrm{z})||p(z|\theta)), \quad (15)$$

where $\mathrm{KL}(q||p) = \mathrm{E}_{\mathrm{z}\sim q(z)}[\ln(q(\mathrm{z})/p(\mathrm{z}))]$ is the Kullback-Leibler (KL) divergence. The latter is a measure of the distance between the two distributions, as we will further discuss in Sec. V-D (see [59], [60]). The analytical advantages of the ELBO $\mathcal{L}(q,\theta)$ over the original log-likelihood are that: (*i*) it entails an expectation of the logarithm of the model $p(x|z,\theta)$, which, as mentioned, is typically a tractable function; and (*ii*) the average is over a fixed distribution $q(z)$, which does not depend on the model parameter θ.

Using Jensen's inequality, it can be seen that the ELBO (15) is a global lower bound on the log-likelihood function, that is,

$$\ln p(x|\theta) \ \geq \mathcal{L}(q,\theta). \quad (16)$$

An illustration of the lower bounding property of the ELBO can be found in Fig. 12. An important feature of this inequality is that the ELBO "touches" the log-likelihood function at values θ_0, if any, for which the distribution $q(z)$ satisfies the equality

$$q(z) = p(z|x,\theta_0). \quad (17)$$

In words, the ELBO is tight if the variational distribution is selected to equal the posterior distribution of the hidden variables given the observation x under the model parameter θ_0. Stated less formally, in order to ensure that the ELBO is tight at a value θ_0, one needs to solve the problem of inferring the distribution of the hidden variables z given the observation x under the model identified by the value θ_0.

The property (16) leads to the natural idea of the Expectation-Maximization (EM) algorithm as a means to tackle the ML problem. As illustrated in Fig. 13, EM maximizes the ELBO iteratively, where the ELBO at each iteration is computed to be tight at the current iterate for θ. More formally, the EM algorithm can be summarized as follows[5]. The model vector is initialized to some value θ^{old} and then for each iteration the following two steps are performed.

[5]EM is an instance of the more general Majorization-Minimization algorithm [61].

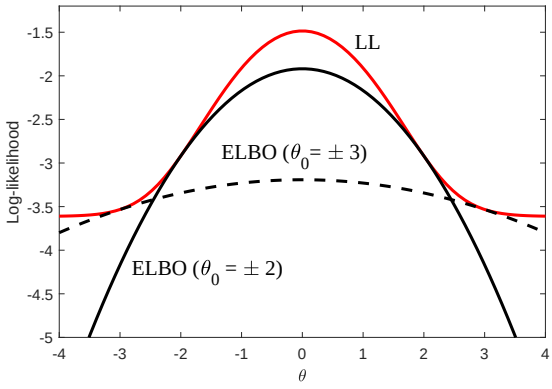

Fig. 12. The ELBO (15) is a global lower bound on the log-likelihood that is tight at values of the model parameters θ_0 for which equality (17) holds.

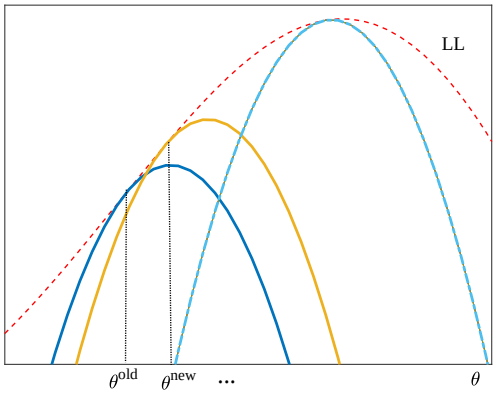

Fig. 13. Illustration of the EM algorithm: At each iteration, a tight ELBO is evaluated in the E step by solving the problem of estimating the latent variables (via the posterior distribution $p(z|x,\theta)$), and then the ELBO is maximized in the M step by solving a problem akin to supervised learning with the estimated latent variables.

- *Expectation, or E, step:* For fixed parameter vector θ^{old}, solve the problem

$$\underset{q}{\text{maximize}}\ \mathcal{L}(q, \theta^{\text{old}}). \tag{18}$$

  The solution of this problem is given by $q^{\text{new}}(z) = p(z|x, \theta^{\text{old}})$. In fact, as discussed, the tightest (i.e., largest) value of the ELBO is obtained by choosing the variational distribution $q(z)$ as the posterior of the latent variables under the current model θ^{old}. This step can be interpreted as estimating the latent variables z, via the predictive distribution $p(z|x, \theta^{\text{old}})$, assuming that the current model θ^{old} is correct.
- *Maximization, or M, step:* For fixed variational distribution $q^{\text{new}}(z)$, solve the problem

$$\underset{\theta}{\text{maximize}}\ \mathcal{L}(q^{\text{new}}, \theta) = \mathrm{E}_{z\sim q^{\text{new}}(z)}\left[\ln p(x, z|\theta)\right]. \tag{19}$$

  This optimization is akin to that carried out in the corresponding supervised learning problem with known latent variables z with the difference that these are randomly selected from the fixed variational distribution $q^{\text{new}}(z)$ obtained in the E step.

Given that the EM algorithm maximizes at each step a lower bound on the log-likelihood that is tight at the current iterate θ^{old}, EM guarantees decreasing objective values along the iterations, which ensures convergence to a local optimum of the original problem. We refer to [57], [58] for detailed examples.

The EM algorithm is generally impractical for large-scale problems due to the complexity of computing the posterior of the latent variables in the E step and of averaging over such distribution in the M step. Many state-of-the-art solutions to the problem of unsupervised learning with probabilistic models entail some approximation of the EM algorithm. Notably, the E step can be approximated by parametrizing the variational distribution with some function $q(z|\varphi)$, or $q(z|x,\varphi)$ to include the dependence on x, and by maximizing ELBO over the variational parameters φ. This approach underlies the popular variational autoencoder technique [7]. In the M step, instead, one can approximate the expectation in (19) using Monte Carlo stochastic approximation based on randomly sampled values of z from the current distribution $q(z)$. Finally, gradient descent can be used to carry out the mentioned optimizations for both E and M steps (see, e.g., [62]).

D. Advanced Learning Methods

As discussed in the previous section, ML is generally prone to overfitting for supervised learning. For unsupervised learning, the performance of ML depends on the task of interest. For example, consider the tasks of density estimation or of generation of new samples (see Sec. V-A). In order to illustrate some of the typical issues encountered when applying the ML criterion, in Fig. 14 we report a numerical result for a problem in which the true data distribution $p(x)$ is multi-modal and the model distribution $p(x|\theta)$ is assumed to be a mixture of Gaussians, i.e., a directed generative model. The ML problem is tackled by using EM based on samples generated from the true distribution (see [19] for details). The learned distribution is seen to be a rather"blurry" estimate that misses the modes of $p(x)$ in an attempt of being inclusive of the full support of $p(x)$. Being a poor estimate of the true distribution, the learned model

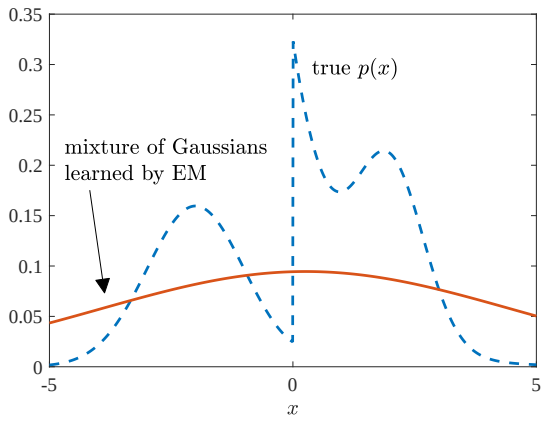

Fig. 14. Illustration of the limitations of ML unsupervised learning, here obtained via the EM algorithm: The ML solution tends to be blurry, missing the modes of the true distribution $p(x)$.

can clearly also be problematic for sample generation in the sense that samples generated from the model would tend to be quite different from the data samples. In the rest of this section, we briefly review advanced learning methods that address this limitation of ML.

In order to move beyond ML, we first observe that ML can be proven to minimize the KL divergence

$$\mathrm{KL}(p_{\mathcal{D}}(x)||p(x|\theta)) = \mathrm{E}_{\mathrm{z}\sim p_{\mathcal{D}}(x)}\left[\ln \frac{p_{\mathcal{D}}(x)}{p(x|\theta)}\right] \tag{20}$$

between the empirical distribution, or histogram, of the data

$$p_{\mathcal{D}}(x) = \frac{N[x]}{N}, \tag{21}$$

where $N[x]$ counts the number of occurrences of value x in the data, and the parameterized model distribution $p(x|\theta)$. In other words, ML fits the model to the histogram of the data by using the KL divergence as a measure of fitness. Indeed, as mentioned in Sec. V-C, the KL divergence is a quantitative measure of "difference" between two distributions. More precisely, as per (20), the KL divergence $\mathrm{KL}(p||q)$ quantifies the difference between two distributions $p(x)$ and $q(x)$ by evaluating the average of the LLR $\ln(p(x)/q(x))$ with respect to $p(x)$.

Consider now the problem illustrated in Fig. 15, in which a discriminator wishes to distinguish between two hypotheses, namely the hypothesis that the data x is a sample from distribution $p(x)$ and the hypothesis that it is instead generated from $q(x)$. To fix the ideas, one can focus as an example on the case where $p(x)$ and $q(x)$ are two Gaussian distributions with different means. To this end, the discriminator computes a statistic, that is, a function, $T(x)$ of the data x, and then decides for the former hypothesis if $T(x)$ is sufficiently large and for

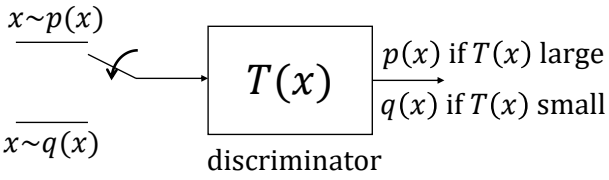

Fig. 15. Discriminator between the hypotheses $\mathrm{x} \sim p(x)$ and $\mathrm{x} \sim q(x)$ based on the statistic $T(x)$. The performance of the optimal discriminator function $T(x)$ under different design criteria yields a measure of the difference between the two distributions.

the latter hypothesis otherwise. Intuitively, one should expect that, the more distinct the two distributions $p(x)$ and $q(x)$ are, the easier it is to design a discriminator that is able to choose the correct hypothesis with high probability.

The connection between the hypothesis testing problem in Fig. 15 and the KL divergence becomes evident if one recalls that the LLR $\ln(p(x)/q(x))$ is known to be the best statistic $T(x)$ in the Neyman-Pearson sense [63]. The KL divergence is hence associated to a particular way of evaluating the performance of the discriminator between the two distributions. Considering a broader formulation of the problem of designing the discriminator in Fig. 15, one can generalize the notion of KL divergence to the class of f-divergences. These are defined as

$$D_f(p||q) = \max_{T(x)} \mathrm{E}_{\mathrm{x}\sim p(x)}[T(\mathrm{x})] - \mathrm{E}_{\mathrm{x}\sim q(x)}[g(T(\mathrm{x}))], \tag{22}$$

for some concave increasing function $g(\cdot)$. The expression above can be interpreted as measuring the performance of the best discriminator $T(x)$ when the design criterion is given by the right-hand side of (22), i.e., $\mathrm{E}_{\mathrm{x}\sim p(x)}[T(\mathrm{x})] - \mathrm{E}_{\mathrm{x}\sim q(x)}[g(T(\mathrm{x}))]$, for a given function $g(\cdot)$. Note that this criterion is indeed larger for a discriminator that is able to output a large value of the statistic $T(x)$ under $p(x)$ and a small value under $q(x)$. The KL divergence corresponds to a specific choice of such function (see [19] for details).

In order to move beyond ML, one can then consider fitting the model distribution to the data histogram by using a divergence measure that is tailored to the data and that captures the features of the empirical distribution that are most relevant for a given application. Such a divergence measure can be obtained by choosing a suitable function $g(\cdot)$ in (22) and by optimizing (22) over a parameterized (differentiable) discriminator function $T_{\varphi}(\mathrm{x})$. Integrating the evaluation of the divergence with the problem of learning the model parameters yields the

min-max problem

$$\min_{\theta} \max_{\varphi} \mathrm{E}_{\mathrm{x}\sim p_{\mathcal{D}}(x)}[T_{\varphi}(\mathrm{x})] - \mathrm{E}_{\mathrm{x}\sim p(x|\theta)}[g(T_{\varphi}(\mathrm{x}))]. \quad (23)$$

This can be famously interpreted as a game between the learner, which optimizes the model parameters θ, and the discriminator, which tries to find the best function $T_{\varphi}(\mathrm{x})$ to distinguish between data and generated samples. The resulting method, known as GAN, has recently led to impressive improvements of ML for sample generation [64].

VI. Applications of Unsupervised Learning to Communication Systems

In this section, we highlight some applications of unsupervised learning to communication networks.

A. At the Edge

1) Physical Layer: Let us first consider some applications of *autoencoders* at the physical layer as implemented by the network edge nodes. A fundamental idea is to treat the chain of encoder, channel, and decoder in a communication link as an autoencoder, where, with reference to Fig. 11(d), the input message is x, the transmitted codewords and received signals represent the intermediate representation z, and the output of the decoder should match the input [30]. Note that, for this particular autoencoder, the mapping $p(x|z)$ can only be partially learned, as it includes not only the encoder but also the communication channel, while the conditional distribution $p(x|z)$ defining the decoder can be learned. We should now ask when this viewpoint can be beneficial in light of the criteria reviewed in Sec. I-C.

To address this question, one should check whether a model or algorithm deficit exists to justify the use of machine learning tools. Training an autoencoder requires the availability of a model for the channel, and hence a model deficit would make this approach inapplicable unless further mechanisms are put in place (see below). Examples of algorithm deficit include channels with complex non-linear dynamical models, such as optical links [65]; Gaussian channels with feedback, for which optimal practical encoding schemes are not known [66]; multiple access channels with sparse transmission codes [67]; and joint source-channel coding [68].

Other applications at the physical layer leverage the use of autoencoders as compressors (see Sec. V-B) or denoisers. For channels with a complex structure with unavailable channel models or with unknown optimal compression algorithms, autoencoders can be used to compress channel state information for the purpose of feedback on frequency-division duplex links [69]. Autoencoders can also be used for their capacity to denoise the input signal by means of filtering through the lower dimensional representation z. This is done in [70] for the task of localization on the basis of the received baseband signal. To this end, an autoencoder is learned for every reference position in space with the objective of denoising signals received from the given location. At test time, the location that corresponds to the autoencoder with the smallest reconstruction error is taken as an estimate of the unknown transmitting device.

We now review some applications of the *generative models* illustrated in Fig. 11(a). A natural idea is that of using generative models to learn how to generate samples from a given channel [71], [72]. This approach is sound for scenarios that lack tractable channel models. As a pertinent example, generative models can be used to mimic and identify non-linear channels for satellite communications [2]. The early works on the subject carried out in the nineties are also notable for the integration of the domain knowledge into the definition of machine learning models (see Sec. IV). In fact, mindful of the strong linear components of the channels, these works posit a learnable model that includes linear filters and non-linearities [2].

Another approach that can be considered as unsupervised was proposed in [73] in order to solve the challenging problem of power control for interference channels. The approach tackles the resulting algorithm deficit by means of a direct optimization of the sum-rate with the aim of obtaining the power allocation vector (as fractions of the maximal available powers) at the output of a neural network. Related supervised learning methods were discussed in Sec. IV. A similar approach – also based on the idea of directly maximizing the criterion of interest so as to obtain an approximate solution at the output of a neural network – was considered in [74] for minimum mean squared error channel estimation with non-Gaussian channels, e.g., multi-path channels.

2) Medium Access Layer: At the medium access layer, generative models have been advocated in [75] as a way to generate new examples so as to augment a data set used to train a classifier for spectrum sensing (see Sec. IV). An unsupervised learning task that has found many applications in communications is *clustering*. For example, in [76], clustering is used to support radio resource allocation in a heterogeneous network.

B. At the Cloud

1) Network Layer: Another typical application of clustering is to enable hierarchical clustering for routing in self-organizing multi-hop networks. Thanks to clustering, routing can be carried out more efficiently by routing

first at the level of clusters, and then locally within each cluster [77]. For an application of the unsupervised learning task of *density estimation*, consider the problem of detecting anomalies in networks. For instance, by learning the typical distribution of the features of a working link, one can identify malfunctioning ones. This approach may be applied, e.g., to optical networks [54].

2) Application Layer: Finally, we point to two instances of unsupervised learning at the application layer that are usually carried out at data centers in the cloud. These tasks follow a conceptually different approach as they are based on discovering structure in graphs. The first problem is community detection in social networks. This amounts to a clustering problem whereby one wishes to isolate communities of nodes in a social graph on the basis of the observation of a realization of the underlying true graph of relationships [78]. Another application is the ranking of webpages based on the graph of hyperlinks carried out by PageRank [19], [79].

VII. Concluding Remarks

In the presence of modelling or algorithmic deficiencies in the conventional engineering flow based on the acquisition of domain knowledge, data-driven machine learning tools can speed up the design cycle, reduce the complexity and cost of implementation, and improve over the performance of known algorithms. To this end, machine learning can leverage the availability of data and computing resources in many engineering domains, including modern communication systems. Supervised, unsupervised, and reinforcement learning paradigms lend themselves to different tasks depending on the availability of examples of desired behaviour or of feedback. The applicability of learning methods hinges on specific features of the problem under study, including its time variability and its tolerance to errors. As such, a data-driven approach should not be considered as a universal solution, but rather as a useful tool whose suitability should be assessed on a case-by-case basis. Furthermore, machine learning tools allow for the integration of traditional model-based engineering techniques and of existing domain knowledge in order to leverage the complementarity and synergy of the two solutions (see Fig. 2).

As a final note, while this paper has focused on applications of machine learning to communication systems, communication is conversely a key element of distributed machine learning platforms. In these systems, learning tasks are carried out at distributed machines that need to coordinate via communication, e.g., by transferring the results of intermediate computations. A recent line of work investigates the resulting interplay between computation and communication [80].

References

[1] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[2] M. Ibnkahla, "Applications of neural networks to digital communications–a survey," *Signal processing*, vol. 80, no. 7, pp. 1185–1215, 2000.

[3] H. J. Levesque, *Common Sense, the Turing Test, and the Quest for Real AI: Reflections on Natural and Artificial Intelligence*. MIT Press, 2017.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38, 1977.

[6] C. Watkins, "Learning form delayed rewards," *Ph. D. thesis, King's College, University of Cambridge*, 1989.

[7] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[8] J. Pearl and D. Mackenzie, *The Book of Why: The New Science of Cause and Effect*. Basic Books, 2018.

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

[10] 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, 2017.

[11] Z. Qin, H. Ye, G. Y. Li, and B.-H. F. Juang, "Deep Learning in Physical Layer Communications," *ArXiv e-prints*, Jul. 2018.

[12] S. Lin and D. J. Costello, *Error control coding*. Pearson Education India, 2001.

[13] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning-based channel decoding," in *CISS 2017*, 2017, pp. 1–6.

[14] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[15] D. Arpit, S. Jastrzębski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio, and S. Lacoste-Julien, "A Closer Look at Memorization in Deep Networks," *ArXiv e-prints*, Jun. 2017.

[16] T. Hastie, R. Tibshirani, and J. Friedman, "Unsupervised learning," in *The elements of statistical learning*. Springer, 2009, pp. 485–585.

[17] R. S. Sutton, A. G. Barto *et al.*, *Reinforcement learning: An introduction*. MIT press, 2018.

[18] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.

[19] O. Simeone, "A brief introduction to machine learning for engineers," *Foundations and Trends in Signal Processing*, vol. 12, no. 3-4, pp. 200–431, 2018.

[20] E. Brynjolfsson and T. Mitchell, "What can machine learning do? Workforce implications," *Science*, vol. 358, no. 6370, pp. 1530–1534, 2017.

[21] S. Kannan, H. Kim, and S. Oh, "Deep learning and information theory: An emerging interface," *IEEE ISIT 2018 Tutorial*.

[22] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[23] A. Bagheri, O. Simeone, and B. Rajendran, "Training probabilistic spiking neural networks with first-to-spike decoding," *arXiv preprint arXiv:1710.10704*, 2017.

[24] J. Chen, L. Song, M. J. Wainwright, and M. I. Jordan, "Learning to explain: An information-theoretic perspective on model interpretation," *arXiv preprint arXiv:1802.07814*, 2018.

[25] M. Polese, R. Jana, V. Kounev, K. Zhang, S. Deb, and M. Zorzi, "Machine Learning at the Edge: A Data-Driven Architecture with Applications to 5G Cellular Networks," *ArXiv e-prints*, Aug. 2018.

[26] G. Paschos, E. Bastug, I. Land, G. Caire, and M. Debbah, "Wireless caching: Technical misconceptions and business barriers," *IEEE Communications Magazine*, vol. 54, no. 8, pp. 16–22, 2016.

[27] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, "Machine learning for wireless networks with artificial intelligence: A tutorial on neural networks," *arXiv preprint arXiv:1710.02913*, 2017.

[28] M. Angjelichinoski, K. F. Trillingsgaard, and P. Popovski, "A statistical learning approach to ultra-reliable low latency communication," *arXiv preprint arXiv:1809.05515*, 2018.

[29] M. Seeger, "A taxonomy for semi-supervised learning methods," MIT Press, Tech. Rep., 2006.

[30] T. J. O'Shea and J. Hoydis, "An introduction to machine learning communications systems," *arXiv preprint*, vol. 1702, 2017.

[31] N. Farsad and A. Goldsmith, "Neural network detection of data sequences in communication systems," *arXiv preprint arXiv:1802.02046*, 2018.

[32] S. Bouchired, D. Roviras, and F. Castanié, "Equalisation of satellite mobile channels with neural network techniques," *Space Communications*, vol. 15, no. 4, pp. 209–220, 1998.

[33] Y. Wang, M. Martonosi, and L.-S. Peh, "A supervised learning approach for routing optimizations in wireless sensor networks," in *Proc. Int. Workshop on Multi-hop ad hoc Networks*. ACM, 2006, pp. 79–86.

[34] G. De Veciana and A. Zakhor, "Neural net-based continuous phase modulation receivers," *IEEE Transactions on Communications*, vol. 40, no. 8, pp. 1396–1408, 1992.

[35] X. Jin and H.-N. Kim, "Deep Learning Detection Networks in MIMO Decode-Forward Relay Channels," *ArXiv e-prints*, Jul. 2018.

[36] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Be'ery, "Deep learning methods for improved decoding of linear codes," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 119–131, 2018.

[37] L. Lugosch and W. J. Gross, "Neural offset min-sum decoding," in *IEEE int. Symp. Information Theory (ISIT 2017)*. IEEE, 2017, pp. 1361–1365.

[38] S. Cammerer, T. Gruber, J. Hoydis, and S. ten Brink, "Scaling deep learning-based decoding of polar codes via partitioning," in *IEEE GLOBECOM 2017*, 2017, pp. 1–6.

[39] S. Schibisch, S. Cammerer, S. Dörner, J. Hoydis, and S. t. Brink, "Online label recovery for deep learning-based communication through error correcting codes," *arXiv preprint arXiv:1807.00747*, 2018.

[40] F. Liang, C. Shen, and F. Wu, "An iterative bp-cnn architecture for channel decoding," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 144–159, Feb 2018.

[41] H. Agirman-Tosun, Y. Liu, A. M. Haimovich, O. Simeone, W. Su, J. Dabin, and E. Kanterakis, "Modulation classification of mimo-ofdm signals by independent component analysis and support vector machines," in *Proc. ASILOMAR 2011*, 2011, pp. 1903–1907.

[42] S.-H. Fang and T.-N. Lin, "Indoor location system based on discriminant-adaptive neural network in ieee 802.11 environments," *IEEE Transactions on Neural networks*, vol. 19, no. 11, pp. 1973–1978, 2008.

[43] Q. Wang, H. Li, Z. Chen, D. Zhao, S. Ye, and J. Cai, "Supervised and Semi-Supervised Deep Neural Networks for CSI-Based Authentication," *ArXiv e-prints*, Jul. 2018.

[44] 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," in *IEEE Signal Processing Advances in Wireless Communications (SPAWC) 2017*, 2017, pp. 1–6.

[45] 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," *ArXiv e-prints*, Aug. 2018.

[46] J. Shu, Z. Xu, and D. Meng, "Small Sample Learning in Big Data Era," *ArXiv e-prints*, Aug. 2018.

[47] A. Balatsoukas-Stimming, "Non-linear digital self-interference cancellation for in-band full-duplex radios using neural networks," *arXiv preprint arXiv:1711.00379*, 2017.

[48] N. Strodthoff, B. Göktepe, T. Schierl, C. Hellge, and W. Samek, "Enhanced Machine Learning Techniques for Early HARQ Feedback Prediction in 5G," *ArXiv e-prints*, Jul. 2018.

[49] V. K. Tumuluru, P. Wang, and D. Niyato, "A neural network based spectrum prediction scheme for cognitive radio," in *IEEE International Conference on Communications (ICC 2010)*, 2010, pp. 1–5.

[50] D. Del Testa, M. Danieletto, G. M. Di Nunzio, and M. Zorzi, "Estimating the number of receiving nodes in 802.11 networks via machine learning techniques," in *IEEE Global Communications Conference (GLOBECOM)*, 2016, pp. 1–7.

[51] H. Okamoto, T. Nishio, K. Nakashima, Y. Koda, K. Yamamoto, M. Morikura, Y. Asai, and R. Miyatake, "Machine-learning-based future received signal strength prediction using depth images for mmwave communications," *arXiv preprint arXiv:1803.09698*, 2018.

[52] 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, 2017.

[53] M. Zorzi, A. Zanella, A. Testolin, M. D. F. De Grazia, and M. Zorzi, "Cognition-based networks: A new perspective on network optimization using learning and distributed intelligence," *IEEE Access*, vol. 3, pp. 1512–1530, 2015.

[54] F. Musumeci, C. Rottondi, A. Nag, I. Macaluso, D. Zibar, M. Ruffini, and M. Tornatore, "A survey on application of machine learning techniques in optical networks," *arXiv preprint arXiv:1803.07976*, 2018.

[55] F. Tang, B. Mao, Z. M. Fadlullah, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "On removing routing protocol from future wireless networks: A real-time deep learning approach for intelligent traffic control," *IEEE Wireless Communications*, vol. 25, no. 1, pp. 154–160, 2018.

[56] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[57] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[58] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.
[59] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.
[60] O. Simeone, "Introducing information measures via inference [lecture notes]," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 167–171, 2018.
[61] Y. Sun, P. Babu, and D. P. Palomar, "Majorization-minimization algorithms in signal processing, communications, and machine learning," *IEEE Transactions on Signal Processing*, vol. 65, no. 3, pp. 794–816, 2017.
[62] A. Mnih and K. Gregor, "Neural variational inference and learning in belief networks," *arXiv preprint arXiv:1402.0030*, 2014.
[63] H. V. Poor, *An introduction to signal detection and estimation*. Springer Science & Business Media, 2013.
[64] I. Goodfellow, "NIPS 2016 tutorial: Generative adversarial networks," *arXiv preprint arXiv:1701.00160*, 2016.
[65] B. Karanov, M. Chagnon, F. Thouin, T. A. Eriksson, H. Bülow, D. Lavery, P. Bayvel, and L. Schmalen, "End-to-end deep learning of optical fiber communications," *arXiv preprint arXiv:1804.04097*, 2018.
[66] H. Kim, Y. Jiang, S. Kannan, S. Oh, and P. Viswanath, "Deepcode: Feedback codes via deep learning," *arXiv preprint arXiv:1807.00801*, 2018.
[67] M. Kim, N. I. Kim, W. Lee, and D. H. Cho, "Deep learning-aided scma," *IEEE Communications Letters*, vol. 22, no. 4, pp. 720–723, April 2018.
[68] E. Bourtsoulatze, D. Burth Kurka, and D. Gunduz, "Deep Joint Source-Channel Coding for Wireless Image Transmission," *ArXiv e-prints*, Sep. 2018.
[69] C.-K. Wen, W.-T. Shih, and S. Jin, "Deep learning for massive mimo csi feedback," *IEEE Wireless Communications Letters*, 2018.
[70] C. Xiao, D. Yang, Z. Chen, and G. Tan, "3-d ble indoor localization based on denoising autoencoder," *IEEE Access*, vol. 5, pp. 12 751–12 760, 2017.
[71] T. J. O'Shea, T. Roy, and N. West, "Approximating the void: Learning stochastic channel models from observation with variational generative adversarial networks," *arXiv preprint arXiv:1805.06350*, 2018.
[72] H. Ye, G. Y. Li, B.-H. F. Juang, and K. Sivanesan, "Channel agnostic end-to-end learning based communication systems with conditional gan," *arXiv preprint arXiv:1807.00447*, 2018.
[73] F. Liang, C. Shen, W. Yu, and F. Wu, "Towards Optimal Power Control via Ensembling Deep Neural Networks," *ArXiv e-prints*, Jul. 2018.
[74] D. Neumann, T. Wiese, and W. Utschick, "Learning the mmse channel estimator," *IEEE Transactions on Signal Processing*, 2018.
[75] K. Davaslioglu and Y. E. Sagduyu, "Generative adversarial learning for spectrum sensing," *arXiv preprint arXiv:1804.00709*, 2018.
[76] A. Abdelnasser, E. Hossain, and D. I. Kim, "Clustering and resource allocation for dense femtocells in a two-tier cellular ofdma network," *IEEE Transactions on Wireless Communications*, vol. 13, no. 3, pp. 1628–1641, 2014.
[77] A. A. Abbasi and M. Younis, "A survey on clustering algorithms for wireless sensor networks," *Computer communications*, vol. 30, no. 14-15, pp. 2826–2841, 2007.
[78] E. Abbe, A. S. Bandeira, and G. Hall, "Exact recovery in the stochastic block model," *arXiv preprint arXiv:1405.3267*, 2014.
[79] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.
[80] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Redundancy techniques for straggler mitigation in distributed optimization and learning," *arXiv preprint arXiv:1803.05397*, 2018.