Deep Learning for Intelligent Wireless Networks: A Comprehensive Survey

Qian Mao, *Student Member, IEEE,* Fei Hu, *Member, IEEE,* and Qi Hao, *Member, IEEE*

***Abstract*—As a promising machine learning tool to handle the accurate pattern recognition from complex raw data, deep learning (DL) is becoming a powerful method to add intelligence to wireless networks with large-scale topology and complex radio conditions. DL uses many neural network layers to achieve a brain-like acute feature extraction from high-dimensional raw data. It can be used to find the network dynamics (such as hotspots, interference distribution, congestion points, traffic bottlenecks, spectrum availability, etc.) based on the analysis of a large amount of network parameters (such as delay, loss rate, link SNR, etc.). Therefore, DL can analyze extremely complex wireless networks with many nodes and dynamic link quality. This article performs a comprehensive survey of the applications of DL algorithms for different network layers, including physical layer modulation/coding, data link layer access control/resource allocation, and routing layer path search and traffic balancing. The use of DL to enhance other network functions, such as network security, sensing data compression, etc., is also discussed. Moreover, the challenging unsolved research issues in this field are discussed in detail, which represent the future research trends of DL-based wireless networks. This article can help the readers to deeply understand the state-of-the-art of the DL-based wireless network designs, and select interesting unsolved issues to pursue in their research.**

***Index Terms*—Wireless Networks, Deep Learning (DL), Deep Reinforcement Learning (DRL), Protocol Layers, Performance Optimization.**

I. Introduction

HUMAN brains possess powerful data processing capabilities. Every day we confront numerous data from the external world. Under a complex environment, a large number of object features are first collected by our sense organs. Then the brain extracts the abstract characteristics from those feature data and finally makes a decision. In many fields computers have already shown comparable or even more powerful capabilities compared to human beings, such as the game playing, auto control, voice and image recognition, etc. The approach for the computer to achieve these abilities is very similar to what human brain does, which has been developed as an eye-catching technology, i.e., Deep Learning (DL) [1]. In the DL process, first, computers need to learn from experiences and build up certain training model. This training process allows computers to determine appropriate weight values between neural nodes, which are able to extract the features from the input data. Once the neural network has been trained, an appropriate decision is able to be made to achieve high reward. This idea has shown great success in many real-world control scenarios, such as voice recognition [2], [3], image recognition [4], [5], [6], [7], semantic analysis [8], [9], language interpretation [10], [11], game control [12], drug discovery [13], and biomedical sciences [14], [15], [16], etc.

DL is a subclass of machine learning which uses cascaded layers to extract features from the input data and eventually forms a decision. The application of DL should consider four aspects: (1) How to represent the state of the environment in suitable numerical formats, which will be taken as the input layer of the DL network; (2) How to represent/interpret the recognition results, i.e., the physical meaning of the output layer of the DL network; (3) How to compute/update the reward value, and what is the proper reward function that can guide the iterative weight updating in each neural layer; (4) The structure of the DL system, including how many hidden layers, the structure of each layer, and the connections between layers.

Currently, many DL systems are tied with Reinforcement Learning (RL) models [17], which comprises three parts: 1) an environment which can be described by some features, 2) an agent which takes actions to change the environment, and 3) an interpreter which announces the current state and the action the agent takes. Meanwhile, the interpreter announces the reward after the action takes effect in the environment, as shown in Fig. 1. The goal of the RL is to train the agent in such a way that for a given environment state, it chooses the optimal action that yields the highest reward. Therefore, one of the main differences between DL and RL is that the former usually learns from examples (e.g., training data) to create a model to classify data, however, the latter trains the model by maximizing the reward associated with different actions.

DL has already shown astonishing capabilities in dealing with many real-world scenarios, such as the success of Alpha Go, the face recognition on mobile phones, etc. Researchers in computer network areas also cast strong interests in DL applications. By using DL model the complex network environment can be represented, abstract features can be obtained, and a better decision can be achieved finally for the computer network nodes to achieve improved network quality-of-service (QoS) and quality-of-experience (QoE).

Wireless networks yield complex features, such as communication signal characteristics, channel quality, queueing state of each node, path congestion situation, etc. On the other hand, there are many network control

Qian Mao is with the Department of Electrical and Computer Engineering, The University of Alabama, Tuscaloosa, AL, 35401 USA; e-mail: qmao3@crimson.ua.edu

Fei Hu (Corresponding author) is with the Department of Electrical and Computer Engineering, The University of Alabama, Tuscaloosa, AL, 35401 USA; e-mail: fei@eng.ua.edu

Qi Hao is with the Department of Computer Science and Engineering, Southern University of Science and Technology, China

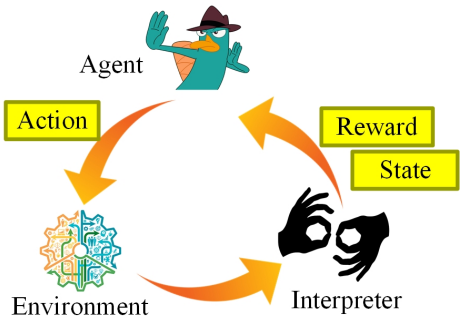


Fig. 1: Schematic Diagram of Reinforcement Learning

targets having significant impacts on the communication performances, such as resource allocation, queue management, congestion control, etc. To handle the complicated situations, machine learning technique has been extensively explored [18]. Chen et al. presented a comprehensive summary towards the ML applications in wireless networks, including wireless communications and networking using Unmanned Aerial Vehicles (UAVs), wireless virtual reality, mobile edge caching and computing, spectrum management and co-existence of multiple radio access, Internet of Things, etc [19]. The applications of ML in these areas present astonishing improvement compared to traditional methods.

On the other hand, since modern wireless networks are becoming more and more complex, more desires are brought to the learning system, such as higher computing capacity, bigger datasets, faster and more intelligent learning algorithms, more flexible input mechanism [19], etc. To meet these urges, deep learning applications in wireless networks has drawn lots of interests. DL equips the wireless network with a 'human brain': it accepts a large number of network performance parameters, such as link signal-to-noise ratios (SNRs), channel holding time, link access success/collision rates, routing delay, packet loss rate, bit error rate, etc., and performs deep analysis on the intrinsic patterns (such as congestion degree, interference alignment effect, hotspot distributions, etc.). Such patterns can be used to perform the protocol controls in different protocol layers. For example, the routing layer may start to look for a new alternate path; the transport layer can shrink the congestion window size, and so on. Compared to ML, DL tends to provide improved performances to abstract in-depth patterns from the input knowledge and to make more accurate decisions.

The success of applying DL for wireless networking is due to the following three similarities between DL and human brain:

(1) Tolerance of incomplete or even erroneous input raw data: The human brain can tolerate distorted samples. For example, we can still recognize the image shape of '1' even some sections of '1' are missing, and we can recognize people from an obscure face image even when some pixels are missing. Likewise, DL uses deep neural network to tolerate missing or distorted input data. This capability is important to wireless networks since it is not possible to accurately collect all the radio links' states due to the channel fading, node mobility, and control channel failure.

(2) Capability of handling large amount of input information: Human brain can simultaneously absorb multiple types of complex information and makes a good judgement. For example, we can use sound, image, and smell to detect the coming of a dog. Likewise, DL can simultaneously accept a huge amount of performance data from multiple protocol layers (such as 1000 nodes' queueing status data and link interference matrix), and then determines the concrete congestion place in a large network. DL will play a critical role in big data wireless transmissions due to its capability of analyzing the performance parameters of huge traffic flows.

(3) Capability of making control decisions: Our brains learn things and guide our behaviors. Passive learning may not be the final goal of network analysis. Using the learning results to guide the proper network control is the ultimate goal. With the Markov decision model, DL is able to evolve into deep reinforcement learning (DRL) model, which can use the updates of system states, reward function, and policy seeking, to make a suitable network control based on the maximum reward calculation. Thus we can use DRL to achieve large-scale wireless network control.

In this article, a comprehensive survey towards the applications of DL in wireless networks is presented. Fig. 2 shows the taxonomy we have used when reviewing various uses of DL for each network aspect. Our contributions in this review consist of 3 important aspects:

(1) *DL applications in different layers*: We will systematically analyze the benefits of adopting DL/DRL for the network feature extraction in different layers. As shown in Fig.2, in physical layer, DL can be used for interference alignment. It can also be used to classify the modulation modes, design efficient error correction codes, etc. In the data link layer, DL can be used for resource (such as channels) allocation, link quality evaluation, and so on. In network (routing) layer, it can help to seek an optimal routing path. In higher layers (such as application layer), it is used to enhance data compression and multi-session scheduling. We will provide the core design ideas for each DL application, and compare different solutions.

(2) *DL advantages in security and other network functions*: Besides the above protocol stack, we will also discuss the advantages of using DL in other network functions. One critical area is security and privacy protection. Today the intrusion detection becomes more challenging due to the network scale increase and huge amount of traffic passing through the attack detector/filters. DL is an ideal tool to perform large-scale network profile analysis to detect the potential intrusion events. We will explain how DL can be used to classify the packets into benign/malicious types, and how it can be integrated with other machine learning schemes such as unsupervised clustering, to achieve a better anomaly detection effect.

(3) *Future trends*: Since this field is still far from maturity and many issues are not solved yet, we will introduce 10 challenging problems on the use of DL to enhance some of the popular wireless networks, such as cognitive radio networks (CRNs), software-defined networks (SDNs), dew/fog comput-

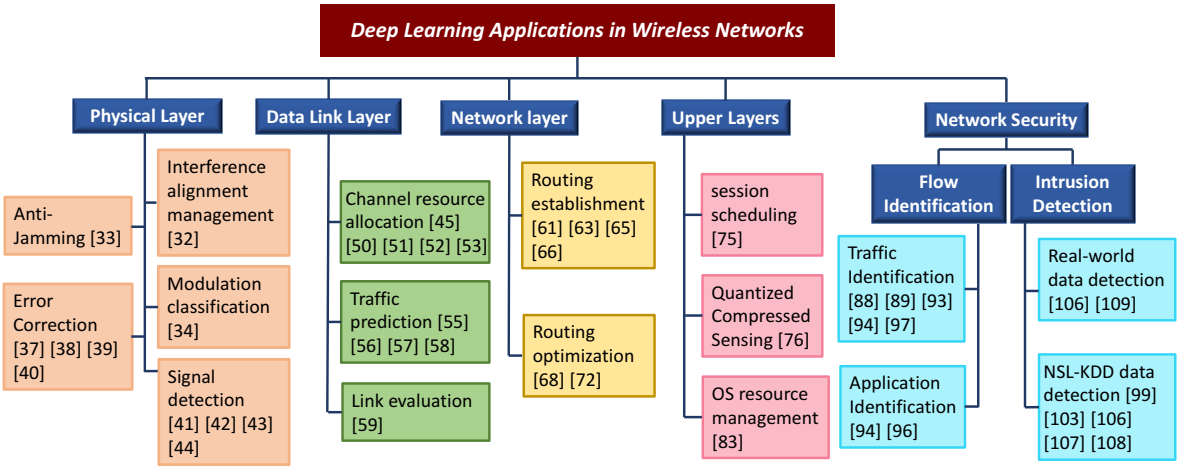


Fig. 2: Taxonomy of Deep Learning Applications in Wireless Networks

ing, etc. We will provide the context, motivation, problem statement, and concrete unsolved issues for each of those 10 problems. They are helpful to the readers who are seeking for new research directions.

Roadmap: The rest of this paper is organized as follows: In section II, to prepare for the discussions of DL applications for wireless network functions, we first explain the fundamental math models of DL, including its relations with general machine learning and graph-based learning framework. Then we move to the discussions of DL-based physical layer enhancements in terms of signal interference and modulation classification in section III. Section IV discusses the importance of DL in data link layer design. Some typical MAC design examples are explained with DL-based enhancements. In Section V the DL-based routing layer operations such as path establishment and optimization are described. The utilizations of DL for security and other network functions are discussed in Section VI. **Section VII summarizes some DL implementation platforms that have been extensively used in wireless network research.** Ten challenging research issues to be solved next are stated in Section VIII, followed by the concluding marks in Section IX.

II. Fundamentals of Deep Learning

DL originated from Machine Learning (ML). In this section, we first analyze the differences and relationship between those two techniques. Then, a brief introduction towards DL principles is presented.

A. From Machine Learning to Deep Learning

Both ML and DL solve real-world problems with neural networks. A typical ML system is composed of three parts: 1) Input layer, which takes pre-processed data as the system input. The features of the real-world data (e.g., pixel values, shape, texture, etc.) need to be pre-processed and identified by humans so that the ML system can deal with them. 2) Feature extraction and processing layer, in which a single layer of data processing is used to extract the data patterns. Currently, Support Vector Machines (SVM), Principal Component Analysis (PCA), Hidden Markov Model (HMM), etc., are extensively used for feature extraction. 3) Output layer, which spills out the results of classification, regression, clustering, density estimation, or dimensionality reduction, depending on the task of the ML model. The schematic structure of ML is shown in Fig. 3 (a).

The original data input into the learning system could be quite diverse, varying from natural information such as image, audio and video, to various quantitative event descriptions. Although the input of the learning system may be different, the core data learning module requires that the input data has a uniformed form, based on which the input events are classified. Therefore, to enable the learning process to "recognize" the input data, the original natural data needs to be pre-processed, i.e., the raw data needs to be transformed into a suitable representation or feature vector, which can be accepted by the ML classification system. This pre-processing needs to be sophisticatedly designed in such a way that the features of the original natural data related with classification are well preserved. And the classification accuracy is significantly affected by the data pre-processing schemes.

Machine learning systems usually have only one hidden layer between the input and output layers. This type of learning system is also referred to as shallow learning network, which provides arbitrary function approximator with enough hidden units in one hidden layer and learns more-or-less independent features from the input layer.

For example, Chen et al. proposed a radio map learning system based on the shallow learning network, which uses machine learning method to exploit the segmentation models and signal strength models of the UAV-assisted wireless networks and reconstructs a finely structured radio map to improve the service coverage [20].

On the contrary, most deep learning systems have more than one hidden layers between the input and output layers, where the input of an upper layer is the output of its lower layer, such as the learning networks proposed in [21], [22], [23], [24]. DL technique avoids the sophisticated input data pre-processing by employing multiple hidden layers between the input and the output layers, as shown in Fig. 3 (b). The natural data is input into the learning system in their raw form. The DL system then automatically extracts appropriate representations for classification or detection purpose. Starting with the natural data, each layer extracts different features from the input data, gradually amplifying features that are more relevant to decision making and suppressing irrelevant features. Each layer is connected to neighboring layers with different weights attached to the connection. To determine the values of the weights, a large number of samples are sent to the system for training purpose, which could be either supervised learning or unsupervised learning. In the supervised learning, a gradient vector is computed for each weight, indicating the amount of error change with the variation of that weight. According to the gradient vector, the weight is adjusted to decrease the error.

B. Deep Learning Framework

Human beings spontaneously interact with the environment by using a combination of reinforcement learning and hierarchical sensory processing system, to accomplish many tasks such as object recognition [25], conditioning and choice-making [26], etc. Inspired by animal behavior, deep reinforcement learning was proposed and has drawn much attention in computer intelligence. A DL model includes two crucial elements: forward feature abstraction and backward error feedback. The training process usually needs both elements, while the verification process solely implements the former.

Forward feature abstraction: Assume there are N layers in the DL network, as shown in Fig. 4. For the j-th node in layer i, denoted as n_{ij}, the output is obtained through two steps. First, node n_{ij} computes a weighted sum of all its inputs, denoted as z_{ij}. Then z_{ij} is sent to a non-linear function $f()$ to obtain the output y_{ij} of node n_{ij}.

$$z_{ij} = \sum_{k=1}^{L_{i-1}} w_{kj}^{i}, y_{ij} = f(z_{ij}), \tag{1}$$

where w_{kj}^{i} is the weight from node $n_{i-1,k}$ to node n_{ij} and L_{i-1} is the number of nodes for layer $i-1$. For the choice of the non-linear function $f()$, the rectified linear unit (ReLU) $f(z) = \max(0, z)$, the hyperbolic tangent function $f(z) = [\exp(z) - \exp(-z)]/[\exp(z) + \exp(-z)]$, and the logistic function $f(z) = 1/[1 + \exp(-z)]$ are the popular options [7].

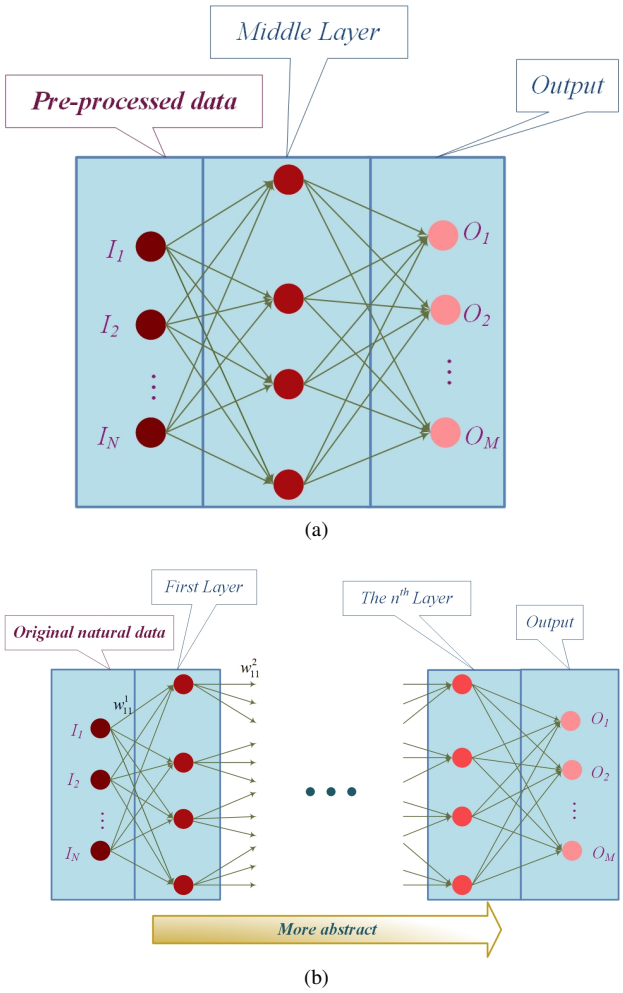


Fig. 3: Schematics of Machine Learning and Deep Learning (a) Machine Learning; (b) Deep Learning

Backward error feedback: The initial weights are random values or empirical values. To improve the accuracy of the final output of the learning system, these weight values are adjusted by backward error feedback technique, i.e., the classification accuracy is feedbacked, according to which the connection weights are modified. For a node of the deepest layer, say, node n_{Nj}, the error derivative is $y_{Nj} - t_{Nj}$, where y_{Nj} and t_{Nj} are the generated output and the correct output, respectively. Then the error derivative of lower layer connection is

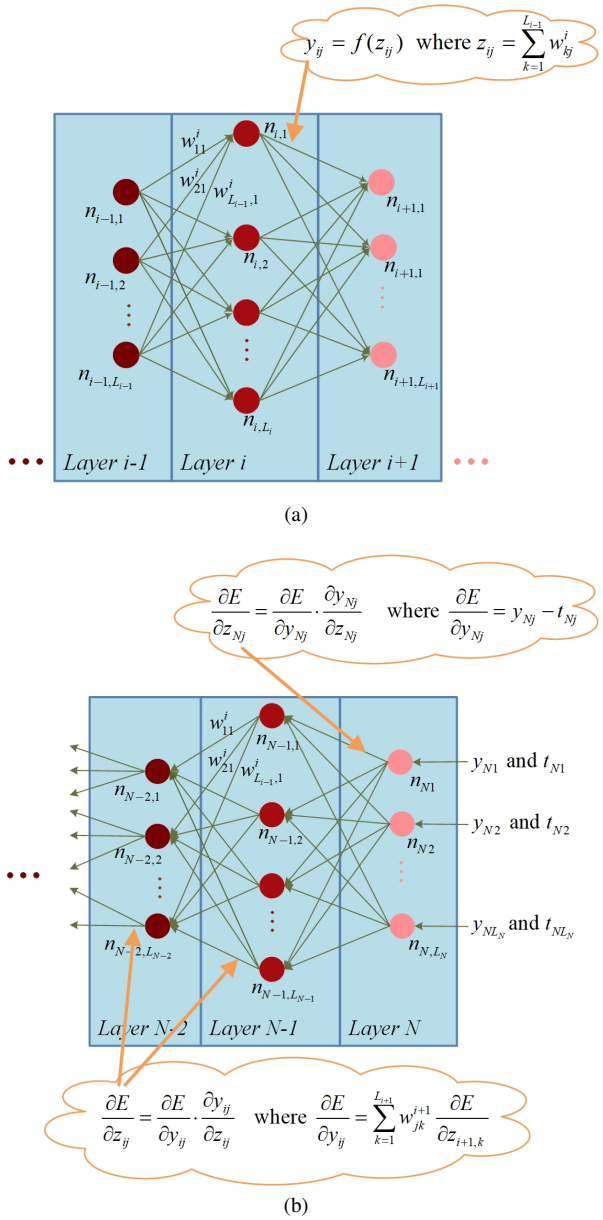
$$\frac{\partial E}{\partial z_{Nj}} = \frac{\partial E}{\partial y_{Nj}} \frac{\partial y_{Nj}}{\partial z_{Nj}}, \tag{2}$$

where $\partial E/\partial y_{Nj} = y_{Nj} - t_{Nj}$ and $j = 1, 2, \cdots, L_N$.

For the j-th node of layer i ($i = 1, 2, \cdots, N-1$), first a weighted sum of the error derivatives of all the inputs (from deeper layer) to the node is computed, denoted as $\partial E/\partial y_{ij}$. Then the error derivative of the lower layer connection is

$$\frac{\partial E}{\partial z_{ij}} = \frac{\partial E}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial z_{ij}}, \tag{3}$$

where $\frac{\partial E}{\partial y_{ij}} = \sum_{k=1}^{L_{i+1}} w_{jk}^{i+1} \frac{\partial E}{\partial z_{i+1,k}}$, $i = 1, 2, \cdots, N-1$, and $j = 1, 2, \cdots, L_i$.



(a)

(b)

Fig. 4: Deep Learning Operations
(a) Forward feature abstraction (b) Backward error feedback

In practice, stochastic gradient descent (SGD) is extensively used [7], since SGD finds a good set of weights with relatively fast speed. The training process of SGD is composed of many rounds, each of which is trained by a small set of samples, and the final gradient is the average of each round.

In circumstances that some useful environment features can be handcrafted, or the environment's state space is low-dimensional and can be fully observed, the performances of reinforcement learning is limited. Furthermore, reinforcement learning tends to be unstable or even diverge when a nonlinear function approximator is used to represent the reward. To solve these problems, deep Q-network (DQN) was proposed, which employs two novel strategies to overcome the instability problem of deep learning, i.e., experienced replay and iterative update [27], [28]. In DQN, the agent interacts with the environment through a sequence of actions, with the goal of maximizing the cumulative reward. Assuming the environment's state is s at moment t and the agent takes action a according to policy π, then, the environment transfers to the next state at moment $t+1$ according to environment's transferring probability P. Meanwhile, a reward r_t is given at moment t. The goal of the agent is to maximize the cumulative reward Q_t, i.e. [28],

$$\max_{\pi} Q_t = \max_{\pi} \mathrm{E}\left(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s_t = s, a_t = a, \pi\right), \tag{4}$$

where γ is a future reward discount, since the current action a_t impacts not only on the current reward, but also on the future reward with a diminuendo strength. In [27], the reward is represented as $Q_t(s, a, \theta_i)$, where θ_i is the weight of the Q-network at iteration i. To replay the experience, for each moment t, the agent's experience, $e_t = (s_t, a_t, r_t, s_{t+1})$, is stored in the experience pool $U(D)$. Each time when the agent needs to adopt an action during the learning process, a sample of the stored experience is randomly chosen by the agent. Thus,

$$L_i(\theta_i) = \mathrm{E}_{(s,a,r,s')\in U(D)}\left(\left(r + \gamma \max_{a'} Q(s', a', \theta_i^-) - Q(s, a, \theta_i)\right)^2\right), \tag{5}$$

where θ_i^- is the weight of the Q-network used to compute the target at iteration i, which is only updated with the weight θ_i every c steps and is fixed between individual updates (c is a constant). The parameterization of reward Q for each action is achieved by a neural network, where each possible action is provided a separate output unit. Therefore, for each possible action, only the state representation serves as the input to the network, yielding the predicted Q value for a specific action.

A classical application of DQN is video game (Atari 2600) [27], as shown if Fig. 5. For instant t, the state is the screenshot of the game, denoted as x_t. (Note that the internal state of the game is not accessible by the observer. Instead, only the screen can be observed.) The action is the operation that the Atari emulator takes, denoted as a_t, and the reward is the game score gained due to the action, denoted as r_t. To learn the strategy with fully observation, the state input to the DQN at instant t is a finite Markov Decision Process (MDP), indicating both the current observation and the previous observations and actions, as shown in Table I. Assuming the game terminates at instant T, the future discounted reward at instant t is $R_t = \sum_{i=t}^{T} r_i \gamma^{i-t}$. Apparently, the goal of the agent at instant t is to take an action that maximizes the future rewards with the current observation representation and policy π, i.e., $\max_{\pi} \mathrm{E}(R_t | s_t = s, a_t = a, \pi)$, which can be represented as the following equation,

$$Q^*(s, a) = \mathrm{E}_{s'}\left(r + \gamma \max_{a'} Q^*(s', a') | s, a\right), \tag{6}$$

where s' is the observation of the next instance and a' is the action taken at the current instance. In practice, an approximator is used to estimate the action reward, i.e., using $Q(s, a, \theta_i^-)$ to replace $Q^*(s, a)$, where θ_i^- is the weights of

the neural network in an iteration before i. (For instance, $\theta_i^- = \theta_{i-1}$.) Therefore, an approximate estimated reward value is

$$y = r + \gamma \max_{a'} Q(s', a', \theta_i^-), \tag{7}$$

For each round of the Q-network training, say round i, the training is implemented by adjusting the weights with the aim of reducing the mean square error of (7).

C. Deep Learning for Graph-Structured Data

In many practical applications the data often has the structured features, i.e., nodes are connected with each other spatially or temporally, or both. For instance, when predicting the behavior of a person in the kitchen, the interactions between the person and the appliances are connected spatially or temporally. Under such a circumstance, considering the spatio-temporal relations among nodes in the DL framework, we can use the graph-based structure to achieve the promising performance [29].

Currently, the graph-structed data is usually generalized as Convolutional Neural Networks (CNNs). A CNN is a sequence of layers, each of them transforms one volume of activations to another through a differentiable function. Usually there are three main types of layers in a CNN architecture, i.e., convolutional layer, pooling layer and fully connected layer. The main difference between the graph-oriented CNN and regular CNN is that the former builds graphs for each neural node of the learning system, which is achieved by selecting neighbors and determining the connection weights with the neighbors for each real-world node, as shown in Fig. 6 (a) and (b). To represent graphs in DL models, the input data is denoted as vertices and edges, i.e., $G = (V, E, A)$, where V represents the set of vertices, E represents the set of edges, and A is the weighted adjacency matrix. Then the graph is input into the learning system monolithically.

The graph DL can be conducted in either spectral domain or spatial domain [30]. The spatial approach generalizes CNN using the graph's spatial structure, capturing the essence of the convolution as an inner product of the parameters with spatially close neighbors, as shown in Fig. 6 (b). Bruna et al. [31] uses multi-scale clustering to define the network architecture, in which the convolutions are defined for each cluster. However, the spatial approaches tend to have difficulties in finding a shift-invariance convolution for non-grid data. To overcome this problem, Hechtlinger et al. [32] proposed a spatial CNN, which uses the relative distance between nodes. Assume $G = (V, E)$ is a graph, where $V = (X_1, X_2, \cdots, X_N)$ is a set of N features (vertices) and E is a set of edges. To select the neighbors for a node, a graph transition matrix, P, is used, of which element p_{ij} denotes the probability of moving from node X_i to node X_j. Meanwhile, the expected number of visits, Q^k, is calculated as $Q^k = \sum_{i=0}^{k} p_{ij}^k$, where $p_{ij}^k \in P^k$ is the probability of moving from node X_i to node X_j. The convolution for the node is conducted upon the top α neighbors with the highest expected visit numbers (α is a constant). Therefore, the weights are decided according to the distance indicated by the transition matrix.

The graph-oriented CNN can also perform on spectral domain graph, i.e., the input graph is first transmitted from graph domain to spectral domain, then the spectral domain graph is sent to CNN for training and testing purpose, as shown in Fig. 6 (c). For instance, in [31] and [33], the spectral decomposition of the graph Laplacian is first used to derive the eigenvectors of the spatial domain graph, then the convolution is implemented upon the spectral graphs. Furthermore, in [34] and [35], a ChebNet is proposed, which uses Chebyshev polynomials of the Laplacian to learn the filter structures for the graph data. Lee et al. [36] proposed a DL scheme performed in spectral domain, which incorporates transfer learning to allow training data and testing data to be drawn from different feature spaces and distributions. **Transfer learning stores knowledge gained from solving one set of problems and applies it to a different but related problem set. By using transfer learning in deep learning networks, the intrinsic information learned by the DL network is transferred from the source domain to the target domain, thereby building a model for a new but related task in the target domain without using new data [37]. In Lee's scheme, a graph is first generated by the co-occurrence graph estimation (CoGE) [29] or supervised graph estimation (SGE) [33]. Then, the intrinsic geometric characteristics of the graph is extracted via Laplacian matrix, where three Laplacian operators are used for comparison purpose. Following that, the convolutional networks are applied to the graph, for which the weights of the DL model are determined through the training process. To learn various data features, the convolution operation is re-defined with spectral information from spatial domain. This operation allows the DL model to transfer the data-driven structural features from the original domain to an appropriate spectral domain, thereby the intrinsic geometric information of the spatial graphs is effectively extracted.**

III. Deep Learning for Physical Layer Design

DL plays important roles in the Physical Layer (PL) of wireless networks. For instance, DL can help to determine the most suitable modulation/encoding schemes according to the comprehensive analysis of the complex radio conditions, including spectrum availability, interference distribution, node mobility, application types, etc. In the following discussions, we will provide some typical DL applications for PL function control.

A. DL for Interference Alignment

Interference Alignment (IA) has attracted extensive interests nowadays, for its improved channel utilization by allowing multiple transmitter-receiver pairs communicating via the same radio resources. In Multi-Input Multi-Output (MIMO) networks, IA uses linear precoding technique to align transmission signals in such a way that the interference signal lies in a reduced dimensional subspace at each receiver [38], [39]. This coordination between the transmitter and receiver

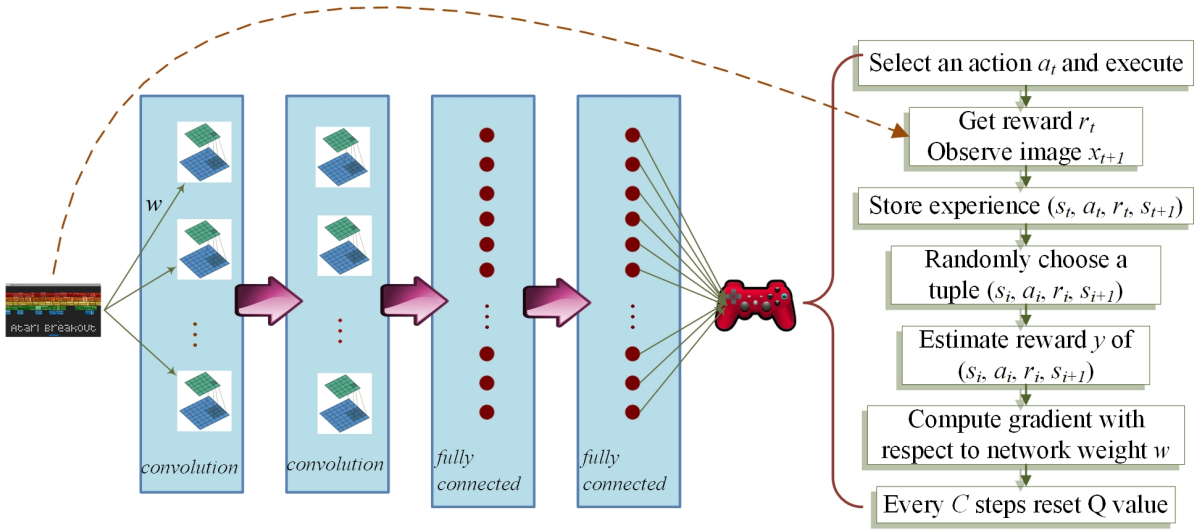


Fig. 5: Schematic of Deep Q-Learning for Video Game

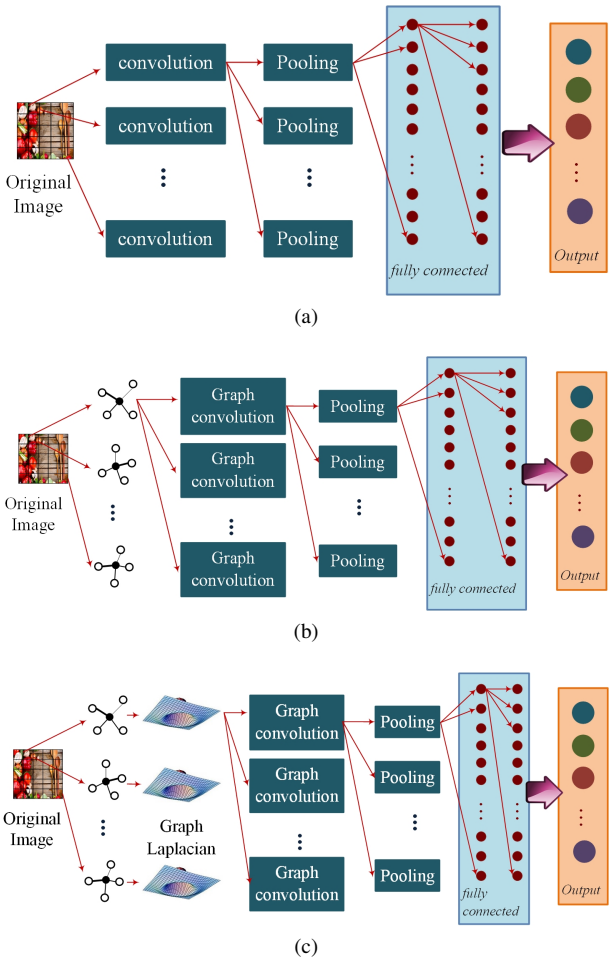


Fig. 6: Convolutional Neural Networks
(a) Grid Structure CNN; (b) Graph Structure CNN in Spatial Domain; (c) Graph Structure CNN in Spectral Domain

breaks the throughput limitation imposed by MIMO antennas' interference problem.

He et al. [40] proposed to use deep Q-learning to obtain the optimal IA user selection policy in the cache-enabled opportunistic IA networks, as shown in Fig. 7. In this scheme, a central scheduler collects the channel condition and the cache status of each user, and allocates channel resources to each user. All the users are connected to the central scheduler via a backhaul network with a total capacity of C_{total}. The channel is time-varying characterized by finite-state Markov model. Each transmitter is equipped with a cache, which stores an amount of frequently requested information. This in-network caching design efficiently reduces the transmission of duplicate contents. Assume that there are L candidates wanting to join the IA network, an action is determined in each time slot, indicating which candidates are chosen to be allocated communication resources based on their current Signal-to-Noise Ratios (SNRs). The system state at time slot t is defined as $x(t) = \{\gamma_1(t), c_1(t), \gamma_2(t), c_2(t), \cdots, \gamma_L(t), c_L(t)\}$, where $\gamma_i(t)$ and $c_i(t)$ denote the channel state and the cache state of candidate i, respectively ($i = 1, 2, \cdots, L$). System action is represented as $a(t) = \{a_1(t), a_2(t), \cdots, a_L(t)\}$, where $a_i(t) = 0$ indicates that candidate i is not selected to be allocated communication resources, and $a_i(t) = 1$ indicates being selected. The reward function is defined as to maximize the throughput of the IA network. If the requested content is in the local cache, the candidate is provided the maximum data rate by the IA. However, if the content is not in candidate's cache, certain amount of bandwidth needs to be used for content transmission.

Note that in this scheme, a central scheduler is needed, which collects channel state information as the input of the deep Q network and implements the resource allocation computation. This structure arouses vulnerability of the entire system. Furthermore, the employment of the central scheduler is intractable.

B. DL for Jamming Resistance

In cognitive radio networks, when the secondary users (SUs) tries to join the network, they need to 1) avoid interfering with

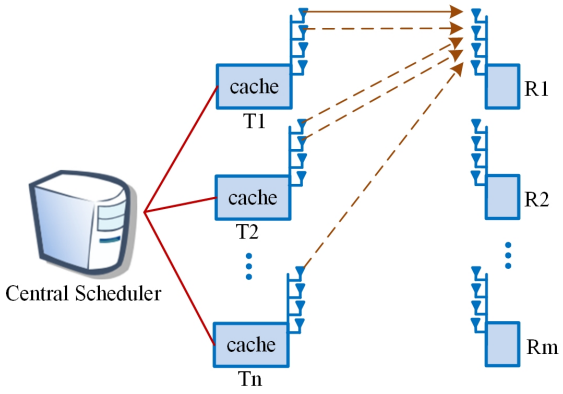


Fig. 7: Cache-Enabled Opportunistic IA Networks

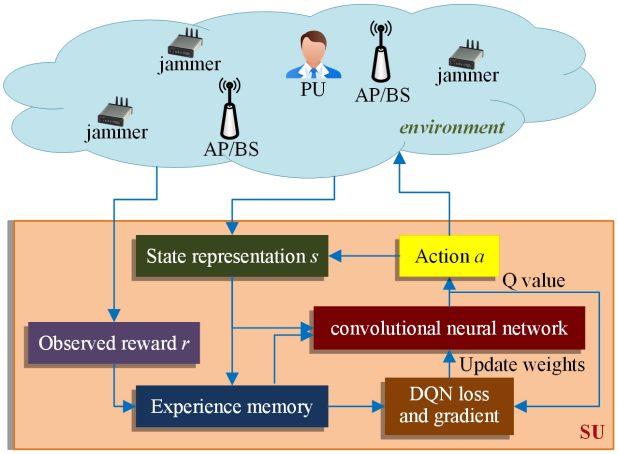


Fig. 8: DQN-based Anti-Jamming System

the primary users (PUs) and 2) counteract jammers. Spread spectrum is one of the most popular anti-jamming techniques. However, smart and cooperative jammers can still block some channels and eavesdrop the control channel.

Han et al. [41] proposed a deep Q-learning based, two-dimensional anti-jamming scheme executed by the SUs. It utilizes both frequency hopping and SU's mobility to confront smart jammers, as shown in Fig. 8. In this scheme, the action executed by the SU is represented as $a_t \in \{0, 1, \cdots, N\}$, where $a_t = 1, 2, \cdots, N$ indicates which frequency channel the SU is going to access (frequency hopping) , and $a_t = 0$ indicates that the SU will leave the area and find another Access Point (AP) due to heavy jamming (mobility). **However, it is not clearly addressed how does the SU move. An efficient moving strategy of SU is crucial to find an optimal access point and to avoid duplicated request.** The scheme is summarized in Table I, where N is the number of frequency channels, W is memory length, r is the utility of the SU. Considering the huge number of frequency channels and the time constraint of the decision making process, a convolutional neural network adopted in [27] was used to estimate the reward for each action, which consists of two convolutional layers and two fully connected layers (see Fig. 5).

C. DL for Modulation Classification

Modulation classification identifies the modulation type for the received signals. To improve the accuracy of the classification for complex modulation signals, Peng et al. [42] proposed a CNN-based DL scheme. Since different modulation methods may have particular constellation diagrams, this scheme uses AlexNet to classify constellation diagrams, thereby pinpointing the modulation method. AlexNet is a CNN based deep learning model that comprises thousands of neurons and millions of connections, and it can classify 1.2 million images into 1000 classes [43], [44]. Simulation shows that this scheme can accurately differentiate QPSK, 8PSK, 16QAM and 64QAM signals and has comparable accuracy compared to the traditional modulation classification schemes such as cumulant based scheme and Support Vector Machine (SVM) based scheme. **The DL-based modulation classification is a promising topic. However, merely considering the graphic pattern of the constellation diagram may limits the classification effect. The classification performance could be improved if the image classification is aimed by modulation parameters analysis.**

D. DL for Physical Coding

In addition, deep learning is also used in error correcting codes. In [45] and [46], the belief propagation (BP) decoding algorithm of the low-density parity-check (LDPC) codes is improved by DL. Tanner graph is extensively used in the BP decoding process. However, it is a challenging task to build an efficient parity check matrix, which can be expressed by the edges of the Tanner graph. Nachmani et al. [46] assigned weight to each edge of the Tanner graph, then these weights are trained using stochastic gradient descent. Using this Tanner graph trained by the DL, the Bit Error Rate (BER) is significantly decreased. In [47] and [48], the performance of polar codes is improved by using a decoding algorithm trained by DL. These works represent a promising application of DL, i.e., to learn a structure-based decoding network. In addition, DL networks are also used in signal detection scenarios, such as multiple input multiple output (MIMO) signal detection [49], [50] and chemical signal detection [51]. Using the detection models optimized by DL, the transmission signals are more accurately deduced and the BER is decreased as a consequence.

O'Shea et al. [52] proposed a novel idea which treats the physical layer as an end-to-end autoencoder. The autoencoder includes the functions of modulation, error correcting coding, signal classification, etc. Then the autoencoder is trained as a CNN. This approach is tested in a single end-to-end communication system, multiple-transmitter/receiver system and radio transformer networks. Compared to the traditional modulation methods (e.g., BPSK and QAM) and error correcting code (e.g., Hamming code), the autoencoder decreases the block error rate (BLER) by 1-5 times in multiple-transmitter/receiver systems, and decreases the BLER by 1-7 times in Rayleigh fading communication scenarios (with radio transformer networks). A comparison on the DL applications in error correcting codes and signal detection is present in Table II.

TABLE I: Parameters of Deep Learning Scheme for Anti-Jamming

Item	Content	Characteristics
State Representation	$s_t = [(\lambda_{t-W-1}, SINR_{t-W-1}, a_{t-W}), (\lambda_{t-W}, SINR_{t-W}, a_{t-W+1}), \cdots, (\lambda_{t-2}, SINR_{t-2}, a_{t-1}), (\lambda_{t-1}, SINR_{t-1})]$	λ indicates the presence of PU (1 if the PU is present). $SINR$ indicates the signal-to-interference-plus-noise ratio.
Action	$a_t \in \{0, 1, \cdots, N\}$	Determine which channel to stay. SU leaves the AP if $a_t = 0$.
Optimal Reward	$Q*(s,a) \approx E_{s'}\left(r + \gamma \max_{a'} Q(s', a', \theta_i^-)\right)$	Using previous network weights as approximation
Loss function	$L(\theta) = E_{s,a,r,s'}\left(\left(r + \gamma \max_{a'} Q(s', a', \theta_i^-) - Q(s, a, \theta_i)\right)^2\right)$	

TABLE II: Comparison of Deep Learning Applications in Error Correcting Codes and Signal Detection

Scheme	Application Scenario	Algorithm	Performance
Nachmani's scheme [46]	Decoding for low-density parity-check (LDPC) codes	BP-RNN decoder	Decreased the BER by 0.2dB and 0.6dB compared to BP feed-forward decoding for a (63,45) BCH code with regular parity check matrix and cycle reduced parity check matrix, respectively.
Gruber's scheme [47]	Decoding for polar codes and random codes	Neural network decoding	The BER yielded by a large NN (larger than 512-256-128) is lower than the maximum a posteriori (MAP) for both polar code and random code.
Cammerer's scheme [48]	Decoding for polar codes	Neural network decoding	The BER is similar to successive cancellation (SC) decoding and conventional belief propagation (BP) decoding but the latency is much lower.
Samuel's scheme [49]	Signal detection in MIMO system	CNN	Decreased BER by 1-16 times in fixed channel and by 0.5-7.5 times in varying channel conditions compared to traditional methods.
Jeon's scheme [50]	Signal detection in MIMO system	Minimum-mean-distance/ minimum-center-distance detections	Decreased BER by 6-30 times with one-bit ADC compared to traditional methods.
Farsad's scheme [51]	Signal detection	CNN, LSTM3 and CNN-LSTM3	Decreased BER by 4-100 times with different symbol intervals compared to non-DL system.
O'Shea's scheme [52]	autoencoder	CNN	Decreased block error rate (BLER) by 1-5 times in multiple-transmitter/receiver systems and by 1-7 times in Rayleigh fading systems, compared to traditional methods.

E. A Brief Discussion on DL Application in Physical Layer

In wireless networks, interference alignment and jamming resistance are two of the trickiest problems, considering the large number of nodes, the nodes' mobility, the variation of channel conditions, the complex frequency usage, etc. DL is an ideal tool to deal with the complicated problems, since it abstracts the intrinsic patterns from hybrid and vast physical layer parameters. In addition, modulation and error correction coding are basic functions of the physical layer, which tend to demand huge computations in modern networks, such as Orthogonal frequency-division multiplexing (OFDM) modulation, Trellis coded modulation (TCM), Turbo codes, LDPC codes, etc. The performances of these operations could be significantly improved by DL technique. However, most physical layer problems have strict limitation towards reaction time, therefore, the employment of the DL server and the computational complexity control are crucial issues of the DL applications in physical layer.

IV. DL For Data Link Layer

A. DL for Spectrum Allocation

LTE-U allows Small Base Station (SBS) to access the unlicensed spectrum, thereby providing an efficient solution in radio spectrum utilization. To efficiently and proactively allocate the unlicensed spectrum, Challita et. al [53] proposed a resource allocation scheme for LTE in unlicensed spectrum (LTE-U) by using Reinforcement Learning and Long Short-Term Memory (RL-LSTM). In this scheme, the time domain is divided into multiple time windows, denoted as T. Each window is further divided into multiple time epochs, denoted as t. Assume there are J SBSs and $M - J$ WiFi stations. Each node has a LSTM encoder unit, which learns a vector representing the historical traffic loads of the SBS or the WiFi station [54], as shown in Fig. 9. All LSTMs comprise a traffic encoder. Following that, a Multi-Layer Perceptron (MLP) abstracts all the historical traffic load vectors as a single vector, which indicates the traffic values of all SBSs and WiFi stations on all the unlicensed channels. Finally, an action decoder interprets the abstract vector into multiple predicted action sequences for the SBSs. For a SBS j, the goal is to maximize the total throughput, u_j, during its allocated airtime

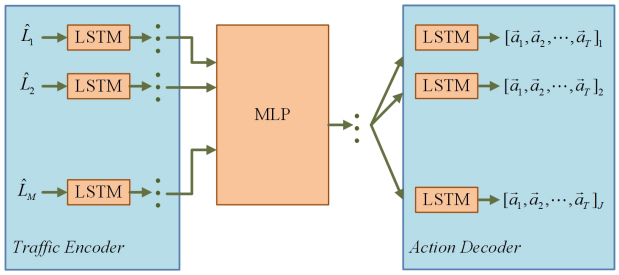


Fig. 9: RL-LSTM based Scheme for Spectrum Allocation

with the selected channel C and the time window T, i.e.,

$$u_j(a_j, a_{-j}) = \sum_{t=1}^{T} \sum_{c=1}^{C} \alpha_{j,c,t} \gamma_{j,c,t} \tag{8}$$

where a_j denotes the action vector of SBS j, a_{-j} denotes the action vector of all other SBSs except j, $\alpha_{j,c,t}$ is the achievable airtime fraction of SBS j on channel c in time epoch t, and $\gamma_{j,c,t}$ is a channel-related parameter. To achieve the optimization goal, the RL algorithm is used to train the weights of the traffic encoder and the action decoder, for which the reward is defined as the approximation of the SBS's throughput, $\hat{u}_j(a_j, a_{-j})$. By maximizing the expected reward $\hat{u}_j(a_j, a_{-j})$ according to the gradient with respect to the policy parameters, the weights of the RL neural network can be trained [55], [56]. The simulations were run upon the dataset provided by [57]. Compared to the traditional reactive allocation approaches, this scheme increases the average airtime allocated for LTE-U by around 18%.

Cloud Radio Access Network (RAN) is proposed for future cellular networks, e.g., 5G, which is a centralized, cloud-computing-based radio access network. In a cloud RAN, there are a central Base Band Unit (BBU) pool in the cloud and many distributed Remote Radio Heads (RRHs) near the users. The RRHs only maintain basic transmission functions, compressing and forwarding users' radio signals to the BBUs via fronthaul links. The resource allocation problem, i.e., how to minimize power consumption of the RRHs while satisfying users' demands, has become one of the main tasks in cloud RANs. To tackle this problem, Xu et al. [58] proposed a DL-based scheme for power-efficient resource allocation in RANs. In the scheme, the decisions are made via two steps: first, using a deep Q-learning algorithm to determine which RRHs should be turned on or switched into sleep status; second, using convex optimization algorithm to calculate the beamforming weights from the RRH to the user for all the active RRHs. The parameters of the first step, i.e., Q-learning operation, are shown in Table III. The state representation of time slot t is $s_t = (m_1, m_2, \cdots, m_R, d_1, d_2, \cdots, d_U)$, where $m_i \in \{0, 1\}$ denotes whether the i-th RRH is active or sleep, R is the total number of RRHs, $d_j \in [d_{\min}, d_{\max}]$ represents the demand of the j-th active RRH. In each time slot, the DRL agent determines which RRH is active. The immediate reward is defined as the gap between the maximum possible power consumption $P_{\max}$ and the actual power consumption, i.e., $P_{\max} - P(A, S, G)$, where the actual power consumption $P(A, S, G)$ comprises actual power consumption and transition power amount (due to sleep/active switch), i.e.,

$$\begin{aligned} P(A, S, G) = \sum_{r \in A} \sum_{u \in U} \frac{1}{\eta_l} |w_{r,u}|^2 + \sum_{r \in A} P_{r,active} + \\ \sum_{r \in S} P_{r,sleep} + \sum_{r \in G} P_{r,transition}, \end{aligned} \tag{9}$$

where $w_{r,u}$ is the beamforming weight from RRH r to user u, η_l is the drain efficiency constant of the power amplifier, A, S and G represent the sets of active, sleep and transition RRHs, respectively, U is the user set, and $P_{r,active}$, $P_{r,sleep}$ and $P_{r,transition}$ are RRH's power consumptions in active, sleep and transition modes, respectively.

For the active RRHs selected by the first step, the DRL agent computes the optimal beamforming weights by solving the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{r \in A} \sum_{u \in U} \frac{1}{\eta_l} |w_{r,u}|^2 \\ \text{subject to} \quad & SINR_u \geq \gamma_u, \qquad u \in U \\ & \sum_{u \in U} \frac{1}{\eta_l} |w_{r,u}|^2 \leq P_r, \quad r \in A \end{aligned} \tag{10}$$

where R_u is user u's demanded data rate, P_r is RRH r's maximum allowable transmit power, and $\gamma_u = \Gamma_m(2^{R_u/B} - 1)$ (B is the channel bandwidth, and Γ_m is the SNR gap depending on modulation). Compared to the single BS association approach, this scheme is shown to satisfy the users' demands better when the amount of demand is high. And compared to the full coordinated association approach, this scheme consumes less power.

Sun et al. [59] proposed a DL-based wireless resource allocation scheme. This scheme puts the resources into a "black box" and trains a DNN in such a way that the power allocation for each transmitter is optimized and the system throughput is maximized. The input layer of the DNN is fully connected, the multiple hidden layers use the Rectified Linear Unite (ReLU), $\max(x, 0)$, as the activation function, and the output layer uses $\min(\max(x, 0), P_{\max})$ to intake the resource constraints, where x is the input of a neural node and $P_{\max}$ is the power budget of each transmitter. The proposed DL-based power allocation scheme is tested in Gaussian interference channel (IC) and multi-cell interfering multiple-access channel (IMAC), respectively, and it is shown that compared to random power allocation and maximum power allocation, the DL-based scheme provides much higher throughput, and compared to WMMSE [60], the throughput of the DL-based scheme is close to WMMSE while the computation time is much shorter.

To maximize the channel utility for multi-user wireless networks with less computation and limited observations, Naparstek et al. [61] proposed a deep multi-user reinforcement learning approach. Assume that there are N users randomly accessing K orthogonal channels. At each time slot, a user accesses a channel with a certain probability. If there is no interference during the channel access and the message is successfully received, a positive ACK will be received by the

TABLE III: Parameters of Deep Q-Learning Scheme for Resource Allocation

Item	Content	Characteristics
State Representation	$s_t = (m_1, m_2, \cdots, m_R, d_1, d_2, \cdots, d_U)$	
Action	Pick one RRH and set it active	
Immediate Reward	$P_{\max} - P(A, S, G)$	Using previous network weights as approximation

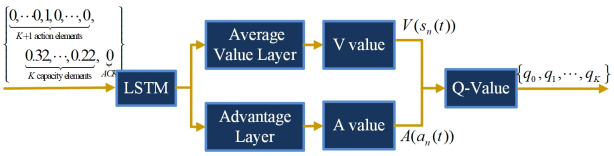


Fig. 10: Multi-User DQN for Spectrum Access Optimization

source node. To optimize the channel utility, the action of user n at time slot t is defined as a vector of size $K+1$, i.e., $a_n(t) = (0, 0, \cdots, 0, 1, 0, \cdots, 0)$, where the single 1 indicates the channel chosen by the user. (If the first element is 1, it indicates that no channel is chosen.) Meanwhile, define $a_{-n}(t) = \{a_i(t)\}|_{i \neq n}$ as the action of any other user except user n. The ACK message serves as the observation, i.e., $o_n(t) = 1$ indicates a successful delivery and $o_n(t) = 0$ indicates a failed delivery. For user n, the history is defined as $H_n(t) = (a_n(1), \cdots, a_n(t), o_n(1), \cdots, o_n(t))$, and the policy $\sigma_n(t)$ is the set of weights when mapping from history $H_n(t-1)$ to action $a_n(t)$. The accumulated discounted reward of user n is $R_n = \sum_{t=1}^{T} \gamma_{t-1} r_n(t)$, where $r_n(t)$ is the reward of user n at time slot t, which depends on both $a_n(t-1)$ and $a_{-n}(t-1)$, γ is the discount factor, and T is time duration. For an arbitrary user, say user n, the goal of training is to find a policy that maximizes the expected accumulated reward for the user, i.e., $\max_{\sigma_n} \mathrm{E}[R_n|\sigma_n]$.

The architecture of the multi-user DQN for spectrum access optimization of user n is shown in Fig. 10. The input is composed of user n's action $a_n(t-1)$, the capacity of each channel, and the observation of user n, $o_n(t-1)$. An LSTM is adopted to maintain internal states and to aggregate observations, since the network state is partially observed and depends on multiple users. Since some states are independent of the users' action, a duel DQN is adopted to achieve accurate estimation. The V-value DQN estimates the average Q-value of the state $V(s_n(t))$. The A-value DQN estimates the advantage of each action $A(a_n(t))$. Then the final Q-value of the action $a_n(t)$ is composed of $V(s_n(t))$ and $A(a_n(t))$. The output of the system is a vector with size $K+1$, each element of which indicates the estimated Q-value for the transmitted message in the corresponding channel, including no transmission (indicated by the first element of the output). It was shown that this scheme almost doubled the average channel utilization compared to traditional slotted-Aloha scheme [62].

Most DL-based spectrum allocation schemes first estimate channel conditions. Based on the channel-related parameters the rewards of an action (e.g. throughput, power consumption, etc.) are determined by the deep learning system. For example, in (11), the channel-related parameters are used to calculate the throughput of the small base station, and in Xu's scheme [58], the channel-related parameters are used to calculate the achievable data rate for users. However, the real channels might be so complicated that the channel estimations are not accurate, which may cause bias to the DL training, thereby resulting in deteriorate decisions of the spectrum allocation. Therefore, an accurate estimation towards the channel condition is a crucial and challenging question in the DL-based spectrum allocation models.

B. DL for Traffic Prediction

Most exiting schemes that optimize resource allocation assume some given factors, such as traffic load, spectrum usage, etc. However, Wang et al. [63] pointed out that these factors could vary significantly both temporally and spatially. Therefore, simply assuming constant values for these parameters may deteriorate the effect of resource allocation. To solve this problem, a spatiotemporal modeling scheme based on hybrid DL was proposed in [63] to predict the traffic in cellular networks. In this scheme, an auto encoder model, which consists of a Global Stacked Auto Encoder (GSAE) and multiple Local Stacked Auto Encoders (LSAEs), is used for spatial modeling. Meanwhile, the long short-term memory units (LSTMs) are adopted for temporal modeling, as shown in Fig. 11. When predicting the traffic of a cell, the historical data of both the cell itself (red hexagon) and its neighboring cells (green hexagons) are collected. Each cell has its LSAE for representation encoding. Meanwhile, a GSAE takes all the cell data and produces a global representation. The local representation is combined with global representation to produce spatial modeling and prediction. The output of spatial modeling is then sent to LSTM for temporal modeling and prediction. Using this spatiotemporal modeling scheme, the traffic of a large LTE network with 2,844 base stations (BSs) and a coverage of 6,500 km^2 is precisely predicted, and the parameters of Mean Square Error (MSE), Mean Absolute Error (MAE) and Log Loss are measured to evaluate the prediction performance. It was shown that such a scheme had a significant improvement compared to Auto Regression Integrated Moving Average (ARIMA) [64] and Support Vector Regression (SVR) [65], [66], which are two most widely used methods for time series analysis.

To predict the traffic for a cell, the traffics of both the cell itself and its neighbors are input into the LSAE and GSAE. The size of the neighboring region should be carefully balanced between the prediction accuracy and the computation. In the simulations in [63], a 11×11 square is used as a neighboring region, i.e., for each cell, the traffics of its 120 neighboring cells are considered. Another tricky

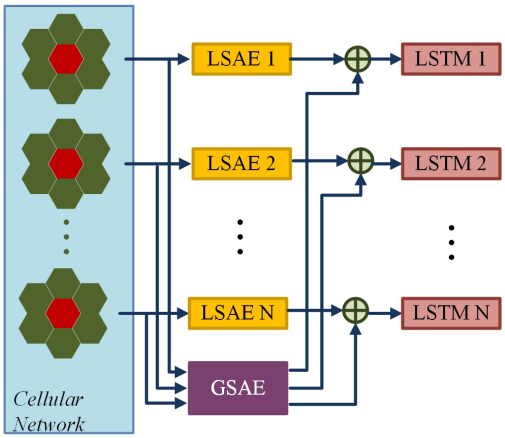


Fig. 11: Spatiotemporal Hybrid Modeling System for Traffic Prediction

issue of traffic prediction is the temporal correlation, which may have periodicity in the range of day, week, month, and year.

C. DL for Link Evaluation

Due to the huge size and complex structure of modern networks (such as multi-layer structure, heterogeneous characteristics, hybrid network resources, etc.), the scale of network optimization problem tends to be enormous. Therefore, reducing the computational complexity is a critical problem. For the link evaluation based optimization problem, Liu et al. [67] proposed to reduce the problem size instead of to reduce the algorithm complexity. In their scheme, one possible status of all virtual links is defined as a network pattern (denoted as set A), and the goal is to minimize the overall power consumption by scheduling all the patterns appropriately. This optimization goal can be achieved by solving a Linear Programming (LP) problem, for which the objective function is $\min E = \sum_{a \in A} P_a t_a$. Here P_a is the power consumption of pattern a, and t_a is the active time. However, the problem scale is huge due to the large number of virtual links. To reduce the problem size, the authors suggest that many virtual links of the network would not be scheduled, or merely carry a small amount of traffic. If these links are excluded from the LP problem, the computation will be magnificently decreased without much degradation of the optimization objective. Therefore, a Deep Belief Network (DBN) [68] is first used before the LP model, which evaluates the link quality. The input of the DBN is flow information, which is represented by a flow demand vector $X = (x_1, x_2, \cdots, x_N)$, where N is the total number of network nodes. For a flow with a demand of d_c and travelling from the source node n_s to the destination node n_d, the element of X is $x_i = d_c$ if $i = n_s$, $x_i = -d_c$ if $i = n_d$, otherwise $x_i = 0$ ($i = 1, 2, \cdots, N$). The output of the DBN is the evaluation values of all links, $Y = (y_1, y_2, \cdots, y_M)$, where M is the number of links in the entire network. Each element of Y, denoted as y_i, indicates the probability of the input flow belonging to link i ($i = 1, 2, \cdots, M$). Apparently, the higher the value of y_i is, the more likely that link i will be used by the flow. Based on the evaluation results, the links that are not likely to be scheduled for a flow will be excluded from the link optimization process. This approach efficiently reduces the problem size of link optimization. Simulation results show that the scheme reduces the computation cost by at least 50% without decreasing the optimization performance.

D. A Brief Discussion on DL Application in Data Link Layer

The applications of DL in data link layer are mostly focusing on resource allocation, traffic prediction, and link evaluation problems, which yield promising performance improvement, as shown in Table IV. **Considering the large size of modern network, DL system usually needs to read tremendous DLL parameters to make a decision. Therefore, how to limit computation and data size are huge challenges for the deep learning applications in DLL. Meanwhile, accurate estimations towards the channel conditions are crucial for the deep learning system to make accurate DLL decisions, which is challenging due to the fast channel variations and the time limit of the decision-making process.**

V. Routing Layer

Modern routing protocols developed for wireless networks are basically categorized into four types: routing-table-based proactive protocols, on-demand reactive protocols, geographical protocols, and ML/DL-based routing protocols. DL-based routing protocols have been extensively studied in the past several years due to its superior performance for complex networks.

A. Lifetime-Aware Routing based on RL

Underwater sensor network usually confronts two big challenges, i.e., a large propagation delay due to the use of acoustic channels and the stringent power usage due to the high power consumption and inconvenience of battery charging. To deal with these challenges, a balanced routing protocol that distributes traffic evenly among all sensors was suggested in [69], which proposed an adaptive, energy-efficient, and lifetime-aware routing scheme, called QELAR, based on Q-learning algorithm. For the Q-learning model $[S, A, P_a(s, s'), R_a(s, s')]$, where S, A, P and R are the set of states, actions, state transition probabilities and rewards, the value of taking action a in state s under a policy π, $Q_\pi(s, a)$, is defined as

$$\begin{aligned} Q_\pi(s,a) &= E_\pi\{R_t | s_t = s, a_t = a\} \\ &= E_\pi\{\sum_{k=0}^{\infty} \gamma_k r_{t+k} | s_t = s, a_t = a\}. \end{aligned} \tag{11}$$

The optimal value of state s is defined as $V^*(s) = \max_a Q^*(s, a)$. To consider the nodes' energy condition, the scheme assumes that the residual energy of a node is $E_{res}(s)$, the initial energy of a node is $E_{init}(s)$, and the average residual energy in the group including the node is $\overline{E}(s)$. If a packet is successfully transferred from node s to node s', the reward is

$$R_{a'}(s, s') = -g - \alpha_1[c(s) + c(s')] + \alpha_2[d(s) + c(s')], \tag{12}$$

TABLE IV: Comparison of Deep Learning Applications in Data Link Layer

Scheme	Application Scenario	Algorithm	Goal	Performance
Challita's scheme [53]	unlicensed spectrum allocation in LTE-U	RL-LSTM	maximize airtime allocated for LTE-U	Increased average airtime by about 18% compared to reactive approach
Xu's scheme [58]	Cloud Radio Access Network	Deep Q-learning	Minimize power consumption	Decreased power consumption by 0-20% compared to single BS association and full coordinated association
Sun's scheme [59]	Gaussian IC and Multi-cell IMAC	DNN	Optimize power allocation to increase throughput	Compared to WMMSE, the throughput is similar while the computation time is much shorter
Naparstek's scheme [61]	Multi-user wireless network	Deep Q-Network with LSTM	Maximize total channel utility	Doubled average channel utilization compared to slotted-Aloha protocol
Wang's scheme [63]	LTE network	Hybrid deep learning (auto-encoder & LSTM)	Traffic prediction	Decreased prediction errors by 18%-40% compared to ARIMA and SVR
Liu's scheme [67]	Generic wireless network	Deep belief net	Link evaluation	Decreased computation cost by 50% by eliminating unused links from LP

where $c(s) = 1 - E_{\text{res}}(s)/E_{\text{init}}(s)$ and $d(s) = \frac{2}{\pi}\arctan(E_{\text{res}}(s) - \overline{E}(s))$ are residual energy-related rewards, α_1 and α_2 are their weights, and g is a punishment coefficient due to power consumption when a node attempts to forward a packet. On the other hand, if the packet forwarding from node s to node s' fails, the reward is

$$R_{a'}(s, s) = -g - \beta_1 c(s) + \beta_2 d(s), \tag{13}$$

where β_1 and β_2 are weights. Then the overall reward r_t in (14) is

$$r_t = P_{a'}(s, s')R_{a'}(s, s') + P_{a'}(s, s)R_{a'}(s, s). \tag{14}$$

where $P_{a'}(s, s')$ is the transition probability from node s to node s' with action a' and $P_{a'}(s, s)$ is the transition probability from node s to node s with action a' (failed data forwarding). For instance, in the network as shown in Fig. 12, node s_1 wants to send packets to node s_4. Initially, all the Q values and V values are set as 0s, and let $\gamma = 0.5$ and $g = 1$. If the nodes' residual energy is not considered, $\alpha_1 = \alpha_2 = 0$. For node s_1, since its immediate neighbors are nodes s_2 and s_3, it calculates the following Q values:

$$\begin{aligned} Q(s_1, a_2) &= r_t + \gamma(P^{a_2}_{s_1 s_2}V(s_2) + P^{a_2}_{s_1 s_1}V(s_1) \\ &= -1 + 0.5V(s_2) = -1 \\ Q(s_1, a_3) &= r_t + \gamma(P^{a_3}_{s_1 s_3}V(s_3) + P^{a_3}_{s_1 s_1}V(s_1) \\ &= -1 + 0.5V(s_3) = -1 \end{aligned} \tag{15}$$

Thus, node s_1 updates its V values as $V(s_1) = \max_a Q(s_1, a) = -1$ by choosing either node s_2 or node s_3, since $Q(s_1, a_2) = Q(s_1, a_3)$. Node s_2 then forwards the packets to node s_4 and updates its V value through the same procedure. The path of the first packet and all the V values of each node (after the packet has been delivered) are shown in Fig. 12 (a).

For the second packet, node s_1 calculates the Q values of its neighbors as follows:

$$\begin{aligned} Q(s_1, a_2) &= r_t + \gamma(P^{a_2}_{s_1 s_2}V(s_2) + P^{a_2}_{s_1 s_1}V(s_1) \\ &= -1 + 0.5(-1) = -1.5 \\ Q(s_1, a_3) &= r_t + \gamma(P^{a_3}_{s_1 s_3}V(s_3) + P^{a_3}_{s_1 s_1}V(s_1) \\ &= -1 + 0.5V(0) = -1 \end{aligned} \tag{16}$$

Therefore, node s_1 updates its V values as $V(s_1) = \max_a Q(s_1, a) = -1$, and chooses the node with a larger Q value, which is s_3, to forward the packet. In this way, the previous packet forwarding conducted by node s_2 acts as a 'penalty' in (17), which causes node s_1 to choose node s_3 to forward the current packet. Node s_3 then calculates the Q values of its neighbors as $Q(s_3, a_1) = -1.5$, $Q(s_3, a_2) = -1.5$ and $Q(s_3, a_5) = -1$. Thus node s_3 forwards the packet to node s_5 and updates its V values as $V(s_3) = -1$. This procedure is repeated for each packet. Finally, the V values of each node converge to stable status, as shown in Fig. 12 (c). To balance the tasks among nodes, the residual energy of each node should be considered. Therefore, in (15) and (16), $\alpha_1, \alpha_2, \beta_1, \beta_2 \in (0, 1]$. In this circumstance, the V value of each node may converge as shown in Fig. 12(d), where the number tagged to each node represents the residual energy. Compared to the Vector-Based-Forwarding (VBF) scheme [70], which is a popular routing protocol for Underwater Sensor Networks, the lifetime of QELAR is 20% longer.

QELAR forms the routing topology based on the task balance among nodes, which significantly increases the batteries' lifetime if the link conditions are perfect. However, in real-life network, many factors may deteriorate link quality, such as a large queue in node's data sending buffer, high mobility, weak signal strength, interference, etc. The deteriorated link quality may decrease the end-to-end transmission quality and increase packet retransmissions. As a consequence, the batteries' lifetime is shortened. Therefore, considering other factors together with task balance might be a good routing strategy, especially when

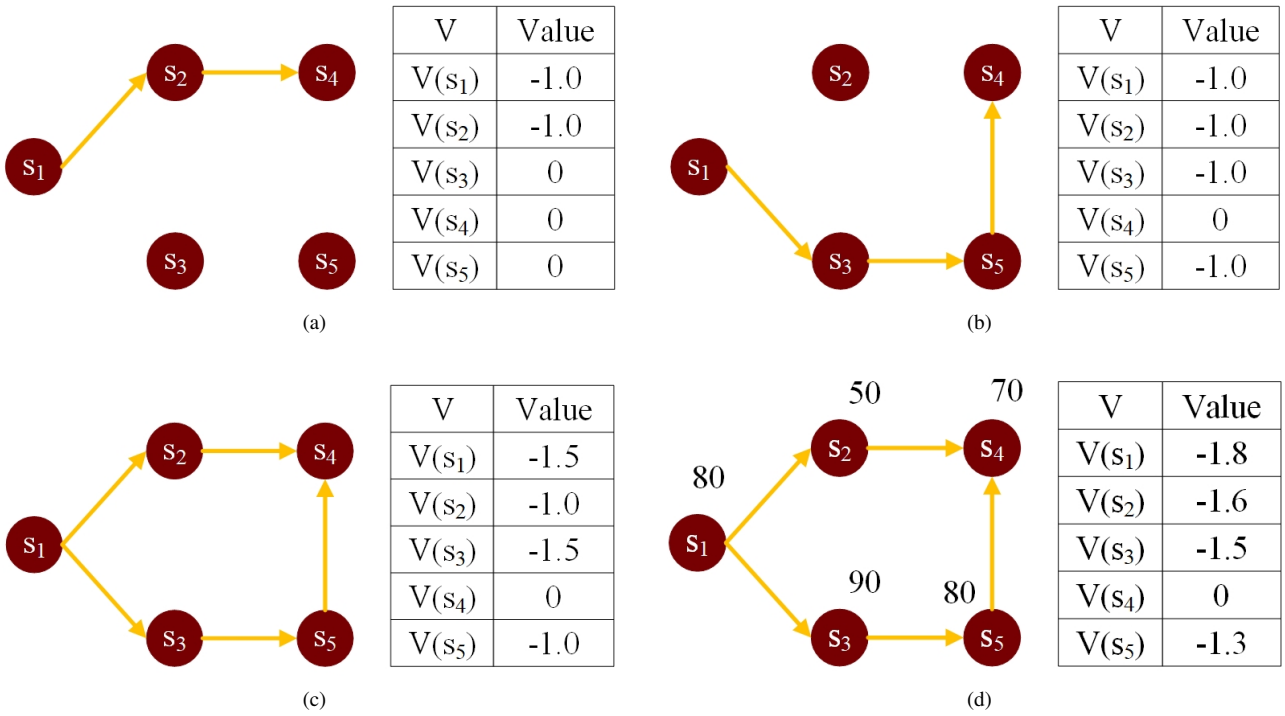


Fig. 12: Reward Value Variation of QELAR Scheme
(a) first packet (without energy consideration) (b) second packet (without energy consideration) (c) converged V values (without energy consideration) (d) converged V values (with energy consideration)

the size of the wireless network is large or the payload is heavy.

B. DL for Routing Path Search

In many networks the conditions of routers vary from time to time, including the saturated caches, overloaded routers, malfunctional hardware, etc. All these factors may cause the deterioration of the routers' performance. In a network with numerous routers, the data forwarding capability varies in each local network region at each time slot. Since a communication session may involve many hops from the source to the destination, the routing algorithms confront magnificent challenges in terms of finding a global optimal path among many candidate nodes in a highly dynamic network environment, and some nodes may provide good local transmission performance while deteriorate the global end-to-end routing performance.

Finding a global optimal path demands heavy computation load. DL can be an efficient approach to relieve the path search burden. Kato et al. [71] proposed an DL approach for the traffic control in heterogeneous networks. First, the traditional routing protocols such as Open Shortest Path First (OSPF) are executed in the network for performance collection purpose. Once enough parameters have been collected, a supervised training process is implemented. In the training phase, each node trains M models, where M is the number of potential receivers in the entire network. The training procedure is initialized by a greedy layer-wise training method and is fine-tuned by a backpropagation algorithm [72]. Once the training is finished, the optimal path can be found in the running phase. To find the optimal path from a source to the destination, the DL model needs to be run for k rounds by the source node, where k is the number of hops from the source to the destination. For each round, the history of the traffic patterns of all the routers in the network serves as the input and only one router is chosen as the output.

Fig. 13 shows an example. Assume there are 10 routers in the entire network, the source node is n_1 , and the destination node is n_{10}. The first step of the running phase is to find the optimal router immediately next to the source. To do so, the DL model $DL_{1,10}$ is run by n_1. The input of $DL_{1,10}$ is vector $A = [\alpha_1, \alpha_2, \cdots, \alpha_{10}]$, where α_i is the traffic pattern of router i. At the output side, one out of 10 routers is chosen, indicating the first hop in the path chosen by the source, which is n_3 in the example. Following that, node n_1 runs model $DL_{3,10}$ to find the second hop node. This operation is repeated until the destination node is reached. Compared to OSPF, this scheme decreases the routing overhead by about 70% and increases the throughput by about 2%. **Apparently, there is no need for a central controller to implement the DL-based routing algorithm, which increases the flexibility. However, the source node has to train many DL models, which demands huge computation power and storage for every node.** Mao et al. [73] integrated this method with programmable routers and achieved good performances.

C. DL for Other Routing Performance Optimizations

Natural disasters and terrorist attacks may damage the communication infrastructures. In these situations, the col-

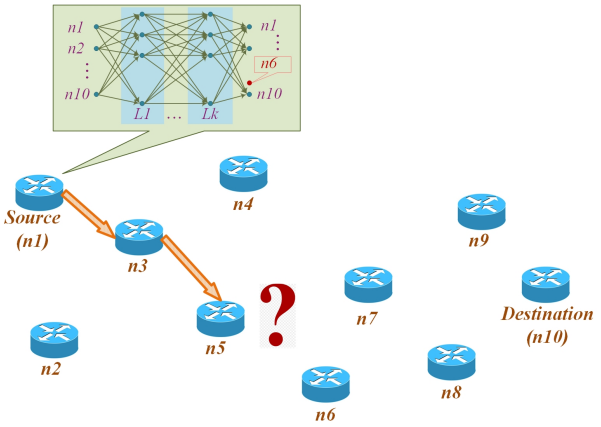


Fig. 13: A Deep Learning based Routing Protocol

laborations between infrastructure devices and wireless nodes (e.g., ad-hoc nodes) are crucial to maintain effective communications. The routing design in such a hybrid wireless network is challenging. Lee [74] proposed a DL-based routing scheme, which treats connectivity as the routing priority. In their scheme, the degree of each node, which indicates the connectivity of a node, is first evaluated using DL algorithm. Following that, a virtual route is generated by Viterbi algorithm [75] with the consideration of node degree. Then, an IP-based routing procedure is implemented to establish the route in the hybrid network. This scheme increased the reachability compared to AODV, OLSR, and ZRP routing protocols. **Note that there is a demand of a Route Information Server (RIS) in this scheme, which determines the node degree using deep learning algorithm and particular hardware.**

Stampa et al. [76] used deep reinforcement learning to optimize the routing performance with the aim of reducing transmission delay. **The DRL network uses traffic matrix as the state, a path from the source to the destination as the action, and the mean of end-to-end delays as the reward. Note that the scheme only considers the traffic matrix, i.e., the bandwidth requests of the traffic flows, as the state and doesn't consider other network factors, such as nodes' queue size, link quality, etc. The routing results may be further optimized if more conditions are considered.** To test the performance, they used the OMNeT++ discrete event simulator [77], [78] to collect transmission delay with given traffic and routing parameters [79]. Their experimental results showed a significant improvement on transmission delay with various traffic intensities, compared to the benchmark routing scheme.

Valadarsky et al. [80] applied machine learning and DRL respectively for network routing. In the DRL approach, the environment of the network is described by the demand matrix (DM), of which element d_{ij} indicates the traffic demand between the source node i and the destination node j. ($i, j = 1, 2, \cdots, N$, and N is the number of nodes in the entire network.) The reward of the DRL is the link utilization rate. In each time slot, the agent chooses a routing scheme based on the routing strategy and DMs. Then the DRL system learns a mapping from DMs to the routing schemes in such a manner that the discounted reward is maximized. Their simulations use the open-source implementation of TRPO [81], [82], and it is shown that learning from the historical routing schemes with the consideration of the demand matrix and link utilization provides an efficient approach for the agent to smartly choose the optimal routing topology for future data forwarding.

Valadarsky's scheme has some similarities with Stampa's scheme, but using different reward object. From their work we see that how to choose reward function is a crucial issue for the DL applications in routing layer. According to the network characteristics and environmental features, designers choose the most important attribute to optimize, which could be throughput, end-to-end delay, link utilization, flow completion time, etc.

D. A Brief Discussion on DL Application in Routing Layer

Centralized routing versus distributed routing is a tricky choice for DL-based routing schemes. This is because that the deep learning process demands tremendous parameters as input to make a decision as well as huge computation to train the neural network.

If a centralized routing strategy is adopted, three main issues should be carefully addressed. First, large amount of network environment data, such as nodes' energy condition, queue size, signal strength, etc., have to be sent to the central controller. In this circumstance, transmission load yielded by the environment data is huge, and too much overhead decreases the good throughput of the network. Second, routing topology needs to be built up within limited time. However, the channel environment data may be delayed when transmitted to the central controller, thereby causing the delay of the routing formation. Third, less flexibility, i.e., a central node running DL algorithm is not always available. For instance, the routing method proposed in [74] adopts a centralized DL strategy, which trains the DL model in the base station and classifies nodes' connectivity level thereby. However, for some ad hoc network, it is difficult to find a central server that has huge computation power as well as an appropriate geographic location.

On the other hand, if a distributed routing strategy is used, each node (or each source node) has to train several DL models. Therefore, huge computation power and storage are needed for every node. For instance, the distributed DL strategy has been adopted by QELAR [69] and Kato's scheme [71], where the source node triggers the DL process and generates routing topology using the trained models.

Therefore, for the DL-based routing design, a sophisticated choice between the centralized and distributed strategy is crucial, which should be made based on plenty of considerations, such as the network structure and size, routing algorithm, deep learning method, etc.

VI. DL FOR OTHER NETWORK FUNCTIONS

A. Vehicle Network Scheduling

Vehicular Ad-Hoc Network (VANET) provides a fully connected network among vehicles and infrastructures, and is the

foundation to establish an intelligent transportation system. There are two types of communications in VANETs, i.e., Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I), where the infrastructures are usually composed of Road-Side Units (RSUs). The prior communication in VANETs is Driving-Safety-Related (DSR) services. However, in V2I communications, there are many non-DSR services, such as web browsing and online games. To guarantee QoS performance, Atallah et al. [83] proposed a DRL-based scheduling scheme among the RSUs, which targets to reduce the energy consumption of the road-side units while providing a safe driving environment. The DRL agent is deployed at the RSUs and interacts with the VANET environment. Assuming that there are M vehicles in a RSU's coverage, at time slot i, the action a_i taken by the agent is either to receive DSR messages from the vehicle, represented as $a_i = 0$, or to send non-DSR messages to a vehicle upon a download service request, represented as $a_i = j$, where $j = 1, 2, \cdots, M$ indicates which vehicle is downloading non-DSR messages. If the RSU chooses to transmit data to a vehicle, the reward is measured by the number of transmitted bits, and the cost is composed of two parts: 1) the power consumed by the RSU, and 2) the waiting time of a DSR message potentially occurred during this non-DSR communication period. On the other hand, if the RSU chooses to receive DSR messages, the induced cost is the power consumption of the message receiving operations. The DRL-based scheme can 1) minimize the delay of DSR messages, 2) maximize the service amount, including both DSR and non-DSR messages, and 3) extend the battery lifetime of RSUs. Their simulations showed that the DQN-based approach achieves better performances compared to Random Vehicle Selection (RVS) algorithm, Least Residual Time (LRT) algorithm, and Greedy Power Conservation (GPC) algorithm in terms of RSU battery lifetime, RSU busy time, and incomplete requests ratio.

B. Sensor Data Reduction

Wireless sensor networks have strict constraints on the size of data it transmits due to its limited network capacity. For instance, the Implanted Medical Devices (IMDs) are usually constrained by power consumption. Unfortunately, the network tends to be overwhelmed by a large amount of sensor data. To reduce the data size, Quantized Compressed Sensing (QCS) technique is extensively used. Assuming that the original data measured by sensors is $X \in \mathbb{R}^N$, then the compressed data is obtained as $Y = \Phi X$, where $Y \in \mathbb{R}^M$ and Φ is the measurement matrix with a size of $M \times N$ ($M < N$). To retrieve the original data from Y, the input X must be sparse and the measurement matrix Φ must satisfy the Restricted Isometry Property (RIP), which raises big challenges for computation. Sun et al. [84] thus proposed a Binary-Weighed, Non-uniform Quantizer, and Deep Neural Network (BW-NQ-DNN) for QCS. Instead of using a randomly or deterministically generated measurement matrix, BW-NQ-DNN learns a measurement matrix from the previous experiences (i.e., training data). Furthermore, the BW-NQ-DNN is used for the optimization of non-uniform quantizer. The BW-NQ-DNN is composed of a compression net, a quantization net, and a recovery net, as shown in Fig. 14.

The compression network is fully connected, i.e., it collects the original N-dimensional data X and represents it with a M-dimensional data Y, with each element of Y being connected with all the elements of X. To reduce the hardware implementation complexity, the weights of the compression network are constrained as $w_{ij} \in \{-1, +1\}$. The quantization network is composed of nonlinear compand and gradient cancelation. Each element of Y, say y_i, is nonlinearly companded as $\phi(y_i) = \sum_{j=-K}^{K} c_i \Psi(y_i/\Delta - j)$, where c_i is the coefficient of the nonlinear compand function, Ψ is a basis function, and Δ is a constant. All the components of Y are companded by the same process. Without loss of generality, only the process of y_1 is shown in Fig. 14. To avoid the discontinuity of the derivative of the quantized value ϕ, a straight-through estimator [85] that considers saturation effect, denoted as ES, is used upon the gradient $\partial C/\partial \phi$. Finally, the compressed and quantized measurements can be recovered through a nonlinear recovery network based on Multi-Layer Perceptron (MLP) [86] architecture. Since the whole system aims at the optimization of the Signal to Noise and Distortion Ratio (SNDR) for the recovered messages, the cost function is chosen as the Mean Squared Error (MSE) between the recovered message, $\hat{X}$, and the original message, X. To update the parameters of the compression, quantization and recovery networks, the Stochastic Gradient Descent (SGD) algorithm [87] is used in the backward propagation of each part. Compared to the popular QCS schemes such as SDNCS [88], BPDQ [89] and QVMP [90], the DL-based BW-NQ-DNN provides higher SNDR and classification accuracy.

C. Hardware Resource Allocation

Operating system supports some application layer tasks for the network communications. And hardware resource allocation significantly impacts the communication performance. Mao et al. [91] proposed to use reinforcement learning in resource management. Assume there are D resources serving M tasks, and task i lasts for t_i seconds and is finished in c_i seconds ($i = 1, 2, \cdots, M$). The slowdown value of task i is defined as $s_i = c_i/t_i$, which can effectively serve as an evaluation index for the resource allocation. The goal of resource management is to minimize the average slowdown value. In DL-based scheme, the reward at each timestep is defined as $\sum_{i \in J} -s_i$, where J is the current task set. The state of the DL system is the current resource allocation plus the resource demands of all the tasks. The action space is $\{0, 1, 2, \cdots, M\}$, where at each timestep, action $a = 0$ indicates the agent does not schedule any task and $a = i$ ($i = 1, 2, \cdots, M$) indicates that the agent schedules task i for a specific resource. Simulation shows that the DL-based resource allocation outperforms the popular methods, such as the Shortest Job First (SJF) scheme, Packer and Tetris in terms of the average task slowdown value.

D. Network Security

Traffic inference and intrusion detection are crucial issues for cyber security. The decision-making process of these prob-

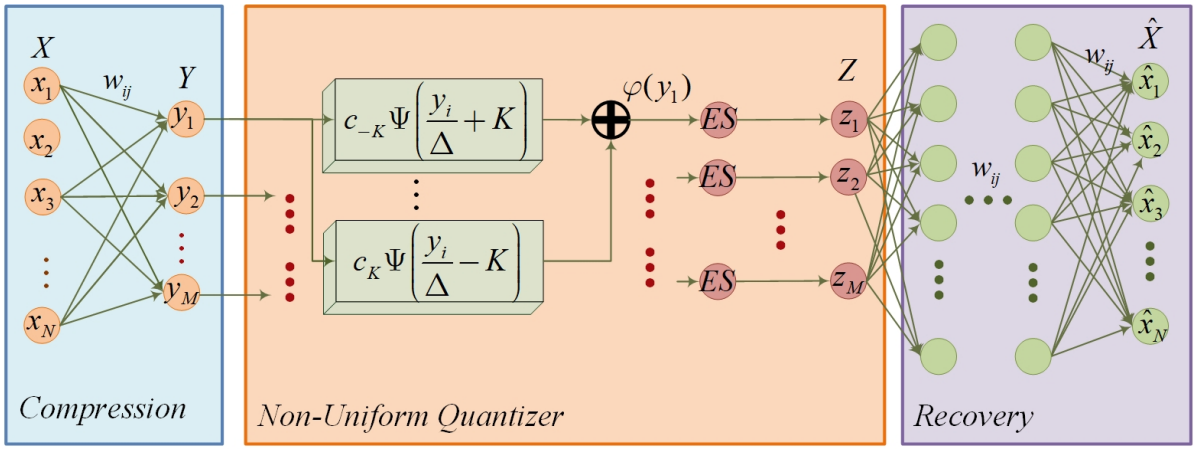


Fig. 14: BW-NQ-DNN Framework

lems requires an analysis of a large number of network features and an abstraction towards the attack-related characteristics. DL schemes have shown promising performances in these tasks.

1) Traffic Identification: Flow inference aims to describe the original flow features generated at the transmitter side according to the received packets at the receiver side. It is crucial for intrusion detection, traffic monitoring, queue management, etc. An easy way to identify traffic is to classify them by port numbers. However, many recent applications, such as P2P traffic and video calls, may use port numbers that are initially assigned to other traffic types. Therefore, more accurate ways are required to identify traffic types. In the past several years, traffic identification methods using statistical models [92], [93] or machine learning [94], [95], [96] have been extensively studied, for which the traffic features such as time interval between packets, packet size, etc., are exploited to analyze traffic types. Due to the complexity of the networks, the patterns of the received flows may have nonlinear alterations, which makes flow inference challenging.

In 2014, Gwon et al. [97] proposed a DL-based flow inference scheme, which classifies the received packet patterns and infers the original properties (e.g., burst size and inter-burst gaps). Note that although the flow inference is achieved by exploiting MAC layer parameters, it analyzes the TCP/UDP/IP flows. The inference system is composed of two independent layers, each of which comprises a feature extractor (FE) and a classifier (CL). For each layer, the sparse coding [98] is used to extract features from time-series data, and the max pooling [99] is used to reduce the number of features for the purpose of computation reduction. The two-layer structure allows both the local features (such as run and gap sizes) and the global features (such as periodicity) to be extracted by the learning system. Simulations show that the deep learning based flow inference scheme has high true positive rate and low false positive rate, compared to ARMAX-least squares [100], Naive Bayes classifiers and Gaussian mixture models.

In 2015, Wang [101] proposed a DL-based traffic identification scheme. In this scheme, the TCP flow is used for traffic identification, since the bytes of different protocol payloads represent different distributions. Therefore, the bytes of TCP sessions are first normalized from integers (ranging from 0-255) to decimals (ranging from 0-1). Then the normalized data is sent to ANN as the input for traffic identification. Simulation shows that this scheme can distinguish 25 most popular protocols, such as SSL, HTTP-Connect, MySQL, SMB, etc., most of which have a precision of higher than 95%.

Lotfollahi et al. [102] proposed a DL-based traffic classification method, namely, deep packet, which not only distinguishes traffic type (such as streaming, P2P, etc.) but also classifies application types (such as Spotify, Bit Torrent, etc.). The 'ISCX VPN-nonVPN traffic dataset' [96] was adopted in their experiments. Two DL methods, i.e., convolutional NN and stacked autoencoder NN, are applied. The simulation platform is built based on Keras library [103] and Tensorflow, and the scheme is shown to achieve 97.0% precision for traffic type classification and 95.4% accuracy for application type classification, both outperforming the general ML-based schemes [96], [104]. A comprehensive comparison upon the traffic classification performances among four ANNs, i.e., backpropagation-based multilayer perceptron (BB-MLP), resilient-backpropagation-based multilayer perceptron (RBB-MLP), recurrent neural network (RNN), and deep learning stacked autoencoder (SAE), has been presented by Oliveira et al. [105].

***Discussion*: To identify traffic accurately, two crucial issues should be carefully considered. First, researchers need to decide on which layer the DL algorithm is implemented. Many traffic identification schemes are implemented upon transport layer or IP layer, such as [101]. However, some DL-based schemes analyze MAC layer or application layer features to identify the traffic type. For instance, Gwon's scheme [97] uses the runs-and-gaps model upon MAC layer as the input of the DL model and identifies the traffic type thereby. Another example is Lotfollahi's scheme [102], which provides both traffic characterization and application identification to meet various requirements. The second issue is to determine what features are used for DL analysis. The choice of data features used for**

DL models significantly impacts the accuracy of traffic identification. For instance, [101] uses the scaled values of the TCP flow data, especially the first 25 values of the payload, as the input of the DL network. Although from intuition and experience, these payload values indicate the traffic type to a certain extent, more intrinsic features, such as the correlation and distribution features of the payload values, may further improve the identification performances of the DL network.

2) Intrusion Detection: Network Intrusion Detection (NID) protects networks from malicious attacks by detecting software intrusions. Traditional NID schemes are mostly based on user signatures. However, this method needs the administration center to maintain a large number of user's signatures. Currently, the anomaly-based detection is extensively studied, which analyzes network activities and marks out abnormal data access as an intrusion. Since the utter goal of NID is to classify network traffics into many categories (i.e., normal traffic and various abnormal traffic types) according to numerous traffic features, it is an ideal choice to use DL approaches to detect network intrusions by learning traffic features [106].

Niyaz et al. [107] proposed a flow-based, Self-taught Learning (STL) [108] approach to detect network intrusion. In their scheme, the NSL-KDD dataset [109], [110], a benchmark for network intrusion, is used for training and testing. The network traffic provided by NSL-KDD dataset includes normal flows and various anomalous flows, including Denial-of-Service (DoS) attack flow, Remote-to-Local (R2L) attack flow, User-to- Root (U2R) attack flow, Probe attack flow, etc. For each traffic, forty-one features are provided, including the average packet number per flow, average time duration per flow, protocol types (e.g., TCP, UDP), etc. The scheme in [107] chooses 22 out of 41 features for the DL process, which consists of two stages: 1) an Unsupervised Feature Learning (UFL) process, which is based on sparse auto-encoder, and 2) a supervised learning process with the goal of classification. Auto-encoder is a feedforward non-recurrent neural network with an input layer, an output layer and one or several hidden layers, as shown in Fig. 15 (a). Specifically, the node number of the input layer and the output layer is the same, which is larger than the node number of the hidden layer(s). The goal of the output layer is to reconstruct the input. Therefore, the cost function is composed of an average of sum-of-square errors upon all the inputs, a weight decay term to avoid over-fitting, and a sparsity penalty term to maintain a low activation values. Using the trained DL network, the testing traffic is classified as two types, i.e., normal traffic and anomalous traffic. Simulations showed that the STL scheme achieves 88.39% accuracy for 2-class detection (normal and anomaly), and 79.1% accuracy for 5-class detection (normal and four different attack categories), which are higher than the accuracies achieved by the Soft-Max Regression (SMR) scheme.

To reduce the number of features the learning system uses, Tang et al. [111] proposed a DL-based NID scheme for software defined networking (SDN). The data used to train the learning network is also chosen from NSL-KDD dataset. However, only six features of each traffic flow are considered in order to reduce the computation cost, i.e., flow's duration, protocol type (e.g., TCP, UDP), number of data bytes from source to destination, number of data bytes from destination to source, number of connections to the same host, and number of connections to the same service. A deep neural network with three hidden layers is adopted. Although there are four malicious traffic types in the dataset, this scheme only classifies the traffic as two types (normal and anomalous), and it achieved an accuracy of 74.67%. Compared the schemes in [107] and [111], it can be seen that the detection accuracy depends on the traffic features the system used for training and classification to a large extent, and one of the most challenging topics is to select appropriate features to balance the detection accuracy and computation cost.

Another type of DL network used for intrusion detection is Deep Boltzmann Machine (DBM) [112], in which each node is bidirectionally connected with the nodes of other layers, as shown in Fig. 15 (b). To decrease the computation cost of the gradient, the intra-layer links (the red dotted lines in Fig. 15 (b)) are abandoned to use, yielding a Restricted Boltzmann Machine (RBM) [113]. As a matter of fact, in many real applications, the network detectors may not know what features the anomalous traffic possesses. Thus, Fiore et al. [114] proposed a Discriminative RBM (DRBM) based intrusion detection method, which is a semi-supervised learning system, i.e., the system is trained only by normal traffic data. The trained network is tested by real-world traffic collected from a workstation for 24 hours and KDD CUP 1999 dataset, respectively, both of which include normal and anomalous traffic. Simulation results show that, when the learning system is trained and tested with the real-world data, a high accuracy (about 94%) is obtained. However, when the DBRM is trained with KDD dataset and tested with real-world data, the accuracy is as low as 84% around.

If we limit the node connections only between the layers, the DBM is transformed to Deep Believe Network (DBN). Alternatively, DBN can be formed by cascading a stack of Restricted Boltzmann Machine in serial. Furthermore, one or more additional layers can be added to perform classification after a supervised learning process. Therefore, DBN is often pre-trained by unlabeled data (unsupervised learning) and then fine-tuned by labelled data (supervised learning), as shown in Fig. 15 (c). Gao et al. [115] used a DBN to detect network intrusion, for which the learning network is trained with KDD CUP 1999 dataset via three stages. The first stage pre-processes data, for which the features of the traffic are digitized and normalized. The second stage pre-trains the DBN, i.e., the weights of a stack of RBMs were learned through an unsupervised greedy contrastive divergence algorithm. Finally, the weights of the entire DBN are fine-tuned through the back-propagation of error derivatives by the labeled data. In their simulations, 41 features of each KDD CUP 1999 traffic flow are first mapped to 122 attributes, then several DBNs with different structures are established. Each DBN in their simulations has 122 input elements and 5 output elements (1 normal traffic and 4 different attack traffics). However, the hidden layer structures are different, varying from as shallow as 122-5 (no hidden layer) to as deep as 122-150-90-50-5

(three hidden layers with 150, 90, 50 nodes respectively). Apparently, the deeper the DBN becomes, the better detection accuracy can be achieved. For the 122-150-90-50-5 DBN, the accuracy reaches 93.49%, outperforming SVM (86.82%) and NN (82.3%). Besides, Alom et al. [116] applied the similar method to NSL-KDD dataset.

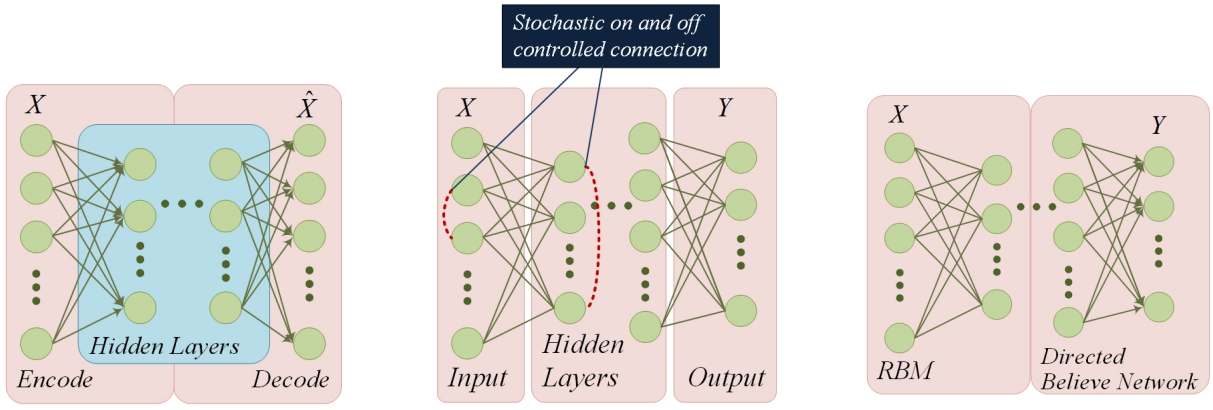
Using similar DBN structure as shown in Fig. 15 (c), Kang et al. [117] proposed an intrusion detection scheme for In-Vehicle Controller Area Network (CAN). Each CAN packet includes 12 bits of arbitration field, 6 bits of control field, 0-8 bytes of data field, etc. Kang's scheme utilizes the data field as the learning object. The data field is composed of mode information (such as controlling wheels) and value information (such as the wheel angle) of the Electronics Control Unit (ECU), and they yield different bit distributions. Since there are different attack scenarios, the learning system first uses the mode information to identify the attack scenarios, then trains the learning network for each scenario. The DBN has less than 64 input nodes (each node corresponds to a bit of the data field but with reduced number of bits considering the semantics redundancy), several hidden layers, and 2 output nodes (indicating normal and anomalous scenarios). In the testing phase, the attack scenario of each CAN packet is first determined by matching the mode information, then the corresponding trained model is used to determine whether the packet is normal or anomalous. Experiments show that the scheme achieves 97.8% accuracy, outperforming Support Vector Machine (SVM) and Artificial Neural Network (ANN).

***Discussion*: A comparison towards the DL-based intrusion detection is shown in Table V. Note that for the output classification number, the number of 2 indicates normal traffic and anomaly traffic, and the number of n ($n > 2$) indicates a normal traffic and $n-1$ different anomaly traffic types. From the comparison we see that there are several challenges in the topic. 1) How many intrusion types the DL network detects. Apparently, the more types the network detects, the lower accuracy will be given. For instance, Niyaz's scheme [107] achieves 88.39% accuracy with two-type detection (normal and anomaly) and 79.1% accuracy with five-type detection (normal and four anomaly types). Thus, designers need to carefully balance the detection accuracy and the number of detection types. 2) How to select the network features as the input of the DL network. Many current DL-based intrusion detection schemes use KDD dataset, which provides 41 features for each traffic flow. Apparently, using all the 41 features for DL analysis yields a huge burden from the computation's point of view. Therefore, many NID schemes select a part of features to detect intrusion. For instance, Niyaz's scheme [107] uses 22 features while Tang's scheme [111] uses only 6 features. As a consequence, Niyaz's scheme achieves 88.39% accuracy with two-type detection, while Tang's scheme yields 74.67% accuracy. 3) What dataset is used for DL network training. As we see from Table V, most current schemes use NSL-KDD dataset, which is an improved version of KDD Cup 99 dataset (proposed in 1999). NSL-KDD dataset reduces some redundant records of the KDD Cup 99 dataset, which makes the sizes of training set and testing set reasonable. The NSL-KDD dataset includes normal traffic and four attacking categories, i.e., DoS, U2R, R2L, and probing. However, with the accelerated development of network attack techniques, new intrusion methods appear with astonishing speed. The DL models trained by KDD dataset may yield deteriorated performance in detecting real-world data, as shown in [114].**

VII. DL-Based Wireless Platform Implementation

There are abundant DL implementation methods, some of which have been performed in wireless networks. In the following, a summary of DL implementations is presented. Those methods have been used in wireless testbeds.

- 1) *Matlab Neural Network Toolbox*. This toolbox includes the most popular DL algorithms, such as ANN, CNN, DBN, SAE, and convolutional autoencoders (CAE). The input layer takes the original raw data. The hidden layers perform convolution, pooling, or ReLU functions upon the raw data. The convolution operation is composed of a set of convolutional filters, which extract certain features from the input data. The pooling operations perform the following operations: nonlinear sampling upon the output of the convolutional filters, reducing the dimensions of the parameters, and controlling the complexity of the deep learning network. The ReLUs map negative values to zero and keep positive values, thereby improving the training efficiency. By repeating these three functions in the hidden layer and training the parameters of the functions, specific features are extracted efficiently for the classification purpose. Then, the output layer performs classification upon the features. A softmax function is typically adopted for classification.
- 2) *TensorFlow* [118]. It is an open-source software originally developed by Google Brain team. TensorFlow is written in Python, C++, and CUDA, and it is supported by Linux, macOS, Windows, and Android systems. It is a symbolic math library composed of nodes and edges. The nodes in the graph represent mathematical operations, and the edges represent the connections (tensors) between nodes. TensorFlow is a flexible, flow-based programming model. Although it is originally developed for conducting ML and deep neural network algorithms, TensorFlow is capable of conducting many other flow-based implementations.
- 3) *Caffe (Convolutional Architecture for Fast Feature Embedding)* [119]. It is an open-source software tool, and was developed by Berkeley AI Research (BAIR) and community contributors. Caffe is a DL framework targeting image classification and segmentation. It has the features of expressive architecture, extensible code, high speed, and modularity. Caffe supports CNN, RCNN, LSTM and has fully connected neural network structures. There are a variety of functions to be chosen to build a DL network using Caffe, including convolution, pooling, inner products, ReLU, logistic unit, local response normalization, element-wise operations, softmax, and hinge.



(a) (b) (c)

Fig. 15: Deep Learning Networks Used by Intrusion Detection
(a) Deep Auto-encoder (b) Deep Boltzmann Machine (c) Deep Believe Network

TABLE V: Comparison of Deep Learning Applications in Intrusion Detection

Scheme	Learning Model	Dataset	Input No.	Output No.	Accuracy
Niyaz's scheme [107]	Self-taught Learning composed of an auto-encoder and a soft-max regression process	NSL-KDD	22	2 or 5	88.39% for 2-class and 79.1% for 5-class
Tang's scheme [111]	Deep neural network	NSL-KDD	6	2	75.75%
Fiore's Scheme [114]	Discriminative Restricted Boltzmann Machine	KDD CUP 1999 and real world traffic	28	2	94%
Gao's scheme [115]	Deep Believe Network	KDD CUP 1999	40	5	93.49%
Kang's scheme [117]	Deep Believe Network	In-Vehicle CAN traffic	<64	2	97.8%

- 4) *Theano (Keras)* [120]. It an open-source Python library that allows users to symbolically define, optimize, and evaluate mathematical expressions such as multi-dimensional arrays. Users can use Theano to implement and train NN models on fast concurrent graphics processing unit (GPU) architectures. The network is built by apply nodes and variable nodes, which represent mathematical operations and tensors, respectively.
- 5) *Keras* [103]. It is an open-source neural network library that can run upon TensorFlow, CNTK, Theano, etc. Keras is originally developed by a Google engineer, Francois Chollet. It provides neural network elements such as layers, objectives, optimizers, activation functions, etc., and it supports convolutional networks, recurrent networks, and the combination of the two types. In 2017, Google's TensorFlow team decided to support Keras in TensorFlow's core library.
- 6) *WILL* [121]. It is an open-source neural network toolkit created by Prevision Limited Company (Hong Kong). It supports convolution, pooling layers, full-connection, and some popular functions such as reLU, sigmoid, tanh and softmax.
- 7) *Customized models*. Many DL-based networks use special functions or neural network structures, and some systems run upon embedded systems. For those applications, developers implemented the algorithms by customized DL systems. The language used to develop these systems varies from C, C++, Matlab, Python, to Java, depending on the features of the learning system, the library used by the learning process, and the communication patterns with the other simulators (such as wireless network simulators).

In addition, there are lots of other popular deep learning software, such as MXNet [122], developed by Distributed Machine Learning Community, Torch [123], Microsoft Cognitive Toolkit [124], developed by Microsoft Research, etc. However, few of them can be found in network applications. Table VI presents a comparison of deep learning software platforms used in wireless networks.

VIII. Future Research Trends

In order to help current researchers to identify unsolved issues in this important field, in this section we will explain 10 challenging issues and point out the future research trends. Although those 10 issues do not represent all the unsolved research topics on DL-based wireless networking, they have long-term dominant impacts on today's popular wireless infrastructures, including cognitive radio networks, software-defined networks, dew/cloud computing, big data networks, etc.

A. (Challenge 1) DL for Transport Layer Optimizations

Congestion control is the main function of transport layer. However, the existing congestion control methods are mostly

TABLE VI: Comparison of Deep Learning Applications in Intrusion Detection

Simulation Platform	Written in	Platform	Developer(s)	Used by
Matlab Neural Network Toolbox	C, C++, Java, Matlab	Linux, macOS, Windows	MathWorks	[105]
Tensorflow	Python, C++, CUDA	Linux, macOS, Windows, Android	Google Brain Team	[45] [58] [59] [102]
Caffe	C++ with Python and Matlab bindings	Linux, macOS, Windows	Berkeley Vision and Learning Center	[42]
Theano	Python	Cross-platform	The University of Montreal	[47]
Keras	Python	Linux, macOS, Windows	Francois Chollet (a Google engineer)	[52] [102]
WILL	C++	Windows	Prevision Limited Company	[71] [73]
Customized	C, C++, Matlab, Python, or Java	Linux, Window, or embedded systems	/	[53] [60] [63] [69] [80] [83] [84] [91] [97]

based on the end-to-end ACK or NACK feedback to indirectly deduce the congestion occurrences. For example, TCP uses ACK feedback to infer the congestion event. The most accurate way is to directly analyze the queues in each node of the routing path to pinpoint exactly which node's queue has overflow event, which indicates the congestion in that node.

Apparently, a single node's queue cannot reflect the congestion distribution in the entire path. It is important to perform multi-queue co-modeling between different nodes to detect the 'congestion propagation'. For example, one node may have very light congestion (i.e., overflow occurs sparsely) in an earlier stage; however, multiple sparse congestions could be accumulated into a serious congestion later on another node. Multi-queue co-modeling can help in finding such an accumulation pattern.

Particularly, DL can be used to perform large-scale network queueing analysis. Assume that each node reports their queue status (such as size, input traffic rate, outgoing traffic rate, etc.) to a central node via an out-of-band control channel, the central node can then run DL to analyze the queues' data accumulation status. For instance, it can find out whether the traffic gets accumulated in a particular node's queue and may cause overflow with a high probability. DL can also help to find an optimal solution to relieve the congestion situation. For example, it can find a node with relatively small queue size during most of the time, and that node may accept a higher incoming traffic rate; or, it may find another set of nodes near the RED zone (i.e., a network area with congested queues), to establish a back-up path to divert the congested traffic from the main path.

Many interesting research issues can be investigated in the above scenario. For example, how do we come up with a time-evolving DL algorithm to detect the multi-queue evolution pattern? How do we define the congestion threshold, i.e., what queue size is a good indication of RED zone? How do we integrate the congestion control scheme with the back-up routing path establishment protocol? If the congestion occurs in multiple nodes which are not neighbors, how do we control the traffic rates in the congested/non-congested nodes to achieve a smooth flow in the entire path? etc.

B. (Challenge 2) Using DL to Facilitate Big Data Transmissions

Today there are many big data applications such as large-scale smart city monitoring, national healthcare management, air pollution monitoring, etc. Wireless transmission of big data is necessary in remote sensing and harsh environments without the deployment of wires. For example, in a large city, numerous mobile phones can send their data (such as user trajectory, user behaviors, patient healthcare data, etc.) to nearby wireless base stations (which can be Wi-Fi access points, cellular network towers, 5G routers, etc.), and eventually reach the cloud servers.

The transmission of big data is a challenging task due to the following 3 reasons: First, there are no standards/protocols to specify the wireless network operations that can efficiently deliver >100T bits of data per second. Second, existing routing protocols cannot provide a 'thick' data pipe to concurrently deliver >1T packets each second. Third, the network status is extremely difficult to monitor in real-time, due to the huge traffic density in very short time (recall that big data has 4Vs features, i.e., Volume, Velocity, Veracity, and Variety).

DL is a promising method to analyze big data transmission dynamics in terms of routing delay analysis of big flows, traffic balancing among nodes, and link access control. For example, we can use DL to analyze the spatio-temporal patterns of huge traffic in each hop, and find out the hot spots of the network with the largest amount of big data traffic. By comparing hop-to-hop big traffic delivery delays, we get to know the average link quality/stability in each hop, and can then determine the traffic allocation in different links to avoid possible traffic bottleneck.

Again, many research issues need to be investigated. For example, how does the DL help to build/maintain a thick routing pipe that can deliver >1T packets per second? How do we apply DL to predict the link failure in some hops? What are the appropriate MAC parameters (such as backoff window size, time slot length, RTS/CTS timing, etc.) to adapt to the QoS requirements for the huge velocity/volume of big traffic among a group of neighboring nodes? and so on.

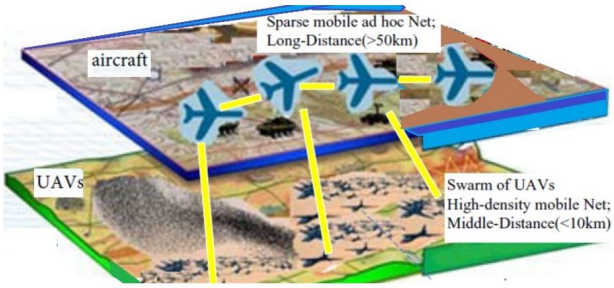


Fig. 16: BW-NQ-DNN Framework

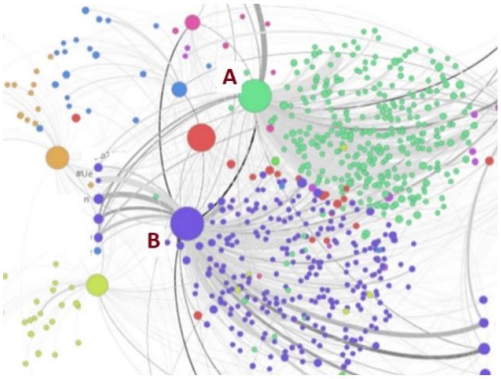


Fig. 17: BW-NQ-DNN Framework

C. (Challenge 3) DL-based Network Swarming

There are many interesting wireless network swarming applications. A typical example is UAV swarming, i.e., a large number of UAVs establish various network topologies to achieve different missions (such as environment monitoring, enemy hunting, forest fire control, etc.), as shown in Fig.16. Other application scenarios include swarming by undersea vehicles to explore sea resources; collaborated war-fighting robots, etc.

A challenging issue in swarming control is the node placement based on both mission and communication requirements. In the beginning of the network deployment, all nodes were randomly distributed. Those nodes may have different density distributions (some places have more densely distributed nodes while other places may be sparse). Then the issue is: how do we guide each node's mobility trajectory to achieve two purposes: (1) forming the desired swarming shape; (2) maintaining good communication architecture? Note that some nodes may have stronger communication capability than others, for example, they may have more powerful directional antennas, higher speed of radio links, faster CPU speed, etc. Those nodes may be placed into the "bridging" positions in the formation (such as the locations of A and B in Fig.17). Those bridging places play critical roles in swarming shape maintenance as well as communication connectivity enhancement. Once they are broken, different swarming regions become isolated.

The entire swarming network may have different cluster distributions, i.e., some clusters have more nodes (higher density) the others. This makes node placement becoming a difficult task. What types of nodes should be moved to different clusters or between those clusters? DL could be used here to describe the topology profile, such as node communication/computation capacity, distance to the desired swarming position (for each node), mobility speed, etc. Then the bridging points can be weighted (i.e., determining their importance levels) based on their locations between different types of clusters. The nodes can be dispatched to those places based on the topology profile.

D. (Challenge 4) Pairing DL with Software-Defined Network (SDN)

The software-defined network (SDN) becomes a promising networking scheme due to its greatly simplified routing/switching management via centralized control. It adopts separate control panel (CP) and data panel (DP) management. The CP consists of one or multiple controllers that send the traffic forwarding control rules to the DP. The flow table(s) in the DP accept those rules to perform data forwarding functions. In the SDN the conventional vendor-specific routers/switches have been replaced by the universal, simple data forwarding devices (called switches) in the DP. The DP does not run protocol-specific data link/physical layer protocols. Instead, they just simply interpret the flow table rules and use the rules to forward the packets to the next switch in the DP.

From the above SDN features we can see that DL and SDN could be "a match made in the heaven" due to two reasons: (1) The CP has the global network monitoring function. This is because that each switch/router in the DP can feedback its data forwarding performance (such as queueing delay, link rate, packet loss rate, etc.) to the CP in real-time. Thus the CP is able to build the global network profile. (2) The CP has the centralized control. DL can be executed in the CP controller(s) to analyze the network profile and determine the rule changes of the flow table in DP.

Some research issues exist in the above scenario: First, we need to solve the network profile formation issue. What types of parameters need to be collected from the DP? What types of big data architecture is suitable to network profile description, e.g., big tensor, big graph, or big time series? What purposes should the DL be used for, e.g., routing decision, queue control, or schedule control? How do multiple CP controllers coordinate with each other to achieve a consistent view of the entire network? and many other issues.

E. (Challenge 5) Distributed DL Implementation in Wireless Nodes

Although DL is a powerful scheme to extract the network dynamics/patterns, it may bring much burden if running in a single network node. While by decreasing the DL algorithm complexity, we can relieve the overhead of the running node. Another efficient method is to distribute the DL computation load to multiple nodes, i.e., using distributed DL implementation.

Some challenging issues need to be solved when using distributed DL model. First, which parts of the DL algorithm can be decomposed into distributed tasks? The gradient neural

network layer weight updating/error propagation can be outsourced to a group of neighboring nodes based on the proper allocation of 'neural cells' to different nodes. Second, how do different wireless nodes exchange the input parameters and output data (i.e., calculation results) with each other via MAC protocols? Note that such a MAC should minimize the channel access collisions by using either well scheduled RTS/CTS exchanges or TDMA-based transmissions. Third, which node is responsible for the final DL output layer assembly? How does this node ensure that the distributed algorithm converges into a stable result?

F. (Challenge 6) DL-Based Cross-Layer Design

While the above sections have covered different individual layers in terms of DL applications, cross-layer design may be a more efficient approach to fully explore all the layers' information for long-term network performance optimization. As a matter of fact, each layer could provide many valuable performance metrics (some listed in Table VII) which can be used to achieve cross-layer global network performance optimization. For example, the channel quality, antenna beam orientation, node mobility, etc., can be used to determine the traffic allocation in each beam of the antenna. The routing layer hop-by-hop delay and packet loss rate can be used to determine the transport layer congestion window size. The MAC layer one-hop link access success/failure information can be used to determine the routing path selection, and so on.

The DL is a perfect tool to fuse the above various cross-layer metrics and extract the intrinsic network patterns for protocol optimization. By using big tensor concept, we can carefully arrange the above metrics into multi-type tensor records, and then apply tensor decomposition to extract the essential patterns. Those patterns can tell us whether the network will have significant packet loss in the near future, and classify the network topology into hotspots and light traffic areas. The patterns can also indicate the link interference distribution across the whole network, and help us to avoid the high interference areas.

Based on the DL pattern extraction results, different layers should co-operate with each other to perform cross-layer optimization. For example, if the DL indicates that a group of nodes form a high-packet-loss 'dark hole', the MAC layer should use much stronger FEC to overcome the bit errors in that area; the routing layer can re-establish a new path to detour around such a hole; and the transport layer can use much smaller congestion control window size.

G. (Challenge 7) DL-Based Application Layer Enhancement

So far, we have not discussed much on the DL-based application layer (AL) enhancement since this survey focuses on the core network protocol design issues. However, the AL has significant impacts on other layers from mission requirements viewpoint. For example, if our mission is to deliver a HDTV flow to multicast users, the AL should specify all the QoS and QoE (quality of experience) requirements for the video stream. Then the lower layers can take those QoS/QoE parameters as the performance goal and adjust their corresponding protocol operations.

DL can be used to learn the network status based on the collected network performance parameters (see Table VII). Then DL outputs the suggested performance goal change in AL. For example, if the network can only provide >100ms of end-to-end delay, it will suggest the AL to use different video coding methods to meet the network limits.

Additionally, DL can be directly used to improve AL performance. For example, it can be used to analyze the webpage display performance (refreshing rate, display speed, image resolution, etc.). It can also be used to perform cyber security analysis to detect spam emails and malicious web sites.

The challenging issue here is to define a low-complexity DL model based on the AL performance goal or application profile data, and solve the DL problem to generate a series of useful results that can be interpreted by the lower layers for protocol operation control purpose. For example, if the AL has a video streaming application, how do we define the AL model to translate the QoS/QoE performance goals into the concrete congestion control and routing parameters? How does the AL classify different applications into various cross-layer protocol design options? etc.

H. (Challenge 8) DL-based Dew-Fog-Cloud Computing Security

We have summarized the application of DL algorithms for network security such as intrusion detection. This is a critical field and will continue to attract many research interests since numerous new attacks keep coming up. Here we would like to point out that DL will play an important role in the security of a promising network infrastructure, called dew/fog/cloud computing (DFC-C). This new network architecture has the following two important features: (1) Collecting data via dew computing units: The large amount of dew computing devices (such as sensors, RFID chips, etc.) can be deployed everywhere, and get connected via wireless networks (such as Zigbee-based systems). The fog computing infrastructure consists of a series of long-distance wireless relays such as Wi-Max nodes or cell phone towers, to deliver the aggregated dew computing data to any cloud server.

From security viewpoint, the above dew-fog-cloud architecture exposes many attack opportunities to the adversaries. For example, one can 'pollute' partial dew computing data by falsifying the sensor data, or mislead the routing path selection in the fog computing segments by claiming a better path, and so on.

To handle the large-scale dew computing sources and concurrent fog computing routing topology, DL is a natural choice to parse all the wireless nodes/links parameters and deduce the possible attack positions and types. For example, we can use all the dew nodes' sensor data as the samples, and run DL-based data clustering test to find out a potential data sample poisoning attack. The challenging issue here is to clearly define a DL model with the proper input/output layer interpretations based on a particular network security

TABLE VII: Network Parameters Considered in Cross-Layer Design

Category	Parameters
Channels	Channel ID, channel bandwidth, channel holding time, handoff channel sequence, channel quality (fading level, Doppler effect, SNR, etc.), channel switching time, etc.
Spectrum sharing modes	Licensed band sharing, unlicensed band sharing, exclusive or cooperative sharing; interference temperature, sharing bandwidth, sharing time, leased bandwidth, etc.
Application & Traffic	QoS parameters (delay, jitter, throughput, etc.), QoE (MOS, PSNR, etc.).
Network architecture	Network topology (cluster-based? Star? Mesh network?), network scale, node density, network connectivity, SDN-assisted? Centralized or distributed? Cloud-supported? etc.
Transport layer	Congestion level, queue size, sliding window size, end-to-end reliability level, congestion bottleneck location, TCP connection duration, retransmission timeout setup, etc.
Routing layer	Multi-path/single-path, multi-cast /uni-cast, # of hops, path throughput, path delay, average link quality, packet drop rate (PDR), re-routing time, path stability, etc.
MAC layer	Access collision rate, backoff time, link BER, robustness to hidden terminal problem, superframe length, access type (TDMA-based on random access), error correction rate, etc.
PHY layer	Wave features, SNR, modulation modes, coding methods, Shannon capacity, etc.
Directional antennas	# of beams, beam steering/switch time, beam angle, beam gap, steered or fixed, single-direction or multi-beam, MIMO control matrix, max radiation distance, etc.
Node property	Mobility speed, mobility modes (random walk or regularized), transmission power, reception sensitivity, max queue length, packet processing time, max hop distance, etc.

problem. Different security/privacy problems mean that the DL should have different input/output/gradient parameter updating structures. For example, the privacy preservation emphasizes the protection of the sensitive data attributes (such as patients' names), and various ID-hiding models can be used to define the DL gradient weight updating process.

I. (Challenge 9) From DL to DRL: Applications for Cognitive Radio Network Control

DL focuses on 'passive' data learning to recognize the intrinsic patterns hidden in the data. However, it does not have concrete 'reactions' for each of the extracted data patterns. Deep reinforcement learning uses Markov decision models to guide the choices of different 'actions' based on the state transition models. Therefore, in many practical applications DRL plays more important roles than DL algorithms.

Here we emphasize the benefits of DRL for cognitive radio network (CRN) control. The CRN is an important type of wireless network due to its flexible spectrum access, i.e., the nodes can grab any available (free) channel to send out data, and can timely vacate the channel if the primary user (PU) is coming back to use the channel again.

DRL can be used to control the following CRN operations: (1) Spectrum sensing: the DRL model can be used to determine the channel scanning order. Some channels with the higher chance of being idle should be scanned first. Note that spectrum scanning is a time-consuming process if thousands of channels need to be scanned and analyzed. By first checking the free channels, we can save spectrum sensing time. (2) Spectrum handoff: When the PU comes back, the node should switch to other channels. The channel switching timing and which channel to switch to, are two critical issues to be solved. Should we wait for the PU's coming event to decide the channel switching operation, or can we predict the timing of PU's transmissions and search for backup channels beforehand? Obviously, the latter has a better communication quality and can avoid some packet loss events.

Proper DRL models need to be clearly defined based on various CRN operation requirements. For example, the DRL may need to be integrated with queuing models to determine the spectrum handoff delay, i.e., how long a user can occupy the existing channel based on the PU traffic analysis, and when the user should start to look for a new channel, and so on.

J. (Challenge 10) Efficient DL/DRL Implementations in Practical Wireless Platforms

The above DL/DRL algorithms eventually need to be implemented in practical wireless network products. However, the pure theoretical understandings cannot be simply programmed in the wireless devices due to the following challenges:

(1) *Difficulty to collect network parameters for DL input layers*: All DL algorithms require the training and testing phases. In each phase, the input layer of the deep neural network consists of the data samples' parameters. The more complete the samples are (in terms of data attributes), the more accurately the DL can recognize the network features. Many network parameters come from MAC and routing layers, which involve many relay nodes' responses. However, those nodes may not have fast feedback about their communication status due to the unpredictable link delays and radio interference. Therefore, the DL models should be designed to tolerate certain parameter miss or data errors in the input layers.

(2) *The resource limits of the wireless devices*: Many wireless products have limited memory and CPU capabilities. They do not allow complex algorithms to be programmed into their existing protocols. Since DL has iterative execution nature, it may elongate the system response time. The DL algorithms should minimize the intermediate computation parameters to save the memory space. The algorithms should be optimized to reduce the execution time.

(3) *Incomplete training sample collections*: DL requires complete or nearly complete training samples to accurately recognize the network patterns. However, the training samples may be very limited due to the difficulty to collect so many data points for each possible network status. This requires that DL should have the capability of adding new samples after the failure of recognizing a new pattern. The new added samples can improve the accuracy of the DL models.

In addition, the network engineers/programmers should carefully define the DL data formats since different network parameters have very different data attributes and formatting requirements. Some proper numerical representations and data normalization methods should be defined clearly to aggregate multiple network parameters into the same DL input layer.

IX. Conclusions

This paper has comprehensively reviewed the methodologies of applying DL schemes for wireless network performance enhancement. In a nut shell, (1) DL/DRL is very useful for intelligent wireless network management due to its human-brain-like pattern recognition capability. With the hardware performance improvement of today's wireless products, its adoption becomes easier. (2) It plays critical roles in multiple protocol layers. We have summarized its applications in physical, MAC and routing layers. It makes the network more intelligently realize the change of the entire topology and link conditions, and helps to generate more appropriate protocol parameter controls. (3) It can be integrated with today's various wireless networking schemes, including CRNs, SDNs, etc., to achieve either centralized or distributed resource allocation and traffic balancing functions. This article also lists ten important research issues that need to be solved in the near future in this field. They cover some promising wireless applications such as network swarming, CRN spectrum handoff, SDN flow table update, dew/fog computing security, etc. This paper will help readers to understand the state-of-the-art of DL-enhanced wireless networking protocols and find some interesting and challenging research topics to pursue in this critical field.

References

[1] J. Patterson and A. Gibson, "*Deep Learning: A Practitioner's Approach*," O'Reilly Media, Inc., 2017.

[2] G. Hinton, L. Deng, D. Yu, et al., "Deep neural networks for acoustic modeling in speech recognition", *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82-97, Nov. 2012.

[3] T. N. Sainath, A.-R. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for LVCSR", In *Proc. 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2013)*, Vancouver, BC, Canada, May 2013, pp. 8614-8618.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," In *Proc. the 25th International Conference on Neural Information Processing Systems (NIPS 2012)*, vol. 1, Lake Tahoe, Nevada, USA, Dec. 2012, pp. 1097-1105.

[5] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1915-1929, Oct. 2012.

[6] J. Tompson, A. Jain, Y. LeCun, and C. Bregler, "Joint training of a convolutional network and a graphical model for human pose estimation," In *Proc. the 27th International Conference on Neural Information Processing Systems (NIPS 2014)*, vol. 1, Montreal, Canada, Dec. 2014, pp. 1799-1807.

[7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436-444, May 2015.

[8] R. Collobert, J. Weston, L. Bottou, et al., "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12, pp. 2493-2537, Aug. 2011.

[9] A. Bordes, J. Weston, and S. Chopra, "Question answering with subgraph embeddings," *eprint arXiv:1406.3676*, Sept. 2014.

[10] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," In *Proc. the 27th International Conference on Neural Information Processing Systems (NIPS 2014)*, Montreal, Canada, Dec. 2014, pp. 3104-3112.

[11] S. Jean, K. Cho, R. Memisevic, and Y. Bengio, "On using very large target vocabulary for neural machine translation," In *Proc. the 53rd Annual Meeting of the Association for Computational Linguistics (ALC 2015)*, Beijing, China, Jul. 2015, pp. 1-10.

[12] D. Silver, A. Huang, C. J. Maddison, A. Guez, et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484-489, Jan. 2016.

[13] J. Ma, R. P. Sheridan, A. Liaw, G. E. Dahl, and V. Svetnik, "Deep neural nets as a method for quantitative structure-activity relationships," *J. Chem. Inf. Model.*, vol. 55, no. 2, pp. 263-274, Jan. 2015.

[14] M. Helmstaedter, K. L. Briggman, S. C. Turaga , V. Jain, et al., "Connectomic reconstruction of the inner plexiform layer in the mouse retina," *Nature*, vol. 500, no. 7461, pp. 168-174, Oct. 2014.

[15] M. K. K. Leung, H. Y. Xiong, L. J. Lee, and B. J. Frey, "Deep learning of the tissue-regulated splicing code," *Bioinformatics*, vol 30, no. 12, pp. 121-129, Jun. 2014.

[16] H. Y. Xiong, B. Alipanahi, L. J. Lee, H. Bretschneider, et al., "The human splicing code reveals new insights into the genetic determinants of disease," *Science*, vol. 347, no. 6218, pp. 144-151, Jan. 2015.

[17] R. S. Sutton and A. G. Barto., "Reinforcement Learning: An Introduction," *Springer Science+Business Media, LLC*, 1992.

[18] M. A. Alsheikh, S. Lin, D. Niyato, and H.-P. Tan, "Machine learning in wireless sensor networks: Algorithms, strategies, and applications," *IEEE Commun. Surv. Tutor.*, vol. 16, no. 4, pp. 1996-2018, Nov. 2014.

[19] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, "Machine learning for wireless networks with artificial Intelligence: A Tutorial on Neural Networks," *eprint arXiv: 1710.02913*, Oct. 2017.

[20] J. Chen, U. Yatnalli, and D. Gesbert, "Learning radio maps for UAV-aided wireless networks: A segmented regression approach," in *Proc. IEE International Conference on Communications (ICC 2017), Signal Processing for Communications Symposium*, May 2017, Paris, France.

[21] Y. Xiao, Z. Han, D. Niyato, and C. Yuen, "Bayesian reinforcement learning for energy harvesting communication systems with uncertainty," in *Proc. IEEE International Conference on Communications (ICC 2015), Next Generation Networking Symposium*, London, UK, June 2015.

[22] M. Bennis and D. Niyato, "A Q-learning based approach to interference avoidance in self-organized femtocell networks," in *Proc. IEEE Global Communications Conference (GLOBECOM 2010) Workshops, Workshop on Femtocell Networks*, Miami, FL, USA, December 2010.

[23] M. Chen, M. Mozaffari, W. Saad, C. Yin, M. Debbah, and C. S. Hong, "Caching in the sky: proactive deployment of cache-enabled unmanned aerial vehicles for optimized quality-of-experience," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 5, pp. 1046-1061, May 2017.

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

[25] T. Serre, L. Wolf, and T. Poggio, "Object recognition with features inspired by visual cortex," in *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005)*, San Diego, CA, USA, Jun. 2005, pp. 994-1000.

[26] T. V. Maia, "Reinforcement learning, conditioning, and the brain: Successes and challenges," *Cognitive, Affective, & Behavioral Neuroscience*, vol. 9, no. 4, pp. 343-364, Dec. 2009.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529-533, Feb. 2015.

[28] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, pp. 279-292, May 1992.

[29] S. S. Sonawane and P. A. Kulkarni, "Graph based representation and analysis of text document: A survey of techniques," *Int. j. comput. appl.*, vol. 96, no. 19. pp. 1-8, Jun. 2014.

[30] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," *IEEE Sig. Proc. Mag.*, vol. 34, no. 4, pp. 18-42, May 2017.

[31] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs," in *Proc. 2nd International Conference on Learning Representations (ICLR 2014)*, Banff, Canada, Apr. 2014, pp. 1-14.

[32] Y. Hechtlinger, P. Chakravarti, and J. Qin, "A Generalization of Convolutional Neural Networks to Graph-Structured Data," *eprint arXiv:1704.08165*, Apr. 2017.

[33] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph structured data," *eprint arXiv:1506.05163*, Jun. 2015.

[34] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Conference on Advances in Neural Information Processing Systems (NIPS 2016)*, vol. 29, Barcelona, Spain, Dec. 2016, pp. 3837-3845.

[35] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *Proc. 5th International Conference on Learning Representations (ICLR 2017)*, Toulon, France Apr. 2017, pp. 1-14.

[36] J. Lee, H. Kim, J. Lee, and S. Yoon, "Transfer learning for deep learning on graph-structured data," in *Proc. the 31st AAAI Conference on Artificial Intelligence (AAAI 2017)*, San Francisco, USA, Feb. 2017, pp. 1-7.

[37] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345-1359, Oct. 2010.

[38] V. R. Cadambe and S. A. Jafar, "Interference alignment and degrees of freedom of the K-user interference channel," *IEEE Trans. Inform. Theory,* vol. 54, no. 8, pp. 3425-3441, Aug. 2008.

[39] N. Zhao, F. R. Yu, M. Jin, Q. Yan, and V. C. M. Leung, "Interference alignment and its applications: A survey, research issues and challenges," *IEEE Commun. Surveys Tutorials*, vol. 18, no. 3, pp. 1779-1803, Mar. 2016.

[40] Y. He, C. Liang, F. R. Yu, N. Zhao, and H. Yin, "Optimization of cache-enabled opportunistic interference alignment wireless networks: a big data deep reinforcement learning approach," in *Proc. 2017 IEEE International Conference on Communications (ICC 2017)*, Paris, France, May 2017, pp. 1-6.

[41] G. Han, L. Xiao, H. V. Poor, "Two-dimensional anti-jamming communication based on deep reinforcement learning," in *Proc. 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2017)*, New Orleans, LA, USA, Mar. 2017, pp. 2087-2091.

[42] S. Peng, H. Jiang, H. Wang, H. Alwageed, and Y. Yao, "Modulation classification using convolutional neural network based deep learning model," in *Proc. 26th Wireless and Optical Communication Conference (WOCC 2017)*, Newark, NJ, USA, Apr. 2017, pp. 1-5.

[43] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Proc. 25th International Conference on Neural Information Processing Systems (NIPS 2012)*, Lake Tahoe, Nevada, USA, Dec. 2012, pp. 1097-1105.

[44] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, et al., "Caffe: Convolutional Architecture for Fast Feature Embedding," in *Proc. 22nd ACM international conference on Multimedia (MM 2014)*, Orlando, Florida, USA, Nov. 2014, pp. 675-678.

[45] E. Nachmani, Y. Beery, and D. Burshtein, "Learning to decode linear codes using deep learning," in *Proc. 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton 2016)*, Monticello, Illinois, USA, Sept. 2016, pp. 341-346.

[46] E. Nachmani, E. Marciano, D. Burshtein, and Y. Beery, "RNN decoding of linear block codes," *eprint arXiv:1702.07560*, Feb. 2017.

[47] T. Gruber, S. Cammerer, J. Hoydis, and S. Brink, "On deep learning-based channel decoding," in *Proc. IEEE 51st Annual Conference on Information Sciences and Systems (CISS 2017)*, Baltimore, MD, USA, Mar. 2017, pp. 1-6.

[48] S. Cammerer, T. Gruber, J. Hoydis, and S. t. Brink, "Scaling deep learning-based decoding of polar codes via partitioning," *eprint arXiv:1702.06901*, Feb. 2017.

[49] N. Samuel, T. Diskin, and A. Wiesel, "Deep MIMO detection," *eprint arXiv:1706.01151*, Jun. 2017.

[50] Y. S. Jeon, S. N. Hong, and N. Lee, "Blind detection for MIMO systems with low-resolution ADCs using supervised learning," in *Proc. 2017 IEEE International Conference on Communications (ICC 2017)*, Paris, France, May 2017, pp. 1-6.

[51] N. Farsad and A. Goldsmith, "Detection algorithms for communication systems using deep learning," *eprint arXiv:1705.08044*, Jul. 2017.

[52] T. O'Shea and J. Hoydis, "An introduction to deep learning for the physical layer," *eprint arXiv:1702.00832*, Jul. 2017.

[53] U. Challita, L. Dong, and W. Saad. "Deep learning for proactive resource allocation in LTE-U networks." in *Proc. 23th European Wireless Conference (European Wireless 2017)*, Dresden, Germany, May 2017, pp. 1-6.

[54] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," in *Proc. International Conference on Learning Representations (ICLR 2015)*, San Diego, CA, USA, May 2015, pp. 1-8.

[55] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229-256, May 1992.

[56] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *12th International Conference on Neural Information Processing Systems (NIPS 1999)*, Denver, CO, USA, Dec. 1999, pp. 1057-1063.

[57] M. Balazinska and P. Castro, "Characterizing mobility and network usage in a corporate wireless local-area network," in *Proc. 1st International Conference on Mobile Systems, Applications, and Services*, San Francisco, CA, USA, May 2003, pp. 303-316.

[58] Z. Xu, Y. Wang, J. Tang, J. Wang, and M. C. Gursoy, "A deep reinforcement learning based framework for power-efficient resource allocation in cloud RANs," in *Proc. 2017 IEEE International Conference on Communications (ICC 2017)*, Paris, France, May 2017, pp. 1-6.

[59] H. Sun, X. Chen, Q. Shi, M. Hong, et al., "Learning to optimize: Training deep neural networks for wireless resource management," in *Proc. IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC 2017)*, Sapporo, Japan, Jul. 2017, pp. 1-6.

[60] Q. Shi, M. Razaviyayn, Z.-Q. Luo, and C. He, "An iteratively weighted MMSE approach to distributed sum-utility maximization for a MIMO interfering broadcast channel," *IEEE Trans. Signal Process.*, vol. 59, no. 9, pp. 4331-4340, Apr. 2011.

[61] O. Naparstek and K. Cohen, "Deep multi-user reinforcement learning for distributed dynamic spectrum access," *eprint arXiv:1704.02613*, Apr. 2017.

[62] K. Cohen, A. Leshem, and E. Zehavi, "Game theoretic aspects of the multi-channel ALOHA protocol in cognitive radio networks," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 11, pp. 2276-2288, Nov. 2013.

[63] J Wang, J. Tang, Z. Xu, Y. Wang, et al., "Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach," in *Proc. IEEE Conference on Computer Communications (INFOCOM 2017)*, Atlanta, GA, USA, May 2017, pp. 1-10.

[64] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, OTexts, Sept. 2014.

[65] H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola and V. Vapnik, "Support vector regression machines," in *Proc. 9th International Conference on Neural Information Processing Systems (NIPS 1996)*, Denver, CO, USA, Dec. 1996, pp. 155-161.

[66] J. A. K. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Process. Lett.*, Vol. 9, No. 3, pp. 293-300, Jun. 1999.

[67] L. Liu, Y. Cheng, L. Cai, S. Zhou, Z. Niu, "Deep learning based optimization in wireless network," in *2017 IEEE International Conference on Communications (ICC 2017)*, Paris, France, May, 2017, pp. 21-25 .

[68] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural comput.*, vol. 18, no. 7, pp. 1527-1554, Jul. 2006.

[69] T. Hu and Y. Fei, "QELAR: A machine-learning-based adaptive routing protocol for energy-efficient and lifetime-extended underwater sensor networks," *IEEE Trans. Mobile Comput.*, vol. 9, no. 6, pp. 796-809, Jun. 2010.

[70] P. Xie, J.-H. Cui, and L. Lao, "VBF: Vector-based forwarding protocol for underwater sensor networks," in *Proc. 5th international IFIP-TC6 conference on Networking Technologies, Services, and Protocols (Networking 2006)*, Coimbra, Portugal, May 2006, pp. 1216-1221.

[71] N. Kato, Z. M. Fadlullah, B. Mao, F. Tang, et al., "The deep learning vision for heterogeneous network traffic control: proposal, challenges, and future perspective," *IEEE Wirel. Commun.*, vol. 24, no. 3, pp. 146-153, Jun. 2017.

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

[73] B. Mao, Z. M. Fadlullah, F. Tang, N. Kato, et al., "Routing or computing? The paradigm shift towards intelligent computer network packet transmission based on deep learning," *IEEE Trans. Comput.*, vol. 66, no. 11, pp. 1946-1960, Nov 2017.

[74] Y. Lee, "Classification of node degree based on deep learning and routing method applied for virtual route assignment," *Ad Hoc Netw.*, vol. 58, pp. 70-85, Apr. 2017.

[75] Q. You, Y. Li, M. S. Rahman, and Z. Chen ,"A near optimal routing scheme for multi-hop relay networks based on Viterbi algorithm," in *Proc. 2012 IEEE International Conference on Communications (ICC 2012)*, Ottawa, ON, Canada, Jun. 2012, pp. 1-6.

[76] G. Stampa, M. Arias, D. Sanchez-Charles, V. Muntes-Mulero, A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization," *eprint arXiv:1709.07080*, Sept. 2017.

[77] A. Varga, "The OMNeT++ discrete event simulation system," in *Proc. the 15th European Simulation Multiconference (ESM 2001)*, Prague, the Czech Republic, Jun. 2001, pp. 1-7.

[78] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *Proc. 1st international conference on simulation tools and techniques for communications, networks and systems & workshops (Simutools 2008)*, Marseille, France, Mar. 2008, pp. 1-10.

[79] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, et al., "Knowledge-defined networking training datasets," Universitat Politecnica de Catalunya, [Online] Available: http://knowledgedefinednetworking.org.

[80] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "A machine learning approach to routing," *eprint arXiv:1708.03074*, Aug. 2017.

[81] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," in *Proc. 32nd International Conference on Machine Learning (ICML 2015)*, Lille, France, Jul. 2015, pp. 1-9.

[82] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proc. 33rd International Conference on Machine Learning (ICML 2016)*, New York, NY, USA, Apr. 2016, pp. 1-14.

[83] R. Atallah, C. Assi, M. Khabbaz, "Deep reinforcement learning-based scheduling for roadside communication networks," in *Proc. 15th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt 2017)*, Paris, France, May 2017, pp. 1-8.

[84] B. Sun, H. Feng, K. Chen, X. Zhu, "A deep learning framework of quantized compressed sensing for wireless neural recording," *IEEE Access*, vol. 4, pp. 5169-5178, Sept. 2016.

[85] T. Tieleman and G. Hinton, "Lecture 6.5: rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Netw. Mach. Learn.* vol. 4, no. 2, pp. 26-31, Oct. 2012.

[86] S. O. Haykin, *Neural Networks and Learning Machines*, 3th Edition, Pearson Higher Ed, Nov. 2008.

[87] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. 19th International Conference on Computational Statistics (COMPSTAT 2010)*, Paris, France, Aug. 2010, pp. 177-187.

[88] J. Zhang, Y. Suo, S. Mitra, S. Chin, et al., "An efficient and compact compressed sensing microsystem for implantable neural recordings," *IEEE Trans. Biomed. Circuits Syst.*, vol. 8, no. 4, pp. 485-496, Jan. 2014.

[89] L. Jacques, D. K. Hammond, and J. M. Fadili, "Dequantizing compressed sensing: When oversampling and non-Gaussian constraints combine," *IEEE Trans. Inf. Theory*, vol. 57, no. 1, pp. 559-571, Dec. 2010.

[90] Z. Yang, L. Xie, and C. Zhang, "Variational Bayesian algorithm for quantized compressed sensing," *IEEE Trans. Signal Process*, vol. 61, no. 11, pp. 2815-2824, Jun. 2013.

[91] H. Mao, M. Alizadeh. I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th ACM Workshop on Hot Topics in Networks (HotNets 2016)*, Atlanta, GA, USA, Nov. 2016, pp. 50-56.

[92] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Traffic classification through simple statistical fingerprinting," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 1, pp. 5-16, Jan. 2007.

[93] X. Wang and D. J. Parish, "Optimised multi-stage TCP traffic classifier based on packet size distributions," in *Proc. Third International Conference on Communication Theory, Reliability, and Quality of Service (CTRQ 2010)*, Athens/Glyfada, Greece, Jun. 2010, pp. 98-103.

[94] R. Sun, B. Yang, L. Peng, Z. Chen, L. Zhang, and S. Jing, "Traffic classification using probabilistic neural networks," in *Proc. Sixth International Conference on Natural Computation (ICNC 2010)*, Yantai, China, Aug. 2010, pp. 1914-1919.

[95] H. Ting, W. Yong, T. Xiaoling, "Network traffic classification based on kernel self-organizing maps," in *Proc. 2010 International Conference on Intelligent Computing and Integrated Systems (ICISS 2010)*, Guilin, China, Oct. 2010, pp. 310-314.

[96] A. H. Lashkari, G. D. Gil, M. S. I. Mamun, A. A. Ghorbani, "Characterization of encrypted and VPN traffic using time-related features," in *Proc. 2nd International Conference on Information Systems Security and Privacy (ICISSP 2016)*, Rome, Italy, Feb. 2016, pp. 407-414.

[97] Y. L. Gwon and H. T. Kung, "Inferring origin flow patterns in Wi-Fi with deep learning," in *Proc. 11th International Conference on Autonomic Computing (ICAC 2014)*, Philadelphia, PA, USA, Jun. 2014, pp. 73-83.

[98] B. A. Olshausen and D. J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607-609, Jun. 1996.

[99] Y. L. Boureau, J. Ponce, and Y. Lecun, "A theoretical analysis of feature pooling in visual recognition," in *Proc. 27th International Conference on Machine Learning (ICML 2010)*, Haifa, Israel, Jun. 2010, pp. 1-8.

[100] L. Ljung, *System Identification: Theory for the User*, 2nd Edition, Prentice Hall, Jan. 1999.

[101] Z. Wang, "The applications of deep learning on traffic identification," in *Proc. Black Hat USA 2015*, Las Vegas, Aug, 2015, pp. 1-10.

[102] M. Lotfollahi, R. S. H. Zade, M. J. Siavoshani, and M. Saberian, "Deep packet: a novel approach for encrypted traffic classification using deep learning," *eprint arXiv:1709.02656*, Sept. 2017.

[103] F. Chollet, et al., "Keras: Deep Learning for humans," [Online] available: https://github.com/fchollet/keras.

[104] B. Yamansavascilar, M. A. Guvensan, A. G. Yavuz, and M. E. Karsligil, "Application identification via network traffic classification," in *Proc. 2017 International Conference on Computing, Networking and Communications (ICNC 2017)*, Santa Clara, CA, USA, Jan. 2017, pp. 843-848.

[105] T. P. Oliveira, J. S. Barbar, and A. S. Soares, "Computer network traffic prediction: a comparison between traditional and deep learning neural networks," *Int. J. Big Data Intelligence*, vol. 3, no. 1, pp. 28-37, Jan. 2016.

[106] E. Hodo, X. Bellekens, A. Hamilton, C. Tachtatzis, and R. Atkinson, "Shallow and deep networks intrusion detection system: a taxonomy and survey," *eprint arXiv:1701.02145*, Jan. 2017.

[107] Q. Niyaz, W. Sun, A. Y. Javaid, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proc. 9th EAI International Conference on Bio-inspired Information and Communications Technologies (BICT 2015)*, New York City, NY, USA, Dec. 2015, pp. 21-26.

[108] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, "Self-taught learning: transfer learning from unlabeled data," in *Proc. 24th International Conference on Machine Learning (ICML 2007)*, Corvalis, OR, USA, Jun. 2007, pp. 759-766.

[109] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. 2nd IEEE Symposium on Computational Intelligence for Security and Defence Applications (CISDA 2009)*, Ottawa, ON, Canada, Jul. 2009, pp. 1-6.

[110] NSL-KDD dataset, [online] available: http://nsl.cs.unb.ca/nsl-kdd/.

[111] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *Proc. 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM 2016)*, Fez, Morocco, Oct. 2016, pp. 1-6.

[112] R. Salakhutdinov and G. Hinton, "deep Boltzmann machines," *Artif. Intell.*, vol. 5, no. 2, pp. 448-455, Jan. 2009.

[113] G.E. Hinton, "A practical guide to training restricted Boltzmann machines," *Momentum*, vol.9, no. 1, pp.1-21, Aug. 2010.

[114] U. Fiore, F. Palmieri, A. Castiglione, and A. D. Santis, "Network anomaly detection with the restricted Boltzmann machine," *Neurocomputing*, vol. 122, pp. 13-23, Dec. 2013.

[115] N. Gao, L. Gao, Q. Gao, and H. Wang, "An intrusion detection model based on deep belief networks," in *Proc. 2nd International Conference on Advanced Cloud and Big Data (CBD 2014)*, Huang Shan, China, Nov. 2014, pp. 247-252.

[116] M. Z. Alom, V. Bontupalli, and T. M. Taha, "Intrusion detection using deep belief networks," in *Proc. National Aerospace and Electronics Conference (NAECON 2015)*, Dayton, OH, USA, Jun. 2015, pp. 339-344.

[117] M-J. Kang and J-W. Kang, "Intrusion detection system using deep neural network for in-vehicle network security," *PLoS ONE*, vol. 11, no. 6, pp. 1-17, Jun. 2016.

[118] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, et al., "Tensorflow: large-scale machine learning on heterogeneous distributed systems," *eprint arXiv:1603.04467*, Mar. 2016.

[119] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, et al., "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM international conference on Multimedia (MM 2014)*, Orlando, FL, USA, Nov. 2014, pp. 675-678.

[120] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, et al., "Theano: A Python framework for fast computation of mathematical expressions," *eprint arXiv:1605.02688*, May 2016.
[121] "WILL API," [Online] available: https://scarsty.gitbooks.io/will/content/.
[122] T. Chen, M. Li, Y. Li, M. Lin, et al., "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," in *Proc. NIPS Workshop on Machine Learning Systems (NIPS 2016)*, Barcelona, Spain, Dec. 2016, pp. 1-6.
[123] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *Proc. BigLearn NIPS Workshop (NIPS 2011)*, Granada, Spain, Dec. 2011, pp. 1-6.
[124] "Comparison of deep learning software," [Online] Available: https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software.