Ad Hoc Networks 93 (2019) 101913

Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc

Machine learning for wireless communications in the Internet of Things: A comprehensive survey

Jithin Jagannath[a,b,*], Nicholas Polosky[a], Anu Jagannath[a], Francesco Restuccia[b], Tommaso Melodia[b]

[a] ANDRO Advanced Applied Technology, ANDRO Computational Solutions, LLC, Rome, NY 13440, USA
[b] Department of Electrical and Computer Engineering, Northeastern University, Boston, MA, 02115, USA

ARTICLE INFO

Article history:
Received 23 January 2019
Revised 27 May 2019
Accepted 5 June 2019
Available online 11 June 2019

Keywords:
Machine learning
Deep learning
Reinforcement learning
Internet of Things
Wireless ad hoc network
Spectrum sensing
Medium access control
Routing protocol

ABSTRACT

The Internet of Things (IoT) is expected to require more effective and efficient wireless communications than ever before. For this reason, techniques such as spectrum sharing, dynamic spectrum access, extraction of signal intelligence and optimized routing will soon become essential components of the IoT wireless communication paradigm. In this vision, IoT devices must be able to not only learn to autonomously extract spectrum knowledge on-the-fly from the network but also leverage such knowledge to dynamically change appropriate wireless parameters (*e.g.*, frequency band, symbol modulation, coding rate, route selection, etc.) to reach the network's optimal operating point. Given that the majority of the IoT will be composed of tiny, mobile, and energy-constrained devices, traditional techniques based on *a priori* network optimization may not be suitable, since (i) an accurate model of the environment may not be readily available in practical scenarios; (ii) the computational requirements of traditional optimization techniques may prove unbearable for IoT devices. To address the above challenges, much research has been devoted to exploring the use of machine learning to address problems in the IoT wireless communications domain. The reason behind machine learning's popularity is that it provides a general framework to solve very complex problems where a model of the phenomenon being learned is too complex to derive or too dynamic to be summarized in mathematical terms.

This work provides a comprehensive survey of the state of the art in the application of machine learning techniques to address key problems in IoT wireless communications with an emphasis on its ad hoc networking aspect. First, we present extensive background notions of machine learning techniques. Then, by adopting a bottom-up approach, we examine existing work on machine learning for the IoT at the physical, data-link and network layer of the protocol stack. Thereafter, we discuss directions taken by the community towards hardware implementation to ensure the feasibility of these techniques. Additionally, before concluding, we also provide a brief discussion of the application of machine learning in IoT beyond wireless communication. Finally, each of these discussions is accompanied by a detailed analysis of the related open problems and challenges.

1. Introduction

Internet of Things (IoT) - the term first coined by K. Ashton in 1999 [1] has hence emerged to describe a network of interconnected devices - sensors, actuators, mobile phones, among others - which interact and collaborate with each other to attain common objectives. IoT will soon become the most pervasive technology worldwide. In the next few years, cars, kitchen appliances, televisions, smartphones, utility meters, intra-body sensors, thermostats, and almost anything we can imagine will be accessible from anywhere on the planet [2]. The revolution brought by the IoT has been compared to the building of roads and railroads during the Industrial Revolution of the 18th to 19th centuries [3] – and is expected to radically transform the education, healthcare, smart home, manufacturing, mining, commerce, transportation, and surveillance fields, just to mention a few [4].

As the IoT gains momentum in every aspect of our lives, the demand for wireless resources will accordingly increase in an unprecedented way. According to the latest Ericsson's mobility report, there are now 5.2 billion mobile broadband subscriptions

* Corresponding author.
E-mail addresses: jjagannath@androcs.com, jagannath.j@northeastern.edu (J. Jagannath), npolosky@androcs.com (N. Polosky), ajagannath@androcs.com (A. Jagannath), frestuc@northeastern.edu (F. Restuccia), melodia@northeastern.edu (T. Melodia).

https://doi.org/10.1016/j.adhoc.2019.101913
1570-8705/

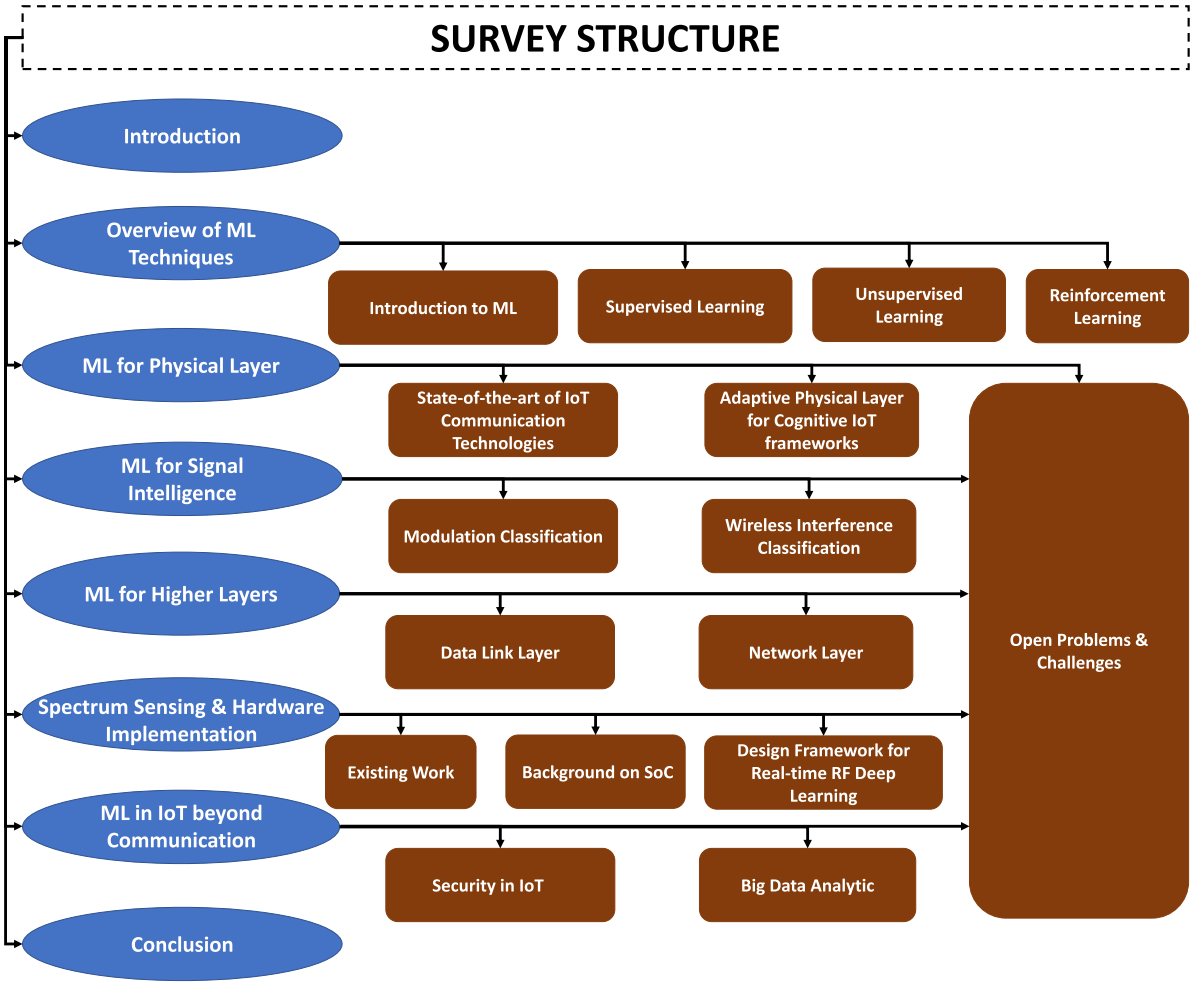

Fig. 1. Overall organization of the survey.

worldwide, generating more than 130 exabytes per month of wireless traffic [5]. Moreover, over 50 billion devices are expected to be in the IoT by 2020, which will generate a global network of "things" of dimensions never seen before [6]. Given that only a few radio spectrum bands are available to wireless carriers [7], technologies such as radio frequency (RF) spectrum sharing through beamforming [8–10], (DSA) [11–15] and anti-jamming technologies [16–18] will become essential in the near future. These technologies usually require coordination among wireless devices to optimize spectrum usage – often, they need to be implemented in a distributed manner to ensure scalability, reduce overhead and energy consumption. To address this challenge, machine learning machine learning (ML) has been widely recognized as the technology of choice for solving classification or regression problems for which no well-defined mathematical model exists.

The recent introduction of ML to wireless communications in the IoT has in part to do with the new-found pervasiveness of ML throughout the scientific community at large, and in part to do with the nature of the problems that arise in IoT wireless communications. With the advent of advances in computing power and ability to collect and store massive amounts of data, ML techniques have found their way into many different scientific domains in an attempt to put both of the aforementioned to good use. This concept is equally true in wireless communications. Additionally, problems that arise in wireless communication systems are frequently formulated as classification, detection, estimation, and optimization problems; for all of which ML techniques can provide elegant and practical solutions. In this context, the application of ML to wireless communications seems almost natural and presents a clear motivation [19–21].

The objective of this paper is to provide a detailed insight into the influence ML has had on the IoT and the broader context of wireless ad hoc networks wireless ad hoc networks (WANETs). Our hope is to elicit more research in the field to solve some of the key challenges of modern IoT communication systems. To begin, we provide an overview of the ML techniques in Section 2. In Sections 3 and 4, we discuss the applications of ML to physical layer to improve the communication and acquire signal intelligence respectively. Next, in Section 5, we discuss how ML has been exploited to advance protocol design at the data-link and network layers of the protocol stack. In Section 6, we discuss the implications of hardware implementations in the context of ML. Thereafter, in Section 7, we provide a brief discussion on the recent application of ML to IoT beyond wireless communication. Finally, the conclusion of this paper is provided in Section 8. The overall structure of the survey paper is depicted in Fig. 1.

2. Overview of machine learning techniques

Before we begin, we would like to introduce some standard notations that will be used throughout this paper. We use boldface upper and lower-case letters to denote matrices and column vectors, respectively. For a vector $\mathbf{x}$, x_i denotes the ith element, $||\mathbf{x}||$ indicates the Euclidean norm, $\mathbf{x}^\intercal$ its transpose, and $\mathbf{x} \cdot \mathbf{y}$ the Euclidean

inner product of **x** and **y**. For a matrix **H**, H_{ij} will indicate the (i,j)th element of **H**. The notation $\mathcal{R}$ and $\mathcal{C}$ will indicate the set of real and complex numbers, respectively. The notation $\mathbb{E}_{x\sim p(x)}[f(x)]$ is used to denote the expected value, or average of the function $f(x)$ where the random variable x is drawn from the distribution $p(x)$. When a probability distribution of a random variable, x, is conditioned on a set of parameters, $\boldsymbol{\theta}$, we write $p(x;\boldsymbol{\theta})$ to emphasize the fact that $\boldsymbol{\theta}$ parameterizes the distribution and reserve the typical conditional distribution notation, $p(x|y)$, for the distribution of the random variable x conditioned on the random variable y. We use the standard notation for operations on sets where $\cup$ and $\cap$ are the infix operators denoting the union and intersection of two sets, respectively. We use $S_k \subseteq S$ to say that S_k is either a strict subset of or equal to the set S and $x \in S$ to denote that x is an element of the set S. $\varnothing$ is used to denote the empty set and $|S|$ the cardinality of a set S. Lastly, the convolution operator is denoted as *.

All the notations used in this paper have been summarized in Table 1. The notations are divided into sections based on where they first appear and if they have been re-defined. Similarly, we also provide all the acronyms used in this paper in Table 2

2.1. Introduction to machine learning

The primary purpose of this section is to provide a brief overview of the field of ML itself as well as provide a fundamental description of the algorithms and techniques presented as solutions to the wireless communications problems introduced in subsequent sections. This section aims to be as rigorous as necessary to allow the reader to understand how the presented algorithms are applied to wireless communications problems but does not aim to give an all-encompassing, comprehensive survey of the field of ML. Interested readers are urged to refer to [22–24] for a comprehensive understanding of ML. The material presented in this section is given from a probabilistic perspective, as many of the concepts of ML are rooted in probability and information theory. The rest of Section 2.1 provides a kind of road map for Section 2 as a whole.

2.1.1. Taxonomy

Most introductory texts in ML split the field into two subdivisions: supervised learning and unsupervised learning. We follow suit and will make the distinction of which subdivision each presented algorithm falls under. As will be shown in later sections of this paper, many problems in WANET can be solved using an approach called reinforcement learning (RL). RL in its most fundamental form can be viewed as a third and separate subdivision of ML, thus we will denote representative algorithms as such. It is important to note that many advanced RL algorithms incorporate techniques from both supervised and unsupervised learning yet we will still denote these as RL algorithms.

Another common type of learning discussed in ML literature is that of deep learning (DL). We view DL techniques not as a separate subdivision of ML but as a means to achieve the ends associated with each of the three subdivisions stated above. DL typically refers to the use of a deep neural network (DNN), which we present with more rigor later in Section 2.2.4. Thus the "Deep" qualifier denotes an algorithm that employs a deep neural network to achieve the task. (ex: A deep reinforcement learning (DRL) algorithm would use a DNN in a RL framework)

2.1.2. A note on modularity

The concept of modularity is pervasive throughout engineering disciplines and is certainly prevalent in communications. We adopt this precedent throughout this text and present each of the algorithms using a common learning algorithm framework. This framework is primarily composed of the model, the optimization algorithm, the loss function, and a data set.

At its core, a learning algorithm is any algorithm that learns to accomplish some goal given some data to learn from. A common formalism of this definition is given in [25]: "A Computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks T, as measured by P, improves with experience E." While this definition of a learning algorithm is commonly agreed upon, formal definitions of a task, experience, and performance measure are less endemic within the ML community, thus we provide examples of each.

In the context of ML, tasks usually define some way of processing an object or data structure. A classification task is the process of assigning a class label to an input object or data structure. While different **examples** (objects) within the data set will give rise to different class labels, the task of assigning a given example a label is the same for the entire data set. Other examples of tasks addressed in this text include regression (assigning a real value to an example) and structured output (assigning a separate data structure, with a pre-defined form, to an example).

The performance measure, P, essentially defines the criteria by which we evaluate a given learning algorithm. In the case of classification, the performance is typically the accuracy of the algorithm, or how many examples the algorithm assigns the correct class label to divided by the total number of examples. It is common practice to divide the entire available data set into two separate data sets, one used for training the algorithm and one used to test the algorithm. The latter, called the test set, is kept entirely separate from the algorithm while training and is used to evaluate the trained algorithm. The performance measure is often a very important aspect of the learning algorithm as it will define the behavior of the system.

The experience, E, that a learning algorithm has while learning essentially characterizes the algorithm into one of the three subdivisions defined earlier. Supervised learning algorithms are provided with a data set that contains examples and their associated labels or targets. An unsupervised learning algorithm experiences data sets containing only examples and attempts to learn the properties of the data set. RL algorithms experience examples produced by the environment with which they interact. The environment often provides feedback to the RL algorithm along with examples.

2.2. Supervised learning

2.2.1. Overview

Recall from the previous discussion that in a supervised learning setting the learning algorithm experiences a data set containing examples and their respective labels or targets. An example will typically be denoted as x and its label, or target, as y. Together, we have training examples $(x,\ y) \in D$ existing in our data set D. In supervised learning problems, we attempt to learn to predict the label y from the example x, or equivalently, estimate the conditional distribution $p(y|x)$. Taking this approach, we will want to obtain a model of this conditional distribution and we will denote the parameters of such a model as $\boldsymbol{\theta}$. Assuming a set of i.i.d data $D = \{x_1, x_2, \ldots x_n\}$ drawn from the data generating distribution $p_{data}(x)$, the maximum likelihood estimator of the parameters, $\boldsymbol{\theta}$, of a model of the data generating distribution is given as,

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} p_{model}(D;\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \prod_{i=0}^{n} p_{model}(x_i;\boldsymbol{\theta}) \tag{1}$$

where p_{model} is a function space of probability distributions over the parameters $\boldsymbol{\theta}$. To make the above more computationally appealing, we can take the logarithm on both sides, as this does not

Table 1
Definition of notations.

Notations	Definitions
Section 2	
x, $\mathbf{x}$	Training example; vector
y, $\mathbf{y}$	Training target; vector
$\hat{y}$, $\hat{\mathbf{y}}$	Training target estimate; vector
D	Set of training data
θ, $\boldsymbol{\theta}$	General model parameter; vector
$k(\cdot, \cdot)$	Kernel function
$G(\cdot)$	Gini impurity
$H(\cdot)$	Entropy function
$\mathcal{L}(\cdot, \cdot, \cdot)$	Loss function
$\mathbf{w}$	Model weight vector
$\mathbf{W}$, $\mathbf{U}$, $\mathbf{V}$	Model weight matrix
b, w_0	Model bias term
$\mathbf{b}$, $\mathbf{c}$	Model bias vector
$\sigma(\cdot)$	Sigmoid activation function
K	Convolution kernel
I	Input image
$S(\cdot, \cdot)$	CNN feature map
L	Neural network layer
C_k	Cluster k
$\boldsymbol{\mu}_k$	Centroid of a cluster k
$d_j(\cdot)$	Discriminant function for a neuron j
$I(\mathbf{x})$	Index of minimum occurrence of discriminant function for $\mathbf{x}$
$T_{j,I(\mathbf{x})}$	Topological neighborhood function of $I(\mathbf{x})$ at neuron j
$S_{i,j}$	Distance from neuron i to neuron j
$\eta(t)$	Learning rate parameter; a function of time
γ	Reward discount parameter
$\gamma(\cdot)$	Reward discount parameter
S	State space
A	Action space
$P_a(\cdot, \cdot)$	State transition function
$R_a(\cdot, \cdot)$	Reward function
r	Observed reward
s	Observed state
a	Performed action
$q_\pi(\cdot, \cdot)$	Action-value function
Section 3	
$\mathfrak{s}_i$, $\mathfrak{s}_{-i}$	Strategy of player i and strategy of all players except i
$U_i(\mathfrak{s}_i, \mathfrak{s}_{-i})$	Utility dependent on $\mathfrak{s}_i$ and $\mathfrak{s}_{-i}$
$\mathbb{P}$	the set of players
$\mathfrak{S}_i$	the set of strategies of player i
p_i	Penalty of player i for inducing interference $\mathcal{I}_i(\mathfrak{s}_i, \mathfrak{s}_{-i})$ to other players
$V_{c,d}$	Value table for each channel device pair
η	Throughput learning rate of value table
$\mathcal{C}(\epsilon, \omega)$	Collision function which depends on exploration factor ϵ and other parameters ω.
C^*	Collision threshold
$L(\epsilon)$	Loss function
s_n	System State
g_n	Channel gain
b_n	Buffer occupancy
n	Index of the block
N	Maximum number of packets in the buffer
B	Size of the packet in bits
P_a	Poisson distribution where a is the number of packets arriving at the buffer
ν	Expected number of packets that will arrived in one block
p_n	Number of packets leaving the buffer in the n^{th} block
d_n	Number of packets dropped from the buffer in the n^{th} block
M	Number of constellation points
m_n	Bits per symbol in the n^{th} block
N_{sym}	Number of symbols in a block
N_0	Noise Spectral Density
ϵ_*	Acceptable BER threshold
P_n	Transmission power in the n^{th} block
$\bar{P}$	Long term average power consumption
$\mathfrak{T}$	System Throughput
$\mathfrak{P}_d$	Packet drop probability
r_n	Reward per block
$\mathfrak{r}(t), \mathfrak{r}(n)$	Continuous and discrete representations of received signal
$\mathfrak{a}(t)$	Modulated amplitude as a function of time t
$\phi(t)$	Modulated phase as a function of time t
$g(t)$	Additive white Gaussian noise as a function of time t
$\mathcal{A}(.), \mathcal{P}(.)$	Amplitude and Phase distortion functions
α_a, β_a, α_ϕ, β_ϕ	Scalar values representing channel parameters

(*continued on next page*)

Table 1 (*continued*)

Notations	Definitions
I_ρ	Information potential
$\mathfrak{G}_\sigma(.)$	Gaussian kernel with standard deviation σ
ρ	Entropy order
y_i	Adaptive system output
d_i	Desired system output
e_i	Error measure between actual and desired system output
L	Mean squared error loss
m_i	Transmitted symbol
$\mathfrak{r}_i$	Received symbol
μ_χ, var_χ	Mean and variance of mini-batch χ
$m(n)$	Discrete representation of baseband OFDM modulated signal
$M(k)$	Discrete frequency domain representation of $m(n)$
$R(k)$, $H(k)$,	Discrete frequency domain representation of received signal $\mathfrak{r}(n)$,
$G(k)$	channel response $h(n)$, and additive white Gaussian noise $g(n)$
$y_{i,e=(v,c)}$	Output of Neuron $e = (v, c)$ in the hidden layer i
z_v	Final v^{th} output of the DNN
i, d	Antenna element and antenna element spacing
a_k, θ_k, ϕ_k,	Amplitude, incident angle, initial phase, and
f_0	initial frequency of k^{th} incident signal
$\mathbf{R}(n), R_{mm'}$	Spatial correlation matrix and its respective diagonal element
$\mathbf{\Theta}$, $\mathbf{F}$	Incident angle matrix and hidden layer matrix
Section 4	
N_s	Number of samples
$\gamma_{\max}$	Maximum value of the power spectral density of the normalized centered-instantaneous amplitude
C_{lk}	l^{th} order, k^{th} conjugate cumulant
δ_0	Deviation of normalized amplitude from the unit circle
$\mathbf{x}_k^{IQ}$	$k^t h$ raw signal training example; I/Q representation
$\mathbf{x}_k^{A/\Phi}$	k^{th} raw signal training example; amplitude and phase representation
$\mathbf{x}_k^F$	k^{th} raw signal training example; frequency domain representation
$\mathbf{r}_k$	Received signal, vector form
r_{q_n}	Received signal quadrature value at index n
r_{i_n}	Received signal in-phase value at index n
$x(n)$	Transmitted signal, function of time
$y(n)$	Transmitted signal, function of time
Section 5	
N	Total number of nodes in the network
N_T	Total number of time slots
T	Set of time slots
SA	Slot assignment matrix
μ_{xi}	Fuzzy state, a degree that time slot t_x is assigned to node i
U	Fuzzy x-partition matrix
ρ	Channel utilization
$deg(i)$	Degree of edges incident to i
E	Energy function
α, β	Positive coefficients
f	Fuzzification parameter
d_{ij}	Parameters used to define connectivity between i and j
c_r	Collision rate
P_{req}	Packet request rate
t_w	Average packet wait time
p_t	Probability of an active DoS attack
Γ_{th}	Chosen threshold
t	Time slot
h	Channel number
$a_i(t)$	Node i's action at time slot t
R_i	Reward for the action
$\mathcal{T}$	Temperature
$z(t)$	Channel observation
$\mathfrak{h}$	State history length
EX_t	Set of experience samples at time t
ux	Upstream neighbor
$\mathcal{K}$	Set of nodes
$\mathcal{E}$	Set of unidirectional wireless link
$\mathcal{G}(\mathcal{K}, \mathcal{E})$	Directed connective graph
γ_{ij}	Score associated with edge (i, j)
l	Number of neurons
$\tilde{\delta}$	Normalized advance towards the sink
$\tilde{E}$	Normalized residual energy
R_C	Constant reward if the node is able to reach sink directly
R_D	Penalty suffered if no next-hop is found
R_E	Penalty if existing next-hop has residula energy below the threshold
ϵ	Probability of exploration

(*continued on next page*)

Table 1 (*continued*)

Notations	Definitions
P_{ij}^{j}	Transition probability
α_1, α_2, β_1, β_1	Tunable weights
c	Constant cost associated with consumption of resources like bandwidth, etc.
E_i^{res}	Residual energy
E_i^{ini}	Initial energy
E_i	Energy cost function associated with E_i^{res} and E_i^{ini}
$\bar{E}_i$	Average residual energy
D_i	Measure of the energy distribution balance
SK	Set of sinks
SK_p	Subset of sinks
$H_{SK_p}^{\mathcal{NB}}$	Routing information through all neighboring nodes in $\mathcal{NB}$

Table 2
Definition of acronyms.

Acronym	Meaning
3GPP	3rd Generation Partnership Project
5G	5th Generation
6LOWPAN	IPv6 over low power wireless personal area networks
A3C	Asynchronous advantage actor critic
AC	Actor–Critic
ACK	Acknowledgement
AM	Amplitude modulation
AMC	Automatic modulation classification
ANN	Artificial neural network
AP	Access point
ASIC	Application specific integrated circuit
AWGN	Additive white Gaussian noise
AXI	Advanced eXtensible Interface
BEP	Belief propagation
BER	Bit error rate
BLE	Bluetooth low energy
BP	Back-propagation
BPSK	Binary phase shift keying
BPTT	Back-propagation through time
BSP	Broadcast scheduling problem
BSSID	Basic service set identifier
CART	Classification and regression trees
CPFSK	Continuous phase frequency shift keying
CPU	Central processing unit
CR	Cognitive radio
CR-IoT	Cognitive radio-based IoT
CSMA	Carrier sense multiple access
CSMA/CA	Carrier sense multiple access/collision avoidance
CDMA	Code division multiple access
CE	Cognitive engine
CMAC	Cerebellar model articulation controller
CNN	Convolutional neural network
CR-VANET	Cognitive Radio-Vehicular Ad Hoc Networks
DARPA	Defense Advanced Research Projects Agency
DBN	Deep belief network
DBSCAN	Density-based Spatial Clustering of Applications with Noise
DCNN	Deep convolutional neural network
DCPC	Distributed constrained power control
DMA	direct memory access
DoA	Direction of arrival
DoS	Denial of service
DRL	Deep reinforcement learning
DSA	Dynamic spectrum access
DSB	Double-sideband modulation
DL	Deep learning
DLMA	Deep reinforcement learning multiple access
DNN	Deep neural network
DP	Dynamic programming
DQN	Deep Q-network
EAR	Energy-Aware Routing
EM	Expectation-Maximization
FDMA	Frequency division multiple access
FHNN	Fuzzy hopfield neural network
FIFO	First-in first-out
FPGA	Field-programmable gate array
FROMS	Feedback Routing for Optimizing Multiple Sinks
FSK	Frequency shift keying
GA	Genetic algorithm

(*continued on next page*)

Table 2 (*continued*)

Acronym	Meaning
GRU	Gated recurrent unit
GFSK	Gaussian frequency shift keying
GMM	Gaussian Mixture Model
GMSK	Gaussian minimum shift keying
GPSR	Greedy Perimeter Stateless Routing
HDL	Hardware description language
HLS	High-level synthesis
HMFPM	Hybrid QoS Multicast Routing Framework-Based Protocol for Wireless Mesh Network
HNN	Hopfield neural network
II	Initiation interval
IoT	Internet of things
IPC	Intelligent Power Control
I/Q	In-phase/quadrature
JQP	Join query packet
JRP	Join reply packet
LATA	Local Access and Transport Area
LANET	Visible light ad hoc network
LMR	Land Mobile Radio
LO	Local oscillator
LoRa	Long Range
LoRaWAN	Long Range Wide Area Network Protocol
LoS	Line of sight
LS	Least-squares
LSTM	Long short term memory
LTE	Long term evolution
LTE-A	Long term evolution-advanced
M2M	Machine-to-machine
MAC	Medium access control
MAP	Maximum a posteriori
MANET	Mobile ad hoc network
MIMO	Multiple input multiple output
MDP	Markov decision process
ML	Machine learning
MLP	Multi-layer perceptron
MMSE	Minimum mean square error
MST	Multi-stage training
M-QAM	M-ary quadrature amplitude modulation
MVDR	Minimum variance distortionless response
MUSIC	Multiple signal classification
NACK	Negative acknowledgement
NB-IoT	Narrowband IoT
NCNN	Noisy chaotic neural network
NDP	Node disconnection probability
NE	Nash equilibrium
NLP	Natural language processing
NOMA	Non-orthogonal multiple access
NSG	Non-cooperative strategic game
OFDM	Orthogonal frequency-division multiplexing
OSPF	Open shortest path first
PAM	Pulse-amplitude modulation
PCA	Principal component analysis
PL	Programmable logic
POMDP	Partially observable markov decision process
PS	Processing system
PSD	Power spectral density
PSK	Phase shift keying
PSO	Particle swarm optimization
PU	Primary user
QARC	Video Quality Aware Rate Control
QAM	Quadrature amplitude modulation
QoE	Quality of experience
QoS	Quality of service
QPSK	Quadrature phase shift keying
RAM	Random access memory
RBF	Radial basis function
RBFNN	Radial basis function neural network
RF	Radio frequency
RFID	Radio frequency identification
RL	Reinforcement learning
RLGR	Reinforcement Learning based Geographic Routing
RN	Residual network
RNN	Recurrent neural network
RSS	Received signal strength
RSSI	Received signal strength indication
SAG	Smart application gateway
SAX	Simple aggregation approximation

(*continued on next page*)

Table 2 (*continued*)

Acronym	Meaning
SC	Smart connectivity
SC2	Spectrum Collaboration Challenge
SC-FDE	Single carrier frequency domain equalization
SGD	Stochastic gradient descent
SIR	Sensor Intelligence Routing
SoC	System on chip
SOM	Self-organizing map
SNR	Signal-to-noise-ratio
SSB	Single-sideband modulation
SVC	Sequential vertex coloring
SVM	Support vector machine
SVR	Support vector regression
SU	Secondary user
TDMA	Time division multiple access
UAN	Underwater acoustic network
UF	Unrolling factor
UAV	Unmanned aerial vehicle
VANET	Vehicular ad hoc network
VQPN	Video quality prediction network
VQRL	Video quality reinforcement learning
WANET	Wireless ad hoc network
WASN	Wireless ad hoc sensor network
WBAN	Wireless body area networks
WBFM	Wideband Frequency Modulation
WIC	Wireless interference classification
WSN	Wireless sensor network

change the optimization problem, which gives us,

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \sum_{i=0}^{n} \log(p_{model}(x_i; \boldsymbol{\theta})) \tag{2}$$

Additionally, we can divide the right hand side of the equation by n, as this does not change the optimization problem either, and we obtain the expectation of the log-probability of the model over the empirical data generating distribution,

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{x\sim\hat{p}_{data}} \log(p_{model}(x_i; \boldsymbol{\theta})) \tag{3}$$

Alternatively, we could formulate the maximum likelihood estimation as the minimization of the KL divergence between the empirical data generating distribution and the model distribution given as,

$$D_{KL}(\hat{p}_{data}||p_{model}) = \mathbb{E}_{x\sim\hat{p}_{data}}[\log(\hat{p}_{data}(x)) - \log(p_{model}(x))] \tag{4}$$

Since the data generating distribution is not a function of the model, we can solve the same minimization problem by minimizing

$$-\mathbb{E}_{x\sim\hat{p}_{data}} \log(p_{model}(x)) \tag{5}$$

which is exactly equivalent to the maximization problem stated in the maximum likelihood formulation. The above is referred to as the negative log-likelihood of the model distribution and minimizing it results in the minimization of the cross-entropy between the data generating distribution and the model distribution. The significance of this is two-fold. Firstly, the terms cross entropy and negative log-likelihood are often used in literature to describe the loss functions that are being used to evaluate a given ML model and the above minimization problem is what is being referred to. Secondly, this gives rise to the narrative that the model associated with the maximum likelihood estimate is, in fact, the same model that most closely resembles the empirical data distribution. This is important considering what we want our model to do, namely, produce correct labels or targets for data drawn from the data generating distribution that the model has not seen before.

For completeness, the maximum likelihood estimator for the conditional distribution, which provides a label's probability given an example, is given as,

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \sum_{i=0}^{n} \log(p_{model}(y_i|x_i; \boldsymbol{\theta})) \tag{6}$$

for i.i.d examples, x_i.

Often times, regularization on the parameters of the model is desirable, as regularization can lead to better generalization of the model. This is most frequently seen in the different types of neural network models that will be described later in this section. Building on the maximum likelihood perspective of the loss function, we can show that adding a regularization function to our optimization function can be seen as inducing a prior over the model parameters and subsequently changing our estimator to the maximum a posteriori (MAP) point estimate. Inducing a prior probability on the model parameter results in the following optimization problem,

$$\boldsymbol{\theta}_{MAP} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|D) = \arg\max_{\boldsymbol{\theta}} \log(p(D; \boldsymbol{\theta})) + \log(p(\theta)) \tag{7}$$

Here, we have made use of Bayes' Rule, the properties of logarithm, and the fact that the optimization problem does not depend on the data generating distribution. If we wish to put a Gaussian prior on the parameters, $p(\boldsymbol{\theta}) \sim \mathcal{N}(0, \frac{1}{\lambda} I^2)$ we obtain a log prior proportional to $\lambda\boldsymbol{\theta}^T\boldsymbol{\theta}$, which yields the popular L2-Regularization scheme. Again. we have made use of the fact that the Gaussian prior does not depend on the data distribution and contains constants that do not affect the optimization problem. Thus, the L2-Regularizer can be seen as a cost associated with the magnitude of the model's parameters as well as the placement of a Gaussian prior on the model parameters.

2.2.2. Support vector machines

The support vector machine (SVM) was initially developed to perform the task of binary classification. Since their introduction into the ML community, SVMs have been successfully extended to perform regression and multi-class classification tasks as well. SVMs are non-parametric models, meaning that the number of parameters that compose the model is not fixed whilst constructing the model. In contrast, a parametric model would have a fixed

number of tunable parameters defined before constructing the model. We will first define the SVM in the context of linear regression and then expand upon extensions to the algorithm later in the section. It is important to note here the change in notation of the model parameter vector from $\boldsymbol{\theta}$ to $\mathbf{w}$. Throughout the remaining parts of this section, $\mathbf{w}$ is typically used when the literature surrounding the algorithm refers to the parameter vector as a *weight* vector and $\boldsymbol{\theta}$ for a general parameter vector. The decision to forgo notation uniformity was made in an attempt to keep our notation consistent with each algorithm's original presentation, making the text more accessible to readers who may already be familiar with some of the algorithms.

Linear regression is perhaps one of the most well known and prevalent linear predictive models known throughout the ML and statistical community. It is typically formulated as follows,

$$y_i = \mathbf{w}^T \mathbf{x_i} + w_0 \tag{8}$$

where y_i are the target values, $\mathbf{x_i}$ are individual training examples and weights, $\mathbf{w}$, are the model parameters. A common approach to solving such a problem is to vectorize the output and input variables and solve the normal equations, giving a closed form solution for the minimum mean square error (MMSE). A typical approach to adapt this algorithm to perform classification tasks is the well known logistic regression given as,

$$p(y = 1|\mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^T \mathbf{x}) \tag{9}$$

where σ is the logistic *sigmoid* function given as,

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{10}$$

One favorable property of logistic regression is that it has a well defined probabilistic interpretation that can be viewed as maximizing the likelihood of the conditional distribution $p(y|\mathbf{x})$. An alternative formulation for a linear classifier is given in what is known as the perceptron algorithm [26]. The perceptron algorithm aims to find a hyperplane in the input space that linearly separates inputs that correspond to different classes. It does so using a zero-one loss function, meaning that the model is penalized equally for every point in the training data that it classifies incorrectly. An obvious shortcoming is that the algorithm converges to any hyperplane that separates the data; it need not be the *optimal* hyperplane.

The linear SVM [27] attempts to find the hyperplane that best separates the data, where the optimal hyperplane maximizes the margin between the nearest points in each class on either side of the plane. While this solution is better, the true power of SVMs comes from the kernelization of the linear SVM, which allows the model to find nonlinear boundaries between different classes by representing the input data in a higher dimensional space. Kernelization of an algorithm is a process by which the parameters of the model are written in terms of a linear combination of the input vectors, which allows the computation of the inner product between a new input vector and the parameter vector of the model to be written as an inner product of the new input and the training inputs. A kernel function can then be substituted for the inner products between training vectors, which can be intuitively interpreted as a function that returns a real value representing the similarity between two vectors. The kernelization of the SVM leads to the kernel SVM [28]. The most common kernels used to kernelize SVMs are the linear, polynomial, and radial basis function (RBF) kernels, given as,

$$k(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{x_i}^T \mathbf{x_j}, \tag{11}$$

$$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i}^T \mathbf{x_j} + 1)^d, \text{ and} \tag{12}$$

$$k(\mathbf{x_i}, \mathbf{x_j}) = e^{-\frac{(\mathbf{x_i} - \mathbf{x_j})^2}{\sigma^2}} \tag{13}$$

respectively, where σ is a user defined parameter.

2.2.3. Decision trees

Decision trees can be employed for both the tasks of classification and regression. Decision tree algorithms are similar to nearest neighbor type algorithms in a sense that labels for examples lying near each other in input space should be similar; however, they offer a much lighter weight solution to these problems.

A decision tree is essentially nothing more than an aggregation of if conditions that allow a new example to traverse the tree. The tree is traversed until happening upon a leaf node, which would specify the output label. Decision trees can be constructed in a number of different ways, but a common approach is to create trees that minimize some measure of impurity while splitting the data. There are many such impurity measures but each of them essentially conveys how non-homogeneous the data in either child node would be if a given split of the data were to occur. A child node containing only training examples of the same label is referred to as a pure leaf and decision trees are often constructed to contain only pure leaves.

We now discuss two of the most popular impurity functions used in decision tree construction. We first define the training data as $D = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n)\}$, $y_i \in \{1, \ldots, c\}$ where c is the number of classes. Additionally, we have $D_k \subseteq D$ where $D_k = \{(\mathbf{x}, y) \in D : y = k\}$ and $D = D_1 \cup \ldots \cup D_c$. We then define the fraction of inputs in D with label k as,

$$p_k = \frac{|D_k|}{|D|} \tag{14}$$

and the Gini Impurity of a leaf node and a tree, respectively as,

$$G(D) = \sum_{k=1}^{c} p_k(1 - p_k), \text{ and} \tag{15}$$

$$G^T(D) = \frac{|D_L|}{|D|} G^T(D_L) + \frac{|D_R|}{|D|} G^T(D_R) \tag{16}$$

where $D = D_L \cup D_R$, $D_L \cap D_R = \varnothing$. The idea is then to choose splits in the tree that minimize this measure of impurity. Another popular impurity function is the entropy function. The entropy of the tree has its derivation in using the KL-divergence between the tree label distribution and the uniform distribution to determine how impure it is. Leaving the derivation to the interested reader, we define,

$$H(D) = -\sum_{k} p_k \log(p_k), \tag{17}$$

$$H^T(D) = \frac{|D_L|}{|D|} H^T(D_L) + \frac{|D_R|}{|D|} H^T(D_R) \tag{18}$$

as the entropy of a leaf and the tree respectively. While decision trees can be strong classifiers on their own, they often benefit from a technique called bagging. We omit the statistical derivation of the benefits of bagging and simply state the essence of bagging: by training many classifiers and considering the average output of the ensemble we can greatly reduce the variance of the overall ensemble classifier. Bagging is often done with decision trees as decision trees are not very robust to errors due to variance in the input data.

Perhaps the most popular bagged algorithm is that of the Random Forest. Random forests are bagged decision trees generated by the following procedure,

- Sample m datatsets $D_1, \ldots, D_m$ from D with replacement.
- For each D_i train a decision tree classifier $h_i(\cdot)$ to the maximum depth and when splitting the tree only consider a subset of features k.
- The ensemble classifier is then the mean output decision i.e. $h(\mathbf{x}) = \frac{1}{m} \sum_{i=1}^{m} h_i(\mathbf{x})$

The number of trees m can be set to any number, provided the computational resources are available. If d is the number of features in each training example, the parameter $k \leq d$ is typically set to $k = \sqrt{d}$.

2.2.4. Feedforward neural networks

The original formulation of feedforward neural networks was proposed in [29]. It can be seen as an extension to the previously mentioned perceptron algorithm with an element-wise nonlinear transition function applied to the linear classifier. This nonlinear transition function allows the hyperplane decision boundary to take a nonlinear form, allowing the model to separate training data that is not linearly separable. The formulation for a given layer, l, is as follows,

$$\mathbf{z}^l = \mathbf{W}^{(l)^T}\mathbf{a}^{l-1} + \mathbf{b}^l \tag{19}$$

$$\mathbf{a}^l = \sigma(\mathbf{z}^l) \tag{20}$$

where $\mathbf{a}^{l-1}$ are the outputs from the previous layer and may be referred to as the activation values of the previous layer. In the instance where the layer in question is the input layer, $\mathbf{a}^{l-1}$ would be set as $\mathbf{x}$, the training example input. The current layer's activation values are thus denoted as $\mathbf{a}^l$ and in the case of the output layer, these values would be synonymous with $\hat{\mathbf{y}}$. The layer weight matrix, $\mathbf{W}^{(l)T}$, consists of column weight vectors for each neuron in the layer and $\mathbf{b}^l$ is a column vector containing the bias term for each neuron. One common implementation approach to handling the bias term is to add an additional parameter to each of the weight vectors and append a 1 to the input vector. When a bias term is omitted this formulation can be assumed unless otherwise stated throughout the section.

The nonlinear transition function, σ, is also referred to as the activation function throughout literature and is often chosen from a handful of commonly used nonlinear functions for different applications. The most widely used activation functions are the following,

$$\sigma(z) = \frac{1}{1+e^{-z}}, \tag{21}$$

$$ReLU(z) = \max(0, z), \text{ and} \tag{22}$$

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{23}$$

Additionally, the RBF kernel function described earlier in Section 2.2.2 can be used as an activation function and doing so give rise to the radial basis function neural network (RBFNN) [30]. To increase the complexity of the model, and thus its ability to learn more complex relationships between the input features, network layers can be subsequently added to the model that accept the previous layer's output as input. Doing so results in a DNN. The function of the network as a whole $\phi(\mathbf{x})$ thus becomes,

$$\phi(\mathbf{x}) = \mathbf{W}^{(3)}\sigma(\mathbf{W}^{(2)}\sigma(\mathbf{W}^{(1)}\mathbf{x})) \tag{24}$$

where the weight matrices $\mathbf{W}^{(i)}$ are indexed according to the layer they belong to. Intuitively, this allows the first layer to learn linear functions between the input features, the second layer to learn nonlinear combinations of these functions, and the third layer to learn increasingly more complex nonlinear combinations of these functions. This formulation additionally gives rise to a nice graphical interpretation of the model, which is widely used in literature and given in Fig. 2.

This graphical interpretation is also where the feedforward neural network gets its loose biological interpretation. Each solid line in Fig. 2 denotes a weighted connection in the graph. The input, output, and hidden layers are denoted as such in the graph and a close up of one node in the graph is provided. This close up calls the single node a neuron, but it can equivalently be referred to simply as a unit in this text and throughout literature. The close up also shows the inputs to the neuron, the weighted connections from the previous layer, the weighted sum of inputs, and the activation value, denoted as a_i^{l-1}, w_{ik}^l, z_k^l, and a_k^l, respectively. Occasionally, a neuron employing a given activation function may be referred to as such a unit in this text and throughout literature, i.e. a unit with a *ReLU* activation function may be called a "*ReLU* unit".

The most common way to train neural networks is by way of the stochastic gradient descent (SGD) optimization algorithm. SGD is similar to well-known gradient descent methods with the exception that the true gradient of the loss function with respect to the model parameters is not used to update the parameters. Usually, the gradient is computed using the loss with respect to a single training example or some subset of the entire training set, which is typically referred to as a mini-batch, resulting in mini-batch SGD. This results in the updates of the network following a noisy gradient, which in fact, often helps the learning process of the network by being able to avoid convergence on local minima which are prevalent in the non-convex loss landscapes of neural networks. The standard approach to applying SGD to the model parameters is through the repeated application of the chain rule of derivation using the famous back-propagation algorithm [31].

The last layer in any given neural network is called the output layer. The output layer differs from the inner layers in that the choice of the activation function used in the output layer is tightly coupled with the selection of the loss function and the desired

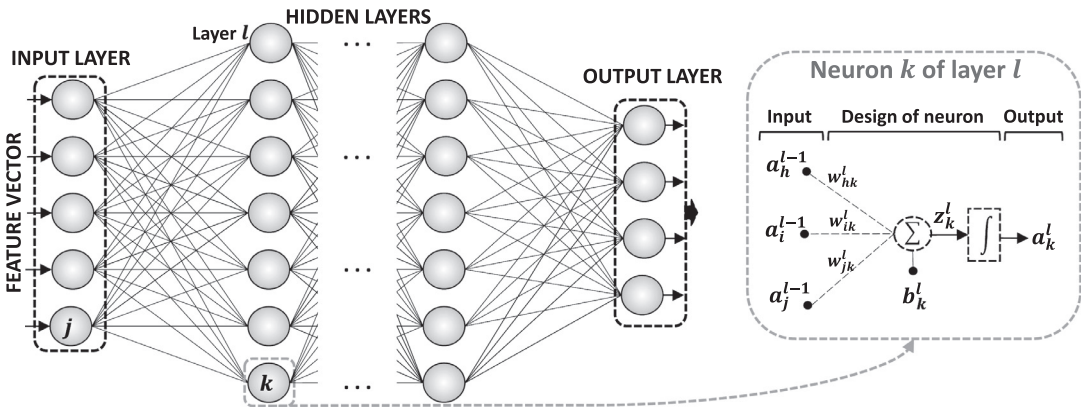

Fig. 2. Standard framework of feed forward neural network.

structure of the output of the network. Generally, the following discussion of output layers and loss functions applies to all neural networks, including the ones introduced later in this section.

Perhaps the simplest of output unit activation functions is that of the linear output function. It takes the following form,

$$\hat{\mathbf{y}} = \mathbf{W}^T\mathbf{a} + \mathbf{b} \tag{25}$$

where $\mathbf{W}$ is the output layer weight matrix, $\mathbf{a}$ are the latent features given by the activation output from the previous layer, and $\hat{\mathbf{y}}$ are the estimated output targets. Coupling a linear output activation function with a mean squared error loss function results in the maximizing the log-likelihood of the following conditional distribution,

$$p(\mathbf{y}|\mathbf{x}) = N(\mathbf{y}; \hat{\mathbf{y}}, I) \tag{26}$$

Another task that we have already touched upon in our discussion of SVMs and perceptrons is that of binary classification. In a binary classification task, the output target assumes one of two values and thus can be characterized by a Bernoulli distribution, $p(y = 1|\mathbf{x})$. Since the output of a purely linear layer has a range over the entire real line, we motivate the use of a function that "squashes" the output to lie in the interval [0,1], thus obtaining a proper probability. We have seen that the logistic *sigmoid* does exactly this and it is in fact the preferred method to obtain a Bernoulli output distribution. Accordingly, the output layer becomes,

$$\hat{y} = \sigma(\mathbf{w}^T\mathbf{a} + \mathbf{b}) \tag{27}$$

The negative log-likelihood loss function, used for maximum likelihood estimation, of the above output layer is given as,

$$\mathcal{L}(\mathbf{y}, \mathbf{x}, \mathbf{w}) = -\log(p(\mathbf{y}|\mathbf{x}; \mathbf{w})) = f((1 - 2\mathbf{y})\mathbf{z}) \tag{28}$$

where $f(x) = \log(1 + e^x)$ is called the *softplus* function and $\mathbf{z} = \mathbf{w}^T\mathbf{x} + \mathbf{b}$ is called the activation value. The derivation of (28) is not provided here but can be found in [22] for the interested reader.

For a multi-class classification task, the desirable output distribution is that of the Multinoulli distribution. The Multinoulli distribution assigns to each class the probability that a particular example belongs to it, requiring the sum over class probabilities for a single example be equal to 1. The Multinoulli distribution is given as the conditional distribution: $\hat{y}_i = p(y = i|\mathbf{x})$. It is important to note that the output, $\hat{\mathbf{y}}$, is now an n-dimensional vector containing the probability that $\mathbf{x}$ belongs to class $i \in [0, n]$ at each index i in the output vector. The targets for such a classification task are often encoded as an n-dimensional vector containing $(n - 1)$ number of0's and a single 1, located at an index j which denotes that the associated training example belongs to the class j. This type of target vector is commonly referred to as a one-hot vector. The output function that achieves the Multinoulli distribution in the maximum likelihood setting is called the *softmax* function and is given as,

$$softmax(\mathbf{z})_i = \frac{e^{\mathbf{z}}}{\sum_j e^{z_j}} \tag{29}$$

where z_j is the linear activation at an output unit j. *Softmax* output units are almost exclusively coupled with a negative log-likelihood loss function. Not only does this give rise to the maximum likelihood estimate for the Multinoulli output distribution but the log in the loss function is able to undo the exponential in the *softmax* which keeps the output units from saturating and allows the gradient to be well-behaved, allowing learning to proceed [22].

2.2.5. Convolutional neural networks

The convolutional neural network (CNN) was originally introduced in [32] as a means to handle grid-like input data more efficiently. The input of this type could be in the form of a time-series but is more typically found as image-based input. The formulation of CNNs additionally has biological underpinnings related to the human visual cortex.

CNNs are very similar to the feedforward networks introduced previously with the exception that they use a convolution operation in place of a matrix multiplication in the computation of a unit's activation value. In this section, we assume the reader is familiar with the concept of the convolution operation on two continuous functions, where one function, the input function, is convolved with the convolution kernel. The primary differences from the aforementioned notion of convolution and convolution in the CNN setting are that the convolution operation is discretized (for practical implementation purposes) and that it is often truly the cross-correlation operation that is performed in CNNs rather than true convolution. This means that the kernel is not typically flipped before convolving it with the input function. This is also primarily done for practical implementation purposes and does not typically affect the efficacy of the CNN in practice.

Convolution in the context of CNNs is thus defined as the following, for an input image I,

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n) \tag{30}$$

where K is the convolution kernel and the output, S, is often referred to as the feature map throughout literature. It is important to note that the above formulation is for two-dimensional convolution but can be extended to input data of different dimensions. The entries of K can be seen as analogues of the weight parameters described previously (Section 2.2.4) and can be learned in a similar manner using SGD and the back-propagation (BP) algorithm. Intuitively, one can imagine having multiple K kernels in a single CNN layer being analogous to having multiple neurons in a single feedforward neural network layer. The output feature maps will be grid-like and subsequent convolutional layers can be applied to these feature maps after the element-wise application of one of the aforementioned nonlinear activation functions.

In addition to convolutional layers, CNNs often employ a separate kind of layer called pooling layers. The primary purpose of a pooling layer is to replace the output of the network at a certain location with a summarization of the outputs within a local neighborhood in the grid. Examples of pooling layers include max pooling [33], average pooling, L^2 norm pooling, and distance weighted average pooling. A max pooling layer would summarize some rectangular region of the input image by selecting only the maximum activation value present in the region as output from the pooling layer. Pooling layers improve the efficacy of CNNs in a few different ways. First, they help make the learned representation of the input invariant to small translations, which is useful when aiming to determine the presence of a feature in the input rather than its location. Second, pooling layers help condense the size of the network since convolutional layers don't inherently do so. A binary classification task taking image data with size $256 \times 256 \times 3$ will need to reduce the size of the net to a single output neuron to make use of the output layer and cost function pairs described previously in Section 2.2.4. Lastly, pooling layers lead to infinitely strong prior distributions making the CNN more statistically efficient [22]. A pictorial representation of a single convolutional layer followed by a pooling layer is given in Fig. 3. The figure depicts a single convolutional layer applied to an input image of a waterfall plot of electroencephalogram data followed by a pooling layer. Subsequent convolutional layers may follow the pooling layer in a deep convolutional neural network (DCNN), and a nonlinear activation function may be applied to $S(i, j)$ prior to the pooling operation.

Some common adaptations applied to CNNs come in the form of allowing information flow to skip certain layers within the network. While the following adaptions were demonstrated on CNNs

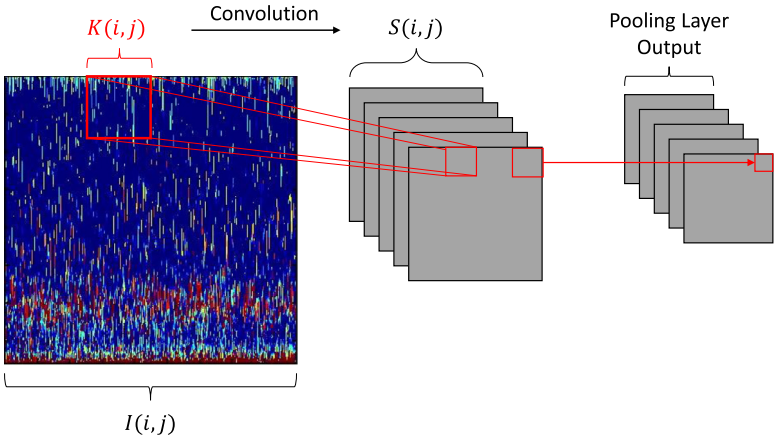

Fig. 3. Convolutional and pooling layers of a CNN.

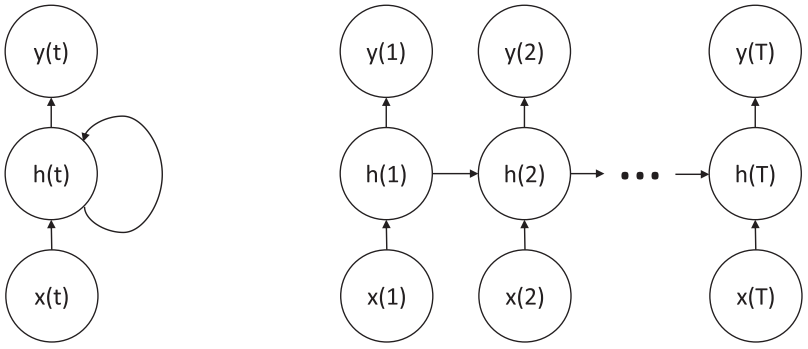

Fig. 4. Equivalent graphical formulations for Recurrent Neural Networks.

and long short term memorys (LSTMs) (a type of recurrent neural network (RNN)), the concepts can be applied to any of the networks presented in this paper. A residual network (RN), or ResNet [34], is a neural network which contains a connection from the output of a layer, say L_{i-2}, to the input of the layer L_i. This connection allows the activation of the L_{i-2} to skip over the layer L_{i-1} such that a "residual function" is learned from layer L_{i-2} to layer L_i. A highway neural network [35] is similar in that it allows a skip connection over layers but additionally applies weights and activation functions to these connections. Lastly, a dense neural network [36] is a network that employs such weighted connections between each layer and all of its subsequent layers. The motivation behind each of these techniques is similar in that they attempt to mitigate learning problems associated with vanishing gradients [37]. For each of these networks, the BP algorithm used must be augmented to incorporate the flow of error over these connections.

2.2.6. Recurrent neural networks

The RNN was first introduced in [31] as a way to handle the processing of sequential data. These types of neural networks are similar to CNNs in the sense that they make use of parameter sharing; however, in RNNs, parameters are shared across time steps or indices in the sequential input. Recurrent nets get their name from the fact that they have recurrent connections between hidden units. We denote this mathematically as follows,

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \boldsymbol{\theta}) \tag{31}$$

where the function f could be considered the activation output of a single unit, $\mathbf{h}^{(i)}$ are called the state of the hidden units at a time i, $\mathbf{x}^{(i)}$ is the input from the sequence at the index i, and $\boldsymbol{\theta}$ are the weight parameters of the network. Note, $\boldsymbol{\theta}$ is not indexed by i, signifying that the same network parameters are used to compute the activation at all indices in the input sequence. Output layers and loss functions appropriate for the desired task are then applied to the hidden unit state $\mathbf{h}$.

Two equivalent graphical representations of RNNs are provided as reference in Fig. 4. The left representation shows the network "rolled up" with a recurrent connection onto itself. The right representation shows the network "unrolled" with the recurrent connections now propagating information forward in time. We now provide the forward propagation equations for the hidden unit and use the *softmax* output layer as an example of how the hidden state would be used as input to the output layer. A loss function can then be applied to the *softmax* output as previously discussed in the paper.

$$\mathbf{a}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b} \tag{32}$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)}) \tag{33}$$

$$\mathbf{o}^{(t)} = \mathbf{V}\mathbf{h}^{(t)} + \mathbf{c} \tag{34}$$

$$\hat{\mathbf{y}}^{(t)} = softmax(\mathbf{o}^{(t)}) \tag{35}$$

The matrices **W, U**, and **V** are the weight matrices shared across hidden units. They are used to weight the connections between hidden units from one time step to the next, between the input and hidden state at the current time step, and the hidden state and output at the current time step. The parameters **b** and **c** are bias term vectors that are shared across time steps.

The loss for a single sequential training example is accumulated over the entire sequence, thus using a negative log-likelihood loss for a sequence $\mathbf{x}^{(t)}$ with output targets $y^{(t)}$ the loss would be,

$$\mathcal{L}(\{\mathbf{x}^{(1)},\dots,\mathbf{x}^{(\tau)}\},\{y^{(1)},\dots,y^{(\tau)}\},\boldsymbol{\theta}) = -\sum_{t}\log(p_{model}(y^{(t)}|\{\mathbf{x}^{(1)},\dots,\mathbf{x}^{(t)}\};\boldsymbol{\theta})) \quad (36)$$

The computation for the gradient of the loss with respect to the model parameters is involved and is out of the scope of this paper. For the interested reader, SGD is commonly employed to train RNNs, employing the back-propagation through time (BPTT) [38] algorithm to compute the gradients.

Many extensions to the described RNN model exist and are worth mentioning. Perhaps the most obvious extension is to add more recurrent layers following the single recurrent layer that was described above, resulting in Deep RNNs [39]. This provides similar advantages that were discussed in the motivation for extending feedforward networks to multiple layers. Additionally, more recurrent connections can be added which may skip over time steps, skip over layers, or even move information backward in time resulting in bidirectional RNNs [40]. These additional recurrent connections would be weighted and a nonlinear activation function could be applied in the same manner that the basic recurrent connection operates.

The most prevalent extensions to the original RNNs are those of the LSTM and gated recurrent unit (GRU), developed originally in [41] and [42], respectively. LSTMs augment the traditional RNN framework by adding a self loop on the state of the network. This self loop is coupled with input, output, and forget gates which control whether input values are written to the state, the state values are forgotten within the state, or the state values are written to the output of the network, respectively. These adaptations allow the network to better "remember" relevant information over longer periods in time. Each of the gates is weighted and have a logistic *sigmoid* activation applied to them, allowing the network to learn how to best use these gates with respect to the task. GRUs operate in a similar fashion but instead use two gates, namely, the update and reset gates. The update gate controls to what degree the state of the network at the given time step is written back to the state variable as well as what parts of the new state to write to the current state. The reset gates control what parts of the current state to use in the next computation of the new state. Both the LSTM and GRU have the ability to retain information over longer time periods and aim to mitigate the negative learning mechanics associated with vanishing gradients.

Recurrent networks can also take forms that are significantly different from the models described above. In particular, a hopfield neural network (HNN) [43] is a special type of recurrent network formulated to recover corrupted patterns. Specifically, it is a recurrent network where each unit is connected to all other units in the graph except for itself. Additionally, the weight between units is shared and each unit in the network encodes a binary state value, typically either 1 or -1. This formulation aims to mimic the forms of associative memory present in human cognition models and is often trained using a form of Hebbian Learning [44]. The famous summarization of Hebbian learning, "cells that fire together wire together" drives the idea that when part of the pattern that the HNN is trained to recognize is present, all of the units associated with that pattern will "fire" and the entire pattern will be represented by the network. Another interesting difference from the previously described RNN structures is that the HNN does not make use of any type of training targets y. This makes the HNN a type of unsupervised learning algorithm, more of which we discuss in further detail in the next section.

2.3. Unsupervised learning

2.3.1. Overview

Unsupervised learning, a separate learning paradigm from the previous described supervised learning, attempts to learn useful properties of the training data rather than learning to map inputs to specific outputs. Examples of unsupervised learning tasks include probability density estimation, denoising, and clustering. Unsupervised learning algorithms only experience the training data examples and are given no target outputs, which are obviously preferable in scenarios when data sets are produced without targets and it would be impractical for a human to go through and label the data set with a target value. Thus, without targets, unsupervised learning algorithms usually try to present the data set in a simpler or easier to understand representation. This simpler representation most commonly manifests itself in the form of lower dimensional representations of data, sparse representations of data, and independent representations of the data.

While some unsupervised learning algorithms draw techniques from previously mentioned supervised learning algorithms, they employ different types of loss functions. Usually, the best types of loss functions to use in unsupervised learning settings will reward the algorithm for preserving information about the input data but penalize the algorithm for not representing the data in one of the three ways discussed in the previous paragraph. The reader may be familiar with the Principal component analysis (PCA) algorithm, which is a great example of a linear unsupervised learning algorithm that aims to decorrelate the input data.

2.3.2. Clustering algorithms

Clustering algorithms are unsupervised learning algorithms that all share a similar goal of attempting to separate the input data set into some number of partitions, or clusters. The process by which these various algorithms group the data points into clusters is specific to each algorithm but is typically based on a metric which may be a function of distance to other data points, density of the surrounding data points, or fit to a probability distribution, among others. Once a clustering algorithm has grouped the input data into clusters, the algorithm is used to categorized new data points into one of the existing clusters. This categorization is computed using the same metric the algorithm initially used to construct the clusters. The primary shortcomings of clustering algorithms arise from the algorithm having a lack of specification about what similarities the clusters should represent in the data. Thus the algorithm may find some grouping of the input data that the designer did not intend for, rendering the resultant classifier ineffective. Next, a few common clustering algorithms are described in further detail.

Lloyd's Algorithm for k-means clustering. Lloyd's algorithm for k-means clustering was initially introduced in [45], and its presentation has since been proliferated to a multitude of sources. The algorithm itself was developed to obtain a solution to the k-means problem, which concerns finding k points (cluster centroids) in the input space which minimize the distance between each training vector and the nearest centroid. Formally the k-means problem is as follows. Given a training data set $D = \{\mathbf{x_1},\dots,\mathbf{x_n}\}$, $\mathbf{x_i} \in \mathcal{R}^d$ and an integer k, find k points $\boldsymbol{\mu_1},\dots,\boldsymbol{\mu_k} \in \mathcal{R}^d$ which minimize,

$$f = \sum_{\mathbf{x_i}\in D}\min_{j\in[k]}||\mathbf{x_i}-\boldsymbol{\mu_j}||^2 \quad (37)$$

Intuitively, minimizing the above expression will attempt to minimize the distance from any given training vector to the nearest cluster centroid. The algorithm developed to find the centroids, the set of $\boldsymbol{\mu_1},\dots,\boldsymbol{\mu_k}$, can be broken out into a two step algorithm that is repeatedly performed until additional iterations no longer further minimize the expression above. We introduce a time parameter t to show how the centroids, and the clusters, $C_1,\dots,C_k$

change as the algorithm progresses. For a random initialization of centroids $\boldsymbol{\mu_1}, \ldots, \boldsymbol{\mu_k}$ the first step, called the assignment step, is given as,

$$C_j^{(t)} = \left\{ \mathbf{x_i} : ||\mathbf{x_i} - \boldsymbol{\mu_j^{(t)}}||^2 \leq ||\mathbf{x_i} - \boldsymbol{\mu_m^{(t)}}||^2 \forall m, 1 \leq m \leq k \right\}, \quad \text{s.t. } C_1 \cap \ldots \cap C_k = \varnothing \tag{38}$$

The following step, called the update step, computes the centroids of the newly assigned clusters as follows,

$$\boldsymbol{\mu_j^{(t+1)}} = \frac{1}{|C_j^{(t)}|} \sum_{\mathbf{x_i} \in C_j^{(t)}} \mathbf{x_i} \tag{39}$$

The presented algorithm will converge once there are no further reassignments of any training vectors to new clusters. Once the algorithm is trained, inference is performed by computing the distance from a new input vector, $\mathbf{r}$, and associating it with cluster j according to,

$$\arg\min_j ||\mathbf{r} - \boldsymbol{\mu_j}||^2 \tag{40}$$

Gaussian Mixture Models (GMMs). Clustering using GMMs in conjunction with the Expectation-Maximization (EM) [24] algorithm is an example of a probability distribution based clustering algorithm and can be seen as an extension to k-means clustering algorithms that allow the clusters themselves to take on different shapes other than perfect circles. This ability is realized through modeling each cluster as a Gaussian distribution with parameterized mean and covariance, and the entire clustered data distribution as a weighted linear combination of Gaussian distributions called a Gaussian mixture. Given a training data set $D = \{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$, $\mathbf{x_i} \in \mathcal{R}^d$ and an integer K, model the distribution of a given data point $\mathbf{x}$ as,

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k}) \tag{41}$$

Where $0 \leq \pi_k \leq 1$, $\sum_k \pi_k = 1$, and $\boldsymbol{\mu_k} \in \mathcal{R}^d$, $\boldsymbol{\Sigma_k} \in \mathcal{R}^{d \times d}$ are the mean vector and covariance matrix of the kth Gaussian distribution in the mixture. Following the maximum likelihood approach introduced in the beginning of this section, the maximum likelihood estimate for the GMM parameters is given as follows,

$$\log(p(\mathbf{X} | \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \log \left[\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x_n} | \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k}) \right] \tag{42}$$

Where $\mathbf{X}$ is a matrix constructed from the concatenation of the input training vectors. By maximizing the log-likelihood function using the EM algorithm, we can obtain the optimal model parameters that give rise to Gaussian distributions that best describe the training input data. To do so we first define,

$$\gamma(z_k) = p(z_k = 1 | \mathbf{x}) = \frac{\pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x} | \boldsymbol{\mu_j}, \boldsymbol{\Sigma_j})} \tag{43}$$

Where $\mathbf{z} \in \mathcal{R}^K$ is a one-hot vector used to reference any one of the K Gaussian components within the mixture. Thus, $\gamma(z_k)$ as defined above can be interpreted as the probability that the kth component of describes the training vector $\mathbf{x}$ best. This formulation is useful for developing the EM algorithm for GMM. In order to perform the EM algorithm, we must first solve for the maximum likelihood estimates of each of the tunable parameters. Setting the derivatives of $\log(p(\mathbf{X} | \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ equal to 0, we obtain the following equations for each of the GMM parameters,

$$\boldsymbol{\mu_k} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(x_n) \tag{44}$$

$$\boldsymbol{\Sigma_k} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x_n} - \mu_{\mathbf{k}})(\mathbf{x_n} - \mu_{\mathbf{k}})^T \tag{45}$$

$$\pi_k = \frac{N_k}{N} \quad \text{where,} \tag{46}$$

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}) \tag{47}$$

Thus, in the expectation step of the EM algorithm, we compute (43) with the current model parameters; obtaining probabilities representing which component distribution best describes each input vector. In the maximization step, we compute (44)–(47) using the previously computed values of $\gamma(z_{nk})$. Doing so obtains an estimate of the distribution parameters for each component distribution that most likely describe each of the training vectors associated with that component. Iterating through both the expectation and maximization steps yields the EM algorithm. E and M steps are typically performed until the log-likelihood of the overall model increases only marginally in any given step.

There are a few well-known difficulties in fitting GMMs with the EM algorithm. Foremost, the log-likelihood function allows for singularities to arise, where one component attempts to describe a single training point. This will send the standard deviation parameter of that component to 0 which will cause the likelihood to tend to infinity. Such a situation can only be avoided by resetting the distribution parameters at fault before restarting the fitting process. The EM algorithm is also computationally expensive and typically needs to iterate many times before convergence occurs. To mitigate the computational requirements, the Lloyd's algorithm described earlier can be used to obtain a better initialization for the component distributions.

Density-based clustering. Density-based clustering algorithms aim to assign clusters to areas in the input training vector space that are particularly dense with respect to the areas around them. Additionally, such algorithms may mark points that lie in a low density area as outliers, not requiring them to belong to any cluster. One of the most popular density-based clustering algorithms is the Density-based Spatial Clustering of Applications with Noise (DBSCAN) algorithm, originally presented in [46]. The DBSCAN algorithm provides six definitions, from which the clusters of the training data set, $D = \{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$, are built. Two input parameters, ϵ and *minpts*, and a distance function are required to be provided to the algorithm by the designer. The usages of each are elucidated in the definitions given below:

- **Definition 1**: The ϵ-neighborhood, $N_\epsilon(\mathbf{x})$, of a training vector $\mathbf{x_i}$ is defined to be the set of all points whose distance from $\mathbf{x_i}$ is less than or equal to ϵ. i.e. $N_\epsilon(\mathbf{x_i}) = \{\mathbf{x_j} \in D | \text{dist}(\mathbf{x_i}, \mathbf{x_j}) \leq \epsilon\}$
- **Definition 2**: Given ϵ and *minpts*, $\mathbf{x_j}$ is directly density reachable from $\mathbf{x_i}$ if $\mathbf{x_j} \in N_\epsilon(\mathbf{x_i})$ and $|N_\epsilon(\mathbf{x_i})| \geq minpts$
- **Definition 3**: A training vector $\mathbf{x_j}$ is density reachable from $\mathbf{x_i}$ if $\exists \mathbf{x_i}, \ldots, \mathbf{x_j}$ such that $\mathbf{x_{k+1}}$ is directly density reachable from $\mathbf{x_k}$.
- **Definition 4**: $\mathbf{x_j}$ is density connected to $\mathbf{x_i}$ if $\exists \mathbf{x_k}$ such that $\mathbf{x_j}$ and $\mathbf{x_i}$ are density reachable from $\mathbf{x_k}$
- **Definition 5**: A set C such that $C \subset D$ and $C \neq \varnothing$, is a cluster if
  - $\forall \mathbf{x_i}, \mathbf{x_j}$: if $\mathbf{x_i} \in C$ and $\mathbf{x_j}$ is density reachable from $\mathbf{x_i}$ then $\mathbf{x_j} \in C$
  - $\forall \mathbf{x_i}, \mathbf{x_j} \in C$: $\mathbf{x_i}$ is density connected to $\mathbf{x_j}$
- **Definition 6**: For clusters $C_1, \ldots, C_k$ of D, noise $= \{\mathbf{x_i} \in D | \forall j : \mathbf{x_i} \notin C_j\}$

The algorithm for finding clusters within the training data set is as follows. First, an initial random training vector is selected from the training data, $\mathbf{x_i}$, and all points within the ϵ-neighborhood of $\mathbf{x_i}$ are retrieved. If $|N_\epsilon(\mathbf{x_i})|$ is less than *minpts* the vector $\mathbf{x_i}$ is added to

the noise set. If $|N_\epsilon(\mathbf{x_i})|$ is greater than or equal to *minpts*, (at least *minpts* number of training examples are directly density reachable from $\mathbf{x_i}$) all points in $|N_\epsilon(\mathbf{x_i})|$ are added to the current cluster index set. Using this initial set, all points that are density reachable from $\mathbf{x_i}$ are then retrieved and added to the current cluster index set. The algorithm then increments the cluster index and repeats the preceding process selecting a new initial point in the training set that has not been associated with either the noise set or any cluster set.

The primary advantage of the DBSCAN algorithm is that the number of clusters need not be specified by the designer of the algorithm. Additionally, there are no constraints on the shape of any given cluster, as is the case implicitly with both Lloyd's algorithm and GMM clustering. DBSCAN also incorporates a noise set, allowing the clusters to be robust to outliers. A disadvantage of the DBSCAN algorithm arises when clusters in the data have very different densities, making it difficult to select the appropriate values for ϵ and *minpts*.

2.3.3. Autoencoders

Autoencoders were first introduced in [47] and have a similar structure to DNNs in that they have an input layer, an output layer, and at least one hidden layer, often called the code layer. Autoencoders, while similar in structure to supervised neural network models, are like other unsupervised learning methods in that they attempt to learn a mapping from the input data to a latent representation that exhibits unique characteristics useful for performing some task. Such latent representations are often learned for the purpose of dimensionality reduction and de-noising; however, in either case, the formulation of the autoencoder splits the model into two parts: the encoder and the decoder. The encoder, usually denoted as f, takes the input data and maps it to a latent representation, or code, $\mathbf{h}$, such that $\mathbf{h} = f(\mathbf{x})$. The decoder, g, then attempts to reconstruct the original input data from latent representation. The training signal for the autoencoder model is thus computed using a loss function assuming the following form,

$$\mathcal{L}(\mathbf{x}, g(f(\mathbf{x})), \boldsymbol{\theta}) \tag{48}$$

and may be any function penalizing the dissimilarity between the two arguments. Such a function will force the encoder to learn a latent representation from which the original input data can be reconstructed by the decoder. While the loss function above necessitates the output layer of the decoder to be the same size as the input layer of the encoder, the code layer of the autoencoder is often smaller than the input and output layers. Such is the case of autoencoders used for dimensionality reduction or feature learning; a diagram of such an autoencoder structure is provided in Fig. 5. This

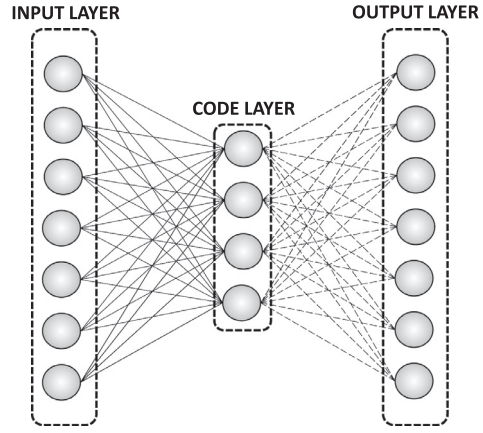

Fig. 5. Generalstructure of an autoencoder used for dimensionality reduction.

ensures that the code learned by the encoder contains only the most salient information of the data distribution that still allows for reconstruction. In dimensionality reduction and feature learning autoencoders, the decoder becomes inert after the model has been trained and only the encoder portion of the model is used to perform the task.

In denoising autoencoder models, the loss function is augmented such that a corrupted version of the input data is given to the encoder, and the loss is computed using the original input and decoder output. For original input, $\mathbf{x}$, and corrupted version, $\tilde{\mathbf{x}}$, the resulting denoising autoencoder loss function is given as,

$$\mathcal{L}(\mathbf{x}, g(f(\tilde{\mathbf{x}})), \boldsymbol{\theta}) \tag{49}$$

The corrupted version of the input data is typically sampled from some corruption process such that each corrupted data is not corrupted in the same way. Unlike dimensionality reduction autoencoders, after the denoising autoencoder model is trained the entire model is kept and used to perform the task.

2.3.4. Self organizing maps

The self-organizing map (SOM) [48] was originally introduced as a type of unsupervised learning algorithm with the goal of performing dimensionality reduction and data clustering. The reader may be familiar with the simple clustering algorithm referred to as k-means clustering, covered in this text in Section 2.3.2, in which each example in the training data is required to belong to one of k different clusters. The obvious pitfall of this algorithm is that the designer of the algorithm must choose the parameter k prior to constructing the model, hence the model's usefulness is contingent on the user's estimate of the appropriate number of clusters. The SOM algorithm avoids this by learning the appropriate number of clusters. Additionally, the SOM algorithm typically aims to represent the training data as a two-dimensional grid, where examples that are near each other in the input topological space are embedded near each other in the two-dimensional latent representation.

The canonical SOM formulation can be viewed as a fully connected single layer feedforward neural network, with units arranged in a two-dimensional grid. As the network sees each input, it computes the similarity between the input vector and each unit in the grid using some discriminant function such as,

$$d_j(\mathbf{x}) = \sum_{i=1}^{N} (x_i - w_{ji})^2 \tag{50}$$

where $d_j(\mathbf{x})$ is the value of the discriminant function at unit j, $\mathbf{w_j}$ is the weight vector associated with unit j, and $i \in [1, N]$ indexes the N dimensional input and weight vectors. This is often called the competitive process of SOMs as it is representative of a type of learning called competitive learning.

Once the discriminant function is computed at each unit for a training example the unit with the least value for the discriminant function is selected for what is called the cooperative process of SOM. The cooperative process attempts to update the neurons in some local neighborhood around the neuron that provides the closest representation of the input vector (i.e. the neuron with the minimal discriminant function). This creates neighborhoods in the map that will activate similarly for similar input values, thus creating clusters within the map. The topological neighborhood is usually defined as,

$$T_{j,I(\mathbf{x})} = \exp\left(\frac{-S_{j,I(\mathbf{x})}^2}{2\sigma^2}\right) \tag{51}$$

where $I(\mathbf{x})$ represents the index in the map where the minimal discriminant function occurred and $S_{j,i}$ denotes the distance from a neuron j to a neuron i. σ is a parameter chosen by the designer

and is typically decayed over time using the following schedule for time-dependence,

$$\sigma_t = \sigma_0 \exp\left(\frac{-t}{\tau_\sigma}\right) \tag{52}$$

Once the topological neighborhood is computed, the weight vectors associated with the units in the neighborhood are updated. This is usually referred to as the adaptive process in the context of SOMs. The change applied to the weight vectors is given as,

$$\delta w_{ji} = \eta(t) T_{j,I(\mathbf{x})}(t)(x_i - w_{ji}) \tag{53}$$

where $\eta(t)$ is the learning rate parameter and is also decayed over time using a similar schedule to that of the σ parameter,

$$\eta_t = \eta_0 \exp\left(\frac{-t}{\tau_\eta}\right) \tag{54}$$

This process is repeated many times for each training example in the training data set, resulting in the SOM.

2.4. Reinforcement learning

2.4.1. Overview

RL is a learning paradigm that can be considered separate from supervised and unsupervised learning. That being said, RL techniques often use ideas and algorithms from both unsupervised and supervised learning. We first describe the problem formulation for RL and then present a solution and how it can be extended to include concepts from other learning paradigms [49].

RL is built on the idea of an agent performing actions within an environment, based on its observations of the environment. The agent generally carries out actions according to a policy, which defines how the agent behaves at a given time. The agent receives reward signals, which define the ultimate goal of the algorithm, from the environment which indicates how well off the agent is at the time step the reward is given. The agent then aims to maximize its cumulative reward by observing its environment and the reward signal received, and then performing actions based on these inputs. The maximization of the cumulative reward is typically defined in terms of a value function. The value function differs from the reward signal in that the reward represents what is a desirable immediate setting and the value function represents how much reward the agent can obtain in the future given the agent's current state. Additionally, RL problems typically define a model of the environment. The model is estimated by the agent to determine the dynamics of the environment and is then subsequently used by the agent to devise some sort of plan about how to act.

RL problems, as described above, are usually formalized mathematically using finite markov decision process (MDP). The tuple (S, A, $P_a(\cdot,\cdot)$, $R_a(\cdot,\cdot)$) defines the dynamics of the MDP as well as the state and action spaces, S and A. At a given time step, an agent observes a state s, chooses an action a, receives a reward r, and transitions to a new state s'. The functions $P_a(\cdot,\cdot)$ and $R_a(\cdot,\cdot)$ define the transition probabilities between states and reward received from the environment when transitioning to a new state. The transition probability function takes the current state, s, and a possible new state, s' and outputs the probability of transitioning to that new state, conditioned on an action, a. i.e.,

$$P_a(s, s') = Pr(S_{t+1} = s' | S_t = s, A_t = a) \tag{55}$$

R is reward function such that it gives the reward obtained directly after transitioning to state s' from state s via action a and is defined as,

$$R_a(s, s') = E[R_{t+1} | S_t = s, A_t = a] \tag{56}$$

A policy is a function which defines how the agent will act given the state it is currently in. The policy is usually denoted as $\pi(a|s)$. Using such a policy, the agent moves about the environment and can start to construct a value function and action-value function based on the return they observe. The action-value function, q, for a policy, π is given as,

$$q_\pi(s, a) = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \tag{57}$$

where R_t are the observed returns over time and γ is a scaling parameter that is used to weight future returns less heavily than immediate returns. The action-value function can be plainly stated as the expected return starting in a state s, taking the action a, and subsequently following the policy π. Obtaining values for state action pairs allows for the agent to plan how to act in its environment. Equipped with the optimal action-value function, the solution to the MDP is merely choosing the action with the greatest action value.

2.4.2. Q-Learning

The method of Q-Learning was introduced in [50] and is what is called an off-policy control algorithm. The off-policy qualifier simply denotes that the algorithm does not depend on the policy the agent uses to navigate the environment. The Q-Learning is algorithm is defined by the following update rule,

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right] \tag{58}$$

Using such an update scheme for action-value pairs will lead to the approximation of the optimal action-value function independent of the policy being followed.

As one would imagine, the state and action spaces of some RL problem become extremely vast, making the storage of the action-value function for all state-action pairs impractical. One way to overcome this issue is to introduce a function approximation method which learns to provide values for state-action pairs. This function approximator could be one of the different types of ML algorithms discussed previously in this section. Such is the case in the famous deep Q-network (DQN) [51], where a deep convolutional neural network was used to approximate the action-value function when learning to play Atari games.

2.4.3. REINFORCE

In the previously presented Q-Learning algorithm, the agent moves about the environment according to some predetermined policy so that it may learn to accurately approximate the action-value function for the state and action spaces belonging to the environment. After learning the action-value function, the agent then navigates through the environment by selecting the actions that map to the greatest action-value function in the given state. The REINFORCE algorithm is inherently different, in that it attempts to learn the optimal policy directly and is thus characterized as a policy gradient method. This distinction specifies that the training signal is in fact a gradient with respect to the parameterized policy function and that the algorithm makes use of the policy gradient theorem [49], given as,

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla \pi(a|s; \boldsymbol{\theta}) \tag{59}$$

where $J(\boldsymbol{\theta})$ is a performance measure usually defined as some function of cumulative reward or reward rate, $\boldsymbol{\theta}$ is the policy parameterization vector, and $\mu(s)$ is a distribution over states which denotes the probability of being in any given state.

From the policy gradient theorem, we wish to obtain an expression that specifies exactly how the policy parameters are updated. To do so, we need an expression that provides information about

how the performance measure is affected by performing a specific action, A_t, in a specific state, S_t. Augmenting the policy gradient theorem to allow the parameter update to be computed for every action taken at every state requires the distribution weighted sum to be replaced by the expectation under the policy π of the gradient [52]. Doing so results in,

$$\nabla J(\boldsymbol{\theta}) = E_\pi\left[\sum_a q_\pi(S_t,a)\nabla\pi(a|S_t;\boldsymbol{\theta})\right] \tag{60}$$

$$= E_\pi\left[\sum_a \pi(a|S_t;\boldsymbol{\theta}) q_\pi(S_t,a)\frac{\nabla\pi(a|S_t;\boldsymbol{\theta})}{\pi(a|S_t;\boldsymbol{\theta})}\right] \tag{61}$$

$$= E_\pi\left[q_\pi(S_t,A_t)\frac{\nabla\pi(A_t|S_t;\boldsymbol{\theta})}{\pi(A_t|S_t;\boldsymbol{\theta})}\right] \tag{62}$$

$$= E_\pi\left[G_t\frac{\nabla\pi(A_t|S_t;\boldsymbol{\theta})}{\pi(A_t|S_t;\boldsymbol{\theta})}\right] \tag{63}$$

Where G_t is the cumulative reward at time t. This gives rise to the policy parameter update,

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t\frac{\nabla\pi(A_t|S_t;\boldsymbol{\theta})}{\pi(A_t|S_t;\boldsymbol{\theta})} \tag{64}$$

Where α is a step size parameter. Intuitively, such an update moves the parameter vector in a direction that increases the probability of taking action A_t in state S_t, proportional to the return received for doing so, normalized by the probability of choosing that action. The above algorithm formulation allows for the policy function to be any differentiable function approximator, which is often one of the neural network structures previously describes in this section. Coupling a neural network with a *softmax* output function additionally provides the output as a distribution, which is desirable as the policy at a given state should be a distribution over actions.

2.4.4. Actor–critic methods

Actor–Critic methods [49] are policy gradient methods that learn a state-value function in addition to the learned policy. The actor, a differentiable function approximator for the policy, learns the optimal policy in a similar fashion described in the REINFORCE algorithm with exception that an eligibility trace is used to update the policy parameters allowing for online learning. The critic should be a differentiable state-value function approximator and also learns using eligibility traces, thus allowing the entire algorithm to learn online.

An eligibility trace vector, $\mathbf{z}$, is a simple way of accumulating parameters that need updating over some time. For an actor policy, $\pi(A|S;\boldsymbol{\theta})$, parameterized by $\boldsymbol{\theta}$, and a critic action-value function, $\hat{v}(S;\mathbf{w})$, parameterized by $\mathbf{w}$, the respective eligibility trace vector updates for the parameters are given as,

$$\mathbf{z}^{\mathbf{w}} \leftarrow \gamma\lambda^{\mathbf{w}}\mathbf{z}^{\mathbf{w}} + \nabla\hat{v}(S;\mathbf{w}) \tag{65}$$

$$\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \gamma\lambda^{\boldsymbol{\theta}}\mathbf{z}^{\boldsymbol{\theta}} + \nabla\ln(A|S;\boldsymbol{\theta}) \tag{66}$$

Where $\lambda^{\mathbf{w}}$ and $\lambda^{\boldsymbol{\theta}}$ are trace decay parameters, and γ is discounting parameter. For episodic actor–critic methods, eligibility trace vectors should be initialized to a zero-vector at the start of each episode. Accordingly, for each time step in the episode an action A_t is sampled from the policy approximator and taken in state S_t, the agent moves to a new state, S'_t, and is given reward R_t. Thus, the parameter updates for the episodic actor–critic algorithm are given for each time step as follows,

$$\delta \leftarrow R_t + \gamma\hat{v}(S'_t;\mathbf{w}) - \hat{v}(S_t;\mathbf{w}) \tag{67}$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta\mathbf{z}^{\mathbf{w}} \tag{68}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\delta\mathbf{z}^{\boldsymbol{\theta}} \tag{69}$$

Where $\alpha^{\mathbf{w}}$ and $\alpha^{\boldsymbol{\theta}}$ are parameter space step sizes for each function approximator. Again, both function approximators for actor–critic methods can be implemented with any differentiable model described within this section and is often some neural network structure. In [49] pseudocode for actor–critic methods can be found along with their extensions to continuous RL problems and problems with continuous action spaces.

3. Machine learning for physical layer

3.1. State-of-the-art of IoT communication technologies

IoT is a broad, emerging trend and hence applies to several key modern concepts that employ several technologies. In fact, 5th Generation (5G) and IoT complement each other in that 5G wireless networks will catalyze the growth of future IoT systems. Achieving the IoT vision has been a subject of extensive research to identify and standardize the communication protocols, ubiquitous connectivity, data storage, computation and analytics, IoT gateway and cloud management, dynamic service delivery, among others [53,54]. The capabilities offered by IoT are countless and find vast applications to improve the economic and social well-being of humans such as smart home, smart lighting systems, smart healthcare, assisted driving, environmental monitoring, mobile ticketing, etc. IoT enables interconnection of various heterogeneous devices which communicate with each other without human intervention in what is known as machine-to-machine (M2M) communication [55]. The limitless possibilities of IoT through Massive and Critical IoT will influence several aspects of everyday life. Massive IoT involves the large deployment of smart devices in smart agricultural monitoring, smart grid, smart surveillance systems, smart home, etc. which require *low-cost user equipment, low energy consumption, and scalability for massive deployment*. Critical IoT, on the other hand, applies to critical operations such as remote healthcare monitoring, smart traffic surveillance, smart industrial operations which requires *low latency, highly reliable and safe end-user experience*. Such large deployments as in Fig. 6 generate an enormous amount of sensed data and requires seamless communication with each other and to the cloud. A critical consequence of such large deployments is *spectrum congestion* which can hinder and prevent the seamless interconnected operation as intended for the IoT applications. The large data generated from these devices require *high-speed connection* to the cloud while interaction among the devices involving control signaling can be satisfied by *low-speed wireless links*. Further, IoT devices are resource-constrained in terms of available energy and computational resources. Consequently, a fundamental requirement for IoT applications is the low power operation such that the deployed devices need not be replaced frequently. There have been several standardization efforts to support emerging IoT communication. Few of these are Zigbee [56], IPv6 over low power wireless personal area networks (6LOWPAN) [57], RPL routing protocol, bluetooth low energy (BLE) [58], EPCGlobal [59], WirelessHart [60], ISA100.11a [61], MiWi [62], Long Range Wide Area Network Protocol (LoRaWAN), narrowband IoT (NB-IoT), enhanced-Machine Type Communications, and Extended Coverage-Global System for Mobile Communications for IoT. Among these, Zigbee, 6LOWPAN, WirelessHart, ISA100.11a and MiWi employ IEEE 802.15.4 Physical and medium access control (MAC) layers while LoRaWAN adopts the Long Range (LoRa) physical layer. The communication protocols at various layers of the IoT protocol stack are shown in Fig. 7. These standards portray the shared interest and

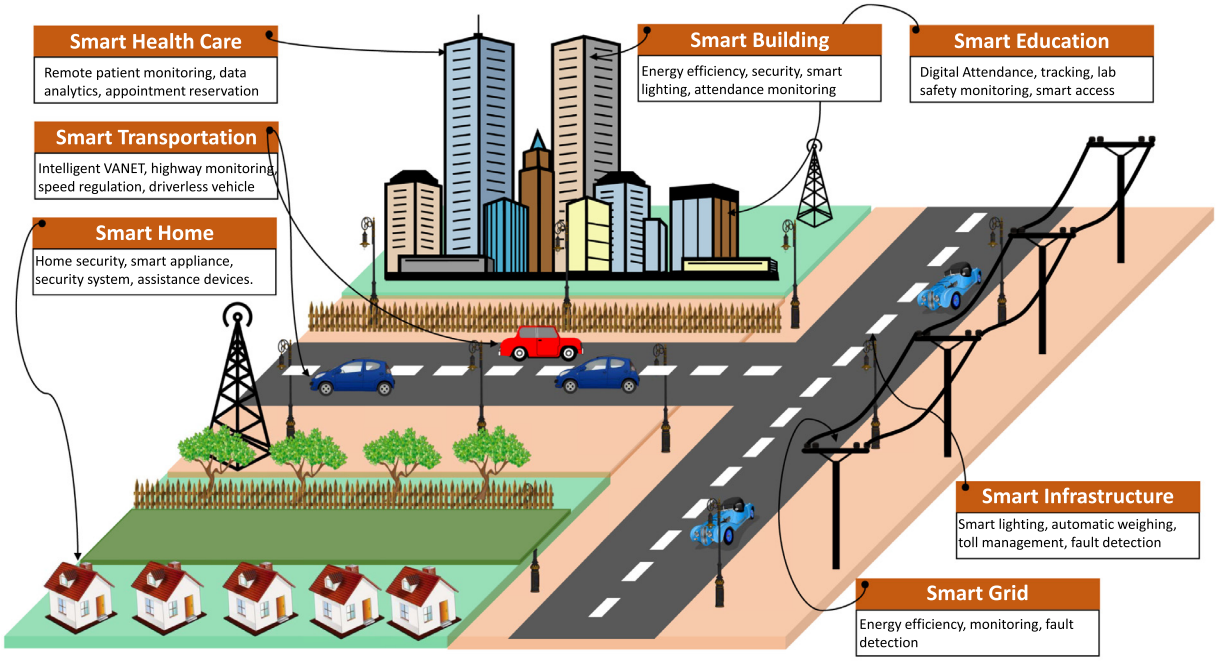

Fig. 6. IoT network enabling smart city.

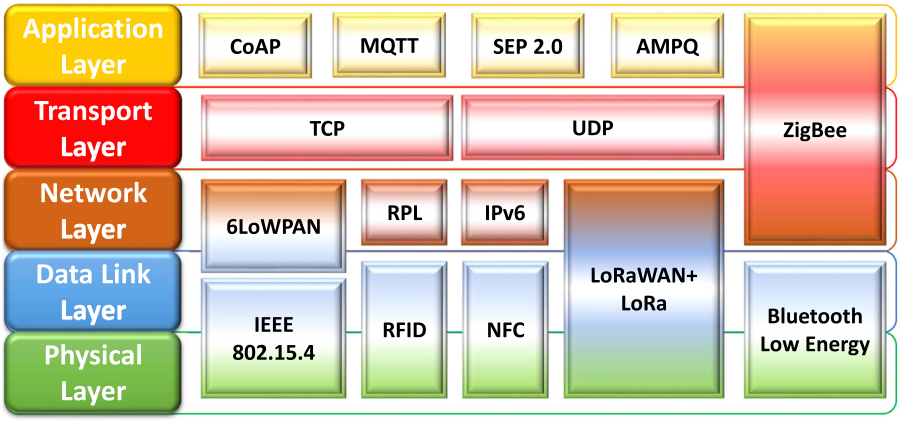

Fig. 7. IoT protocol stack.

vision shared by standardization institutions and interest groups around the world in realizing the IoT vision.

The Ericsson report for massive IoT [63] project the number of connected smart devices around the world will reach 28 billion by 2021. Such a surging number of devices pose a significant constraint on the wireless communication capacity of current and future deployments. The current static spectrum utilization policies lead to inefficient use of spectrum [53,64]. Several research works have been conducted in this regard to demonstrate the benefits of dynamic spectrum sensing, opportunistic spectrum access, and cooperative communications [13,65–69]. Such studied interactions between devices with strategic spectrum access methodologies introduce cognitive radio (CR) networks. Realizing the extent of capabilities that can be achieved with cognition, a new paradigm termed cognitive IoT has been introduced. Such cognitive radio-based IoT (CR-IoT) systems has been studied by [53,55,64,70–73]. There are several ongoing standardization efforts to incorporate CR techniques for IoT communication such as ETSI Reconfigurable Radio systems [74], ECMA-392 [75], IEEE 802.22b [76], IEEE 802.11af [77]. They allow dynamic spectrum sensing, spectrum access, and spectrum management. ECMA-392 is a cross-layer scheme that interfaces the MAC and physical layers and enables wireless home and business networks to dynamically use TV white spaces. The CR aspect will have wide applications in disaster response and management, wireless body area networks (WBAN), smart-healthcare facilities, vehicular networks, smart grid, among others. In this regard, [78] discusses the challenges and requirements in realizing Cognitive Radio-Vehicular Ad Hoc Networks (CR-VANET). The authors of [79] explored the applicability of ML techniques and proposed a learning architecture for CR-VANET. The CR based smart grid architectures has been studied in [80–82]. The works in [83–85] integrates CR to WBAN architectures. The work in [86] explores the potential benefits of incorporating CR in public safety and emergency response communications. The cognitive-IoT aspect must address several key issues to allow efficient communication between the devices, viz., 1. Resource-constrained IoT

devices, 2. Communication between heterogeneous hardware, 3. Dense deployments in confined space, 4. Interference between the devices, 5. Heterogeneous connectivity requirements, 6. Communication privacy and security, and 7. Large data management.

The authors of [71] presented the COGNICOM+ concept, a hybrid architecture that jointly use cognitive engine (CE) and smart connectivity (SC) to allow optimal use of local gateways and cloud computing. The authors present the software and hardware architecture required to support the COGNICOM+ concept. The envisioned IoT hybrid architecture houses the CE and SC in a smart application gateway (SAG) that is local to the connected devices. The CE is envisioned to employ compressed DL and game theory built on a CNN application specific integrated circuit (ASIC) accelerators. The authors introduce SAG to perform local computing unlike cloud and fog computing aiming to reduce latency and costs while improving capacity, scalability, privacy, and security. The SC module collects spectrum sensing data from the deployed devices which are relayed to the CE. The CE gathers the collaborative spectrum sensing data to detect unoccupied spectrum bands and dynamically access them. Such collaborative spectrum sensing and accessing maximize spectrum utility. The CE applies reasoning to make a strategic decision to maximize certain user-defined objective.

In game-theoretic sense, the authors consider each SAG as an agent in a multi-agent non-cooperative strategic game (NSG). Let the set of players (SAGs) be denoted as $\mathbb{P}$ and the strategy of player (i) be $\mathfrak{s}_i \in \mathfrak{S}_i$. Let $\mathfrak{s}_{-i}$ denote the strategy of all players except i. The strategy in the COGNICOM+ aspect refers to decisions on transmit power, data rate, accessible frequency bands, and interference to primary users. Each player has an associated utility $U_i(\mathfrak{s}_i, \mathfrak{s}_{-i})$ which is resultant of their own strategy and strategies of other players. The NSG can be expressed as $\mathcal{G} = \left\{\mathbb{P}, \{\mathfrak{S}_i\}_{i\in\mathbb{P}}, \{U_i(\mathfrak{s}_i, \mathfrak{s}_{-i})\}_{i\in\mathbb{P}}\right\}$. Now, if the players are operating in a greedy fashion, each player searches for the optimum strategy $\mathfrak{s}_i^*$ that maximizes their utility such that

$$\max_{\mathfrak{s}_i\in\mathfrak{S}_i} U_i(\mathfrak{s}_i, \mathfrak{s}_{-i}). \tag{70}$$

A fundamental concept in NSG is Nash equilibrium (NE) [87] where each player adopts their best possible strategy while being fully aware of the strategies of other players. In NE, neither player gains a unilateral incentive by deviating from the strategy. The authors propose to adopt a distributed optimization strategy in a more cooperative manner where each player optimizes its strategy to maximize their modified utility function,

$$\widetilde{U}_i(\mathfrak{s}_i, \mathfrak{s}_{-i}) \triangleq w_i U_i(\mathfrak{s}_i, \mathfrak{s}_{-i}) - p_i \mathcal{I}_i(\mathfrak{s}_i, \mathfrak{s}_{-i}), \tag{71}$$

where w_i represent the weights of player i and p_i is the penalty for inducing interference $\mathcal{I}_i(\mathfrak{s}_i, \mathfrak{s}_{-i})$ to other players. In this way, the collaborative operation of SAGs imparts a balance between the greedy maximization of self utility and the interference caused to other players.

The authors propose to use compressed DCNN in the CE. The CNN compression is achieved by weight/activation compression, model compression, and CNN computation acceleration in convolutional layers. Weight compression can be achieved by quantizing the pre-trained weights or quantizing during training process which significantly reduces the memory requirements. Additionally, input feature maps can be compressed by converting floating point to fixed point resulting in significant power and computational gains. Model compression is achieved by pruning less significant connection from the CNN. A similar strategy is employed in SqueezeNet [88] whereby smaller convolution filters (1×1, 3×3) are employed resulting in microarchitectures called Fire modules. The Fire modules are reconfigured by choosing between 1×1 and 3×3 filters forming larger CNN macroarchitectures. SqueezeNet has been shown to achieve accuracy comparable to AlexNet [89] with $50\times$ fewer samples and less than 0.5 MB model size ($\equiv 510\times$ smaller than AlexNet). Finally, CNN computation acceleration can be attained by compressing each convolutional layer by an equivalent low-rank approximations and adapting the upper layers until desired prediction performance is met.

The authors of [90] presented end-to-end dynamic spectrum management facilitated by IoT big data and ML algorithms. The authors propose the ML enabled IoT spectrum management system comprised of spectrum sensing and measurement collector, deep analytics for spectrum activity learning, and spectral reasoning and decision-making. The authors implemented the proposed spectrum management framework on the testbed [91] which connects to distributed sensors via IoT service covering frequencies 70 MHz to 6 GHz. The sensor management, data storage, ML decision-making are performed in the cloud. The Land Mobile Radio (LMR) band which ranges from 70 MHz to 1 GHz is considered in their experiment. The LMR band spans the very high frequency, ultra high frequency, and public safety channels. The incoming spectrum access requests could either be LMR service type or M2M applications. The spectrum sensing data from the deployed sensors contain measured energy levels in the LMR bands. The energy level above a preset level identifies as an occupied channel. The spectrum sensing data is processed in conjunction with the license database information to generate a channel occupancy time series for each channel every hour. The incoming spectrum sensing data is passed through usage characterization module which along with the candidate channel feature forecast module [92] generates candidate and training channels. The candidate channels have unused spectrum bands that can be shared with other users. The training channels represent all spectrum occupancy patterns of the incoming request. The candidate and training channels form the spectrum-sharing training dataset comprised of features and sharing labels. The sharing labels represent the sharing performance of the users such as the delay incurred by users and the channel loading with respect to the channel capacity. The channel is deemed to be overloaded if the loading label exceeds 1 and available to share otherwise. A sharing predictor is trained using the training dataset which assigns candidate channel and label (sharing performance). The authors used K-means clustering prior to training the predictor with the gradient boosting tree (XGBoost) [93] algorithm. The candidate channels are ranked for spectrum sharing for each incoming request as per the predicted sharing labels. Subsequently, the refining process improves the accuracy and robustness of the sharing label prediction of the ranked candidate channels before predicting the final match. The authors compared the predictor performance trained with XGBoost, random forest and SVM algorithms and demonstrated faster training speed and improved accuracy with XGBoost.

The work in [72] proposed a CR network architecture that employs multi-stage online learning techniques to perform spectrum assignment to IoT devices with an aim to improve their throughput and energy efficiency. The authors considered a IoT network with primary users (PUs) and secondary users (SUs). The PUs are the licensed users who have the primary right of accessing the channels while SUs can opportunistically access the channels as it becomes available. The PUs are categorized into idle and active states depending on whether they are actively transmitting. A collision can occur if a SU sensed the channel idle and starts transmission while the PU moves into an active state and starts transmission simultaneously. This happens as the PU has the exclusive right to the channel and will use it without sensing its availability. If a collision occurs the PU retransmits the data until successful transmission prior to switching back to an idle state. The authors considered a central node that has access to all the IoT devices in the network which will perform the channel assignment based on the channel

sensing data from them. The PU traffic model is considered to be either generalized Pareto or hyper-exponential while the SU traffic could be either event-driven, periodic, or high volume payload exchange. The proposed approach comprises a channel order selection for sensing, and OFF time prediction for each channel. The OFF time prediction allows the SUs to access the channels without sensing. The authors exploit the fact that the central node has access to all the IoT in the network and can gather the channel sensing data from them to model the traffic characteristics. The central node assigns one channel at a time to save energy consumed in sensing all channels. If the sensed channel is deemed available by the central node, the SU will access and proceed to transmission. If the transmission was a success, the corresponding throughput is returned to the central node else if a collision occurs it will inform the central node and switch to wait state. A value table ($V_{c,d}$) for each channel (c) and device (d) is maintained at the central node with their corresponding throughput ($\mathfrak{T}$). The value table is updated as,

$$V_{c,d} \leftarrow \eta\mathfrak{T} + (1-\eta)V_{c,d}, \tag{72}$$

where η is the learning rate that affects the priority given to the latest and past observations. The value table signifies the quality of each channel to each device. The authors adopt a hill climbing strategy to randomly swap some entries in the value table and recalculate the value of the resultant configuration. If the new channel-device configurations offer better quality compared to the previous, the new configuration is saved while discarding the previous. This swapping continues until there are no new configurations available that could improve the quality. The hill climbing will work only if the value table maintained at the central node is correctly estimated. However, this knowledge is unavailable initially and requires an exploration strategy to build the value table. Accordingly, an $\epsilon-$ greedy strategy is adopted to randomly explore different configurations for a fraction of time. Further, to predict the OFF time of the channels, the central node is required to learn the PU traffic distribution. Accordingly, a non-parametric Bayesian learning method is employed to perform online learning of the PU traffic distribution. Subsequently, a function $\mathcal{C}(\epsilon,\omega)$ representing the number of observed collisions which is dependent on the exploration factor ϵ and other factors ω is used. The objective is to achieve a value close to a predetermined threshold level (C^*) for $\mathcal{C}(\epsilon,\omega)$. In order to achieve this objective, a loss function $L(\epsilon) = loss(C^*, \mathcal{C}(\epsilon,\omega))$ is optimized using SGD. The gradient loss function with respect to ϵ is expressed as,

$$\frac{\partial L(\epsilon)}{\partial \epsilon} = \frac{\partial\ loss(C^*, \mathcal{C}(\epsilon,\omega))}{\partial \epsilon}\frac{\partial \mathcal{C}(\epsilon,\omega)}{\partial \epsilon}. \tag{73}$$

However, the functional relationship between $\mathcal{C}(\epsilon,\omega)$ and the parameters ϵ, ω are unknown. In order to circumvent this, the authors adopted a Simultaneous Perturbation Stochastic Approximation [94] that allows performing SGD while the functional relationship is unknown. The predicted OFF time allows the central node to assign skip period to the IoT devices enabling them to use the channel directly without sensing. The authors demonstrated using simulations that the proposed approach requires less channel sensing and achieve comparable throughput while not exceeding the collision threshold C^*.

3.2. Adaptive physical layer for cognitive IoT frameworks

The signal processing techniques that enable the physical layer functionalities have a direct impact on the data rate and sensitivity of the radio. With the increasing number of IoT devices that communicate over networks, some of which stream multimedia data, there is a growing need for high speed, low latency, and higher capacity systems. IoT devices are often deployed densely with several devices interconnected and communicating in the same spectrum posing severe constraints on bandwidth. To enable communication in such dense IoT networks, several challenges such as interference, power and bandwidth constraints come into play. Adaptive signal processing is a well-researched topic aimed around suppressing interference and noise from received attenuated signal samples by estimating the interference plus noise covariance from the received samples and suppressing their effect to improve the spectral efficiency of the system [95–97]. Another well-known approach to increase spectral efficiency is to adjust the modulation and coding scheme *on-the-fly* based on instantaneous channel conditions. The promising capabilities of multiple input multiple output (MIMO) systems to increase channel capacity has led to their adoption in wireless communication standards. Significant performance gain can be achieved by learning and estimating the varying channel dynamics and nullifying the channel' effect from the received signal samples to estimate the actual transmitted bits, in what is commonly known as adaptive channel equalization. Research surrounding the physical layer has historically been aimed at pushing the boundaries against the norm to provide increased agility to the radios, subsequently enhancing their performance. Enabling the radios with cognitive skills at the physical layer can revolutionize the wireless communication capability of the IoT devices. The ML based solutions can transform the IoT framework into cognitive IoT that can adaptively decide which actions are necessary to achieve a certain objective based on parameters learned by the system. This section will explore the various aspects of signal processing at the physical layer and how ML based solutions can offer a better alternative.

3.2.1. Adaptive rate and power control

RL based solutions have been extensively used in wireless communications to estimate the dynamic system model on the fly [98–106]. In the context of the physical layer, RL based solutions can extensively improve the system data rate, bit error rate, goodput (*i.e.*, the amount of useful information that successfully arrived at the destination over the time-varying channel) and energy efficiency [107–110]. Adaptive rate control can serve as a useful tool to selectively adapt the data rate depending on the instantaneous channel conditions. Such flexibility aids the system in leveraging the channel statistics to its benefit, essentially maximizing the channel utilization. IoT devices are often battery powered and hence constrained in power. Each layer of the protocol stack must be designed to reduce the energy consumption and prolong the device' lifetime. Therefore, adaptive power control at the physical layer is imperative to the longevity of the device and consequently the IoT network lifetime.

In [111], an adaptive rate control strategy based on RL is proposed to learn the dynamically varying channel conditions. The time-varying fading channel is modeled as a finite state Markov chain, whose channel state transition probabilities are unknown but the instantaneous channel gains can be estimated. Now the optimization problem forms a MDP which can be solved in dynamic programming (DP). However, the DP approach is suited best for static systems and hence would not be suitable for a dynamic system where the channel statistics vary with time.

In this work, the authors propose to use Q(λ)-learning [49] to track the varying environmental changes in pursuit of the optimal control policy online. Q(λ)-learning is a popular RL based algorithm used to solve MDP when the system's state transition probabilities are unknown. The Q(λ)-learning algorithm is similar to the standard Q-learning except that it updates the learning rate based on the Q value of the state-action pair. The incremental learning process involves the learning agent transitioning from system state of one block to another at the next block by choosing an action. For each chosen action, the agent observes the reward and

modifies its control policy with an aim to maximize the expected reward for future actions. This foresighted iterative learning process will repeat at each block and the agent will eventually converge at the optimal policy.

In this context, the objective is to find a rate-control scheme that maximizes the system throughput subject to a fixed bit error rate (BER) constraint and long-term average power constraint. The system state is characterized by the instantaneous channel gain and buffer occupancy as $s_n = \{g_n, b_n\}$. The receiver estimates the channel gain and feeds back to the transmitter. In a practical system, this could be accomplished by having the receiver and transmitter exchange estimated statistics via control packets using a separate control channel. Consider the transmission buffer is a first-in first-out (FIFO) queue that can hold a maximum of N packets each of size B bits. The packet arrival process to the buffer follows a Poisson distribution $P_a = \frac{\nu^a e^{-\nu}}{a!}$, where a is the number of packets that arrived at the buffer and ν is the expected number of packets that will arrive in one block. The number of packets dropped from buffer in the n^{th} block can be expressed as $d_n = \max[b_{n-1} - p_{n-1} + a_n - N, 0]$, where p_n is the number of packets leaving the buffer in the n^{th} block.

Consider an M-ary quadrature amplitude modulation (M-QAM) system which, based on the learning agent's rate-control policy, can change the number of bits per symbol ($\log_2(M)$). There are numerous ways to change a system's transmission rate; (i) vary coding rate, (ii) vary modulation scheme, i.e., constellation size, and (iii) careful combination of both. Let us denote the bits per symbol in the nth block as $m_n = \{1, 2, 3, \ldots, K\}$ and the number of symbols in a block as N_{sym}. Then, the number of packets that can be transmitted in the nth block is $p_n = \frac{m_n N_{sym}}{B}$, referred to herein as rate. For a W bandlimited system operating in an additive white gaussian noise environment with noise spectral density N_0, the minimum transmission power required to maintain an acceptable BER (ϵ_*) in the nth transmission block is,

$$P_n \geq \frac{WN_0}{g_n} \frac{(-\log 5\epsilon_*)(2^{p_n B/N_{sym}} - 1)}{1.5}. \tag{74}$$

Now, the long-term average power consumption can be expressed as,

$$\bar{P} = \lim_{n \to \infty} \frac{1}{n} \sum_{i=0}^{n} P_i. \tag{75}$$

The rate control scheme must now aim to maximize the system throughput ($\mathfrak{T} = \nu(1 - \mathfrak{P}_d)$) subject to the BER and average power constraints. Here, $\mathfrak{P}_d$ is the packet drop probability. This escalates to a dual objective optimization; maximizing system throughput and minimizing average power. This multi-objective optimization will be solved to arrive at a Pareto-optimal solution (rate control policy). Q(λ)-learning aims to find the optimal control policy by estimating an action-value function for each state-action pair. The action-value function is the long-term discounted reward if the system starts at state s_n taking an action p_n. The reward per block for taking an action/transmission rate p_n at a state s_n has a Lagrangian form which essentially implies the system gets a larger reward if the packet drops and transmission power is lower. The negative cost (reward) per block can be expressed as,

$$r_{n+1} = -[E(d_{n+1}) + \lambda P_n]. \tag{76}$$

The Q(λ)-learning can be solved in a way similar to the standard Q-learning except here the learning rate (ρ) is updated based on the state-action pair which is kept a constant in the standard Q-learning.

$$\rho = r_{n+1} + \gamma Q(s_{n+1}, p^*_{s_{n+1}}) - Q(s_n, p_n), \tag{77}$$

where γ is the discount factor and $p^*_{s_{n+1}}$ is the action which maximizes the action-value function $Q(s_{n+1}, p^*_{s_{n+1}})$. The Q($\lambda$)-learning demonstrates faster convergence compared to the standard Q-learning. The authors demonstrated the ability of learning agent to acclimate to the varying wireless channel to learn and adapt the rate control policy best suited for the channel conditions.

The authors of [107] attempt to solve the link adaptation problem of single carrier frequency domain equalization (SC-FDE) systems. SC-FDE systems use cyclically prefixed M-QAM to allow frequency domain equalization at the receiver. The authors approach the problem from a classification perspective such that the optimum modulation and coding scheme that would deliver the highest goodput for the current channel conditions would correspond to the best classification of the multidimensional data. The feature vectors considered include estimated post-processing signal-to-noise-ratio (SNR), estimated channel coefficients, and noise variance. PCA is used for dimensionality reduction such that an orthogonal transformation maps the features from a higher dimensional space to lower dimension. The kNN algorithm is used to classify the reduced dimensional feature vectors. A significant drawback of using kNN algorithm is that it requires storing the previously observed values which is memory intensive and computationally expensive. For a low power wireless device, such an algorithm is a poor choice for real-time operations [108]. tackle this problem to perform real-time link adaptation of MIMO-orthogonal frequency-division multiplexing (OFDM) systems by using online kernelized support vector regression (SVR). SVR attempts to minimize the generalization error bound to achieve generalized performance rather than minimizing training error like SVM. SVR requires minimal memory and computational power and was demonstrated to adapt quickly to varying channel conditions in their simulations. For every packet, the receiver observes the packet failure/success, channel measurements and the modulation and coding scheme corresponding to that packet. To prevent memory explosion, the authors use a sparsification algorithm [112] such that only linearly independent samples are preserved in the dictionary. The SVR algorithm finds the linear regression function that corresponds to the minimum mean squared loss function. The authors compared the performance of online kNN *versus* online SVR to demonstrate the monotonically increasing memory and time consumption with online kNN while it remained constant for online SVR.

A RL based solution is proposed in [109] to achieve adaptive rate and power control for point-to-point communication and extend it to a multi-node scenario. The receiver is assumed to feedback channel gain and packet success/fail status (acknowlegement (ACK)/negative acknowledgement (NACK)) to the transmitter allowing it to choose the modulation and transmitter power based on the obtained information. Accordingly, the authors formulate the objective to maximize the throughput per total consumed energy considering the channel conditions, queue backlog, modulation and transmit power. The authors incorporate buffer processing cost/energy into the total energy consumption cost such that there is a cost incurred for buffer overflows. Imposing buffer processing cost can be viewed as a quality of service (QoS) factor. The formulated MDP is solved using the Actor–Critic AC algorithm [49] which involves two parts: actor and critic. The actor decides the action and the critic estimates the state-value function and the error which criticizes the actor's action. The actor selects the action based on Gibbs *softmax* method [49] such that the action corresponding to the highest conditional probability of state-action is chosen. The authors demonstrated the throughput achieved with Actor–Critic AC algorithm is twice that of a simple policy where the highest modulation order that maintains a predefined link SNR is chosen.

Another notable application of ML in improving real-time video streaming is presented in [113]. The authors propose Video Quality Aware Rate Control (QARC), a DL based adaptive rate control

scheme to achieve high video quality and low latency. The complex challenge posed by the varying video quality and dynamic channel conditions is solved by two RL based models; a video quality prediction network (VQPN) and a video quality reinforcement learning (VQRL). The VQPN predicts future video quality based on previous video frames whereas the VQRL algorithm adopts the asynchronous advantage actor critic (A3C) RL [114] method to train the neural network. VQRL accepts historic network status and video quality predictions from VQPN as inputs.

The authors use a neural network in this video streaming application motivated by the effectiveness of the neural network in the prediction of time sequence data. The VQPN model adopts CNN layers to perform feature extraction of the input video frames to obtain spatial information. The CNN layers are followed by a two layered RNN which extracts temporal characteristics of the video frames in the past k sequences. The output of the VQPN is the prediction of video quality for the next time slot. The weights are updated based on the mean squared error loss function between the actual video quality score and the estimated video quality score. Specifically, the VQPN has 5 layers to perform feature extraction; a convolution layer with 64 filters each of size 5 with stride 1, a 3×3 average pooling layer, a second convolution layer with 64 filters of size 3 with stride 1, a 2×2 max-pooling layer and a hidden layer with 64 neurons. The output of the feature maps represents time series data which is fed into the RNN. The RNN comprises a GRU layer with 64 hidden units which then connects to another GRU layer of 64 hidden units. The hidden layer connects to the hidden output of the last GRU layer resulting in a 5-dimensional vector output corresponding to the video quality scores for the bit rates [300, 500, 800, 1100, 1400] kbps. The authors use Adam gradient optimizer to train the VQPN with a learning rate of 10^{-4}. The VQPN was realized using the open source ML library, TensorFlow [115].

In modeling the VQRL, the neural network must be trained to learn the relationship between the video quality and bit rate. The sender serves as a learning agent who observes the future video quality and previous network status in the state space. The network status in the state space is comprised of sender's video transmission bit rate, received bit rate of past k sequences, delay gradient, and packet loss ratio of previous k sequences. The action taken refers to the video bit rate selected for the next time slot. Since in this case the states will be represented by continuous numbers which leads to a fairly large state space, it is unable to store them in a tabular form. As Q-learning cannot be effective in solving large state space problems, the authors have combined RL with a neural network. The authors solve this RL problem using A3C RL algorithm [114] whereby the policy training is achieved by means of a policy gradient algorithm. The authors further propose a multiple-training agent version to accelerate the training process. The multiple agents comprise of a central agent and several forward propagation agents. The forward propagation agent only decides with policy and critic via state inputs and neural network model received by the central agent for each step. The central agent uses the actor–critic algorithm to compute gradients and then updates its neural network model which is then pushed to the forward propagation agent. This can happen asynchronously among all agents with no thread locking between agents. The VQPN is trained and tested on two video datasets; VideoSet (a large scale compressed video quality dataset) and self-collected video sets (live concerts, music videos, short movies). To train VQRL, the authors use packet-level, chunk-level, and synthetic network traces. The quality of experience (QoE) metric is defined as a weighted function of video quality, sender's bit rate and delay gradient measured by the receiver at a time instant n. The QARC algorithm was tested for its efficacy in real-world operating conditions by video-streaming on three different networks (public WiFi, Verizon cellular network and a wide area network between Shanghai and Boston) at a local coffee shop. The client was running on a MacBook Pro laptop connected to a server running on a Desktop machine located in Boston. The authors demonstrated the QARC algorithm outperform Skype and WebRTC in terms of the QoE metric.

Future IoT systems will involve a variety of traffic types ranging from bursty small packets, emergency low-latency transmissions, and high-rate multimedia traffic. Adaptive rate control strategies which intelligently respond to link quality, as well as traffic type, will be imperative for such systems.

3.2.2. Adaptive channel equalization

Another key area where ML algorithms, and more specifically neural network models, have been successfully employed to enhance the physical layer is adaptive channel equalization [116–125]. IoT networks are usually dense comprising of several devices attempting to communicate simultaneously. Such dense deployment with multiple transmissions results in a harsh communication environment. Channel equalization techniques must be employed at the receiver for efficient signal demodulation [124]. employs multi-layer perceptrons (MLPs) to perform non-linear channel equalization of a 16-QAM system. The use of non-linear power amplifiers result in non-linear amplitude and phase distortion resulting in a non-linear channel model as expressed by the following relation,

$$\mathfrak{r}(t) = \mathcal{A}(\mathfrak{a}(t))e^{j[\,\phi(t)+\mathcal{P}(\mathfrak{a}(t))\,]} + g(t), \tag{78}$$

such that $\mathcal{A}(x) = \frac{\alpha_a x}{1+\beta_a x^2}$ and $\mathcal{P}(x) = \frac{\alpha_\phi x}{1+\beta_\phi x^2}$ are the non-linear amplitude and phase distortions and $g(t)$ is the additive white Gaussian noise (AWGN).

The goal of non-linear channel equalization is to estimate the transmitted symbol from the received distorted symbols. The MLP is trained following a minimum error entropy criterion [126]. The adaptive system training aims to minimize/maximize the information potential based on the Renyi's entropy order. Fig. 8 shows an adaptive system learning to update its weights such that the difference (e_i) between the output (y_i) and desired response (d_i) is minimized. The weights (w) are trained based on the gradient of the information potential (I_ρ) as

$$\frac{\partial I_\rho}{\partial w} = \frac{\rho-1}{N^\rho}\sum_j\left(\left[\sum_i \mathfrak{G}_\sigma(e_j-e_i)\right]^{\rho-2}\sum_i \mathfrak{G}'_\sigma(e_j-e_i)\frac{\partial y_i}{\partial w}\frac{\partial y_j}{\partial w}\right), \tag{79}$$

where $\mathfrak{G}_\sigma(.)$ denotes the Gaussian kernel with standard deviation σ. The gradients of the outputs with respect to the weights can be computed using the standard BP algorithm. The proposed adaptive equalizer is composed of two MLPs operating in parallel, say MLP1 and MLP2. MLP1 is trained to learn the mapping of the transmitted signal amplitude $\mathfrak{a}_i, i = 1, 2, \ldots, N$ to the received signal amplitude $|\mathfrak{r}_i|$. Since for a 16-QAM, the transmitted signal amplitude can only have three different amplitude levels, the output of the MLP1 corresponding to the transmitted signal amplitude is compared to the measured $|\mathfrak{r}_i|$. The output that gives the closest estimate to the possible values is chosen as the estimate for transmitted signal amplitude. The MLP2 is trained to learn the mapping from received signal amplitude to the non-linear phase distortion. From the estimated amplitude and phase from MLP1 and MLP2, the in-phase and quadrature components of the transmitted symbol are determined. In this work, authors train the system for an entropy order ($\rho = 3$), steepest ascent for information potential, a Gaussian kernel with $\sigma = 1$ and a dynamic step size. The training initially starts with unitary step size which then updates depending on the weight update such that the value increases when the update yields better performance and vice-versa. The authors

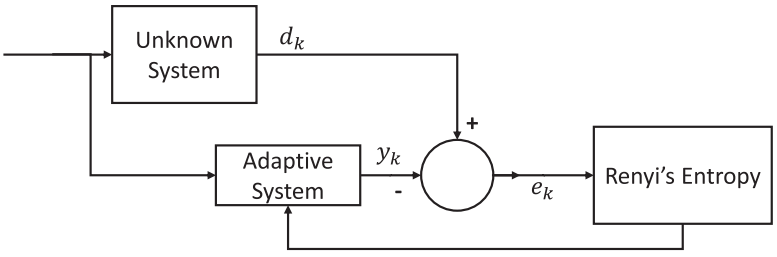

Fig. 8. Adaptive system.

demonstrated in simulations that the information potential maximization approach converges in fewer iterations than the mean squared error technique.

The authors of [125] explore the capabilities of DL for joint channel equalization and decoding. The DL model comprises of an increased number of hidden layers to improve the representation capability of the neural network. Similar to the [124], the channel is assumed to introduce non-linear distortion to the transmitted symbols. The authors train the network to minimize the mean squared error loss ($L = \frac{1}{N}\sum_i (\mathfrak{r}_i - m_i)^2$) between the transmitted symbol (m_i) and received symbol ($\mathfrak{r}_i$) as presented to the network in the training phase. The neuron weights in each layer can be updated using any gradient descent algorithms such as to minimize the loss function. The activation functions could be *ReLU* or *sigmoid* functions. The authors demonstrated the performance of the proposed DL for joint channel equalization and decoding with a six layer neural network comprising of 16, 256, 128, 64, 32 and 8 neurons in each layer. The modulation used is binary phase shift keying (BPSK) for a (16,8) polar code and *ReLU* activation function.

In [127], three DL models for channel decoding are proposed. In this book, we will cite the CNN model for the channel decoder. The CNN employs a convolution operation which significantly reduces the number of parameters allowing the network to be deeper with fewer parameters. The hidden layers use either convolution or pooling. The input to the CNN is batch-normalized [128] such that any layer that previously received input x will receive $BatchNorm(x)$ which is a normalized, scaled and shifted version of original input with respect to a mini-batch.

$$BatchNorm(x) = \theta_1 \hat{x} + \theta_2, \tag{80}$$

where $\hat{x} = \frac{x-\mu_\chi}{\sqrt{var_\chi + \epsilon}}$ is the normalized x over the mini-batch χ with mean (μ_χ) and variance (var_χ), θ_1 and θ_2 are the parameters to be learned.

The batch-normalized CNN is trained with mini-batch SGD to minimize the mean-squared error loss function. The authors observed the CNN decoder offered better performance compared to a MLP but at the cost of increased computational time.

The applicability of DL in channel estimation and signal detection in OFDM systems is demonstrated in [123]. The DL model is trained offline with simulated data to learn the channel distortions and reconstruct the transmitted symbols. Let $m(n)$ be the baseband OFDM modulated symbols transmitted over an N-path multipath channel $\{h(n)\}_{n=0}^{N-1}$ with AWGN $g(n)$ as shown by,

$$\mathfrak{r}(n) = m(n)h(n) + g(n). \tag{81}$$

After removing cyclic prefix and converting back to frequency domain the signal representation translates to,

$$R(k) = M(k)H(k) + G(k). \tag{82}$$

The pilot symbols are transmitted in the first OFDM block followed by user data in the subsequent blocks. The received pilot block and one data block are fed as input to the DL model. During the offline training stage, the model is trained with various received OFDM symbols generated with varying channel conditions under certain statistical profiles. The trained model when deployed for online signal detection, would estimate the signals without explicit channel estimation. The received signal and original transmitted symbols are supplied to the model to train it such that the difference between the model output and the original transmitted data are minimized. The model consists of five layers, three of which are hidden. Each layer comprises 256, 500, 250, 120 and 16 neurons respectively. The *ReLU* function is used as the activation function in all layers to map the input to the outputs of each layer except the last layer where *sigmoid* function is used to map to the interval [0,1].

A DL based method to improve the belief propagation (BEP) algorithm for decoding linear codes is proposed in [129]. BEP also known as Sum-Product algorithm is a message passing algorithm to derive statistical inferences from graphical models such as Bayesian networks. BEP is a form of Maximum A Posteriori (MAP) decoding of linear codes. BEP was first used in information theory by Gallager's iterative decoder for LDPC [130] which was a generalized case of belief propagation. Tanner graph forms a Bayesian network on which BEP operates. The DL model is trained with a single codeword. Conventional BEP decoder is constructed from the Tanner graph which is a graphical representation of the parity check matrix that describes the code [131]. The messages are transmitted over edges such that each edge calculates the outgoing message based on messages received over all its edges except for the transmitting edge.

To enable DL for a BEP decoder, the authors propose an alternative trellis representation where nodes in the hidden layer represent edges in the Tanner graph. If N denote the code block length, the number of neurons in the input layer is a vector of size N. The subsequent layers except for the final output layer i.e., the hidden layers have size E implying the number of edges in the Tanner graph. Each neuron in the hidden element corresponds to the message transmitted over some edge in the Tanner graph. The output layer has a size N that outputs the final decoded codeword. Let $e = (v, c)$ denote the neuron in the hidden layer $i, i \in 1, 2, \ldots, 2L$, l_v is the log-likelihood ratio of the variable node v and $y_{i,e}$ represent the output message from the neuron after $\lfloor \frac{i-1}{2} \rfloor$ iterations. To allow the DL model, to learn based on the inputs, they are assigned weights which will be updated using the SGD method. The output of a neuron in the hidden layer, for an odd i is expressed as,

$$y_{i,e=(v,c)} = \tanh\left(\frac{1}{2}\left(w_{i,v} l_v + \sum_{e'=(v,c'),c' \neq c} w_{i,e,e'} y_{i-1,e'}\right)\right) \tag{83}$$

and for an even i,

$$y_{i,e=(v,c)} = 2\tanh^{-1}\left(\prod_{e'=(v',c),v' \neq v} y_{i-1,e'}\right) \tag{84}$$

and the final vth output of the network is expressed as

$$z_v = \sigma\left[w_{2L+1,v} l_v + \sum_{e'=(v,c')} w_{2L+1,v,e'} y_{2L,e'}, \right] \tag{85}$$

where $\sigma(x) = (1+e^{-x})^{-1}$ is the *sigmoid* function to map the final output codeword in the range [0,1]. The goal is to train the weights $\{w_{i,v}, w_{i,e,e'}, w_{i,v,e'}\}$ to achieve an $N-$dimensional output codeword.

The computationally constrained IoT systems desire low complexity channel equalization approaches. The trained neural networks will be able to perform channel equalization without requiring channel estimation, hence rendering them suitable for the IoT systems.

3.2.3. Adaptive array processing

MIMO systems are a trending physical layer solution to meet the increasing demand for high-speed, high-multiuser capacity communication systems. MIMO systems, due to their antenna arrays, can exploit spatial and temporal diversity to increase the communication data rate and spectral efficiency. Systems with adaptive antenna arrays can perform smart signal processing to combine the signals received at each array and nullify the interference and/or transmit the signals to steer the beam in an intended direction. Multi-user MIMO [132] is already adopted in the developed and evolving communication standards like 3rd Generation Partnership Project (3GPP) long term evolution (LTE), and long term evolution-advanced (LTE-A). Another emerging MIMO technology, Massive MIMO, is the physical layer technology of choice for the latest 5G technology [133]. Massive-MIMO can revolutionize the 5G communication by providing reliable faster communication to more number of users simultaneously. Emerging 5G wireless networks promise ubiquitous connectivity, high data rates, energy efficiency, and spectrum availability. The dense, diverse and heterogenous nature of IoT networks can be fulfilled by the disruptive 5G technologies such as massive MIMO, non-orthogonal multiple access (NOMA), M2M, etc. Beamforming is a prominent MIMO solution to enable communication to the desired device allowing to coexist with the other devices in the dense network. An essential step that enables beamforming is direction of arrival (DoA) estimation that allows the transmitter/receiver to learn the direction to/from which the signal should be directed/arrived. In this section, we will discuss a few prominent adaptive array techniques and how ML solutions can improvise them.

Several works [134–139] address the problem of DoA estimation in array signal processing using artificial neural networks (ANNs). Let us look at each of these solutions. In [135], authors propose the use of a three-layer RBFNN that can learn multiple source-direction findings of a six-element linear antenna array. The RBFNN does not require training with all possible combinations of training sets. The network will generalize when trained with an expected range of input data. In this case, authors trained the network with input data whose DoA is uniformly distributed in the range -90° to 90°. The performance is compared to the conventional multiple signal classification (MUSIC) algorithm for DoA estimation of correlated and uncorrelated signals. The linear antenna array performs the mapping from the angle space to the sensor output space such that

$$o_i = \sum_{k=1}^{K} a_k e^{j2\pi f_0 d \sin\theta_k + \alpha_k}, \tag{86}$$

where $i = 1, 2, \ldots, M$ and k denote the respective antenna element and incident signal respectively, f_0 is the frequency of incident signal, d is the inter-element spacing, θ_k is the angle of arrival of kth signal and ϕ_k is the initial phase of kth incident signal. The RBFNN is trained with N patterns to perform reverse mapping of received array data (o_i) to the angle space (θ_k). The incident array vectors are preprocessed prior to feeding them to the RBFNN. To train the neural network, the antenna array output vectors are generated ($\mathbf{o}(n), n = 1, 2, \ldots, N$). Each of the array output vector is further transformed to the spatial correlation matrix $\mathbf{R}(n)$. Since the diagonal elements of the correlation matrix does not carry any angle information, i.e. $R_{mm'} = \sum_{k=1}^{K} a_k$, only the cross-correlation terms are considered. These cross-correlated terms are arranged into an input vector $\mathbf{v}(n)$. The output node subsequently computes the weighted sum of the hidden layer outputs.

$$z_k(j) = \sum_{i=1}^{N} w_i(k) \mathfrak{G}\left(\|\mathbf{o}(j) - \mathbf{o}(i)\|^2\right), \quad k = 1, 2, \ldots, K, \;\; j = 1, 2, \ldots, N, \tag{87}$$

where $w_i(k)$ represents the ith weight of the network for the kth incident signal and $\mathfrak{G}(.)$ is the Gaussian function performed by the hidden layer. Now, the equation (87) changes to

$$z_k(j) = \sum_{i=1}^{N} w_i(k) e^{-\|\mathbf{o}(j) - \mathbf{o}(i)\|^2 / \sigma_g^2}, \tag{88}$$

where σ_g controls the influence of each basis function. The above equation can be rewritten in matrix form as,

$$\mathbf{\Theta} = \mathbf{WF}. \tag{89}$$

Here, $\mathbf{\Theta}$ and $\mathbf{W}$ are the $K \times L$ angle and weight matrices and $\mathbf{F}$ is the $L \times N$ hidden layer matrix. L is chosen to be less than N to prevent ill-conditioning arising from large matrix. The input vectors $\mathbf{v}(n)$ are normalized according to Eq. (89). Using least-squares (LS) approach, the weights can be obtained as

$$\hat{\mathbf{W}} = \mathbf{\Theta F}^{\dagger}, \tag{90}$$

where $F\dagger$ is the pseudo-inverse given by

$$\mathbf{F}^{\dagger} = \mathbf{F}^T (\mathbf{F}\mathbf{F}^T)^{-1}. \tag{91}$$

Now, the DoA estimate can be obtained as

$$\hat{\mathbf{\Theta}} = \hat{\mathbf{W}}\mathbf{F} = \mathbf{\Theta}^T (\mathbf{F}\mathbf{F}^T)^{-1}\mathbf{F}. \tag{92}$$

The RBFNN is trained with the Normalized cumulative delta rule [140] such that the weight changes are accumulated over several training presentations as specified by the Epoch. The trained RBFNN will give the DoA estimates when presented with the normalized input vector. The authors demonstrated the computational advantage gained by adopting the RBFNN based DoA estimator as opposed to the conventional MUSIC algorithm. The estimation accuracy of the proposed DoA estimator in addition to the computational efficiency are the key merits of the proposed solution and presents itself as a computationally efficient alternative.

In a recent work [134], the DoA estimation is performed using a ANN with three layers; input, hidden and output layers. The authors study the estimation accuracy in terms of the number of neurons in the hidden layer. Unlike the RBF approach adopted by authors of [135], here the input activation function is the *hyperbolic tangent sigmoid transfer function* and the output activation function is the *logarithmic sigmoid*.

In yet another work [141], the authors employ RBFNN to perform adaptive beamforming. Adaptive beamforming is a method of updating weights of an adaptive antenna array such that the antenna radiation pattern will form beams such that strong beam is sent towards intended user's direction and nulls to sources of interference. The authors adopt a two-step approach to tackle this problem. First, the DoA of desired users are determined as in [135] and secondly, the beamformer weights are estimated to direct the beams. Similar to the DoA estimation problem, the authors

Table 3
Summary of applications of ML in physical layer.

Physical layer solution	ML algorithm	Objective
V. T. Nguyen et al. [71].	DCNN	Cognitive communication architecture
Li et al. [90].	XGBoost	End-to-end dynamic spectrum management
T. Tholeti et al. [72].	Non-parametric Bayesian learning	CR architecture for spectrum assignment
Li et al. [111].	RL	Adaptive rate control
Puljiz et al. [107].	kNN	Adaptive rate control
Yun and Caramanis [108]	SVR	Adaptive rate control
Li [142]	RL	Adaptive rate and power control
Huang et al. [113].	RL wt CNN and RNN	Adaptive rate control
Erdogmus et al. [124].	MLP	Non-linear channel equalization
Ye and Li [125]	DNN	Non-linear channel equalization
Lyu et al. [127].	CNN	Channel decoder
Ye et al. [123].	DNN	Channel equalization in OFDM systems
Nachmani et al. [129].	DNN	Improve BEP algorithm for decoding linear codes
Zooghby et al. [135].	RBFNN	DoA estimation
Nleren and Yaldiz [134]	ANN	DoA estimation
Zooghby et al. [141].	RBFNN	Adaptive beamforming

approach this using RBFNN. For K incident signals, let the signal received at an M element linear antenna array at p^{th} time instant be

$$\mathbf{o}(p) = \sum_{k=1}^{K} \mathbf{b}_k s_k(p) + \mathbf{g}(p) = \mathbf{B}\mathbf{s}(p) + \mathbf{g}(p) \tag{93}$$

such that, $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_M]^T$ is the array steering matrix that holds the spatial signal of the kth source, $\mathbf{s} = [s_1, s_2, \ldots, s_K]$ is the signal vector and $\mathbf{g}(p)$ is the noise vector. Here,

$$\mathbf{b}_m = [1, e^{-j2\pi d \sin\theta_k/\lambda}, \ldots, e^{-j2\pi(M-1)d\sin\theta_k/\lambda}]. \tag{94}$$

RBFNN is trained to compute the minimum variance distortionless response (MVDR) beamformer weights such that

$$\hat{\mathbf{w}}_{MVDR} = \frac{\mathbf{R}^{-1}\mathbf{b}_m}{\mathbf{b}_m^H \mathbf{R}^{-1}\mathbf{b}_m}, \tag{95}$$

where $\mathbf{R} = \frac{1}{P}\sum_p \mathbf{o}(p)\mathbf{o}^H(p)$ is the sample averaged covariance matrix computed from P snapshots of the received signal vector. The beamformer output can be denoted as $\mathbf{y}(p) = \hat{\mathbf{w}}_{MVDR}^H \mathbf{o}(p)$. The beamformer vector estimation can be extended to any adaptive antenna array, the model considered in this example is a linear array for notational simplicity. The input and output layer of the RBFNN consists of $2M$ nodes to accommodate the in-phase and quadrature components of the input vector $\mathbf{o}(p)$ and the hidden layer' outputs. Much alike the DoA estimation problem in [135], the RBFNN is trained to perform an input-output mapping from the received vector space to the beamformer weight space. The weights from the input to the hidden layer are identified using unsupervised k-means clustering and those from hidden to output layer follows the supervised *delta* learning rule. During the training phase, the RBFNN is trained with N_t training array output vectors $\mathbf{x}_n(p)$ and their corresponding $\hat{\mathbf{w}}_{MVDR}^n \forall n \in 1, 2, \ldots, N_t$. The array output vector is normalized prior to computing the corresponding covariance matrices $\mathbf{R}_n$. The beamformer weights $\hat{\mathbf{w}}_{MVDR}^n$ are then computed according to Eq. (95). The trained RBFNN can be used to estimate the optimum MVDR beamformer weights for a presented normalized array output vector in a computationally inexpensive manner.

In this section, we explored the various ML techniques pertinent to the physical layer that is currently proposed to enhance the IoT framework. The integration of such ML solutions with the IoT devices would be a prominent step in developing cognitive IoT architectures that can learn, adapt and behave under the varying system and environmental dynamics. Table 3 enlists the various ML algorithms and their corresponding physical layer objective.

3.3. Open problems and challenges

We have discussed the capabilities introduced by integrating cognition into an IoT framework. The CR-IoT framework is truly an invaluable component to keep up with the rising IoT device density and its tailored comeuppances. While CR holds the key in realizing the full potential of future IoT architectures, several open challenges exist that readers can derive motivation from for future research.

3.3.1. Optimizing distributed spectrum utilization

Though a few spectrum utilization techniques have been introduced for CR IoT frameworks, they involve cognition in the cloud/fog or in the gateway. Such centralized spectrum assignment decision-making introduces additional latency to the IoT communications. Further, the scalability of such techniques would be speculative and intractable in terms of latency and the computational and data management load on the centralized access point (cloud/fog/gateway). A lightweight distributed spectrum decision making would be desired for the IoT frameworks whereby each IoT device uses its own cognitive ability to access the spectrum based on its historical and current spectrum sensing data.

Another viable approach would be to perform opportunistic spectrum access decisions based on the geographical location of the spectrum sensing history. Such that even when the IoT device (SU) senses a PU/SU traffic it can identify the radio frequency identification tags and map them to their location. Accordingly, a geolocation-based spectrum occupancy history can be built to predict the traffic and perform efficient transmit power control scheme to use the channel without interfering with the ongoing traffic. The success of such transmissions can be recorded over time to learn the collision and transmit power level records. An efficient ML technique can be used to perform online learning of the spectrum records to optimize the transmit power level to carry out interference-free spectrum sharing.

3.3.2. Mobility support

Exploring efficient communication strategies for mobile IoT applications such as moving vehicles in the connected vehicular network, drones in an unmanned aerial vehicle (UAV) network, mobile smartphone users, among others pose unforeseen challenges to the capacity, spectrum handoffs, cloud connectivity, scalability, etc. Adaptive physical layer technique such as adaptive intelligent beamforming in conjunction with opportunistic spectrum access in the mobile scenario for reliable energy-efficient communication is another area to be explored. Specifically, ML can be exploited

Table 4
Summary of ML solutions for automatic modulation classification.

Classifiers	Model	Representation	Objective
Jagannath et al. [160].	DNN	Feature-Based	7-Class task considering PSKs, FSKs, QAMs
Kulin et al. [147].	DCNN	I/Q, A/Φ, FFT	11-Class task considering PSKs, FSK, QAMs, PAM, DSB, SSB
O'Shea and Corgan [158]	DCNN	I/Q	11-Class task considering PSKs, FSK, QAMs, PAM, DSB, SSB
Shengliang Peng and Yao [159]	DCNN	Constellation	4-Class task considering PSKs and QAMs
West and O'Shea [146]	DCNN, LSTM, RN	I/Q	11-Class task considering PSKs, FSK, QAMs, PAM, DSB, SSB
Karra et al. [148].	DCNN, DNN	I/Q, FFT	11-Class task considering PSKs, FSK, QAMs, PAM, DSB, SSB

to learn the user mobility pattern and data traffic model to assign radio resources such as transmission rate, power and the frequency band based on link reliability, spectrum congestion, spectrum availability, among others.

4. Machine learning for signal intelligence

As IoT devices become more pervasive throughout society the available operational radio frequency (RF) environment will contain more non-cooperative signals than ever seen before. Subsequently, the ability to garner information about signals within a spectrum of interest will become ever more important and complex, motivating the use of ML for signal intelligence in the IoT. ML techniques for signal intelligence typically manifest themselves as solutions to discriminative tasks, and many applications specifically focus on multi-class or binary classification tasks. Problems of these types arise in the context of IoT in many ways including automatic modulation classification (AMC) tasks, wireless interference classification tasks, and signal detection tasks, each of which, is relevant to signal intelligence for the IoT in their own way.

AMC is the task of determining what scheme was used to modulate the transmitted signal, given the raw signal observed at the receiver. Knowledge of the modulation format used by the transmitter is essential for proper demodulation of the received signal at the receiver, thus solutions to AMC tasks are paramount in scenarios where the operational environment may distort the transmitted signal. Such is the case in the IoT, where multipath fading channels are regular in device to device communication. AMC may also find application in next-generation intelligent, adaptive transceiver technology in which the radios rapidly switch between modulations based on the channel conditions without requiring a dedicated feedback channel.

Solutions to wireless interference classification tasks aim primarily to associate a given received signal with an emitter from a known list of emitters. Typical implementations consider emitters that use common communication standards including WiFi, Zigbee, and Bluetooth. Other such solutions consider additional signals that are present in the environment, such as those emanating from a household microwave oven appliance, as they may play an interfering role in some operational environments. Wireless interference classification in this nature is particularly important in the IoT, as IoT devices are often deployed in the homes of users and around other devices that emit RF signals. Classification of a signals emitter can provide insight into the behavior of its use and subsequently its effect on the operability of the local IoT devices.

Problems of signal detection arise in many different areas of communications and the resulting applications of signal detection very widely. In the simplest case, signal detection can be formulated as a binary classification problem with an output corresponding to whether or not a signal is present in the locally sensed RF environment. While interesting solutions exist for the aforementioned problem formulation, within the IoT more complex detection problems often arise in the context of security. Interesting signal detection algorithms can thus be extended to classify the presence of an intruder provided characteristics of their transmission in the environment. Problems of these types are discussed later in Section 4.3.

4.1. Modulation classification

DL solutions to modulation classification tasks have received significant attention in recent years [143–148]. Several DL models are presented in [143] to address the modulation recognition problem. Hierarchical DNNs used to identify data type, modulation class, and modulation order are discussed in detail in [148]. A conceptual framework for end-to-end wireless DL is presented in [147], followed by a comprehensive overview of the methodology for collecting spectrum data, designing wireless signal representations, forming training data and training deep neural networks for wireless signal classification tasks.

The task of AMC is pertinent in signal intelligence applications as the modulation scheme of the received signal can provide insight into what type of communication frameworks and emitters are present in the local RF environment. The problem at large can be formulated as estimating the conditional distribution, $p(y|x)$, where y represents the modulation structure of the signal and x is the received signal.

Traditionally, AMC techniques are broadly classified as maximum likelihood-based approaches [149–153], feature-based approaches [154–156] and hybrid techniques [157]. Prior to the introduction of ML, AMC tasks were often solved using complex hand engineered features computed from the raw signal. While these features alone can provide insight about the modulation structure of the received signal, ML algorithms can often provide a better generalization to new unseen data sets, making their employment preferable over solely feature-based approaches. The logical remedy to the use of complex hand engineered feature-based classifiers are models that aim to learn directly from received signal data. Recent work [158] show that DCNNs trained directly on complex time domain signal data outperform traditional models using cyclic moment feature-based classifiers. In [159], the authors propose a DCNN model trained on the two-dimensional constellation plots generated from the received signal data and show that their approach outperforms other approaches using cumulant-based classifiers and SVMs.

While strictly feature-based approaches may become antiquated with the advent of the application of ML to signal intelligence, expert feature analysis can provide useful input to ML algorithms. In [160], we compute hand engineered features directly from the raw received signal and apply a feedforward neural network classifier to the features to provide a AMC. The discrete time complex-valued received signal can be represented as,

$$y(n) = h(n)x(n) + w(n), \qquad n = 1, \ldots, N \tag{96}$$

where $x(n)$ is the discrete-time transmitted signal, $h(n)$ is the complex valued channel gain that follows a Gaussian distribution and $w(n)$ is the additive complex zero-mean white Gaussian noise process at the receiver with two-sided power spectral density (PSD)

$N_0/2$. The received signal is passed through an Automatic Gain Control prior to the computation of feature values.

The first feature value computed from the received signal is the variance of the amplitude of the signal and is given by,

$$Var(|y(n)|) = \frac{\sum_{N_s}(|y(n)| - \mathbb{E}(|y(n)|))^2}{N_s} \tag{97}$$

where $|y(n)|$ is the absolute value of the over-sampled signal and $\mathbb{E}(|y(n)|)$ represents the mean computed from N_s samples. This feature provides information which helps distinguish frequency shift keying (FSK) modulations from the phase shift keying (PSK) and quadrature amplitude modulation (QAM) modulation structures also considered in the classification task. The second and third features considered are the mean and variance of the maximum value of the power spectral density of the normalized centered-instantaneous amplitude, which is given as,

$$\gamma_{max} = \frac{max|FFT(a_{cn}(n))|^2}{N_s}, \tag{98}$$

where $a_{cn}(n) \triangleq \frac{a(n)}{m_a} - 1$, $m_a = \frac{1}{N_s}\sum_{n=1}^{N_s} a(n)$, and $a(n)$ is the absolute value of the complex-valued received signal. This feature provides a measure of the deviation of the PSD from its average value. The mean and variance of this feature computed over subsets of a given training example are used as two separate entries in the feature vector input into the classification algorithm, corresponding to the second and third features, respectively.

The fourth feature used in our work was computed using higher order statistics of the received signal, namely, cumulants, which are known to be invariant to the various distortions commonly seen in random signals and are computed as follows,

$$C_{lk} = \sum_{p}^{\text{No. of partitions in } l} (-1)^{p-1}(p-1)! \prod_{j=1}^{p} \mathbb{E}\{y^{l_j - k_j} y^{*k_j}\}, \tag{99}$$

where l denotes the order and k denotes the number of conjugations involved in the computation of the statistic. We use the ratio, C_{40}/C_{42} as the fourth feature which is computed using,

$$C_{42} = \mathbb{E}(|y|^4) - |\mathbb{E}(y^2)|^2 - 2\mathbb{E}(|y|^2)^2, \tag{100}$$

$$C_{40} = \mathbb{E}(y^4) - 3\mathbb{E}(y^2)^2. \tag{101}$$

The fifth feature used in our work is called the in-band spectral variation as it allows discrimination between the FSK modulations considered in the task. We define $Var(f)$ as,

$$Var(f) = Var\Big(\mathcal{F}\big(y(t)\big)\Big), \tag{102}$$

where $\mathcal{F}(y(t)) = \big\{Y(f) - Y(f - F_0)\big\}_{f=-f_i}^{+f_i}/F_0$, F_0 is the step size, $Y(f) = FFT(y(t))$, and $[-f_i, +f_i]$ is the frequency band of interest.

The final feature used in the classifier is the variance of the deviation of the normalized signal from the unit circle, which is denoted as $Var(\Delta_o)$. It is given as,

$$\Delta_o = \frac{|y(t)|}{\mathbb{E}(|y|)} - 1. \tag{103}$$

This feature helps the classifier discriminate between PSK and QAM modulation schemes.

The modulations considered in the work are the following: BPSK, quadrature phase shift keying (QPSK), 8PSK, 16QAM, continuous phase frequency shift keying (CPFSK), Gaussian frequency shift keying (GFSK), and Gaussian minimum shift keying (GMSK), resulting in a seven class classification task using the aforementioned six features computed from each training example. To generate the data set, a total of 35,000 examples were collected: 1,000 examples for each modulation at each of the five SNR scenarios considered in the work. Three different feedforward neural network structures were trained at each SNR scenario using a training set consisting of 80% of the data collected at the given SNR and a test set consisting of the remaining 20%. The three feedforward nets differed in the number of hidden layers, ranging from one to three. Qualitatively, the feedforward network with one hidden layer outperformed the other models in all but the least favorable SNR scenario, achieving the highest classification accuracy of 98% in the most favorable SNR scenario. The seemingly paradoxical behavior is attributed to the over-fitting of the training data when using the higher complexity models, leading to poorer generalization in the test set.

This work has been further extended to evaluate other ML techniques using the same features. Accordingly, we found that training a random forest classifier for the same AMC task yielded similar results to the feedforward network classifier. Additionally, the random forest classifier was found to outperform the DNN approach in scenarios when the exact center frequency of the transmitter was not known, which was assumed to be given in the previous work. The random forest classifier was comprised of 20 classification and regression trees (CART) constructed using the gini impurity function. At each split a subset of the feature vectors with cardinality equal to 3 was considered.

An alternative approach to the previously described method is to learn the modulation of the received signal from different representations of the raw signal [147]. train DCNNs to learn the modulation of various signals using three separate representations of the raw received signal. The authors denote the raw complex valued received signal training examples as $\mathbf{r}_k \in \mathcal{C}^N$, where k indexes the procured training data set and N is the number of complex valued samples in each training example. We inherit this notation for presentation of their findings. The data set in the work was collected by sampling a continuous transmission for a period of time and subsequently segmenting the received samples into N dimensional data vectors.

The authors train separate DCNNs on three different representations of the raw received signal and compare their results to evaluate which representation provides the best classification accuracy. The first of the three signal representations are given as a $2 \times N$ dimensional in-phase/quadrature (I/Q) matrix containing real-valued data vectors carrying the I/Q information of the raw signal, denoted $\mathbf{x_i}$ and $\mathbf{x_q}$, respectively. Mathematically,

$$\mathbf{x}_k^{IQ} = \begin{bmatrix} \mathbf{x_i}^T \\ \mathbf{x_q}^T \end{bmatrix} \tag{104}$$

where $\mathbf{x}_k^{IQ} \in \mathcal{R}^{2\times N}$. The second representation used is a mapping from the complex values of the raw received signal into two real-valued vectors representing the phase, Φ and the magnitude, A,

$$\mathbf{x}_k^{A/\Phi} = \begin{bmatrix} \mathbf{x_A}^T \\ \mathbf{x_\Phi}^T \end{bmatrix} \tag{105}$$

where $\mathbf{x}_k^{A/\Phi} \in \mathcal{R}^{2\times N}$ and the phase vector $\mathbf{x_\Phi}^T \in \mathcal{R}^N$ and magnitude vector $\mathbf{x_A}^T \in \mathcal{R}^N$ have elements,

$$x_{\Phi_n} = \arctan\left(\frac{r_{q_n}}{r_{i_n}}\right), x_{A_n} = (r_{q_n}^2 + r_{i_n}^2)^{\frac{1}{2}} \tag{106}$$

respectively. The third representation is a frequency domain representation of the raw time domain complex signal. It is characterized by two real-valued data vectors, one containing the real components of the complex FFT, $\Re(X_k)$, and the other containing the imaginary components of the complex FFT, $\Im(X_k)$, giving,

$$x_k^F = \begin{bmatrix} \Re(X_k)^T \\ \Im(X_k)^T \end{bmatrix} \tag{107}$$

Using these three representations of the raw signal, the authors train three DCNNs with identical structure and compare the accuracy of the resultant models to determine which representation allows for learning the best mapping from raw signal to modulation structure.

The authors use training examples comprised of $N = 128$ samples of the raw signal sampled at 1 MS/s and consider the following 11 modulation formats: BPSK, QPSK, 8-PSK, 16-QAM, 64-QAM, CPFSK, GFSK, 4-pulse-amplitude modulation (PAM), wideband Frequency Modulation (WBFM), amplitude modulation (AM)-double-sideband modulation (DSB), and AM-single-sideband modulation (SSB). Thus, the training targets $\mathbf{y}_k \in \mathcal{R}^{11}$ are encoded as one-hot vectors where the index holding a 1 encodes the modulation of the signal. The authors use a total of 220,000 training examples $\mathbf{x}_k \in \mathcal{R}^{2\times 128}$. Additionally, samples were acquired uniformly over different SNR scenarios ranging from $-20dB$ to $+20dB$.

The CNN structure used for each signal representation is the same, and consists of two convolutional layers, a fully connected layer, and a *softmax* output layer trained using the negative log-likelihood loss function. The activation function used in each of the convolutional layers and the fully connected layer is the *ReLU* function. The CNNs were trained using a training set comprised of 67% of the total data set, with the rest of the data set used as test and validation sets. An Adam optimizer [161] was used to optimize the training processes for a total of 70 epochs. The metrics used to evaluate each of the models include the precision, recall, and F1 score of each model. The authors provide a range of values for the three aforementioned metrics for the CNN models trained on different data representations for three different SNR scenarios: high, medium, and low, corresponding to $18dB$, $0dB$, and $-8dB$, respectively. In the high SNR scenario, the authors report that the precision, recall, and F1 score of each of the three CNN models falls in the range of 0.67–0.86. For the medium and low SNR scenarios, the same metrics are reported in the ranges of 0.59–0.75 and 0.22–0.36, respectively. The authors attribute the relatively low performance to the choice of the time-varying multipath fading channel model used when generating the data.

The authors go on to evaluate the classification accuracy of each of the three models trained using different data representations under similar SNR conditions. Qualitatively, each of the three CNN models performs similarly at low SNR, while the CNN trained on the I/Q representation of data yields a better accuracy at medium SNR and the CNN trained on the amplitude and phase representation yields a better accuracy at high SNR. Interestingly, the CNN trained on the frequency domain representation of the data performs significantly worse than the I/Q and A/ϕ CNNs at high SNR. The authors mention that this could potentially be due to the similar characteristics exhibited in the frequency domain representation of the PSK and QAM modulations used in the classification problem. The primary takeaway from this work is that learning to classify modulation directly from different representations of the raw signal can be an effective means of developing a solution to the AMC task; howeve, the efficacy of the classifier is dependent on how the raw signal is represented to the learning algorithm.

The following table provides the summary of the methods for AMC discussed in this section.

4.2. Wireless interference classification

The task of wireless interference classification (WIC) regards identifying what type of wireless emitters exist in the local RF environment. The motivation behind such a task is that it can be immensely helpful to know what type of emitters are present (WiFi, Zigbee, Bluetooth, etc.) in the environment when attempting to avoid and coexist with interference from other emitters. Solutions to WIC tasks are often similar in nature to AMC techniques. For example, [162] employ DCNNs to classify IEEE 802.11 b/g, IEEE 802.15.4, and IEEE 802.15.1 emitters using a frequency domain representation of the captured signal. WIC tasks may also consider emitters in the environment that are not used in communication systems. In [163], an SVM solution is developed to classify interference in wireless sensor networks (WSNs) from IEEE 802.11 signals and microwave ovens. A recent work [164] shows the use of DCNNs to classify radar signals using both spectrogram and amplitude-phase representations of the received signal. In [165], DCNN models are proposed to accomplish interference classification on two-dimensional time-frequency representations of the received signal to mitigate the effects of radio interference in cosmological data. Additionally, the authors of [166] employ DCNN and LSTM models to achieve a similar end.

In [147], DCNNs are employed for the purpose of the wireless interference classification of three different wireless communication systems based on the WiFi, Zigbee, and Bluetooth standards. They look at five different channels for each of the three standards and construct a fifteen class classification task for which they obtain 225,225 training vectors consisting of 128 samples each, collected at 10 MS/s. A flat fading channel with additive white Gaussian noise is assumed for this classification task.

Three DCNNs were trained and evaluated using the wireless interference classification data set described above. Each of the three CNNs was trained on one of the representations of the data that were presented in the previous section, namely, I/Q, A/Φ, and frequency domain representation. The CNN architectures were also the same as presented previously in Section 4.1.

Each of the three CNNs trained using different data representations was evaluated in a similar fashion to the evaluation method described in Section 4.1, namely, using precision, recall, and F1 score under different SNR scenarios. For the wireless interference classification task, the precision, recall, and F1 score of each of the three CNNs all fell in the interval from 0.98 to 0.99 under the high SNR scenario. For the medium and low SNR scenarios, the analogous intervals were from 0.94–0.99 to 0.81–0.90, respectively.

Additionally, the authors provide an analysis of classification accuracy for each of the three CNN models at varying SNRs. For the task of wireless interference classification, the CNN model trained on the frequency domain representation of the data outperforms the other models at all SNRs, especially in lower SNR scenarios. The authors claim that these findings are due to the fact that the wireless signals considered have more expressive features in the frequency domain as they have different bandwidth, modulation, and spreading characteristics.

The authors of [167] take a different approach to the wireless interference classification task and primarily compare different types of learning models rather than different types of data representation. The models the authors propose include deep feed-forward networks, deep convolutional networks, support vector machines using two different kernels, and a multi-stage training (MST) algorithm using two different learning algorithms. The authors consider 12 different transmitters and collect 1000 packets from each transmitter for a total of 12,000 packets which comprise the entire data set. Each transmitter transmitted the same exact 1000 packets, which were generated using pseudo-random values injected into the modem. All of the transmitters used a proprietary OFDM protocol with a QPSK modulation scheme and a baseband transmitter sample rate of 1.92 MS/s. At the receiver, each packet is represented by 10,000 time domain I/Q samples. Each of the models was trained on data sets consisting of training examples made up of 32, 64, 128, 256, 512, and 1024 samples from each packet, and their performance is compared across data sets. Given the complex-valued received signal,

$$r = (r_1, r_2, \ldots, r_N) \tag{108}$$

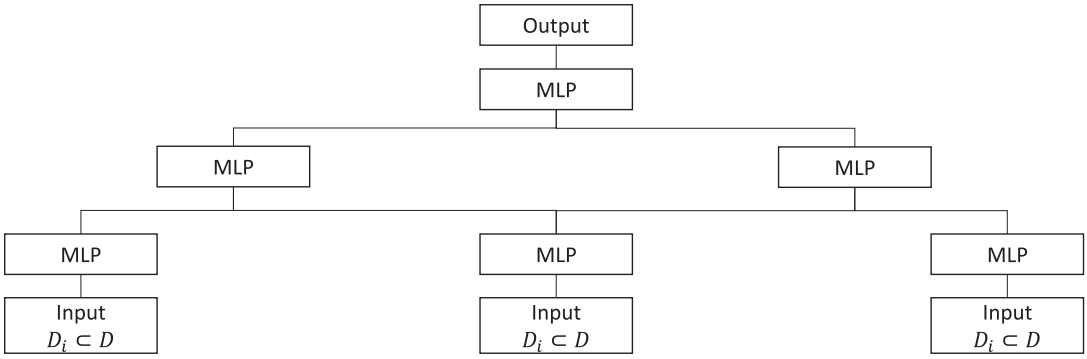

Fig. 9. Adaptation of MST MLP used in [167].

N samples were selected by skipping the first N_0 samples of a packet where $|\Re(r_i)| < \tau$ for some $\tau > 0$ yielding the signal vector x,

$$x = (r_{N_0}, r_{N_0+1}, \ldots, r_{N_0+N-1}) \tag{109}$$

For the DNN, SVM, and MST models each training example was constructed by concatenating the real and imaginary parts of the signal vector, yielding a vector of dimension $2N$. For the CNN model the real and imaginary parts of the signal vector were stacked to generate $2 \times N$ dimensional training vectors.

The DNN architecture considered in the work consisted of two fully connected hidden layers, comprised of 128 *ReLU* units each and an output layer consisting of logistic *sigmoid* units. The network was trained using the Adam optimizer [161] and a mini-batch size of 32.

The CNN model used by the authors was composed of two convolutional layers using 64 (8×2) and 32 (16×1) filters, respectively. Each convolutional layer was input into a max-pool layer with a pool size of 2×2 and 2×1, respectively. The output of the second max-pool layer was fed into a fully-connected layer consisting of 128 *ReLU* units. An output layer employing logistic *sigmoid* units was used on top of the fully-connected layer.

The two SVM architectures analyzed in the work differ only in the kernel function used. The first architecture employed the polynomial kernel and the second employed the Pearson VII Universal Kernel [168]. Both architectures used Platt's Minimization Optimization algorithm to compute the maximum-margin hyperplanes.

The authors also analyze the performance of MST MLPs trained using first order and second order methods. A high-level description of MST MLP is presented here and we refer the interested reader to [169] for a more rigorous derivation. The MST method to training neural networks, as presented in the work, is essentially a hierarchical way to solve an optimization problem by solving smaller constituent optimization problems. To this end, in what is called the first stage, a number of separate MLPs would be trained on different subsets of the training data set. This can be seen in the lowest layer of the hierarchical representation adapted from [167], and provided herein Fig. 9.

Once the first stage is trained, a second stage is trained by taking the concatenation of the network outputs from the first stage as input. Training can continue in this fashion for subsequent stages. One of the advantages of training networks in this way is that the many smaller MLPs comprising the larger classifier can be efficiently trained using second-order optimization methods. Second-order optimization methods such as Newton, Gauss-Newton, or Levenberg-Marquardt methods are usually intractable due to the size of typical networks but can provide better convergence when applicable. The authors train two 3-stage MST systems, one using the first order method of SGD, and one using the second-order Accelerated Levenberg–Marquardt method [170]. Each MST system had the identical structure where stage 1 consisted of 60 MLPs with 2 hidden layers and 10 units in each layer. Stage 2 and 3 had the same architecture and were comprised of 30 MLPs with each MLP consisting of 2 hidden layers made up of 15 units each. All hidden units employed the *tanh* activation function and all output layers contained linear units.

All of the models described above were trained on 10 different iterations of the collected data set and their performance was compared. Five data sets were constructed using training examples made up of 32, 64, 128, 256, and 512 samples and each model was trained twice, using a training set comprised of 90% and 10% of the total data set, for a total of 10 different data sets for each model. In general, the MST system trained using second-order methods on 90% of the training data performed best across all sizes of training examples, yielding a classification accuracy of 100% for each data set. All of the models performed better when trained using 90% of the data set as opposed to 10% of the training data set. Generally, each model performed better when provided with training examples that contained more samples, with the exception of the deep feedforward network model, which the authors attribute to the fact that longer sequences of samples may contain an increasing number of artifacts which the DNN may not be robust to. A summarization of the different models presented in this section is provided in Table 5.

4.3. Open problems

The solutions to the tasks of wireless interference and modulation classification fixate themselves among solutions readily available to be deployed in the IoT. This distinction is primarily a result of the mutual exclusivity that these tasks exhibit with the IoT itself; these problems exist both within and outside the context of the IoT. Contrarily, there are signal intelligence tasks that arise from and are innate to the IoT, which have been studied in comparatively less detail. These tasks, along with their potential to benefit from the application of ML techniques, are described in the rest of this section.

4.3.1. Intrusion detection

Security in the IoT is of utmost importance as the prevalence of connected devices in society and the amount of data collected from individuals increases. Detection of an intruder is often the first step in mitigating efforts from adversaries and can be performed in myriad ways across multiple layers of the protocol stack. As the problem of intruder detection moves from the internet to the IoT, the detection of the physical presence of the intruder among things becomes a salient avenue for mitigation. In [171],

Table 5
Summary of ML solutions for wireless interference classification.

Classifiers	Model	Representation	Objective
Kulin et al. [147].	DCNN	I/Q, A/Φ, FFT	Classification of 15 WiFi, ZigBee, and Bluetooth Transmitters
Selim et al. [164].	DCNN	2D time-frequency, A/Φ	Classification of Radar Signals
Akeret et al. [165].	DCNN	2D time-frequency	Classification of Cosmological Interference
Czech et al. [166].	DCNN, LSTM	2D time-frequency	Classification of Cosmological Interference
Youssef et al. [167].	DNN, DCNN, SVM, MST	I/Q	Classification of 12 OFDM Transmitters
Schmidt et al. [162].	DCNN	FFT	Classification of IEEE 802.11 b/g, IEEE 802.15.4, IEEE 802.15.1 signals
Grimaldi et al. [163].	SVM	Feature Based	Classification of IEEE 802.11 and Microwave Oven signals

received signal strength indication (RSSI) information is collected from deployed security probes in an attempt to detect behaviors and communications that are illegitimate and thus identify devices that may have been compromised or may have entered the local network illegally. The authors collect RSSI information, reception timestamps, and radio activity during a time interval from each of the probes and route them to a central security system, which processes the information using a proposed neural network algorithm, which classifies the presence of an intruder. The authors note significant changes in collected RSSI information in their laboratory but highlight a full implementation of the proposed solution as future work. A primary advantage to RSSI-based intrusion detection is that the proposed solution is protocol agnostic.

4.3.2. Indoor localization

The problem of indoor localization remains a challenging one in the context of the IoT and elsewhere. Generally, RF localization problems arise when trying to estimate the geolocation of a receiving or transmitting radio. In outdoor environments, this is readily accomplished on-board many devices using various geolocating signals such as GPS and GNSS; however, the efficacy of these signals use in geolocation is severely diminished without line of sight (LoS) between the satellites and receivers. Thus, indoor localization becomes an important problem in many applications involving the tracking and location of devices that are associated with human users, as these applications often occur indoors. Examples of applications that benefit from indoor localization capabilities include indoor robotic systems, assisted living systems, health applications, and location-based services. Additionally, in [172], indoor localization of IoT devices is motivated as one of the key enabling technologies in increasing the utilization of the IoT.

Most indoor localization approaches in the IoT aim to make use of information transmitted from the local Wi-Fi access points and employ some form of Wi-Fi fingerprinting. In [173], a clustering based access point selection and RSSI reconstruction algorithm is proposed to obtain the optimal feature set for input to an ML-based localization algorithm. Simulation results are provided using ANN, SVR, and ensemble SVR to obtain localization predictions from the selected RSSI values. In [174], DNNs are proposed in conjunction with a linear discriminant analysis to operate on RSSI and basic service set identifier (BSSID) information to produce both classification and regression location information. Alternatively, [175] suggest utilizing channel state information consisting of subcarrier-level measurements of OFDM channels as opposed to RSSI based fingerprinting and simulation results using CNNs and LSTMs trained on channel state information are provided.

5. Machine learning for higher layers

The requirement for IoT devices to have distributed intelligence is becoming inevitable to tackle the problems emanating from the complexity, dynamic nature of its operations and to ensure scalability. This implies that part of the IoT "smart" devices will require autonomy to react to a wide range of situations pertaining to networking, spectrum access, among others [176]. This is where the role of ad hoc networking becomes a crucial part of IoT. Examples of ad hoc interaction in the context of IoT can include vehicular ad hoc network (VANET) that involves vehicles communicating with each other and roadside infrastructure along with the assistance of various sensors (velocity, temperature, humidity, CO2 emissions, etc.). Similarly, the ability to deploy wireless ad hoc sensor networks (WASNs) will also play a crucial role in the overall IoT architecture [177] that is envisioned to enable smart cities as shown in Fig. 6. The ad hoc networking aspect of IoT will, therefore, find applications in areas such as healthcare, infrastructure management, disaster prevention, and management, and optimizing transportation systems [178–180].

The advancements in the higher layers, especially the data-link and the network layers have played a significant role in enabling IoT devices. The necessity to provide fair and efficient spectrum access has been a key motivating factor for researchers to design MAC protocols for IoT [181,182]. In contrast to centralized designs where entities like base stations control and distribute resources, nodes in ad hoc IoT network have to coordinate resource allocation in a distributed manner. Similarly, to ensure scalability and reduce overhead, distributed designs are usually favored while designing routing algorithms for such networks. Recently, ML has made a significant impact on the design of these layers specifically to enhance scheduling and resource allocation, mitigating attacks like denial of service (DoS) in hostile environments, and efficient routing among others. In this section, we discuss in detail some of the advances made on this front.

5.1. Data link layer

A key functionality of the data link layer is to negotiate the access to the medium by sharing the limited spectrum resources in an ad hoc manner. Traditional MAC protocols designed for WANETs (including IoT networks) include carrier sense multiple access/collision avoidance (CSMA/CA) [183,184], time division multiple access (TDMA) [185,186], code division multiple access (CDMA) [187,188] and hybrid approaches [189–191]. Here, we discuss some of the recent efforts to employ ML to enhance the data link layer.

The broadcast scheduling problem (BSP) is a key problem studied in a TDMA-based network to find an optimal TDMA schedule that provides transmission time slots to of all nodes while minimizing the TDMA frame size [192]. Several ML-based approaches have been proposed to solve this combinatorial optimization of BSP using variations of neural networks. This includes the work of [193] proposing a combination of HNN and genetic algorithm (GA) and [194] using sequential vertex coloring (SVC) and noisy chaotic neural network (NCNN). Subsequently, these solutions were shown to be outperformed by fuzzy hopfield neural network (FHNN) proposed in [195]. Here, we describe how [195] tackles BSP.

Consider N nodes in a network with N_T time slots to share among these nodes. The slot assignment matrix **SA**, in which each element is defined as $SA_{ij} = 1$, if time slot j is assigned to node i, otherwise $SA_{ij} = 0$. The set of time slots to be assigned is given by set $T = \{t_1, t_2, \ldots, t_{N_T}\}$. The fuzzy state, a degree that time slot t_x is assigned to node i is represented by μ_{xi} and matrix of all the fuzzy states, **U** is called a fuzzy c-partition matrix. Next, the channel utilization of node i is defined as the fraction of total time slots assigned to node j from the total TDMA frame given as $\rho_j = (\sum_{j=1}^{N_T} SA_{ij}/N_T)$. Accordingly, the total channel utilization for the network can be given as [196],

$$\rho = \frac{1}{N_T N} \sum_{j=1}^{N} \sum_{i=1}^{N_T} SA_{ij} \tag{110}$$

The lower bound for the frame length is given by $\max_{i \in N} \deg(i) + 1$ where $\deg(i)$ is the number of edges incident to it. In the case of FHNN, an energy function is considered as the distance between the current state of the HNN and its solution state. The objective is to minimize the energy function by solving the optimization problem. In this case, the energy function that considers all the constraints is defined as follows [195],

$$E = \frac{\alpha}{2} \sum_{x=1}^{N_T} \left(\sum_{i=1}^{N} \mu_{xi} - 1 \right)^2 + \beta \sum_{x=1}^{N_T} \sum_{i=1}^{N} \left(\sum_{y=1, y \neq i}^{N_T} d_{iy} \mu_{yi} + \sum_{y=1, y \neq i}^{N_T} d_{iy} \sum_{y=1, y \neq i, k \neq y}^{N_T} d_{yk} (\mu_{xi})^f \left[t_x - \sum_{y=1}^{N_T} \frac{t_y}{\sum_{k=1}^{N_T} (\mu_{ki})^f} (\mu_{yi})^f \right]^2 \right) \tag{111}$$

where α and β are assumed to be positive coefficients, f is the fuzzification parameter, and $d_{iy} = 1$, if there is a connectivity between i and y. The first term in equation (111) ensures that N_T slots can only be distributed among the N classes (nodes). The second term minimizes the inter-class euclidean distance from a sample to the cluster center of all clusters. Accordingly, FHNN aims to classify N_T time slots into N nodes by minimizing E. In simulations, the proposed FHNN based BSP approach outperforms both [193,194] in terms of average time delay. Additionally, authors also show that performance improves with larger f at the expense of increased convergence time.

There have also been efforts to advance the current MAC protocols to react to different kinds of attack like DoS that can debilitate IoT devices. In one such case [197], a MLP is used to modify carrier sense multiple access (CSMA)-based network to identify DoS attack and stay inactive for a duration to preserve the energy of the wireless sensor nodes. As shown in Fig. 10, the MAC layer of each node consists of a MLP that has been trained prior to deployment. The parameters used by MLP include collision rate (c_r), packet request rate (P_{req}) and average packet wait time (t_w). The proposed solution is evaluated using both BP and the particle swarm optimization (PSO) [198] algorithm for training. The authors show that BP has lower computational cost compared to PSO but provides inferior convergence point in terms of quality of the weights. The output of MLP represents the probability that there is an active DoS attack (p_t). Based on the chosen threshold Γ_{th}, the nodes decide to sleep for a predetermined period of time when $p_t > \Gamma_{th}$. The work does not discuss the optimal value for Γ_{th} or the sleep time but provides an example of applying ML to mitigate the effects of such attacks.

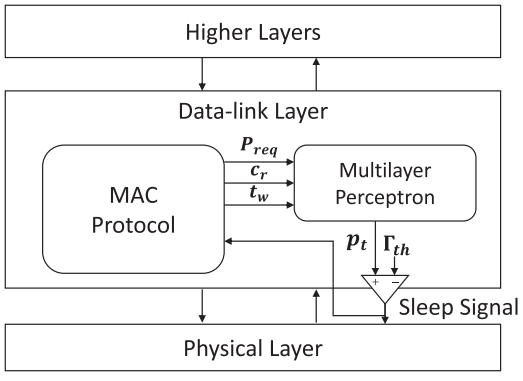

Fig. 10. Block diagram of MLP in MAC against DoS.

Another interesting application where ML, specifically RL, has been successfully applied is in the domain of dynamic spectrum access (DSA) for CR which could be instrumental in enabling modern IoT given the constrained availability of spectrum. An ALOHA-like scheme is developed for CRs by applying a multiagent RL framework [142]. In this work, a secondary user that is able to transmit successfully over an idle channel without collision receives a positive reward and zero otherwise. ACK packets received after the transmission is used to ensure collision-free transmissions. Since the secondary user does not have control over the channel state, the Q-function is defined as the expected reward over a given time slot t. This in turn depends on the state of the overall system, $S(t) = s$ and the node i's action ($a_i(t)$) at time slot t to transmit on channel h. The expectation is taken over the randomness of other secondary user's action and the primary user's activity which can be represented as,

$$Q_{ih}^s = E[R_i | a_i(t) = h, S(t) = s] \tag{112}$$

where R_i is the rewards for action. To ensure good channels are not neglected, the authors propose the use of a Boltzmann distribution for random exploration during the learning phase. Considering temperature $\mathcal{T}$, the exploration probability is given as,

$$P(i \text{ chooses channel } h | \text{ state } s) = \frac{\exp(Q_{ih}^s / \mathcal{T})}{\sum_{k=1}^{N} \exp(Q_{ih}^s / \mathcal{T})} \tag{113}$$

To accomplish this, each secondary user considers both the channel and other secondary users to update its Q-values to choose the best action. It is important to remember that this is an extreme case where no control packets are exchanged between nodes similar to traditional ALOHA. Furthermore, the authors were able to show convergence in limited circumstances even when they extend the full observations to the case of partial observations. Simulations showed how secondary users can learn to avoid collision and outperform a scheme that uses Nash equilibrium.

A similar case is considered in [199] where authors propose a distributed DSA algorithm based on multi-agent reinforcement learning but this time employing DRL. We have seen how Q-learning provides adequate performance when the state-action space is relatively small. As the state-action space grows exponentially for larger problems, the direct application of Q-learning becomes inefficient and impractical as discussed previously. DQN which combines DNN with Q-learning can overcome this challenge. The goal is to enable users to learn a policy while dealing with the large state space without online coordination or message exchanges between users. In [199], the authors model their network state as partially observable for each user and the dynamics being non-Markovian and determined by the multi-user actions, they propose to use LSTM layer that maintains an internal state and aggregate observations over time.

To ensure feasibility, the training is set to happen offline where various training experiences with changing environment and topology are considered. This ensures that the algorithm can be deployed to operate in a distributed manner with the need to be

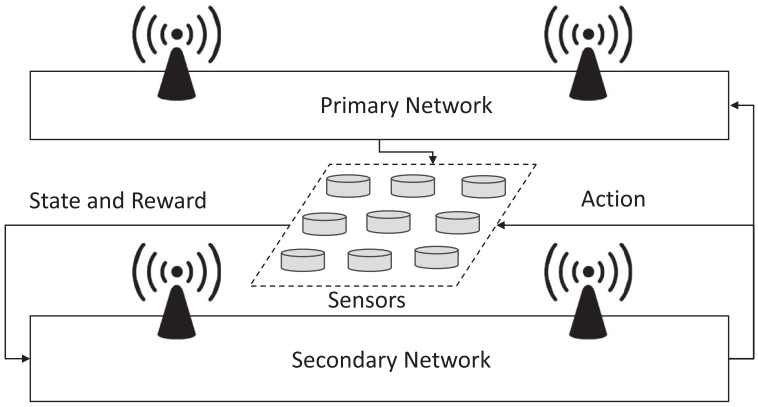

Fig. 11. Framework for power control in cognitive network.

updated only if the operating conditions are significantly different from the training set. After the training phase, each user determines which channel to select and associate "attempt probability" based on its observation. The proposed algorithm is compared against slotted-ALOHA that is assumed to have complete knowledge of the network and hence used optimal attempt probability. The proposed distributed algorithm that only used ACKs to learn outperforms slotted ALOHA by twice the channel throughput. They evaluated the network for two network utilities, (i) network rate maximization and (ii) individual rate maximization. In the case of users whose objective was to maximize the sum rate of the network, some learned to remain idle (sacrifice) incurring zero rate in order to maximize the overall network utility. In contrast, when each user aims to maximize its own rate they converged to a Pareto-optimal sharing policy.

The authors of [200] propose a DQN based framework that provides an Intelligent Power Control (IPC) algorithm for secondary users to coexist with the primary user while ensuring QoS for both. The overall architecture is depicted in Fig. 11. The authors assume the presence of several sensors that are deployed to monitor and convey the received signal strength (RSS) to the secondary users for decision making. The infinite states associated with the continuous RSS impose the need to employ DQNs. During the DQN's training phase, secondary users assume complete knowledge of whether the QoS of every user (primary and secondary) are satisfied. The authors argue this can be achieved by overhearing the ACK packets. Once learning is complete, only the feedback from the sensors is required to determine the optimal power level for the secondary user to access the spectrum while satisfying the QoS constraint of both the networks. IPC is compared against the distributed constrained power control (DCPC) algorithm [201] which is an optimized solution. In contrast to IPC, an optimization-based technique like DCPC requires cooperation between both primary and secondary users. The simulation shows how IPC converges faster compared to DCPC while achieving a near optimal solution.

Realizing the role ML will play in the near future in maximizing the use of scarce spectrum, Defense Advanced Research Projects Agency (DARPA) initiated a three year competition known as Spectrum Collaboration Challenge (SC2). The goal was for teams to propose an ML-based spectrum sharing strategy to allow peaceful coexistence between any unknown heterogeneous wireless networks. A solution inspired from this competition is presented in [202], which explores deep reinforcement learning multiple access (DLMA) for a heterogeneous wireless network consisting of various kind of networks (ALOHA, TDMA) that coexist. To accomplish this, authors use DRL to learn spectrum usage from a series of environmental observations and actions without actually being aware of the type of MAC protocols being operated. The goal is to maximize the total throughput of all the coexisting networks. They exploit neural networks to employ DRL as compared to traditional RL to enable fast convergence and ensure robustness to non-optimal parameters. Fast convergence is essential for wireless networks as convergence time is shorter than coherence time which will give nodes an opportunity to operate using an optimal strategy rather than trying to catch up with the changing efferent every time. Similarly, the lack of knowledge of existing networks makes it difficult to obtain optimal parameters.

The possible actions that can be taken by an agent is $a(t) \in \{wait, transmit\}$. The observation after taking one of these actions can be $z(t) \in \{success, collision, idleness\}$. Accordingly, the channel state at $t+1$ is given as an action-observation pair $c(t+1) \triangleq \{a(t), z(t)\}$. Next, the environmental state at time $t+1$ is given as $s(t+1) \triangleq \{c(t-\mathfrak{h}+2), \ldots, c(t), c(t+1)\}$, where the parameter $\mathfrak{h}$ is the state history length to be tracked by the agent. The reward for transitioning from $s(t)$ to $s(t+1)$ is defined as,

$$r(t+1) = \begin{cases} 1 & , \text{ if } z(t) = success \\ 0 & , \text{ if } z(t) = \{collision\ idleness\} \end{cases} \tag{114}$$

In this work, a DNN is used to approximate the state value function. Assuming ϕ is the parameter vector representing the weights of the DNN, the approximation can be represented as $q(s,\ a;\ \phi) \approx Q^*(s,\ a)$. The authors employ "experience replay" [203] which uses multiple experience samples $(s, a, r(t+1), s(t+1))$ in each training step using the following loss equation,

$$L(\theta) = \sum_{(s,a,r,s') \in EX_t} (y_{r,s'} - q(s,a;\phi))^2 \tag{115}$$

where,

$$y_{r,s'} = r + \gamma \max_{a'} q(s', a'; \phi_t) \tag{116}$$

where EX_t is the set of experience samples used for training at time t. The authors argue the advantage of using DRL over RL by showing a faster convergence rate and a near-optimal strategy being achieved through simulations. They show how the network can learn and achieve near-optimal performance with respect to the sum throughput objective without the knowledge of co-existing MAC (TDMA, ALOHA). The work is further extended [204] by using a residual network [205] in place of the DNN. The authors show how a single RN architecture with fixed depth is suitable to ever-changing wireless network scenarios as compared to the plain DNN which was shown to vary in performance based on the selected number of hidden layers.

These works provide a promising direction towards solving the spectrum crunch that will be experienced with the proliferation of

Table 6
Summary of application of ML in MAC protocols.

MAC protocol	ML algorithm	Objective
Salcedo-Sanz et al. [193].	HNN wt GA	Proposed to solve BSP
Shi and Wang [194]	NCNN wt SVC	Proposed to solve BSP
Shen and Wang [195]	FHNN	Proposed to solve BSP
Kilkarni and Venayagamoorthy [197]	MLP	Tolerance against DoS
Li [142]	RL	ALOHA-like spectrum access
Naparstek and Cohen [199]	DQN wt LSTM	ALOHA-linke spectrum access
Li et al. [200].	DQN	Intelligent power control
Yu et al. [202].	DQN	Non-coopertive heterogenous network
Yu et al. [204].	DQN wt RN	Non-coopertive heterogenous network

IoT devices and 5G networks in the near future. We summarize the discussion of this section in Table 6.

5.2. Network layer

Routing protocols have evolved over the years to accommodate the needs of modern IoT WANETs. The design of the routing protocols primarily depends on the context and objective of the application and can be classified in several ways. Some of these classifications include geographical location based routing [206–209], hierarchical [210], QoS-based [211,212], and recently cross-layer optimized routing [69,213–217]. Similar to earlier discussions, ML has elegantly found its way into this domain by providing a powerful tool to solve some of the problems associated with designing routing algorithms.

One of the earliest attempts to apply ML to routing algorithms is presented in [218] in the context of a traditional wired network including Local Access and Transport Area (LATA) telephone network. The proposed algorithm, referred to as Q-routing, uses a distributed approach which gathers estimated delay information from immediate neighbors to make the routing decision. The proposed Q-learning based routing algorithm can be represented as a variation of Bellman-Ford shortest path algorithm [219,220] that replaces hop count by delivery time and performs the relaxation step online in an asynchronous manner. In [218], the authors clearly showed how Q-routing is able to adapt to varying traffic loads after the initial inefficient learning period. When the load is low, Q-routing converges to using the shortest path and when the load increases, it is capable of handling the congestion more elegantly compared to the shortest path routing that is forced to use static routes.

In a recent effort [221], the need to reenvision router architectures and key routing strategies to meet the requirements of modern networks is highlighted. This was motivated by the advent of the graphical processing unit accelerated software defined routers that are capable of massive parallel computing. Accordingly, authors propose to use DL, specifically, a deep belief network (DBN) based system that uses traffic patterns to determine the routes. The authors demonstrated with simulations the superiority of DBNs over open shortest path first (OSPF) in terms of throughput and average delay per hop. This can be attributed to the reduced overhead as DBNs does not use the traditional rule-based approach. Some of these ideas are extendable to WANETs after careful consideration of the challenges and characteristics of wireless networks.

One of the key challenges that will be faced by IoT devices operating in ad hoc mode is the reliability of routes that can get disconnected due to channel conditions or node failure. The authors of [222] study this problem in the context of multicast routing and apply cerebellar model articulation controller (CMAC) [223]. To ensure reliability, wireless mesh networks need to have the ability to recover from link disruption due to disrupted channel or node failure. The CMAC algorithm was first introduced around the same time that the perceptron algorithm was first introduced. While the CMAC framework can be considered a type of neural network, it is fundamentally different from the ones previously described in this paper. The CMAC architecture can be seen as an attempt to model human associative memory and employs a sort of look-up table technique. The CMAC framework is characterized by a mapping from input space to memory address space (look-up table) and a subsequent mapping from address space to output space. The mapping from input to address space is usually denoted as $S \longrightarrow A$ where S is the input space and A is the address space. Typically, multiple mappings from input to address space are used such that a single input can "activate" multiple addresses in the address space. Each address in the address space contains a weight vector, $\mathbf{w} \in A$, which is used in the subsequent mapping from address space to output space, usually denoted as $A \longrightarrow P$. The function $f : A \longrightarrow P$ is given to be the sum of the weight vectors contained in the activated memory regions. The training of the model can be conducted iteratively over training examples by updating the weight vectors used in the computation of the output by some proportion of the error observed for that training example.

In [222], authors use this concept to learn to estimate the route disconnection expectancy between itself and access points (APs) based on the following three parameters, (i) delay of packets in a node (i.e. sum of queuing delay and processing delay), (ii) number of node disconnections, and (iii) difference in delays between two packets that are separated by a predetermined number of packets. The proposed CMAC uses these three parameters to estimate the node disconnection probability (NDP). Then the NDP estimate enables nodes to predict possible node failure and react faster enabling better throughput, higher packet delivery ratio for multicast packets and provide minimum delay without prior knowledge of the topology.

Q-MAP is another multicast routing algorithm proposed to ensure reliable communication [224]. The algorithm is divided into two phases; in *join query forward* phase nodes use join query packets (JQPs) to explore all the possible routes to the multicast destination and *join reply backward* phase uses join reply packets (JRPs) to establish the optimal route that maximizes the designed Q-value. The JQP can be considered as forwarding agents carrying the possible Q-values downstream, subsequently, JRP packets can be considered as backward agents carrying the optimal decision information upstream to the source.

In traditional unicast routing, the information used to make route decisions (such as resource reservation information, and Q value) are derived from downstream nodes. In contrast, Q-MAP gathers information from the upstream nodes that is used to make the route selection. A simple topology is depicted in Fig. 12 where *src* is the source node and *des* is one of the destinations. In this example, node i needs to choose between j and k as the upstream node. Let us consider node i received a JQP from nodes j and k. Accordingly, node i computes its reinforcement Q-function and resource reservation data. Therefore, in this case, node i updates $Q(i, ux)$ for any such JQP received from any upstream neighbor ux

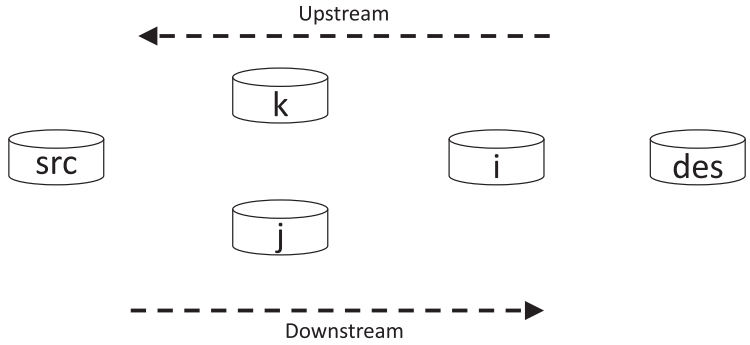

Fig. 12. Framework for power control in cognitive network.

(which in this example is j and k) as follows,

$$Q_t(i, ux) \leftarrow (1-\alpha)Q_{t+1}(i, ux) + \alpha[r + \beta Q_t^*(ux)] \tag{117}$$

Next, node i configures its own JQP and floods the packet downstream. Subsequently, when i receives a JRP from a downstream node (in this scenario *des*), node i will choose an upstream forwarding node that will eventually become part of the optimal route as follows,

$$Q_t^*(i) = \max_{ux} Q_t(i, ux) \;\; \forall \;\; ux \in (i, k) \tag{118}$$

Assuming that j is chosen as the forwarding node, in this case, node i creates its JRP and floods it. Node j receives this JRP and configures itself as the forwarding node for this multicast group by setting its forwarding flag. Each node maintains a forwarding table consisting of a source ID, group ID, forwarding flag indication and a timer field indicating the expiry of the forwarding group. In this manner, the multicast route is selected and maintained by source periodically initiating JQP. If any given receiver does not need to receive from a given source node, it just stops sending JRP for the corresponding multicast group. In this work, the authors do not discuss any experimental results, rather they keep the design general stating that the reward function is designed based on the objective of the network (maximize throughput, minimize energy consumption, minimize latency, etc.) and accordingly the corresponding resource reservation decision taken at each hop can include bandwidth, power or time slot allocation.

An unsupervised learning based routing referred to as Sensor Intelligence Routing (SIR) is proposed in [225]. They modify the Dijkstra's algorithm utilizing SOM. Consider a directed connectivity graph $\mathcal{G}(\mathcal{K}, \mathcal{E})$, where $\mathcal{K} = \{k_0, k_1, \ldots, k_N\}$ is a finite set of nodes, and $(i, j) \in \mathcal{E}$ represents unidirectional wireless link from node k_i to node k_j (for simplicity, they are refer to them as node i and node j). Each edge (i, j) has a score associated with it denoted by γ_{ij} and it is assumed that $\gamma_{ij} = \gamma_{ji}$ which depends on QoS requirements of the network under consideration. In [225], the authors use latency, throughput, error-rate, and duty-cycle to represent a measure of QoS. Accordingly, the authors use these metrics for each link to represent their input of training vectors for a two-layer SOM architecture as shown in Fig. 13. The input layer consists of $l = 4$ neurons, for each input vector of $\mathbf{x}(t) \in \mathcal{R}^l$. The second layer consists of a rectangular grid, where each neuron has l weight vectors connected from the input layer. During the learning phase, competitive learning is used such that the neuron whose vector most closely resembles the current input vector dominates. The SOM clusters these link by assigning each cluster a QoS rating. The learning phase is computationally intensive and hence needs to be performed offline. Meanwhile, execution can run on computational constrained sensor nodes and provide reliable performance as long as the topology and operational characteristics do not change.

The authors compare the proposed solution to Energy-Aware Routing (EAR) [226] and directed diffusion [227]. Directed diffusion is a data-centric routing protocol where the sink first broadcasts a request packet. This is used to set up a gradient (weighted reverse link) pointing to the sink (or the source of request). These gradients are used to find paths which are eventually pruned until the optimal path is determined. In the case of EAR, the source maintains a set of paths chosen by means of a certain probability that is inversely proportional to the energy consumption of that given route. The goal is to distribute traffic over multiple nodes to improve the network lifetime. Simulations show that the advantage of using SIR becomes evident only when nodes in the network start to fail. The parameters used to train SOM enable SIR to choose paths that are less prone to failure thereby providing better delay performance in scenarios where 40% nodes are prone to failure.

An example of RL in geographical routing can be seen in [228]. In Reinforcement Learning based Geographic Routing (RLGR), they proposed a distributed algorithm that utilizes residual energy E_r and location of the neighbors. The MDP is characterized by the state of the packet which is defined by the current node where the packet resides and the action represents the choice of next-hop based on the Q-value ($Q(s, a)$). In this work, the reward function is given by,

$$r = \begin{cases} \alpha\tilde{\delta} + (1-\alpha)\tilde{E}, & \text{if next hop is not sink} \\ R_C, & \text{if next hop is the sink} \\ -R_D, & \text{if no next hop} \\ -R_E, & \text{if next hop available but with low energy} \end{cases} \tag{119}$$

where $\tilde{\delta}$ represents the normalized advance towards the sink, $\tilde{E}$ is the normalized residual energy. The authors consider a constant reward, R_C if the node is able to reach the sink directly. Finally, both R_D and R_E can be considered as the penalty suffered if no next-hop is found or if the existing next-hop has energy below the threshold. The proposed algorithm also uses ϵ to indicate the probability of exploration, i.e. how often the node will choose a random neighbor which may not be the next-hop that has the largest Q-value. For all other occasions (probability of $1-\epsilon$), each node chooses a next-hop that provides the maximum Q-value. Their simulations showed significant improvement in network lifetime comparing RLGR to Greedy Perimeter Stateless Routing (GPSR) [206].

Next, we look at an example beyond RF terrestrial networks. In underwater acoustic networks (UANs), maximizing network lifetime is a key requirement. Accordingly, [229] propose a RL based approach that aims to distribute traffic among sensors to improve the lifetime of the network. In this work, the system state related to a packet is defined as the node that holds the packet. So s_i denotes the state of a packet held by node i. The action taken by node i to forward a packet to node j is denoted as a_j. If this action is successful, the state transitions from s_i to s_j with the

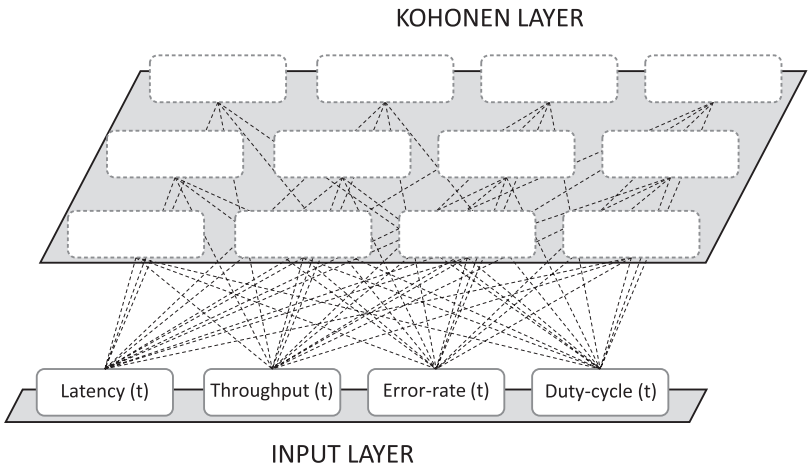

Fig. 13. SOM Architecture used in SIR.

transition probability of P_{ij}^{j} and stays in the same state s_i with transition probability of $P_{ii}^{j} = 1 - P_{ij}^{j}$ if it fails. Though these transition probabilities are unknown, authors argue that this can be estimated at runtime based on history. Accordingly, the overall reward function at time instant t can be defined as follows,

$$r_t = P_{ij}^{j} R_{ij}^{j} + P_{ii}^{j} R_{ii}^{j} \tag{120}$$

where,

$$R_{ij}^{j} = -c - \alpha_1 (E_i + E_j) + \alpha_2 (D_i + D_j) \tag{121}$$

where α_1 and α_2 are tunable weights and c is the constant cost associated with consumption of resource (bandwidth, energy etc.) when a node chooses to transmit. E_i is the cost function associated with residual energy (E_i^{res}) and inital energy (E_i^{ini}). The energy cost function penalizes the system as residual energy decreases and is defined as,

$$E_i = 1 - \frac{E_i^{res}}{E_i^{ini}} \tag{122}$$

Similarly, D_i is defined to measure the energy distribution balance as follows,

$$D_i = \frac{2}{\pi} \arctan(E_i^{res} - \bar{E}_i) \tag{123}$$

where $\bar{E}_i$ is the average residual energy of i and all its direct neighbors. This parameter increases the chance of neighbors with higher residual energy being preferred.

The reward function for the case where a packet forwarding attempt fails is defined as,

$$R_{ii}^{j} = -c - \beta_1 E_i + \beta_2 D_i \tag{124}$$

where β_1 and β_2 are again tunable weights. Authors use Q-learning at each node to enable them to learn about the environment using control packets and take action to improve network lifetime. The proposed solution is shown to outperform the vector-based forwarding protocol [208], a geographical routing protocol designed for UANs by achieving 20% longer lifetime. The authors claim the proposed solution can be applied for various UAN applications by tuning the trade-off between latency and energy efficiency for network lifetime.

Feedback Routing for Optimizing Multiple Sinks (FROMS) [230] is proposed to achieve near-optimal routing from multiple source to multiple sink nodes. The goal of each node is to determine neighbor(s) for next-hop(s) towards the intended subset of sinks $SK_p \subset SK$. The state is defined as a tuple, $S = \{SK_p, H_{SK_p}^{\mathcal{NB}}\}$, where $H_{SK_p}^{\mathcal{NB}}$ is the routing information through all neighboring nodes $\mathcal{NB}$. The action is defined by a set $A_t = \{a_1, a_2, \ldots, a_n\}$, such that $a_i = (nb_i, SK_i)$, where $SK_i \subset SK_p$. The complete action set A must ensure that each sink $sk \in SK_p$ must be considered by exactly one element $a_i \in A$. The Q-value here is defined as follows,

$$Q_t(a) = \left(\sum_{i=1}^{n} Q_t(a_i) \right) - (n-1) \tag{125}$$

where,

$$Q_t(a_i) = \left(\sum_{sk \in SK_i} H_{sk}^{nb_i} \right) - 2(|D_i| - 1) \tag{126}$$

where $H_{sk}^{nb_i}$ is the number of hops to the intended sink $sk \in SK_i$ through neighbor nb_i. $|SK_i|$ denotes the number of sinks in SK_i. The goal of the learning process is to decrease the Q-value as much as possible such that nodes pick the action that corresponds to the lowest Q-value. Accordingly, the reward function is defined as follows,

$$R_t(a_i) = C + \min_a Q(a) \tag{127}$$

where C is the cost of the action. In this manner, the Q-values propagate upstream facilitating the learning process. During the operational phase, it is assumed that nodes overhear neighbor's packets and use the information contained in the packets to update their Q-value. Eventually, the goal is to use routes that will deliver the packets to the desired subset of sinks through the least number of total hops. The authors also explore both greedy exploration and stochastic exploration techniques to avoid local minima. Simulation results validate the ability to learn shared routes to multiple sinks in an efficient manner to decrease the cost per packet compared to directed diffusion [231]. Additionally, they show how exploration can further reduce the cost per packet albeit marginally. We summarize these routing algorithms and the ML techniques they apply in Table 7.

5.3. Open problems and challenges

In this section, we discuss some of the challenges and open problems specifically at the network and data-link layer in the context of IoT.

Table 7
Application of ML in routing protocols.

Routing protocol	ML algorithm	Objective/Comments
Boyan and Littman [218]	RL	Variation of Bellman-Ford proposed for wired network
Mao et al. [221].	DBN	Outperform OSPF due to reduced overhead
Sun et al. [224].	RL	Multicast Routing Algorithm
Pourfakhar and Rahmani [222]	CMAC	Proposed to improve reliability by predicting disconnection probabilities
Barbancho et al. [225].	SOM	Modified version of Dijkstra's Algorithm
Dong et al. [228].	RL	Energy efficient geographical routing
Hu and Fei [229]	RL	Liftime-aware routing for UAN that aims to distribute traffic load among nodes
Forster and Murphy [230]	RF	Near-Optimal routing for multiple source to multiple sinks

5.3.1. Scalability and distributed operation

The exponentially increasing number of IoT devices demand a scalable networking architecture to enable large scale interactions especially in the context of wireless communication. The spectrum congestion will imply more competition for limited resources. While cross-layer approaches [215–217] have been studied in the context of IoT to enable interaction between layers and optimize the utilization of resources, the dimension of the optimization problem space is increasing drastically. This is due to the explosion in operational states (channel, residual energy, traffic level, level of QoS, the density of the neighborhood, the priority of the entity, among others) that must be considered during decision making. This challenge is further exacerbated when a distributed operation is required to reduce the overhead and ensure scalability. In these circumstances, novel ML approaches including DRL needs to be explored in conjunction with network optimization techniques [232,233].

5.3.2. IoT data-link and network layer security

Another key aspect that needs attention at the data link and network layer of the wireless IoT network is the security threat due to various kinds of attacks [234]. In networks like the one established by ZigBee devices, the attacker could eavesdrop and redirect traffic, launching what is known as man-in-the-middle attack [235]. In this attack, the attackers can reduce the performance of the network or even intercept and change the transmitted data. Energy efficiency is a key performance parameter of IoT networks. Keep–Awake attack can be used to drain the battery of IoT devices by sending control packets that constantly revive IoT devices from their dormant sleep cycles [236]. Other attacks at the network layer include selective forwarding and sinkhole (black hole) attack [237,238]. In black hole or sinkhole attack, an attacker's node broadcasts more favorable routes attracting all traffic towards it. Due to the enormous amount of traffic handled by the IoT network it might be challenging to identify such attacks in an efficient and effective manner. The inherent ability of ML to use the "big data" to its advantage can be exploited to explore solutions for these security concerns in IoT networks.

6. Spectrum sensing and hardware implementation

One of the key challenges in enabling real-time inference from spectrum data is how to *effectively* and *efficiently* extract *meaningful* and *actionable* knowledge out of the tens of millions of I/Q samples received every second by wireless devices. Indeed, a single 20 MHz-wide WiFi channel generates an I/Q stream rate of about 1.28 Gbit/s, if I/Q samples are each stored in a 4-byte word. Moreover, the RF channel is significantly time-varying (*i.e.*, in the order of milliseconds), which imposes strict timing constraints on the *validity* of the extracted RF knowledge. If (for example) the RF channel changes every 10ms, a knowledge extraction algorithm must run with latency (much) less than 10ms to both (i) offer an accurate RF prediction and (ii) drive an appropriate physical-layer response; for example, change in modulation/coding/beamforming vectors due to adverse channel conditions, local oscillator (LO) frequency due to spectrum reuse, and so on.

As discussed earlier, DL has been a prominent technology of choice for solving classification problems for which no well-defined mathematical model exists. It enables the analysis of unprocessed I/Q samples without the need of application-specific and computational-expensive feature extraction and selection algorithms [143], thus going far beyond traditional low-dimensional ML techniques. Furthermore, DL architectures are application-insensitive, meaning that the same architecture can be retrained for different learning problems.

Decision-making at the physical layer may leverage the spectrum knowledge provided by DL. On the other hand, RF DL algorithms must execute in *real-time* (*i.e.*, with static, known-a-priori latency) to achieve this goal. Traditional central processing unit (CPU)-based knowledge extraction algorithms [239] are unable to meet strict time constraints, as general-purpose CPUs can be interrupted at-will by concurrent processes and thus introduce additional latency to the computation. Moreover, transferring data to the CPU from the radio interface introduces unacceptable latency for the RF domain. Finally, processing I/Q rates in the order of Gbit/s would require CPUs to run continuously at maximum speed, and thus consume enormous amounts of energy. For these reasons, RF DL algorithms must be closely integrated into the RF signal processing chain of the embedded device.

6.1. Existing work

Most of existing work is based on traditional low-dimensional machine learning [240–244], which requires (i) extraction and careful selection of complex features from the RF waveform (*i.e.*, average, median, kurtosis, skewness, high-order cyclic moments, etc.); and (ii) the establishment of tight decision bounds between classes based on the current application, which are derived either from mathematical analysis or by learning a carefully crafted dataset [245]. In other words, since feature-based machine learning is (a) significantly application-specific in nature; and (b) it introduces additional latency and computational burden due to feature extraction, its application to real-time hardware-based wireless spectrum analysis becomes impractical, as the wireless radio hardware should be changed according to the specific application under consideration.

Recent advances in DL [246] have prompted researchers to investigate whether similar techniques can be used to analyze the sheer complexity of the wireless spectrum. For a compendium of existing research on the topic, the reader can refer to [247]. Among other advantages, DL is significantly amenable to be used for real-time hardware-based spectrum analysis, since different model architectures can be reused to different problems as long as weights and hyper-parameters can be changed through software.

Additionally, DL solutions to the physical layer modulation recognition task have been given much attention over recent years, as previously discussed in this work. The core issue with existing approaches is that they leverage DL to perform offline spectrum analysis only. On the other hand, the opportunity of real-time hardware-based spectrum knowledge inference remains substantially uninvestigated.

6.2. Background on system-on-chip computer architecture

Due to its several advantages, we contend that one of the most appropriate computing platform for RF DL is a system on chip (SoC). An SoC is an integrated circuit (also known as "IC" or "chip") that integrates all the components of a computer, *i.e.*, CPU, random access memory (RAM), input/output (I/O) ports and secondary storage (*e.g.*, SD card) – all on a single substrate [248]. SoCs have low power consumption [249] and allow the design and implementation of *customized hardware* on the field-programmable gate array (FPGA) portion of the chip, also called programmable logic (PL). Furthermore, SoCs bring unparalleled flexibility, as the PL can be reprogrammed at-will according to the desired learning design. The PL portion of the SoC can be managed by the processing system (PS), *i.e.*, the CPU, RAM, and associated buses.

SoCs use the Advanced eXtensible Interface (AXI) bus specification [250] to exchange data (i) between functional blocks inside the PL; and (ii) between the PS and PL. There are three main AXI sub-specifications: *AXI-Lite, AXI-Stream* and *AXI-Full*. AXI-Lite is a lightweight, low-speed AXI protocol for register access, and it is used to configure the circuits inside the PL. AXI-Stream is used to transport data between circuits inside the PL. AXI-Stream is widely used, since it provides (i) standard inter-block interfaces; and (ii) rate-insensitive design, since all the AXI-Stream interfaces share the same bus clock, the high-level synthesis (HLS) design tool will handle the handshake between DL layers and insert FIFOs for buffering incoming/outgoing samples. AXI-Full is used to enable burst-based data transfer from PL to PS (and *vice versa*). Along with AXI-Full, direct memory access (DMA) is usually used to allow PL circuits to read/write data obtained through AXI-Stream to the RAM residing in the PS. The use of DMA is crucial since the CPU would be fully occupied for the entire duration of the read/write operation, and thus unavailable to perform other work.

6.3. A design framework for real-time RF deep learning

One of the fundamental challenges to be addressed is how to transition from a software-based DL implementation (*e.g.*, developed with the Tensorflow [239] engine) to a hardware-based implementation on an SoC. Basic notions of high-level synthesis and a hardware design framework are presented in Sections 6.3.1 and 6.3.2, respectively.

6.3.1. High-level synthesis

HLS is an automated design process that interprets an algorithmic description of a desired behavior (*e.g.*, C/C++) and creates a model written in hardware description language (HDL) that can be executed by the FPGA and implements the desired behavior [251]. Designing digital circuits using HLS has several advantages over traditional approaches. First, HLS programming models can implement almost any algorithm written in C/C++. This allows the developer to spend less time on the HDL code and focus on the algorithmic portion of the design, and at the same time avoid bugs and increase efficiency, since HLS optimizes the circuit according to the system specifications. The clock speed of today's FPGAs is several orders of magnitude slower than CPUs (*i.e.*, up to 200–300 MHz in the very best FPGAs). Thus, parallelizing the circuit's operations is crucial. In traditional HDL, transforming the signal processing algorithms to fit FPGA's parallel architecture requires challenging programming efforts. On the other hand, an HLS toolchain can tell how many cycles are needed for a circuit to generate all the outputs for a given input size, given a target parallelization level. This helps to reach the best trade-off between hardware complexity and latency.

Loop pipelining: In high-level languages (such as C/C++), the operations in a loop are executed sequentially and the next iteration of the loop can only begin when the last operation in the current loop iteration is complete. Loop pipelining allows the operations in a loop to be implemented in a concurrent manner.

Fig. 14 shows an example of loop pipelining, where a simple loop of three operations, *i.e.*, read (RD), execute (EX), and write (WR), is executed twice. For simplicity, we assume that each operation takes one clock cycle to complete. Without loop pipelining, the loop would take 6 clock cycles to complete. Conversely, with loop pipelining, the next RD operation is executed concurrently to the EX operation in the first loop iteration. This brings the total loop latency to 4 clock cycles. If the loop length were to increase to 100, then the latency decrease would be even more evident: 300 *versus* 103 clock cycles, corresponding to a speedup of about 65%. An important term for loop pipelining is called initiation interval (II), which is the number of clock cycles between the start times of consecutive loop iterations. In the example of Fig. 14, the II is equal to one, because there is only one clock cycle between the start times of consecutive loop iterations.

Loop unrolling: Loop unrolling creates multiple copies of the loop body and adjusts the loop iteration counter accordingly. For example, if a loop is processed with an unrolling factor (UF) equal to 2 (*i.e.*, two subsequent operations in the same clock cycle as shown in Fig. 15), it may reduce a loop's latency by a factor of 50%, since a loop will execute in half the iterations usually needed. Higher UF and II may help achieve low latency, but at the cost of higher hardware resource consumption. Thus, the trade-off between latency and hardware consumption should be thoroughly explored.

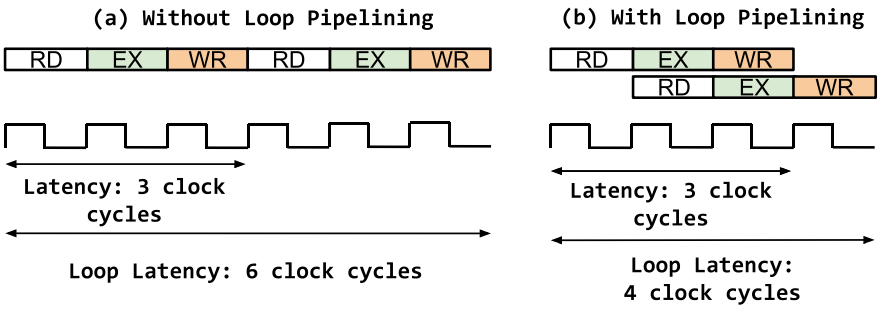

Fig. 14. Loop pipelining.

```
for(int i = 0; i < 10; i++) {
    sum += a[i];
}

for(int i = 0; i < 10; i+=2) {
    sum += a[i];
    sum += a[i+1];
}
```

Fig. 15. Loop unrolling.

6.3.2. Design steps

Our framework presents several design and development steps, which are illustrated in Fig. 16. Steps that involve hardware, middleware (*i.e.*, hardware description logic, or HDL), and software have been depicted with a blue, red, and green shade, respectively.

The first major step of the framework is to take an existing DL model and convert the model in HLS language, so it can be optimized and later on synthesized in hardware. Another critical challenge is how to make the hardware implementation fully reconfigurable, *i.e.*, the weights of the DL model may need to be changed by the *Controller* according to the specific training. To address these issues, we distinguish between (i) the DL model architecture, which is the set of layers and hyper-parameters that compose the model itself, and (ii) the parameters of each layer, *i.e.*, the neurons' and filters' weights.

To generate the HLS code describing the software-based DL model, an *HLS Library*, which provides a set of HLS functions that parse the software-based DL model architecture and generates the HLS design corresponding to the desired architecture. The *HLS Library* supports the generation of convolutional, fully-connected, rectified linear unit, and pooling layers, and operated on fixed-point arithmetic for better latency and hardware resource consumption. The HLS code is subsequently translated to HDL code by an automated tool that takes into account optimization directives such as loop pipelining and loop unrolling. At this stage, the HDL describing the DL core can be simulated to (i) calculate the amount of PL resources consumed by the circuit (*i.e.*, flip-flops, BRAM blocks, etc); and (ii) estimate the circuit latency in terms of clock cycles. After a compromise between space and latency as dictated by the application has been found, the DC core can be synthesized and integrated with the other PL components, and thus total space constraints can be verified. After implementation (*i.e.*, placing/routing), the PL timing constraints can be verified, and finally the whole system can be deployed on the SoC and its functionality tested.

6.4. Open problems and challenges

In this section, we discuss a set of open challenges overcoming which will accelerate the induction of ML techniques to the IoT hardware especially in the context of spectrum sensing.

6.4.1. Lack of large-scale wireless signal datasets

It is well known that learning algorithms require a considerable amount of data to be able to effectively learn from a training dataset. Moreover, to compare the performance of different learning models and algorithms, it is imperative to use the same sets of data. More mature learning fields, such as computer vision and natural language processing (NLP) already have standardized datasets for these purposes [252,253]. However, literature still lacks large-scale datasets for RF ML.

This is not without a reason. Although the wireless domain allows the synthetic generation of signals having the desired characteristics (*e.g.*, modulation, frequency content, and so on), problems such as RF fingerprinting and jamming detection require data that captures the unique characteristics of devices and wireless channels. Therefore, significant research effort must be put forth to build large-scale wireless signal datasets to be shared with the research community at large.

6.4.2. Choice of I/Q data representation format

It is still subject of debate within the research community what is the best data representation for RF deep learning applications. For example, an I/Q sample can be represented as a tuple of real numbers or a single complex number, while a set of I/Q samples can be represented as a matrix or a single set of numbers represented as a string. It is a common belief that there is no one-size-fits-all data representation solution for every learning problem, and that the right format might depend, among others, on the learning objective, choice of the loss function, and the learning problem considered [143].

6.4.3. Choice of learning model and architecture

While there is a direct connection between images and tensors, the same cannot be concluded for wireless signals. For example, while 3-D tensors have been proven to effectively model images (*i.e.*, red, green, and blue channels), and kernels in convolutional layers are demonstrably powerful tools to detect edges and contours in a given image, it is still unclear if and how these concepts

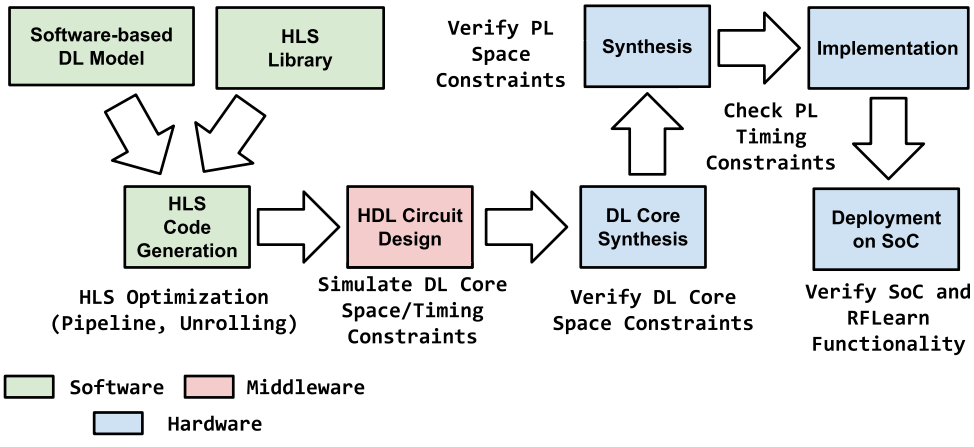

Fig. 16. Ahardware design framework for RF deep learning.

can be applied to wireless signals. Another major difference is that, while images can be considered as stationary data, RF signals are inherently stochastic, non-stationary and time-varying. This peculiar aspect poses significant issues in determining the right learning strategy in the wireless RF domain. For example, while CNN seems to be able to effective at solving problems such as modulation recognition [143,146,148], it is still unclear if this is the case for complex problems such as RF fingerprinting. Moreover, DL has traditionally been used in static contexts [254,255], where the model latency is usually not a concern. Another fundamental issue absent in traditional deep learning is the need to satisfy strict constraints on resource consumption. Indeed, models with a high number of neurons/layers/parameters will necessarily require additional hardware and energy consumption, which are clearly scarce resources in embedded systems. Particular care must be devoted, therefore, when designing learning architectures to solve learning problems in the RF domain.

7. Machine learning in IoT beyond communication

The core objective of this work is to provide a comprehensive account of the applications of ML for communication in IoT. In this section, for the benefit of readers who might be exploring ML for communication in conjunction with other IoT related areas of research, we provide a brief discussion on how ML has been applied to areas like security (beyond communication surfaces) and big data analysis. This is not intended to be as comprehensive as the earlier sections of this survey but provides the adequate resources for readers to understand the broad nature of ML being applied in these areas by pointing them to the relevant resources.

7.1. Security in IoT

Due to the complex and integrative arrangement of IoT devices, it can be prone to a wide range of attacks. Limited computation and power resources, a wide range of accessibility, and a large amount of data being handled leads to challenging circumstances to defend IoT devices from security threats. The interdependent and interconnected environment in which IoT devices operate leads to vast numbers of attack surfaces to monitor and manage.

ML has been leveraged as a powerful tool that can monitor the vast number of IoT devices to detect and alert operators of imminent security threats [256]. One of the key security concern in an IoT network is the presence of intruders that may induce malicious behavior. Several of the ML techniques have been used to detect these forms of attacks. In one of the earliest works [257], the author proposed a robust SVM-based solution to intruder detection. This involved analyzing 1998 DARPA Basic Security Module data set collected at MIT's Lincoln Labs. Recently, a SVM-based hybrid detection method that integrates the misuse detection model and an anomaly detection model has been proposed in [258]. The solution is shown to be computationally efficient, and capable of providing better detection rate for both known and unknown attacks while maintaining a low probability of false alarm. RNN, specifically LSTM have been proposed as an effective tool to detect malicious activity [259] especially for time-series based threats.

Since IoT devices are often connected to Android-based mobile devices in order to enable remote control and configuration, there has been a growth in malware developers. Malware enables developers to control compromised devices to extract private user information or constructing botnets. Several ML techniques have been applied to detect such malware attack. Few examples include a SVM-based malware detection to ensure reliable IoT services [260], malware detection using CNN [261] and an autoencoder-based approach [262].

7.2. Big data analytics

The large amount of data generated and/or flowing through IoT devices have been referred to as smart data [263,264] and have been used to feed various ML tools to enable several applications in traffic, energy management, health, environment, homes, agriculture, among others. The analysis of data can happen data centers (cloud computing) [265], edge devices (edge computing) [266] or edge servers (fog computing) [267] based on the computation requirement, acceptable latency, among other factors.

To identify regular traffic patterns, authors of [268] employs DBSCAN algorithm to analyze various trips using the operator's smart card to detect regular travel patterns and then use K-Means algorithm to classify these travel patterns. This information can then be utilized for city planning and identify the optimal use of budget to add critical infrastructure. In [269], an example of using IoT data in predictive analysis for enhanced decision making has been provided. In this particular example, the authors' goal was to predict energy usage of a building using four ML models used by the WEKA data mining software [270] which includes SVM for regression, two ANN architectures, and linear regression.

Another example of ML being applied to classify big data is provided in [271]. Here, the authors provide a hybrid (unsupervised and supervised learning) solution to classify the multi-variate time series sensor data that includes environmental variables viz. temperature, humidity, light, and voltage. The authors first apply simple aggregation approximation (SAX) representation to the data in order to reduce its dimensions. Next, clustering techniques are applied to learn the target classes and SVM was used thereafter to perform classification.

Management of a large number of IoT devices is also becoming a challenging task taking into consideration the limited resources each of these devices house. Operational indicators of IoT devices that represent the reliability, QoS, productivity, etc. are received from the management protocols. The set of these values are referred to as the state of the IoT device in [272]. To enable better management and mitigate the problems arising from inefficient use of limited resources, the authors propose an ANN-based framework that enables prediction of IoT device state enabling higher efficiency in their decision making process for a wide variety of applications. These are just a subset of applications where data analytics has been exploited using ML. Big data analytics will also find its application in health care, education, smart grid as well as other components forming a smart city.

7.3. Open problems and challenges

7.3.1. Data analytics

The quality of data is a key factor affecting the efficacy of the ML techniques applied for data analytics. The quality of sensors, the environmental condition, protocols and hardware employed along with several other factors may affect the quality of the data generated by IoT devices. This along with the fact that this data is produced in high volume, high velocity and its nature vary based on devices, application, and protocols used by these devices. It becomes an extremely challenging task to assess the quality of incoming data. The computational load required to analyze the data for quality, pre-process to enhance the data and subsequently perform application-specific data analysis in real-time will continue to be a daunting problem as the IoT revolution grows exponentially.

Beyond the quality and computational requirement of handling the data, the overarching legal and ethical concerns of handling data emanating from various IoT devices will also have to be explored in greater depth. It will be challenging to reach consensus in defining the optimal procedure/methodology of handling critical data regarding health, law enforcement or national security

among others. The same data that is collected in a different context may directly impact is accessibility and sensitivity. In certain cases, the location where the data is stored (data centers) for computation can be critical to the application of the agency that generates the data. This may induce further constraints on the computational requirements. Since there is a never-ending struggle between the need of applying a centralized form of secure data handling while requiring more scalable, distributed, and low overhead operations, there will always be open challenges to determine the Pareto-optimal solutions to handle the vast amount of IoT data.

7.3.2. Security

Most ML techniques employed to enhance the security of IoT that rely on supervised learning are predominantly trained using simulated or emulated data. This is due to the fact that it is very challenging to gather training data that has been obtained during real-world attack scenarios. An important research direction is to cooperatively obtain crowd-sourced data set from IoT deployed by different commercial, government and academic entities. This again will be a challenge due to the privacy, propriety and other regulatory, proprietary concerns discussed in the earlier section. Assuming this will be a difficult task in the near future, significant research will be required to design ML techniques that can provide adequate real-world protection even when trained on emulated/simulated data sets.

The next-generation of ML-based solution needs to be able to adapt to the ever-changing landscape of the attack methodologies. Signature-based malware detection may be unable to detect zero-day attack or malware that evolve continuously as in case of metamorphic and polymorphic malware. A new emerging threat that was previously unknown to the malware detector is referred to as Zero-day attack. Though there have been recent efforts to tackle these problems [273,274], there is a significant opportunity to employ ML to mitigate or eradicate the damages caused by ever-evolving security threats.

8. Conclusion

This paper provides a comprehensive account of advances in IoT wireless communication made possible by the application of ML. To accomplish this, we first provide readers with a detailed overview of some of the most prevalent ML techniques that are employed in wireless communication networks. We have done so with the hope that by elucidating the inner workings of some of the ML algorithms relevant to communication in the IoT, we have not only enabled the reader to understand the subsequent text at a deeper level but inspired other researchers to apply the techniques discussed to their own problems in IoT communication. To a lesser degree, we have written the overview with the intent of providing a light foray into ML for the unfamiliar reader. While it is not an all-encompassing field guide to ML, the overview covers many of the popular algorithms from the different sub-fields of ML and aims to provide an intuition surrounding their use.

Next, we presented an overview of the current state-of-the-art of IoT communication, the standardization efforts, challenges and how CR aspect along with ML approaches are exploited to address some of these. CR along with ML is a powerful tool that can take the IoT technologies a step forward in mitigating the myriad problems that arise from large deployments. We provided a glimpse of the subset of works proposed in realizing the IoT vision for the foreseeable future dense, large scale IoT deployment. The COGNICOM+ framework is inspiring but has a long development road ahead to realize the plethora of approaches presented from designing ASIC-based CNN accelerators to developing the fully realized COGNICOM+. Recent works have taken algorithmic designs from simulations to real testbed implementations. More such works are essential to realize the challenges and pave way for future CR-IoT. However, the scalability of such a centralized solution might be challenging for a large and dense deployment. The big data analytics and management will need to be addressed for such centralized approaches when applied to dense deployment. Instilling ML techniques for future CR-IoT enables intelligent resource management such as radio resource optimization via intelligent beamforming, channel equalization, adaptive power and rate control, spectrum allocation and management. Conventional techniques involve optimization techniques performed in an offline/semi-offline manner but ML enables such optimization to be performed in an online fashion in real-time. ML approaches continue to learn and adapt to the varying parameters improving the cognition of the system. Such intelligent online decision making will best fit the future CR-IoT.

Following the discussion of the application of ML to problems in the physical layer, we introduce the use of ML techniques for signal intelligence tasks in the realm of the IoT. We describe how ML, and often DNNs, can be used to enhance the efficacy of the discriminative classification tasks of AMC and wireless interference classification. The common narrative underlying the presentation of these tasks and their respective solutions is that hand-crafted feature-based classifiers of old are outperformed by their DNN counterparts. Not only do the ML and DL solutions presented in this section improve upon classification accuracies, but they also allow for a model to be learned directly on the raw signal representation. The advantages of such a result are two-fold. First, learning a model that operates directly on the raw signal reduces the need for preprocessing of the data, in turn reducing latency and computational load, both of which often have stringent constraints in IoT networks. Second, the use of hand-crafted signal features limits the model's ability to adapt to new input, thus reducing the applicability of the learned model to new data sets. The raw signal representation is the most information-rich representation of the signal and thus reducing it to a set of hand-crafted features reduces the information content. The crux of DL is to allow the algorithm to determine what aspects of and interactions between the data are important for a given task, and thus providing the algorithm with more information (raw signal) allows for a more versatile model. This is important with respect to the IoT as the wireless networks, communication protocols, and RF signals that arise in the IoT are not uniform, placing a premium on solutions that are easily adaptable to new scenarios and problem formulations. Such is the reason motivating the use of ML in signal intelligence problems within the IoT.

Thereafter, we detail the increasing relevance of these techniques in the higher layers of the protocol stack enabling optimized utilization of limited resources which will be key to support the rapid growth of IoT devices. Deploying a dense IoT network may rely on TDMA to broadcast information to each other. In these scenarios, the BSP is essentially a TDMA cycle minimization problem which is known to be NP-complete. In this work, we have seen how ML techniques have been successfully applied to these NP-complete problems which otherwise is challenging to overcome. While some of the solutions designed to overcome BSP provided acceptable results they unfortunately required long computational time to reach the solution. By Applying FHNN to solve BSP, the problem was formulated as one that aims at minimizing the energy function associated with FHNN. This approach outperformed the existing methodologies in terms of convergence rate. This is one example where ML is applied to an intractable problem of wireless communication which in this case was to determine the non-conflicting transmission schedule that maximizes the utilization of the channel. Another key application of ML during medium access is its ability to sense the spectrum and provide insight into the IoT devices regarding possible active attacks. This is then

leveraged by the decision engine to determine appropriate responses to mitigate the attack or alert the presence of a malicious entity in the spectrum of interest.

RL becomes an excellent candidate to enable DSA and other cognitive radio solutions because of the inherent nature of the problem that can be modeled as MDP. Thereafter, Q-learning or even DQN (for large state-action space) can be used to determine optimal action for a given state of the agent (transceiver). These models can be used by the data link layer for power control, negotiating spectrum access and to determining optimal transmission strategies. Similarly, at the network layer, Q-learning is used in varying traffic loads to handle congestion and QoS requirements, optimize network parameters like delay, throughput, fairness, and energy efficiency. In contrast to traditional approaches, ML has also been used to predict route failures enabling more rapid recovery process which can be critical to large distributed IoT networks. A key point to remember in the context of feasibility is that in many cases the learning phase might be computationally intensive and is performed offline. On the other hand, the execution itself can be light-weight thereby making ML based approaches more feasible for IoT devices. Realizing the importance of extending these techniques to hardware implementation, we discuss some steps that can be taken in those directions to ensure a rapid transition of these techniques to commercial hardware.

Finally, we have also looked at a couple of key areas beyond communication where ML is being leveraged as an effective tool in the realm of IoT. Various supervised ML techniques are being employed to detect intruders and malicious behaviors which can be a key application given the risk of such attack on IoT devices. This is usually possible by analyzing the large amount of data associated with IoT. Furthermore, we have also presented some recent efforts of where big data analytics has been performed using ML as it is a significant emerging and motivating factor in the current surge of IoT. The overarching goal of this paper is to enable researchers with the fundamental tool to understand the application of ML in context of wireless communication in the IoT and apprise them of the latest advancements that will, in turn, motivate new and exciting works.

Conflict of interest

None.

References

[1] K. Ashton, That 'Internet of Things' thing, RFID J. (2009).
[2] A. Whitmore, A. Agarwal, L. Da Xu, The internet of things – a survey of topics and trends, Inf. Syst. Front. 17 (2) (2015) 261–274.
[3] Glen Martin (Forbes), How the internet of things is more like the industrial revolution than the digital revolution(https://www.forbes.com/sites/oreillymedia/2014/02/10/more-1876-than-1995/#674c4e0b66d2).
[4] L. Da Xu, W. He, S. Li, Internet of things in industries: a survey, IEEE Trans. Ind. Inf. 10 (4) (2014) 2233–2243.
[5] Ericsson Incorporated, Ericssoninterim mobility report, February 2018, 2018, (https://www.ericsson.com/assets/local/mobility-report/documents/2018/emr-interim-feb-2018.pdf).
[6] Cisco Systems, Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021 White Paper, 2017, (http://tinyurl.com/zzo6766).
[7] Federal Communications Commission [2016], Spectrum crunch(https://www.fcc.gov/general/spectrum-crunch).
[8] H. Shokri-Ghadikolaei, F. Boccardi, C. Fischione, G. Fodor, M. Zorzi, Spectrum sharing in mmwave cellular networks via cell association, coordination, and beamforming, IEEE J. Sel. Areas Commun. 34 (11) (2016) 2902–2917.
[9] M.A. Vázquez, L. Blanco, A.I. Pérez-Neira, Hybrid analog–digital transmit beamforming for spectrum sharing backhaul networks, IEEE Trans. Signal Process. 66 (9) (2018) 2273.
[10] L. Lv, J. Chen, Q. Ni, Z. Ding, H. Jiang, Cognitive non-orthogonal multiple access with cooperative relaying: a new wireless frontier for 5g spectrum sharing, IEEE Commun. Mag. 56 (4) (2018) 188–195.
[11] X. Jin, J. Sun, R. Zhang, Y. Zhang, C. Zhang, Specguard: spectrum misuse detection in dynamic spectrum access systems, to appear, IEEE Trans. Mob. Comput. (2018).
[12] T.M. Chiwewe, G.P. Hancke, Fast convergence cooperative dynamic spectrum access for cognitive radio networks, IEEE Trans. Ind. Inf. (2017).
[13] J. Jagannath, S. Furman, T. Melodia, A. Drozd, Design and experimental evaluation of a cross-layer deadline-based joint routing and spectrum allocation algorithm, IEEE Trans. Mob. Comput. (2018), doi:10.1109/TMC.2018.2866093. 1–1
[14] Federated Wireless, Citizens Broadband Radio Service (CBRS)shared spectrum: an overview, 2018, (https://www.federatedwireless.com/wp-content/uploads/2017/09/CBRS-Spectrum-Sharing-Overview.pdf).
[15] S. Agarwal, S. De, eDSA: energy-efficient dynamic spectrum access protocols for cognitive radio networks, IEEE Trans. Mob. Comput. 15 (12) (2016) 3057–3071.
[16] L. Zhang, F. Restuccia, T. Melodia, S. Pudlewski, Learning to detect and mitigate cross-layer attacks in wireless networks: framework and applications, in: Proc. of IEEE Conf. on Communications and Network Security, 2017. Las Vegas, NV, USA
[17] J.-F. Huang, G.-Y. Chang, J.-X. Huang, Anti-jamming rendezvous scheme for cognitive radio networks, IEEE Trans. Mob. Comput. 16 (3) (2017) 648–661.
[18] G.-Y. Chang, S.-Y. Wang, Y.-X. Liu, A jamming-resistant channel hopping scheme for cognitive radio networks, IEEE Trans. Wirel. Commun. 16 (10) (2017) 6712–6725.
[19] M. Bkassiny, Y. Li, S.K. Jayaweera, A survey on machine-learning techniques in cognitive radios, IEEE Commun. Surv. Tutor. 15 (3) (2013) 1136–1159, doi:10.1109/SURV.2012.100412.00017.
[20] C. Jiang, H. Zhang, Y. Ren, Z. Han, K.C. Chen, L. Hanzo, Machine learning paradigms for next-generation wireless networks, IEEE Wirel. Commun. 24 (2) (2017) 98–105, doi:10.1109/MWC.2016.1500356WC.
[21] M. Chen, U. Challita, W. Saad, C. Yin, M. Debbah, Machine learning for wireless networks with artificial intelligence: a tutorial on neural networks, 2017. arXiv: 1710.02913.
[22] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, 1, MIT Press, 2016.
[23] K.P. Murphy, Machine Learning: A Probabilistic Perspective, The MIT Press, 2012.
[24] C.M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), Springer-Verlag, Berlin, Heidelberg, 2006.
[25] T.M. Mitchell, Machine Learning, McGraw Hill series in computer science, McGraw-Hill, 1997.
[26] F. Rosenblatt, The Perceptron, a Perceiving and Recognizing Automaton Project Para, Report: Cornell Aeronautical Laboratory, Cornell Aeronautical Laboratory, 1957.
[27] V. Vapnik, A. Lerner, Pattern recognition using generalized portrait method, Autom. Remote Control 24 (1963).
[28] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (3) (1995) 273–297, doi:10.1023/A:1022627411411.
[29] F. Rosenblatt, Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms, Report (Cornell Aeronautical Laboratory), Spartan Books, 1962.
[30] D. Broomhead, D. Lowe, Radial basis functions, multi-variable functional interpolation and adaptive networks, Royal Signals and Radar Establishment Malvern (United Kingdom), RSRE-MEMO-4148, 1988.
[31] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back--propagating errors, Nature 323 (1986) 533–536.
[32] Y. LeCun, et al., Generalization and network design strategies, Connect.Perspect. (1989) 143–155.
[33] Y.T. Zhou, R. Chellappa, Computation of optical flow using a neural network, in: IEEE 1988 International Conference on Neural Networks, 1988, pp. 71–78.
[34] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, 2015. arXiv: 1512.03385.
[35] R.K. Srivastava, K. Greff, J. Schmidhuber, Highway networks, 2015. arXiv: 1505.00387.
[36] G. Huang, Z. Liu, K.Q. Weinberger, Densely connected convolutional networks, 2016. arXiv: 1608.06993.
[37] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.
[38] P.J. Werbos, Backpropagation through time: what it does and how to do it, Proc. IEEE 78 (10) (1990) 1550–1560, doi:10.1109/5.58337.
[39] R. Pascanu, Ç. Gülçehre, K. Cho, Y. Bengio, How to construct deep recurrent neural networks, 2013. arXiv: 1312.6026.
[40] M. Schuster, K.K. Paliwal, Bidirectional recurrent neural networks, IEEE Trans. Signal Process. 45 (11) (1997) 2673–2681, doi:10.1109/78.650093.
[41] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780, doi:10.1162/neco.1997.9.8.1735.
[42] K. Cho, B. van Merrienboer, D. Bahdanau, Y. Bengio, On the properties of neural machine translation: encoder-Decoder approaches, ArXiv e-prints (2014).
[43] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, Proc. Natl. Acad. Sci. 79 (8) (1982) 2554–2558, doi:10.1073/pnas.79.8.2554.
[44] D. Hebb, The organisation of behaviour.(1949).
[45] S. Lloyd, Least squares quantization in pcm, IEEE Trans. Inf. Theory 28 (2) (1982) 129–137, doi:10.1109/TIT.1982.1056489.
[46] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise, in: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, in: KDD'96, AAAI Press, 1996, pp. 226–231.

[47] Y. Lecun, PhD thesis: Modeles connexionnistes de l'apprentissage (connectionist learning models), Universite P. et M. Curie (Paris 6), 1987.
[48] T. Kohonen, Self-organized formation of topologically correct feature maps. biological cybernetics, Biol. Cybern. (1982) 59–69.
[49] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction., MIT Press, Cambridge, MA, 1998.
[50] C.J.C.H. Watkins, Learning from Delayed Rewards, King's College, Oxford, 1989 Ph.D. thesis.
[51] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M.A. Riedmiller, Playing atari with deep reinforcement learning, 2013. arXiv: 1312.5602.
[52] R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Mach. Learn. 8 (3) (1992) 229–256, doi:10.1007/BF00992696.
[53] A.A. Khan, M.H. Rehmani, A. Rachedi, Cognitive-radio-based internet of things: applications, architectures, spectrum related functionalities, and future research directions, IEEE Wirel. Commun. 24 (3) (2017) 17–25, doi:10.1109/MWC.2017.1600404.
[54] Q. Wu, G. Ding, Y. Xu, S. Feng, Z. Du, J. Wang, K. Long, Cognitive internet of things: a new paradigm beyond connection, IEEE Internet Things J. 1 (2) (2014) 129–143, doi:10.1109/JIOT.2014.2311513.
[55] P. Rawat, K.D. Singh, J.M. Bonnin, Cognitive radio for M2M and internet of things: a survey, Comput. Commun. 94 (2016) 1–29, doi:10.1016/j.comcom.2016.07.012.
[56] ZigBee Alliance, ZigBee specification, 2008, (http://www.zigbee.org).
[57] D. Minoli, IPv6 Over low power WPAN (6Lowpan), pp. 293–301. 10.1002/9781118647059.ch9
[58] Specification of the Bluetooth System, Covered Core Package, Version: 4.0, The Bluetooth Special Interest Group (2010).
[59] The EPCGlobal Architecture Framework. EPCglobal Final Version 1.3, 2009.
[60] IEC 62591 Ed. 1.0 b:2010, Industrial Communication Networks - Wireless Communication Network and Communication Profiles - WirelessHART™(2010).
[61] Wireless Systems for Industrial Automation: Process Control and Related Applications, ISA Standard ISA-100.11a-2011(2009).
[62] D. Flowers, Y. Yang, Microchip MiWi wireless networking protocol stack, Microchip Technol. (2010).
[63] Ericsson, Cellular Networks for Massive IoT- Enabling Low Power Wide Area Applications, Stockholm, Sweden,pp. 1-13(2016).
[64] A.A. Khan, M.H. Rehmani, A. Rachedi, Whencognitive radio meets the internet of things? in: Proc. of International Wireless Communications and Mobile Computing Conference (IWCMC), 2016, pp. 469–474, doi:10.1109/IWCMC.2016.7577103.
[65] S. Huang, X. Liu, Z. Ding, Opportunisticspectrum access in cognitive radio networks, in: Proc. of IEEE International Conference on Computer Communications (INFOCOM), 2008, pp. 1427–1435, doi:10.1109/INFOCOM.2008.201.
[66] A.W. Min, K. Kim, J. Pal Singh, K.G. Shin, Opportunistic spectrum access for mobile cognitive radios, in: Proc. of IEEE International Conference on Computer Communications (INFOCOM), 2011, pp. 2993–3001, doi:10.1109/INFCOM.2011.5935141.
[67] M. Song, C. Xin, Y. Zhao, X. Cheng, Dynamic spectrum access: from cognitive radio to network radio, IEEE Wirel. Commun. 19 (1) (2012) 23–29, doi:10.1109/MWC.2012.6155873.
[68] Y. Zhang, Dynamic spectrum access in cognitive radio wireless networks, in: Proc. of IEEE International Conference on Communications (ICC), 2008, pp. 4927–4932, doi:10.1109/ICC.2008.923.
[69] J. Jagannath, T. Melodia, A. Drozd, DRS: distributed deadline-based joint routing and spectrum allocation for tactical ad-hoc networks, in: Proc. of IEEE Global Communications Conference (GLOBECOM), 2016. Washington D.C., USA
[70] T. Li, J. Yuan, M. Torlak, Network throughput optimization for random access narrowband cognitive radio internet of things (NB-CR-Iot), IEEE Internet Things J. 5 (3) (2018) 1436–1448, doi:10.1109/JIOT.2017.2789217.
[71] V.T. Nguyen, N. Nguyen-Thanh, L. Yang, D.H.N. Nguyen, C. Jabbour, B. Murmann, Cognitive computation and communication: acomplement solution to cloud for IoT, in: Proc. of International Conference on Advanced Technologies for Communications (ATC), 2016, pp. 222–230, doi:10.1109/ATC.2016.7764778.
[72] T. Tholeti, V. Raj, S. Kalyani, A non-parametric multi-stage learning framework for cognitive spectrum access in IoT networks, 2018. arXiv: 1804.11135.
[73] M.A. Shah, S. Zhang, C. Maple, Cognitive radio networks for Internet of Things: applications, challenges and future, in: Proc. of International Conference on Automation and Computing, 2013, pp. 1–6.
[74] M. Mueck, A. Piipponen, K. Kalliojarvi, G. Dimitrakopoulos, K. Tsagkaris, P. Demestichas, F. Casadevall, J. Perez-Romero, O. Sallent, G. Baldini, S. Filin, H. Harada, M. Debbah, T. Haustein, J. Gebert, B. Deschamps, P. Bender, M. Street, S. Kandeepan, J. Lota, A. Hayar, ETSI reconfigurable radio systems: status and future directions on software defined radio and cognitive radio standards, IEEE Commun. Mag. 48 (9) (2010) 78–86, doi:10.1109/MCOM.2010.5560591.
[75] Standard ECMA-392. MAC and PHY for Operation in TV White Space, (https://www.ecma-international.org/publications/standards/Ecma-392-arch.htm).
[76] IEEE Standard for information technology–Telecommunications and information exchange between systems - Wireless regional area networks (WRAN)–Specific requirements - Part 22: cognitive wireless RAN medium access control (MAC) and physical layer (PHY) specifications:policies and procedures for operation in the TV bands - Amendment 2: enhancement for broadband services and monitoring applications, IEEE Std 802.22b-2015 (Amendment to IEEE Std 802.22–2011 as amended by IEEE Std 802.22a-2014) (2015) 1–299. 10.1109/IEEESTD.2015.7336461.
[77] IEEE Standard for information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: wireless LAN medium access control (MAC) and physical layer (PHY) specifications amendment 5: television white spaces (TVWS) operation, IEEE Std 802.11af-2013 (Amendment to IEEE Std 802.11–2012, as amended by IEEE Std 802.11ae-2012, IEEE Std 802.11aa-2012, IEEE Std 802.11ad-2012, and IEEE Std 802.11ac-2013) (2014) 1–198. 10.1109/IEEESTD.2014.6744566
[78] K.D. Singh, P. Rawat, J.-M. Bonnin, Cognitive radio for vehicular ad hoc networks (CR-VANETs): approaches and challenges, EURASIP J. Wirel. Commun. Netw. 2014 (1) (2014) 49, doi:10.1186/1687-1499-2014-49.
[79] Si Chen, R. Vuyyuru, O. Altintas, A.M. Wyglinski, Learning in vehicular dynamic spectrum access networks: opportunities and challenges, in: Proc. of International Symposium on Intelligent Signal Processing and Communications Systems (ISPACS), 2011, pp. 1–6, doi:10.1109/ISPACS.2011.6146215.
[80] R.C. Qiu, Z. Hu, Z. Chen, N. Guo, R. Ranganathan, S. Hou, G. Zheng, Cognitive radio network for the smart grid: experimental system architecture, control algorithms, security, and microgrid testbed, IEEE Trans. Smart Grid 2 (4) (2011) 724–740, doi:10.1109/TSG.2011.2160101.
[81] R. Deng, J. Chen, X. Cao, Y. Zhang, S. Maharjan, S. Gjessing, Sensing-performance tradeoff in cognitive radio enabled smart grid, IEEE Trans. Smart Grid 4 (1) (2013) 302–310, doi:10.1109/TSG.2012.2210058.
[82] A. Ghassemi, S. Bavarian, L. Lampe, Cognitive radio for smart grid communications, in: Proc. of International Conference on Smart Grid Communications, 2010, pp. 297–302, doi:10.1109/SMARTGRID.2010.5622097.
[83] R. Chavez-Santiago, K.E. Nolan, O. Holland, L. De Nardis, J.M. Ferro, N. Barroca, L.M. Borges, F.J. Velez, V. Goncalves, I. Balasingham, Cognitive radio for medical body area networks using ultra wideband, IEEE Wirel. Commun. 19 (4) (2012) 74–81, doi:10.1109/MWC.2012.6272426.
[84] R. Chávez-Santiago, I. Balasingham, Cognitive radio for medical wireless body area networks, in: Proc. of International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2011, pp. 148–152, doi:10.1109/CAMAD.2011.5941105.
[85] A.R. Syed, K.A. Yau, Oncognitive radio-based wireless body area networks for medical applications, in: Proc. of IEEE Symposium on Computational Intelligence in Healthcare and e-health (CICARE), 2013, pp. 51–57, doi:10.1109/CICARE.2013.6583068.
[86] A. Gorcin, H. Arslan, Publicsafety and emergency case communications: opportunities from the aspect of cognitive radio, in: Proc. of IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks, 2008, pp. 1–10, doi:10.1109/DYSPAN.2008.68.
[87] D. Fudenberg, J. Tirole, Game Theory, MIT Press, Cambridge, MA, 1991. Translated into Chinesse by Renin University Press, Bejing: China.
[88] F.N. Iandola, M.W. Moskewicz, K. Ashraf, S. Han, W.J. Dally, K. Keutzer, Squeezenet: alexnet-level accuracy with 50x fewer parameters and <1MB model size, 2016. arXiv: 1602.07360.
[89] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in: F. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1097–1105.
[90] L. Li, A. Ghasemi, IoT Enabled machine learning for an algorithmic spectrum decision process, IEEE Internet Things J. (2019), doi:10.1109/JIOT.2018.2883490. 1–1
[91] L. Li, D. Boudreau, R. Paiement, I. Labbe, F. Patenaude, P. Chahine, M. Wang, P. Brouillette, A cloud-based spectrum environment awareness system, in: Proc. of International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), 2017, pp. 1–6, doi:10.1109/PIMRC.2017.8292378.
[92] K.E. Baddour, A. Ghasemi, H. Rutagemwa, Spectrumoccupancy prediction for land mobile radio bands using a recommender system, in: Proc. of Vehicular Technology Conference (VTC-Fall), 2018, pp. 1–6, doi:10.1109/VTCFall.2018.8690654.
[93] T. Chen, C. Guestrin, XGBoost: a scalable tree boosting system, in: Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, in: KDD '16, ACM, New York, NY, USA, 2016, pp. 785–794, doi:10.1145/2939672.2939785.
[94] J.C. Spall, Multivariate stochastic approximation using a simultaneous perturbation gradient approximation, IEEE Trans. Autom. Contr. 37 (3) (1992) 332–341, doi:10.1109/9.119632.
[95] P.P. Markopoulos, S. Kundu, D.A. Pados, Small-sample-support suppression of interference to PN-masked data, IEEE Trans. Commun. 61 (7) (2013) 2979–2987, doi:10.1109/TCOMM.2013.043013.120643.
[96] A. Jagannath, A. Amanna, Realizing data driven and hampel preprocessor based adaptive filtering on a software defined radio testbed: A USRP case study, in: Proc. of International Conference on Computing, Networking and Communications (ICNC), 2018, pp. 310–314, doi:10.1109/ICCNC.2018.8390369.
[97] D.A. Pados, G.N. Karystinos, An iterative algorithm for the computation of the MVDR filter, IEEE Trans. Signal Process. 49 (2) (2001) 290–300, doi:10.1109/78.902111.
[98] J.P. Leite, P.H.P. de Carvalho, R.D. Vieira, A flexible framework based on reinforcement learning for adaptive modulation and coding in OFDM wireless systems, in: Proc. of IEEE Wireless Communications and Networking Conference (WCNC), 2012, pp. 809–814, doi:10.1109/WCNC.2012.6214482.

[99] C. Wang, Z. Wang, W. Sun, D.R. Fuhrmann, Reinforcement learning-based adaptive transmission in time-varying underwater acoustic channels, IEEE Access 6 (2018) 2541–2558, doi:10.1109/ACCESS.2017.2784239.

[100] T. Ahmed, F. Ahmed, Y.L. Moullec, Optimization of channel allocation in wireless body area networks by means of reinforcement learning, in: Proc. of IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob), 2016, pp. 120–123, doi:10.1109/APWiMob.2016.7811445.

[101] B. Peng, Q. Jiao, T. Kürner, Angle of arrival estimation in dynamic indoor thz channels with Bayesian filter and reinforcement learning, in: Proc. of European Signal Processing Conference (EUSIPCO), 2016, pp. 1975–1979, doi:10.1109/EUSIPCO.2016.7760594.

[102] X. Liu, Y. Xu, Y. Cheng, Y. Li, L. Zhao, X. Zhang, A heterogeneous information fusion deep reinforcement learning for intelligent frequency selection of hf communication, China Commun. 15 (9) (2018) 73–84, doi:10.1109/CC.2018.8456453.

[103] A.R. Syed, K.L.A. Yau, H. Mohamad, N. Ramli, W. Hashim, Channel selection in multi-hop cognitive radio network using reinforcement learning: An experimental study, in: Proce. of International Conference on Frontiers of Communications, Networks and Applications (ICFCNA), 2014, pp. 1–6, doi:10.1049/cp.2014.1402.

[104] S. Tubachi, M. Venkatesan, A.V. Kulkarni, Predictive learning model in cognitive radio using reinforcement learning, in: Proc. of International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI), 2017, pp. 564–567, doi:10.1109/ICPCSI.2017.8391775.

[105] R. Ranjan, A. Phophalia, Reinforcement learning for dynamic channel allocation in mobile cellular systems, in: Proc. of International Conference on Recent Advances in Microwave Theory and Applications, 2008, pp. 924–927.

[106] S. Singh, A. Trivedi, Anti-jamming in cognitive radio networks using reinforcement learning algorithms, in: Proc. of International Conference on Wireless and Optical Communications Networks (WOCN), 2012, pp. 1–5, doi:10.1109/WOCN.2012.6331885.

[107] Z. Puljiz, M. Park, R.H. Jr., A machine learning approach to link adaptation for SC-FDE system, in: Proc. of IEEE Global Communications Conference (GLOBECOM), 2011, pp. 1–5, doi:10.1109/GLOCOM.2011.6134362.

[108] S. Yun, C. Caramanis, Reinforcementlearning for link adaptation in MIMO-OFDM wireless systems, in: Proc. of IEEE Global Communications Conference (GLOBECOM), 2010, pp. 1–5, doi:10.1109/GLOCOM.2010.5683371.

[109] C. Pandana, K.J.R. Liu, Near-optimal reinforcement learning framework for energy-aware sensor communications, IEEE J. Sel. Areas Commun. 23 (4) (2005) 788–797, doi:10.1109/JSAC.2005.843547.

[110] N. Mastronarde, M. van der Schaar, Joint physical-layer and system-level power management for delay-sensitive wireless communications, IEEE Trans. Mob. Comput. 12 (4) (2013) 694–709, doi:10.1109/TMC.2012.36.

[111] X. Li, H. He, Y. Yao, Reinforcement learning based adaptive rate control for delay-constrained communications over fading channels, in: Proc. of International Joint Conference on Neural Networks (IJCNN), 2010, pp. 1–7, doi:10.1109/IJCNN.2010.5596697.

[112] Y. Engel, S. Mannor, R. Meir, The kernel recursive least-squares algorithm, IEEE Trans. Signal Process. 52 (8) (2004) 2275–2285, doi:10.1109/TSP.2004.830985.

[113] T. Huang, R.-X. Zhang, C. Zhou, L. Sun, QARC: video quality aware rate control for real-time video streaming based on deep reinforcement learning, 2018.

[114] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T.P. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, 2016. arXiv: 1602.01783.

[115] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[116] P.C. Kumar, P. Saratchandran, N. Sundararajan, Communication channel equalization using minimal radial basis function neural networks, in: Proc. of Neural Networks for Signal Processing VIII, 1998, pp. 477–485, doi:10.1109/NNSP.1998.710678.

[117] L. Zhang, X. Zhang, MIMO channel estimation and equalization using three-layer neural networks with feedback, Tsinghua Sci. Technol. 12 (6) (2007) 658–662, doi:10.1109/TST.2007.6071814.

[118] D. Jianping, N. Sundararajan, P. Saratchandran, Communication channel equalization using complex-valued minimal radial basis function neural network, in: Proc. of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium, 5, 2000, pp. 372–377, doi:10.1109/IJCNN.2000.861498.

[119] M. Birgmeier, A neural network trained with the extended Kalman algorithm used for the equalization of a binary communication channel, in: Proc. of IEEE Workshop on Neural Networks for Signal Processing, 1994, pp. 527–534, doi:10.1109/NNSP.1994.366013.

[120] W.R. Kirkland, D.P. Taylor, On the application of feed forward neural networks to channel equalization, in: Proc. of IJCNN International Joint Conference on Neural Networks, 2, 1992, pp. 919–924 vol.2, doi:10.1109/IJCNN.1992.226870.

[121] S.-H. Heo, S.-K. Park, S.-W. Nam, Channel equalization for severe intersymbol interference and nonlinearity with a radial basis function neural network, in: IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339), 6, 1999, pp. 3992–3995 vol.6, doi:10.1109/IJCNN.1999.830797.

[122] M.M.A. Moustafa, S.H.A. El-Ramly, Channel estimation and equalization using backpropagation neural networks in OFDM systems, in: Proc. of IFIP International Conference on Wireless and Optical Communications Networks, 2009, pp. 1–4, doi:10.1109/WOCN.2009.5010528.

[123] H. Ye, G.Y. Li, B. Juang, Power of deep learning for channel estimation and signal detection in OFDM systems, IEEE Wirel. Commun. Lett. 7 (1) (2018), doi:10.1109/LWC.2017.2757490.

[124] D. Erdogmus, D. Rende, J.C. Principe, T.F. Wong, Nonlinear channel equalization using multilayer perceptrons with information-theoretic criterion, in: Neural Networks for Signal Processing XI: Proceedings of the 2001 IEEE Signal Processing Society Workshop (IEEE Cat. No.01TH8584), 2001, pp. 443–451, doi:10.1109/NNSP.2001.943148.

[125] H. Ye, G.Y. Li, Initialresults on deep learning for joint channel equalization and decoding, in: Proc. of Vehicular Technology Conference (VTC-Fall), 2017, pp. 1–5, doi:10.1109/VTCFall.2017.8288419.

[126] D. Erdogmus, J.C. Principe, Generalized information potential criterion for adaptive system training, IEEE Trans. Neural Netw. 13 (5) (2002) 1035–1044, doi:10.1109/TNN.2002.1031936.

[127] W. Lyu, Z. Zhang, C. Jiao, K. Qin, H. Zhang, Performanceevaluation of channel decoding with deep neural networks, in: Proc. of IEEE International Conference on Communications (ICC), 2018, pp. 1–6, doi:10.1109/ICC.2018.8422289.

[128] S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, 2015. arXiv: 1502.03167.

[129] E. Nachmani, Y. Be'ery, D. Burshtein, Learning to decode linear codes using deep learning, in: Proc. of Annual Allerton Conference on Communication, Control, and Computing (Allerton), 2016, doi:10.1109/ALLERTON.2016.7852251.

[130] R.G. Gallager, Low-density parity-check codes, IRE Trans. Inf. Theory IT-8 (1962) 21–28.

[131] R. Tanner, A recursive approach to low complexity codes, IEEE Trans. Inf. Theory 27 (5) (1981) 533–547, doi:10.1109/TIT.1981.1056404.

[132] J. Lee, J.-K. Han, J. Zhang, MIMO technologies in 3GPP LTE and LTE-advanced, EURASIP J. Wirel. Commun. Netw. 2009 (2009) 3:1–3:10, doi:10.1155/2009/302092.

[133] I.F. Akyildiz, S. Nie, S.-C. Lin, M. Chandrasekaran, 5G roadmap, Comput. Netw. 106 (C) (2016) 17–48, doi:10.1016/j.comnet.2016.06.010.

[134] M.F. UNlerşen, E. Yaldiz, Direction of arrival estimation by using artificial neural networks, in: 2016 European Modelling Symposium (EMS), 2016, pp. 242–245, doi:10.1109/EMS.2016.049.

[135] A.H.E. Zooghby, C.G. Christodoulou, M. Georgiopoulos, Performance of radial-basis function networks for direction of arrival estimation with antenna arrays, IEEE Trans. Antennas Propag. 45 (11) (1997) 1611–1617, doi:10.1109/8.650072.

[136] S.H. Zainud-Deen, H.A. Malhat, K.H. Awadalla, E.S. El-Hadad, Direction of arrival and state of polarization estimation using Radial Basis Function Neural Network (RBFNN), in: Proc. of National Radio Science Conference, 2008, pp. 1–8, doi:10.1109/NRSC.2008.4542314.

[137] Y.L. Sit, M. Agatonovic, T. Zwick, Neural network based direction of arrival estimation for a MIMO OFDM radar, in: Proc. of European Radar Conference, 2012, pp. 298–301.

[138] E. Efimov, T. Shevgunov, D. Filimonova, Angle of arrival estimator based on artificial neural networks, in: 2016 17th International Radar Symposium (IRS), 2016, pp. 1–3, doi:10.1109/IRS.2016.7497355.

[139] M. Hirari, K. Gotoh, M. Hayakawa, DOA estimation of distributed sources using neural networks, in: Proc. of Fourth International Conference on Signal Processing (ICSP), 1998, pp. 335–338vol.1, doi:10.1109/ICOSP.1998.770219.

[140] S. Haykin, Neural Networks: A Comprehensive Foundation, Prentice Hall, 1994.

[141] A. Zooghby, C. Christodoulou, M. Georgeiopoulos, Neural network based beamforming for interference cancellation, in: SPIE proceedings series, Society of Photo-Optical Instrumentation Engineers, 1998, pp. 420–429.

[142] H. Li, Multiagent-learning for aloha-like spectrum access in cognitive radio systems, volume 2010, 2010, p. 876216. 10.1155/2010/876216

[143] T.J. O'Shea, T. Roy, T.C. Clancy, Over-the-air deep learning based radio signal classification, IEEE J. Sel. Top. Signal Process. 12 (1) (2018) 168–179, doi:10.1109/JSTSP.2018.2797022.

[144] T.J. O'Shea, J. Hoydis, An introduction to deep learning for the physical layer, IEEE Trans. Cognit. Commun. Netw. 3 (4) (2017) 563–575.

[145] T. Wang, C.-K. Wen, H. Wang, F. Gao, T. Jiang, S. Jin, Deep learning for wireless physical layer: opportunities and challenges, China Commun. 14 (11) (2017) 92–111.

[146] N.E. West, T. O'Shea, Deep architectures for modulation recognition, in: Proc. of IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN), 2017, pp. 1–6, doi:10.1109/DySPAN.2017.7920754. Baltimore, MD, USA

[147] M. Kulin, T. Kazaz, I. Moerman, E.D. Poorter, End-to-end learning from spectrum data: a deep learning approach for wireless signal identification in spectrum monitoring applications, IEEE Access 6 (2018) 18484–18501, doi:10.1109/ACCESS.2018.2818794.

[148] K. Karra, S. Kuzdeba, J. Petersen, Modulation recognition using hierarchical deep neural networks, in: Proc. of IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN), 2017, pp. 1–3, doi:10.1109/DySPAN.2017.7920746. Baltimore, MD, USA

[149] O. Ozdemir, R. Li, P. Varshney, Hybrid maximum likelihood modulation classification using multiple radios, IEEE Commun. Lett. 17 (10) (2013) 1889–1892.
[150] O. Ozdemir, T. Wimalajeewa, B. Dulek, P.K. Varshney, W. Su, Asynchronous linear modulation classification with multiple sensors via generalized EM algorithm, IEEE Trans. Wirel. Commun. 14 (11) (2015) 6389–6400.
[151] T. Wimalajeewa, J. Jagannath, P.K. Varshney, A.L. Drozd, W. Su, Distributed asynchronous modulation classification based on hybrid maximum likelihood approach, in: Proc. of IEEE Conf. on Military Communications (MILCOM), 2015. Tampa, FL, USA
[152] S. Foulke, J. Jagannath, A.L. Drozd, T. Wimalajeewa, P.K. Varshney, W. Su, Multisensor Modulation Classification (MMC) Implementation considerations - USRP case study, in: Proc. of IEEE Conf. on Military Communications (MILCOM), 2014. Baltimore, MD, USA
[153] J. Jagannath, H.M. Saarinen, A.L. Drozd, Framework for Automatic Signal Classification Techniques (FACT) for software defined radios, in: Proc. of IEEE Symposium on Computational Intelligence in Security and Defense Applications (CISDA), 2015. Verona, NY, USA
[154] E.E. Azzouz, A.K. Nandi, Automatic Modulation Recognition of Communication Signals, Kluwer Academic Publishers, Norwell, MA, 1996.
[155] A. Hazza, M. Shoaib, S. AlShebeili, A. Fahd, Automatic modulation classification of digital modulations in presence of HF noise., EURASIP J. Adv. Signal Process. 2012 (2012) 238.
[156] A. Kubankova, J. Prinosil, D. Kubanek, Recognition of Digital modulations based on mathematical classifier, in: Proc. of the European Conference of Systems (ECCS), 2010. Stevens Point, WI
[157] J. Jagannath, D. O'Connor, N. Polosky, B. Sheaffer, L.N. Theagarajan, S. Foulke, P.K. Varshney, S.P. Reichhart, Design and evaluation of hierarchical hybrid automatic modulation classifier using software defined radios, in: Proc. of IEEE Annual Computing and Communication Workshop and Conference (CCWC), 2017. Las Vegas, NV, USA
[158] T.J. O'Shea, J. Corgan, Convolutional radio modulation recognition networks, 2016. arXiv: 1602.04105.
[159] H.W.H.A. Shengliang Peng Hanyu Jiang, Y.-D. Yao, Modulation classification using convolutional neural network based deep learning model., WOCC (2017).
[160] J. Jagannath, N. Polosky, D. O'Connor, L. Theagarajan, B. Sheaffer, S. Foulke, P. Varshney, Artificial neural network based automatic modulation classifier for software defined radios, in: Proc. of IEEE International Conference on Communications (ICC), 2018. Kansas City, MO, USA
[161] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, 2014. arXiv: 1412.6980.
[162] M. Schmidt, D. Block, U. Meier, Wireless interference identification with convolutional neural networks, 2017. arXiv: 1703.00737.
[163] S. Grimaldi, A. Mahmood, M. Gidlund, An svm-based method for classification of external interference in industrial wireless sensor and actuator networks, J. Sens. Actuator Netw. 6 (2017).
[164] A. Selim, F. Paisana, J.A. Arokkiam, Y. Zhang, L. Doyle, L.A. DaSilva, Spectrum monitoring for radar bands using deep convolutional neural networks, 2017. arXiv: 1705.00462.
[165] J. Akeret, C. Chang, A. Lucchi, A. Refregier, Radio frequency interference mitigation using deep convolutional neural networks, Astron. Comput. 18 (2017) 35–39, doi:10.1016/j.ascom.2017.01.002.
[166] D. Czech, A. Mishra, M. Inggs, A CNN and LSTM-based approach to classifying transient radio frequency interference, Astron. Comput. 25 (2018) 52–57, doi:10.1016/j.ascom.2018.07.002.
[167] K. Youssef, L.-S. Bouchard, K.Z. Haigh, H. Krovi, J. Silovsky, C.P. Vander Valk, Machine learning approach to RF transmitter identification, ArXiv e-prints (2017).
[168] B. Üstün, W.J. Melssen, L.M.C. Buydens, Facilitating the application of support vector regression by using a universal pearson vii function based kernel, 2005.
[169] K. Youssef, N.N. Jarenwattananon, L. Bouchard, Feature-preserving noise removal, IEEE Trans. Med. Imaging 34 (9) (2015) 1822–1829, doi:10.1109/TMI.2015.2409265.
[170] L.-S. B. K. Youssef, Training artificial neural networks with reduced computational complexity.
[171] J. Roux, E. Alata, G. Auriol, V. Nicomette, M. Kaaniche, Toward an intrusion detection approach for IoT based on radio communications profiling, in: 2017 13th European Dependable Computing Conference (EDCC), 2017, pp. 147–150, doi:10.1109/EDCC.2017.11.
[172] D. Macagnano, G. Destino, G. Abreu, Indoor positioning: a key enabling technology for iot applications, in: 2014 IEEE World Forum on Internet of Things (WF-IoT), 2014, pp. 117–118, doi:10.1109/WF-IoT.2014.6803131.
[173] Y. Cheng, H. Chou, R.Y. Chang, Machine-learning indoor localization with access point selection and signal strength reconstruction, in: 2016 IEEE 83rd Vehicular Technology Conference (VTC Spring), 2016, pp. 1–5, doi:10.1109/VTCSpring.2016.7504333.
[174] A. Belay Adege, H.-P. Lin, G. Berie Tarekegn, S.-S. Jeng, Applying deep neural network (dnn) for robust indoor localization in multi-building environment, Appl. Sci. 8 (2018) 1062, doi:10.3390/app8071062.
[175] X. Wang, L. Gao, S. Mao, S. Pandey, Csi-based fingerprinting for indoor localization: a deep learning approach, 2016. arXiv: 1603.07080.
[176] D. Miorandi, S. Sicari, F. De Pellegrini, I. Chlamtac, Internet of things: vision, applications and research challenges, Ad Hoc Netw. 10 (7) (2012) 1497–1516.
[177] L. Mainetti, L. Patrono, A. Vilei, Evolution of wireless sensor networks towards the Internet of Things: a survey, in: Proc. of International Conference on Software, Telecommunications and Computer Networks, 2011, pp. 1–6.
[178] D.G. Reina, S.L. Toral, F. Barrero, N. Bessis, E. Asimakopoulou, The Role of Ad Hoc Networks in the Internet of Things: A Case Scenario for Smart Environments, Springer, Berlin, Heidelberg, pp. 89–113.
[179] K. Sood, S. Yu, Y. Xiang, Software-defined wireless networking opportunities and challenges for internet-of-Things: a review, IEEE Internet Things J. 3 (2016) 453–463.
[180] R. Bruzgiene, L. Narbutaite, T. Adomkus, MANET Network in internet of things system, Ad Hoc Netw (2017) 89.
[181] C. Cormio, K.R. Chowdhury, A survey on mac protocols for cognitive radio networks, Ad Hoc Netw 7 (7) (2009) 1315–1329, doi:10.1016/j.adhoc.2009.01.002.
[182] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, Internet of things: a survey on enabling technologies, protocols, and applications, IEEE Commun. Surv. Tutor. 17 (4) (2015) 2347–2376.
[183] S. Lien, C. Tseng, K. Chen, Carrier sensing based multiple access protocols for cognitive radio networks, in: Proc. of IEEE International Conference on Communications (ICC), 2008, pp. 3208–3214, doi:10.1109/ICC.2008.604.
[184] N. Jain, S.R. Das, A. Nasipuri, A multichannel csma mac protocol with receiver-based channel selection for multihop wireless networks, in: Proc. of Tenth International Conference on Computer Communications and Networks (Cat. No.01EX495), 2001, pp. 432–439.
[185] C. Cordeiro, K. Challapali, C-mac: a cognitive mac protocol for multi-channel wireless networks, in: Proc. of IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, 2007, pp. 147–157, doi:10.1109/DYSPAN.2007.27.
[186] M. Hadded, P. Muhlethaler, A. Laouiti, R. Zagrouba, L.A. Saidane, Tdma-based mac protocols for vehicular ad hoc networks: a survey, qualitative analysis, and open research issues, IEEE Commun. Surv. Tutor. 17 (4) (2015) 2461–2492, doi:10.1109/COMST.2015.2440374.
[187] A. Muqattash, M. Krunz, CDMA-based MAC protocol for wireless ad hoc networks, in: Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking &Amp; Computing, in: MobiHoc '03, ACM, New York, NY, USA, 2003, pp. 153–164, doi:10.1145/778415.778434.
[188] S. Kumar, V.S. Raghavan, J. Deng, Medium access control protocols for ad hoc wireless networks: A Survey, Ad Hoc Netw. 4 (3) (2006) 326–358, doi:10.1016/j.adhoc.2004.10.001.
[189] L. Sitanayah, C.J. Sreenan, K.N. Brown, Er-mac: a hybrid mac protocol for emergency response wireless sensor networks, in: Proc. of Fourth International Conference on Sensor Technologies and Applications, 2010, pp. 244–249, doi:10.1109/SENSORCOMM.2010.45.
[190] H. Su, X. Zhang, Opportunistic mac protocols for cognitive radio based wireless networks, in: Proc. of 41st Annual Conference on Information Sciences and Systems, 2007, pp. 363–368, doi:10.1109/CISS.2007.4298329.
[191] J. Jagannath, A. Saji, H. Kulhandjian, Y. Sun, E. Demirors, T. Melodia, A hybrid MAC protocol with channel-dependent optimized scheduling for clustered underwater acoustic sensor networks, in: Proc. of ACM Intl. Conf. on UnderWater Networks and Systems (WUWNet), 2013. Kaohsiung, Taiwan
[192] C.W. Commander, Broadcast scheduling problemBroadcast Scheduling Problem, Springer, Boston, MA, pp. 339–345. 10.1007/978-0-387-74759-0_60
[193] S. Salcedo-Sanz, C. Bousono-Calzon, A.R. Figueiras-Vidal, A mixed neural-genetic algorithm for the broadcast scheduling problem, IEEE Trans. Wirel. Commun. 2 (2) (2003) 277–283, doi:10.1109/TWC.2003.808967.
[194] H. Shi, L. Wang, Broadcast scheduling in wireless multihop networks using a neural-network-based hybrid algorithm, Neural Netw. 18 (5) (2005) 765–771, doi:10.1016/j.neunet.2005.06.013. IJCNN 2005
[195] Y.-J. Shen, M.-S. Wang, Broadcast scheduling in wireless sensor networks using fuzzy hopfield neural network, Expert Syst Appl 34 (2) (2008) 900–907, doi:10.1016/j.eswa.2006.10.024.
[196] G. Wang, N. Ansari, Optimal broadcast scheduling in packet radio networks using mean field annealing, IEEE J. Sel. Areas Commun. 15 (2) (1997) 250–260.
[197] R.V. Kulkarni, G.K. Venayagamoorthy, Neural network based secure media access control protocol for wireless sensor networks, in: Proc. of International Joint Conference on Neural Networks, 2009, pp. 1680–1687, doi:10.1109/IJCNN.2009.5179075.
[198] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proc. of International Conference on Neural Networks (ICNN), 4, 1995, pp. 1942–1948 vol.4, doi:10.1109/ICNN.1995.488968.
[199] O. Naparstek, K. Cohen, Deep multi-user reinforcement learning for dynamic spectrum access in multichannel wireless networks, in: Proc. of IEEE Global Communications Conference (GLOBECOM), 2017, pp. 1–7, doi:10.1109/GLOCOM.2017.8254101.
[200] X. Li, J. Fang, W. Cheng, H. Duan, Z. Chen, H. Li, Intelligent power control for spectrum sharing in cognitive radios: a deep reinforcement learning approach, IEEE Access 6 (2018) 25463–25473, doi:10.1109/ACCESS.2018.2831240.
[201] S.A. Grandhi, J. Zander, R. Yates, Constrained power control, Wirel. Personal Commun. 1 (4) (1994) 257–270, doi:10.1007/BF01098870.
[202] Y. Yu, T. Wang, S.C. Liew, Deep-reinforcement learning multiple access for heterogeneous wireless networks, in: Proc. of IEEE International Conference on Communications (ICC), 2018, pp. 1–7, doi:10.1109/ICC.2018.8422168.
[203] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis,

Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.
[204] Y. Yu, T. Wang, S.C. Liew, Deep-reinforcement learning multiple access for heterogeneous wireless networks, 2017. arXiv: 1712.00162.
[205] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778, doi:10.1109/CVPR.2016.90.
[206] B. Karp, H.T. Kung, GPSR: greedy perimeter stateless routing for wireless networks, in: Proc. of the 6th Annual International Conference on Mobile Computing and Networking, in: MobiCom '00, 2000. New York, NY, USA
[207] J. Jagannath, S. Furman, A. Jagannath, L. Ling, A. Burger, A. Drozd, HELPER: heterogeneous efficient low power radio for enabling ad hoc emergency public safety networks, Ad Hoc Netw. 89 (2019) 218–235.
[208] P. Xie, J.-H. Cui, L. Lao, Vbf: vector-based forwarding protocol for underwater sensor networks, in: F. Boavida, T. Plagemann, B. Stiller, C. Westphal, E. Monteiro (Eds.), NETWORKING 2006. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems, Springer, Berlin, Heidelberg, 2006, pp. 1216–1221.
[209] J. Jagannath, S. Furman, T. Melodia, A. Drozd, "Design and experimental evaluation of a cross-layer deadline-based joint routing and spectrum allocation algorithm", IEEE Trans. Mob. Comput. (2018).
[210] W.R. Heinzelman, A. Chandrakasan, H. Balakrishnan, Energy-efficient communication protocol for wireless microsensor networks, in: Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, 2000. 10 pp. vol.2–
[211] K. Akkaya, M. Younis, An energy-aware qos routing protocol for wireless sensor networks, in: Proc. of International Conference on Distributed Computing Systems Workshops, 2003, pp. 710–715.
[212] K. Akkaya, M. Younis, A survey on routing protocols for wireless sensor networks, Ad Hoc Netw. 3 (2005) 325–349.
[213] V. Srivastava, M. Motani, Cross-layer design: a survey and the road ahead, IEEE Commun. Mag. 43 (2005) 112–119.
[214] L. Kuo, T. Melodia, Cross-layer routing on MIMO-OFDMunderwater acoustic links, in: Proc. of IEEE Conf. on Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2012, pp. 227–235. Seoul, Korea
[215] M.Z. Hasan, F.M. Al-Turjman, H.M. Al-Rizzo, Analysis of cross-Layer design of quality-of-Service forward geographic wireless sensor network routing strategies in green internet of things, IEEE Access 6 (2018) 20371–20389.
[216] C. Xu, J. Feng, Z. Zhou, J. Wu, C. Perera, Cross-layer optimization for cooperative content distribution in multihop device-to-device networks, IEEE Internet Things J. 6 (1) (2019) 278–287, doi:10.1109/JIOT.2017.2741718.
[217] G. Callebaut, G. Ottoy, L. Van der Perre, Cross-layer framework and optimization for efficient use of the energy budget of IoT nodes, arXiv:1806.08624 (2018).
[218] J.A. Boyan, M.L. Littman, Packet routing in dynamically changing networks: a reinforcement learning approach, in: Proceedings of the 6th International Conference on Neural Information Processing Systems, in: NIPS'93, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993, pp. 671–678.
[219] R. Bellman, Onarouting problem, Q. Appl. Math. 16 (1) (1958) 87–90.
[220] D.R. Ford, D.R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, NJ, USA, 2010.
[221] B. Mao, Z.M. Fadlullah, F. Tang, N. Kato, O. Akashi, T. Inoue, K. Mizutani, Routing or computing? The paradigm shift towards intelligent computer network packet transmission based on deep learning, IEEE Trans. Comput. 66 (11) (2017) 1946–1960.
[222] E. Pourfakhar, A.M. Rahmani, A hybrid QoS multicast framework-based protocol for wireless mesh networks, Comput. Commun. 33 (17) (2010) 2079–2092, doi:10.1016/j.comcom.2010.07.026. Special Issue:Applied sciences in communication technologies
[223] J.S. Albus, A new approach to manipulator control: the cerebellar model articulation controller (cmac), ASME J. Dyn. Syst. Meas.Control 15 (2) (1975).
[224] R. Sun, S. Tatsumi, G. Zhao, Q-MAP: a novel multicast routing method in wireless ad hoc networks with multiagent reinforcement learning, in: Proc. of IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering. TENCOM, 1, 2002, pp. 667–670 vol.1, doi:10.1109/TENCON.2002.1181362.
[225] J. Barbancho, C. León, F.J. Molina, A. Barbancho, A new QoS routing algorithm based on self-organizing maps for wireless sensor networks, Telecommun. Syst. 36 (1) (2007) 73–83, doi:10.1007/s11235-007-9061-1.
[226] R.C. Shah, J.M. Rabaey, Energy aware routing for low energy ad hoc sensor networks, in: 2002 IEEE Wireless Communications and Networking Conference Record. WCNC 2002 (Cat. No.02TH8609), 1, 2002, pp. 350–355 vol.1, doi:10.1109/WCNC.2002.993520.
[227] C. Intanagonwiwat, R. Govindan, D. Estrin, Directed diffusion: a scalable and robust communication paradigm for sensor networks, in: Proc. of the Annual International Conference on Mobile Computing and Networking, in: MobiCom '00, ACM, New York, NY, USA, 2000, pp. 56–67, doi:10.1145/345910.345920.
[228] S. Dong, P. Agrawal, K. Sivalingam, Reinforcementlearning based geographic routing protocolfor UWB wireless sensor network, in: Proc. of IEEE Global Telecommunications Conference (GLOBECOM), 2007, pp. 652–656, doi:10.1109/GLOCOM.2007.127.
[229] T. Hu, Y. Fei, Qelar: a machine-learning-based adaptive routing protocol for energy-efficient and lifetime-extended underwater sensor networks, IEEE Trans. Mob. Comput. 9 (6) (2010) 796–809, doi:10.1109/TMC.2010.28.
[230] A. Forster, A.L. Murphy, FROMS: feedback routing for optimizing multiple sinks in WSN with reinforcement learning, in: Proc. of International Conference on Intelligent Sensors, Sensor Networks and Information, 2007, pp. 371–376, doi:10.1109/ISSNIP.2007.4496872.
[231] F. Silva, J. Heidemann, R. Govindan, D. Estrin, Frontiers in Distributed Sensor Networks, CRC Press, Inc., Boca Raton, Florida, USA, p. to appear. Refernece is superseded by [Silva05a].
[232] M. Lee, D. Marconett, X. Ye, S.J.B. Yoo, Cognitivenetwork management with reinforcement learning for wireless mesh networks, in: D. Medhi, J.M. Nogueira, T. Pfeifer, S.F. Wu (Eds.), IP Operations and Management, Springer, Berlin, Heidelberg, 2007, pp. 168–179.
[233] A. Jagannath, J. Jagannath, A. Drozd, Artificial intelligence-based cognitive cross-layer decision engine for next-generation space mission, in: Proc. of IEEE Cognitive Communication for Aerospace Applications (CCAA) Workshop, 2019. [under review].
[234] H.A. Abdul-Ghani, D. Konstantas, M. Mahyoub, A comprehensive IoT attacks survey based on a building-blocked reference model, Int. J. Adv. Comput. Sci. Appl. 9 (3) (2018), doi:10.14569/IJACSA.2018.090349.
[235] J. Markert, M. Massoth, K. Fischer-Hellmann, S. Furnell, C. Bolan, Attackvectors to wireless ZigBee network communications- analysis and countermeasures, in: Proc. of the Seventh Collaborative Research Symposium on Security, E-learning, Internet and Networking (SEIN), 2011.
[236] N. Vidgren, K. Haataja, J.L. Patiño-Andres, J.J. Ramírez-Sanchis, P. Toivanen, Security Threats in ZigBee-Enabled Systems: Vulnerability Evaluation, Practical Experiments, Countermeasures, and Lessons Learned, in: Proc. of 46th Hawaii International Conference on System Sciences, 2013, pp. 5132–5138, doi:10.1109/HICSS.2013.475.
[237] A. Mayzaud, R. Badonnel, I. Chrisment, A taxonomy of attacks in RPL-based internet of things, I. J. Netw. Secur. 18 (2016) 459–473.
[238] P. Pongle, G. Chavan, A survey: attacks on RPL and 6LoWPAN in IoT, in: Proc. of International Conference on Pervasive Computing (ICPC), 2015, pp. 1–6, doi:10.1109/PERVASIVE.2015.7087034.
[239] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., Tensorflow: a system for large-scale machine learning., in: OSDI, 16, 2016, pp. 265–283.
[240] M.L.D. Wong, A.K. Nandi, Automatic digital modulation recognition using spectral and statistical features with multi-layer perceptrons, in: Proc. of the Sixth International Symposium on Signal Processing and its Applications (Cat.No.01EX467), 2, 2001, pp. 390–393, doi:10.1109/ISSPA.2001.950162.
[241] J.L. Xu, W. Su, M. Zhou, Software-defined radio equipped with rapid modulation recognition, IEEE Trans. Veh. Technol. 59 (4) (2010) 1659–1667, doi:10.1109/TVT.2010.2041805.
[242] S.U. Pawar, J.F. Doherty, Modulation recognition in continuous phase modulation using approximate entropy, IEEE Trans. Inf. Forensics Secur. 6 (3) (2011) 843–852, doi:10.1109/TIFS.2011.2159000.
[243] Q. Shi, Y. Karasawa, Automatic modulation identification based on the probability density function of signal phase, IEEE Trans. Commun. 60 (4) (2012) 1033–1044, doi:10.1109/TCOMM.2012.021712.100638.
[244] S. Ghodeswar, P.G. Poonacha, An SNR estimation based adaptive hierarchical modulation classification method to recognize M-ary QAM and M-ary PSK signals, in: Proc. of International Conference on Signal Processing, Communication and Networking (ICSCN), 2015, pp. 1–6, doi:10.1109/ICSCN.2015.7219867. Chennai, India
[245] S. Shalev-Shwartz, S. Ben-David, Understanding Machine Learning: From Theory to Algorithms, Cambridge University Press, 2014.
[246] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436.
[247] Q. Mao, F. Hu, Q. Hao, Deep learning for intelligent wireless networks: a comprehensive survey, to appear, IEEE Commun. Surv.Tutor. (2018), doi:10.1109/COMST.2018.2846401.
[248] R.F. Molanes, J.J. Rodríguez-Andina, J. Fariña, Performance characterization and design guidelines for efficient processor - FPGA communication in cyclone v FPSocs, IEEE Trans. Ind. Electron. 65 (5) (2018) 4368–4377, doi:10.1109/TIE.2017.2766581.
[249] Pete Bennett (EE Times), Thewhy, where and what of low-power SoC design, 2004, (https://www.eetimes.com/document.asp?doc_id=1276973).
[250] Xilinx Inc., AXI Reference Guide, UG761 (v13.1) March 7, 2011, 2011, (https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf).
[251] F. Winterstein, S. Bayliss, G.A. Constantinides, High-level synthesis of dynamic data structures: acase study using vivado hls, in: Proc. of International Conference on Field-Programmable Technology (FPT), 2013, pp. 362–365. Kyoto, Japan
[252] L. Deng, The mnist database of handwritten digit images for machine learning research [best of the web], IEEE Signal Process. Mag. 29 (6) (2012) 141–142.
[253] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: a large-scale hierarchical image database, in: Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, Ieee, 2009, pp. 248–255.
[254] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing systems, 2012, pp. 1097–1105.
[255] 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 Process Mag. 29 (6) (2012) 82–97.
[256] M.A. Al-Garadi, A. Mohamed, A.K. Al-Ali, X. Du, M. Guizani, A survey of machine and deep learning methods for internet of things (IoT) security, 2018. arXiv: 1807.11023.

[257] W. Hu, Robust support vector machines for anomaly detection, in: Proc. of International Conference on Machine Learning and Applications (ICMLA'03, 2003, pp. 23–24.
[258] G. Kim, S. Lee, S. Kim, A novel hybrid intrusion detection method integrating anomaly detection with misuse detection, Expert Syst. Appl. 41 (4, Part 2) (2014) 1690–1700, doi:10.1016/j.eswa.2013.08.066.
[259] P. Torres, C. Catania, S. Garcia, C.G. Garino, An analysis of recurrent neural networks for botnet detection behavior, in: Proc. of IEEE Biennial Congress of Argentina (ARGENCON), 2016, pp. 1–6, doi:10.1109/ARGENCON.2016.7585247.
[260] M.-S.K. Hyo-Sik Ham Hwan-Hee Kim, M.-J. Choi, Linear SVM-based android malware detection for reliable IoT services, J. Appl. Math. 2014 (2014).
[261] N. McLaughlin, J. Martinez del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao, A. Doupé, G. Joon Ahn, Deep android malware detection, in: Proc. of the Seventh ACM on Conference on Data and Application Security and Privacy, in: CODASPY '17, ACM, New York, NY, USA, 2017, pp. 301–308, doi:10.1145/3029806.3029823.
[262] M. Yousefi-Azar, V. Varadharajan, L. Hamey, U. Tupakula, Autoencoder-based feature learning for cyber security applications, in: Proc. of International Joint Conference on Neural Networks (IJCNN), 2017, pp. 3854–3861, doi:10.1109/IJCNN.2017.7966342.
[263] M.S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi, A.P. Sheth, Machine learning for internet of things data analysis: a survey, Digit. Commun. Netw. 4 (3) (2018) 161–175, doi:10.1016/j.dcan.2017.10.002.
[264] A. Sheth, Transforming big data into smart data: Deriving value via harnessing volume, variety, and velocity using semantic techniques and technologies, in: Proc. of IEEE International Conference on Data Engineering, 2014, doi:10.1109/ICDE.2014.6816634. 2–2
[265] B.P. Rimal, E. Choi, I. Lumb, A taxonomy and survey of cloud computing systems, in: Proc. of Fifth International Joint Conference on INC, IMS and IDC, 2009, pp. 44–51, doi:10.1109/NCM.2009.218.
[266] Edge computing: a survey, Future Generation Computer Systems 97 (2019) 219–235, doi:10.1016/j.future.2019.02.050.
[267] R.K. Naha, S.K. Garg, D. Georgekopolous, P.P. Jayaraman, L. Gao, Y. Xiang, R. Ranjan, Fog computing: survey of trends, architectures, requirements, and research directions, 2018. arXiv: 1807.00976.
[268] X. Ma, Y.-J. Wu, Y. Wang, F. Chen, J. Liu, Mining smart card data for transit riders' travel patterns, Transp. Res. Part C 36 (2013) 1–12, doi:10.1016/j.trc.2013.07.010.
[269] W. Derguech, E. Bruke, E. Curry, An autonomic approach to real-time predictive analytics using open data and internet of things, in: Proc. of IEEE International Conference on Ubiquitous Intelligence and Computing and IEEE International Conference on Autonomic and Trusted Computing and IEEE International Conference on Scalable Computing and Communications and Its Associated Workshops, 2014, pp. 204–211.
[270] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The WEKA data mining software: an update, SIGKDD Explor. Newsl. 11 (1) (2009) 10–18, doi:10.1145/1656274.1656278.
[271] M.A. Khan, A. Khan, M.R. Khan, S.R.C. Anwar, A novel learning method to classify data streams in the internet of things, in: Proc. of National Software Engineering Conference, 2014, pp. 61–66.
[272] I. Kotenko, I. Saenko, F. Skorik, S. Bushuev, Neural network approach to forecast the state of the Internet of Things elements, in: Proc. of XVIII International Conference on Soft Computing and Measurements (SCM), 2015, pp. 133–135, doi:10.1109/SCM.2015.7190434.
[273] P.M. Comar, L. Liu, S. Saha, P.-N. Tan, A. Nucci, Combining supervised and unsupervised learning for zero-day malware detection, in: Proc. of IEEE International Conference on Computer Communications (INFOCOM), 2013, pp. 2022–2030.
[274] S. Suthaharan, Big data classification: problems and challenges in network intrusion prediction with machine learning, SIGMETRICS Perform. Eval. Rev. 41 (4) (2014) 70–73, doi:10.1145/2627534.2627557.

Jithin Jagannath is an Associate Senior Scientist/Engineer at ANDRO Computational Solutions, LLC. He is also a Ph.D. Candidate in the Department of Electrical and Computer Engineering at Northeastern University. Mr. Jagannath's expertise and research interests include cognitive radio, optimized networking, automatic modulation classification (AMC), machine learning, testbed design using SDRs, communication protocols for wireless sensor networks including terrestrial, and underwater sensor networks. He joined ANDRO in 2013, previously he was at the Wireless Networks and Embedded Systems Laboratory at the University at Buffalo from where he received his MSEE degree specializing in communication, control and signal processing in 2013. He has extensive experience developing SDR based prototypes. This includes designing, implementing and testing different communication protocols and optimized algorithms on hardware for wireless sensor networks. He is currently the Principal Investigator (PI) of several Small Business Innovation Research (SBIR) with objectives of designing and developing novel solutions in the domain of MIMO wireless communication, signal detection and classification, and machine learning. Previously, he has been the PI of HELPER, a project that aimed to provide off-the-grid connectivity to survivors for disasters through wireless ad hoc network. He has also been the technical lead in multiple SBIR/STTR efforts including cross-layer networking technology, AMC (which employed ANN based solution), cybersecurity, and compressive sensing. He has been the lead author of several peer-reviewed journal and conference publications. He also renders his review services to several leading Elsevier and IEEE Journals and conferences.

Nicholas Polosky is an Associate Computer Scientist/Engineer at ANDRO Computational Solutions, LLC. His areas of expertise and research interests include artificial intelligence, machine learning, cognitive radio, brain computer interfaces (BCI), and automatic modulation classification (AMC). He joined ANDRO in 2016 as an intern and now works full-time at ANDRO after receiving his BS in Computer Science from Cornell University. His previous work includes development of artificial neural networks (ANN) for automatic modulation classification tasks and implementation of long-short term memory networks (LSTM) for use in BCI applications.

Anu Jagannath's research interests include spread spectrum communication, adaptive receiver designs, adaptive spreading codes, medium access control (MAC) and routing protocols for RF and underwater acoustic wireless sensor networks, MIMO systems, and cognitive radio communications. She has been the Technical lead for the "Developing Interference detection and Analysis Device" SBIR effort for the Department of Homeland Security and the lead developer for the "Towards Maximal Spectral efficiency networking" project while also assisting in SBIR/STTR efforts and other projects including but not limited to "Railroad Physical Layer authentication". Her expertise lies in developing; PHY layer waveforms for software-defined radios, Time-Frequency Spectral Analysis tools, adaptive filters for spread spectrum waveforms, channel coding techniques, adaptive beamforming techniques for array signal processing. Prior to joining ANDRO, she has been a researcher in the Wireless Networks & Embedded systems laboratory (WINES lab, SUNY) and conducted her research on Underwater acoustic sensor networks. She graduated from University at Buffalo, SUNY with a MS degree in Electrical Engineering in 2013.

Francesco Restuccia received the Ph.D. degree in computer science from Missouri S&T, Rolla, MO, in 2016. He is currently an Associate Research Scientist with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA, USA. In the past, he has held research positions with the University of Texas at Arlington, Arlington, TX, USA, and the National Research Council of Italy, Rome, Italy. His current research interests include modeling, analysis, and experimental evaluation of wireless networked systems, with applications to pervasive and mobile computing and the Internet of Things. Dr. Restuccia has served on the Technical Program Committee of IEEE INFOCOM 2018, IEEE WoWMoM 2017-2018, IEEE SMARTCOMP 2017, and IEEE LCN 2016-2018. He is a Member of the IEEE and the ACM.

Tommaso Melodia received the Ph.D. degree in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2007. He is the William Lincoln Smith Chair Professor with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA, USA, and the Founding Director of the Institute for the Wireless Internet of Things. He is the Director of Research for the PAWR Project Office, a public-private partnership that is developing four city-scale platforms for advanced wireless research in the U.S. His research is supported mostly by U.S. federal agencies, including the National Science Foundation, the Air Force Research Laboratory, the Office of Naval Research, and the Army Research Laboratory. His current research interests include modeling, optimization, and experimental evaluation of wireless networked systems, with applications to 5G networks and Internet of Things, software-defined networking, and body area networks. Prof. Melodia is an Associate Editor for the IEEE Transactions on Mobile Computing, the IEEE Transactions on Molecular, Biological and Multi-Scale Communications, Computer Networks, and Smart Health. He is a Fellow of the IEEE and Senior Member of the ACM.