network

MDPI

*Review*

# On Attacking Future 5G Networks with Adversarial Examples: Survey

**Mikhail Zolotukhin [1,*], Di Zhang [2], Timo Hämäläinen [1] and Parsa Miraghaei [3]**

[1] Magister Solutions Ltd., 40720 Jyväskylä, Finland
[2] Faculty of Information Technology, University of Jyväskylä, 40014 Jyväskylä, Finland
[3] Faculty of Information Technology and Communication Sciences, Tampere University, 33100 Tampere, Finland
* Correspondence: mikhail.zolotukhin@magister.fi

**Abstract:** The introduction of 5G technology along with the exponential growth in connected devices is expected to cause a challenge for the efficient and reliable network resource allocation. Network providers are now required to dynamically create and deploy multiple services which function under various requirements in different vertical sectors while operating on top of the same physical infrastructure. The recent progress in artificial intelligence and machine learning is theorized to be a potential answer to the arising resource allocation challenges. It is therefore expected that future generation mobile networks will heavily depend on its artificial intelligence components which may result in those components becoming a high-value attack target. In particular, a smart adversary may exploit vulnerabilities of the state-of-the-art machine learning models deployed in a 5G system to initiate an attack. This study focuses on the analysis of adversarial example generation attacks against machine learning based frameworks that may be present in the next generation networks. First, various AI/ML algorithms and the data used for their training and evaluation in mobile networks is discussed. Next, multiple AI/ML applications found in recent scientific papers devoted to 5G are overviewed. After that, existing adversarial example generation based attack algorithms are reviewed and frameworks which employ these algorithms for fuzzing stat-of-art AI/ML models are summarised. Finally, adversarial example generation attacks against several of the AI/ML frameworks described are presented.

**Keywords:** 5G networks; artificial intelligence; deep learning; adversarial machine learning; 5G cybersecurity knowledge base



**Citation:** Zolotukhin, M.; Zhang, D.; Hämäläinen, T.; Miraghaei, P. On Attacking Future 5G Networks with Adversarial Examples: Survey. *Network* **2023**, *3*, 39–90. https://doi.org/10.3390/network3010003

Academic Editor: Markus Fiedler

Received: 17 November 2022
Revised: 9 December 2022
Accepted: 27 December 2022
Published: 30 December 2022



## 1. Introduction

According to IMT-2020 requirements, the 5th generation networks shall provide for 20 times faster communication and 90% less latency compared to 4G-LTE which calls for a series of new technologies being innovated, applied and developed [1]. This revolution takes place in each networking component including radio access network (RAN), core network (CN) and user equipment (UE). Millimetre waves (mmWave) utilising higher frequency, e.g., 24.25–27.5 GHZ demand more agile and precise mobility management to maintain service continuity [2] whereas massive multi-input multi-output (MIMO) antenna technology since applying 64T/64R requires more accurate resource allocation mechanisms and prediction algorithms to enhance the space diversity [3]. Dynamic time-division duplexing (TDD) that aims to improve the spectrum efficiency by continuously adapting the transmission direction needs more real-time reacting algorithms [4]. Despite these frequency, space and domain technologies sound valuable in theory, they are hard to coordinate in effect. The significantly raised management requirements are expected to be solved by introducing artificial intelligence (AI) and machine learning (ML), as proposed in new 3GPP study item "Artificial Intelligence and Machine Learning in NG-RAN: New

Study in RAN3" [5]. Furthermore, AI/ML frameworks are proved to be more resource efficient and they allow mobile network providers to reduce operational expenses (OpEx), especially when facing more complex hard to model systems [6] or embracing new kinds of applications to the wireless communication systems [7]. Furthermore, AI/ML improves the ability of the operators to learn about their networks and customers' needs. This in return leads to better understanding and reasoning when making decisions in different scenarios and environment conditions [8].

As AI/ML has recently become an indispensable part of modern mobile networks, vulnerability of AI/ML models to adversarial effects causes an increasing concern. Due to the shared nature of wireless medium, mobile network applications are vulnerable to adversaries that can manipulate the training and inference processes of AI/ML-driven frameworks over the air. However, due to different channel gain values, an adversary is not capable of observing the input features of the target AI/ML model directly. In addition, since the output of the target model is most of the time used internally by the network component and it is not accessible from outside, the attacker does not have access to the output label. Finally, the adversary usually can only change the input data indirectly by adding some signal perturbations to the existing transmissions [9].

Therefore, from the methodology point of view, the attack resembles radio jamming, i.e., an adversary uses a malicious fake BS (FBS) to emit a radio signal on top of existing transmissions to perturb the input to the model in such a way that its output is incorrect which in turn may negatively affect the functionality of the corresponding network component in which the model is deployed. In the majority of cases, the effect from the attack is similar to the one caused by radio jamming and denial-of-service attacks, i.e., deterioration of the network data rate which can lead to mobile customers being unable to use certain network services as described in FS.30 Security Manual of 5G Cybersecurity Knowledge Base [10]. For this reason, mitigation measurements from RAN-3 of the Baseline Security Controls FS.31 can be employed [11] as well as security techniques described in IR.77 [12] and fraud mitigation measurements [13].

Speaking in more details of the adversarial machine learning attacks, those may take place during both the training or inference stage. During the training, the adversary either poisons the training data directly or injects perturbations to the training samples so that the target model is trained with the resulting malicious samples leading to it making errors later during the inference [14]. There are two main variations of such poisoning attacks: those targeting the model's availability, and those targeting its integrity. The first group of the attacks aims to distort the patterns present in the data in order to make the model inaccurate and therefore useless. The second group of the attacks is more sophisticated as it aims to poison the training data in such an intelligent way that a backdoor is introduced into the target model. A backdoor is a type of input the attacker can leverage to make the model under attack to label data samples with certain characteristics as the target class. As a result, the trained model produces correct results on regular clean data, however, it misclassifies a source category sample as the target category one when the attacker injects the trigger to the source input feature vector [15].

Concerning the poisoning attacks in 5G, an adversary can in theory utilise the end users' equipment to send false signals and messages to the RAN domain after authentication and authorization in the CN domain. If a network provider decides to use the data collected to train an AI/ML model, it ends up with the model that is either inaccurate or, in the worst case scenario, vulnerable to backdoor triggers. Apart from data poisoning via end users' equipment, poisoning attacks may also take place when transmitting model parameters and/or data in certain transmission procedures as RAN can also exchange data with the CN and the operation and maintenance (OAM) domains to collaboratively improve specific end users' quality of experience (QoE) with regards to data rate [16] and to enhance cooperatively the service continuity of the end users [17] as well as RAN throughput [18]. Despite such a threat certainly exists, this attack approach can be successful only in the case of some major flaw in the data processing pipeline. For example, the provider decides to

use the training dataset obtained from a non-trusted source or the adversary has access to some internal components of the AI/ML model during the training stage which allows it to inject poisonous data.

In our research, we focus on adversarial example attacks. This type of attack is a more realistic threat to the AI/ML models deployed in a 5G network since it takes place in the inference stage and therefore requires having access to neither the target model nor the datasets during the training. During such an adversarial example generation attack the adversary aims to supply such input features to the target model that it outputs some wrong result [19]. As mentioned above, adversarial examples in mobile networks are usually generated via adding carefully crafted radio signal perturbations. In this case, the adversary as a rule perturbs the samples on the decision boundary. This allows the attacker to increase the chance of the target model making an error [20]. The algorithms for adversarial example generation usually fall into one of two main categories: white-box or black-box. In the former scenario, the adversary has access to the target model and in some cases even the data used for its training. In the latter case, the attacker is only capable of observing the labels returned by the model for the inputs provided making the attack algorithms from this category more applicable in real-world use case scenarios.

In a 5G network system, white-box attacks in the inference stage would require having the ability to access the target network components, e.g., centralised unit (CU) in RAN, network data analytics function (NWDAF) in CN, management data analytics function (MDAF) in OAM and data storage centre, which needs to initiate an eavesdropping or man-in-the-middle attack in advance. Despite being more efficient, these attack vectors are much more difficult to implement due to defending mechanisms defined in 3GPP SA3 [21]. As mentioned earlier, in a real world wireless networking system the adversary has direct access to neither the target model nor its input feature space and output labels. For this reason, black-box attack scenarios are more realistic in a 5G system as such attacks are much easier to carry out by utilising end users' equipment to send intelligently crafted signals.

As we have recently become interested in this research area, we find that amusing that to our knowledge there is no all-in-one type of document devoted to this topic that would allow a reader to obtain extensive information about every important aspect of the problem including AI/ML applications in 5G networks, datasets and simulators that can be used for experimentation, algorithms for crafting adversarial examples and fuzzing frameworks which provide stable implementations of these algorithms. The purpose of this survey is to fill this gap by providing detailed introduction and references for each of the issues listed. Our contribution is therefore two-fold. First, we summarise and categorise the studies devoted to applications of AI/ML in the mobile networking domain including the data used and the algorithms employed. Second, we survey and analyse potential attacks which can be carried out using adversarial examples in the next generation mobile networks and outline future research directions and use case scenarios to study which might be useful for other researchers interested in this topic.

The rest of the document is organised as follows. Section 2 overviews big data used in 5G. Various AI/ML algorithms that are proposed to be employed in 5G networks are summarised in Section 3. Section 4 surveys a multitude of recent scientific papers devoted to enhancing 5G networks with AI/ML. Both white-box and black-box adversarial example generation attacks against these algorithms are listed in Section 5. Existing tools and frameworks which can be deployed for AI/ML fuzzing are overviewed in Section 6. Transferability of the attack algorithms and tools studied to the 5G network domain is discussed in Section 7. The main points of the survey are discussed in Section 8. Section 9 concludes the paper and outlines future work.

## 2. Big Data in 5G

Based on its source, the mobile network data can be divided into two main categories: network-level and application-level [22]. The former is obtained throughout network infrastructure while the latter as a rule is collected by the mobile devices on the edge.

Network-level mobile data may include global mobile network performance metrics which include but are not limited to throughput, end-to-end delay, and jitter, as well as individual session times, information about the sender and the receiver, communication types, call detail records (CDRs) and many others. Network-level data is usually utilised for network diagnosis and management. Moreover, spatio-temporal patterns extracted from the network-level data can be employed for user mobility analysis and public transportation planning. Network-level data can be further subdivided into the following four sub-classes: radio information, CDR, key performance indicators (KPI), and infrastructure data [8]. The first group includes such radio signal statistics as the serving BS, modulation scheme, spectrum, frequency and signal power. CDR data may contain information about the sender and the receiver, packet inter-arrival times, session start and end times and several others. Frequently used KPIs in the third group are bit/packet error rate, end-to-end delay, jitter, quality of experience (QoE), and throughput. The last group consists of information about equipment holders, infrastructure locations and capabilities.

In distinction from the network-level category, application-level data is recorded by mobile devices' sensors or applications installed on those devices. The data sources may include mobile cameras and video recorders as well as global positioning systems (GPS) and several others. Data gathering, preprocessing, and subsequent distribution of such data to specific locations is carried out on mobile devices themselves [23]. The data as a rule is related to public events and it is collected through crowdsourcing schemes. Such increase of social data can be explained by availability of the mobile Internet which has made multimedia communication accessible for everyone. Furthermore, preferences of individuals or social groups can also be extracted and analysed due to wide spread and popularity of mobile social networks. Another distinct group of application-level data is related to cloud servers and multimedia content stored on them. Users' preferences for specific multimedia content can be analysed and used to predict content demands in the future which in turn allows the network providers to perform recommendation actions in order to improve user experience.

Many studies devoted to application of AI/ML in wireless networks generate the data using various simulation software. Such software includes specialised network simulators such as Vienna LTE-A [24], 6TiSCH [25] and OMNeT++ [26]. Another group of researchers generate the data implementing well known channel models using programming and scripting software, e.g., Python and Matlab. The third data simulation option is using a real radio network testbed. Even though this option allows one to carry out the most realistic experiments, it is obviously the most expensive one. In addition, such an approach is most of the time not scalable in the lab environment. Finally, the data for training and evaluating AI/ML models can be retrieved from publicly available datasets. The datasets used in the studies surveyed and summarised later in this document are listed in Table 1.

**Table 1.** Mobile network datasets used by researchers for training AI/ML models.

| Dataset | Description | Features and Labels | Mobile Network Applications |
|---|---|---|---|
| DeepMIMO [27] | Dataset for mmWave and massive MIMO | Channel matrix, ray-tracing path parameters, line-of-sight status, tx-rx distance path loss, UE and BS locations | Channel estimation in MIMO [28] |
| RadioML [29] | Synthetic dataset consisting of several modulations at varying SNRs | The received signal in in-phase/quadrature (I/Q) format, SNR, modulation type | Automatic modulation classification [30–32] |
| Raymobtime [33] | Realistic dataset with ray tracing, mobility and time evolution | UE positions, ray-tracing, lidar and video image data | Beam selection [34] |
| SPHERE [35] | Activity recognition with multi-modal sensor data | Accelerometer data, features extracted from video, passive sensor data, activity | Activation control [36] |
| Massive MIMO [37] | Data for power allocation in the downlink of a massive MIMO network | UE positions, max-min and max-prod power allocation vectors | Power allocation [37] |
| IBM/Watson [38] | SNMP records for IBM Watson research centre over several weeks | Aggregated network traffic features for each user of several APs | Scheduling [39] |
| CTU [40] | Real botnet, normal and background traffic | Raw packet captures, per-flow network traffic data features, flow labels | Network intrusion detection [41] |
| AWID [42] | Real 802.11 WLAN traces including normal and malicious traffic | MAC layer, radio and general frame information extracted from 802.11 WLAN traffic, attack labels | Network intrusion detection [43] |
| CICIDS18 [44] | Realistic network traffic data for multiple attack scenarios | Raw packet captures, per-flow network traffic data features | Network intrusion detection [45] |
| WSN-DS [46] | Dataset for intrusion detection systems in wireless sensor networks | Node state features: energy consumption, packets sent/received, distance to BS, attack labels | Jamming detection and classification [47] |

## 3. Algorithms

In this section, we summarise multiple AI/ML algorithms which are employed for big data analysis in 5G networks. They are usually grouped into one of the following three general categories depending on whether or not a human supervision is required: supervised, unsupervised and reinforcement. Supervised learning requires large quantities of human-labelled data available to learn a functional mapping between the input features and the output labels. Two main advantages of supervised learning are the convergence speed and high accuracy. Unsupervised learning infers the underlying information structure of the data without any external supervision. As a rule unsupervised learning methods are less accurate than supervised ones, but on the other hand no prior knowledge is required to train an AI/ML model in an unsupervised way. Finally, reinforcement learning (RL) discovers optimal actions through interactions in uncertain time-varying environments with the help of reward signals retrieved by an agent during the learning process. Direct supervision is not required for an RL agent, but shaping an efficient reward function may require considerable manual effort. It is also worth mentioning that training an RL agent is most of the time carried out in a realistic simulation software designed to emulate the corresponding real world environment since often training in the real world environment is either extremely time consuming and expensive or not feasible at all. Furthermore, implementing such a simulation software may require huge amounts of resources which poses additional limitations on this machine learning approach.

### *3.1. Supervised Learning*

Table 2 summarises AI/ML algorithms proposed to be deployed in various mobile network components. As one can notice, the major part of the AI/ML applications is based on deep learning architectures trained in a supervised way. Deep learning approach relies on training multi-layered neural networks the first layers of which look for extraction of features from raw data samples whereas the achievement of the task given is the responsibility of the later layers. The neurons are activated through weighted connections. As a rule, the neuron output is calculated by applying a nonlinear activation function, which allows the network to approximate nonlinear transformations. The learning process is carried out by back-propagating the loss calculated in the output layer towards the input layer. Speaking of the loss function, in classification tasks it is usually categorical cross-entropy, whereas in regression tasks the mean square error (MSE) is commonly used. The simplest deep learning architecture frequently used is a fully-connected neural network. Each neuron in the output or a hidden layer of such a network is connected to each neuron of the previous layer. Fully-connected, or dense, layers have few trainable parameters which allows them to learn faster than other more complicated deep learning structures, however they cannot extract spatio-temporal patterns present in the data which can be a critical issue when dealing with time-series or images.

Convolutional neural networks (CNNs) are often used in image related problems as they are capable of extracting low-level features including edges as well as colour and gradient orientations [48]. The main building block of a CNN is the convolutional layer which calculates the overlap of the layer's filter over its input. As previously, to account for nonlinearity in the data the output value is passed through an activation function. As a rule, the final output of the convolution layer consists of multiple convolutions of the input with different filters. After a convolution operation, pooling can be performed to reduce the dimensionality of the output. Max pooling is probably the most frequently used pooling mechanism. It uses a moving window over its inputs with a given step and calculates the maximum value in each such window. A CNN as a rule includes several convolutional layers mixed with few pooling layers followed by dense layers. In such architecture, the convolutional layers allow for extraction of local features present in the image data supplied whereas the dense layers provide for global overview of the features extracted.

In order to extract features from time-series data, recurrent neural networks (RNNs) are often employed. Such deep learning architectures have one or several recurrent layers. Each recurrent layer has some sort of an internal memory which changes with every new batch of training samples supplied. The value of this memory also affects the output of the layer. As a rule, the output value is simply equal to the weighted sum of the input sample and the internal memory state vector. As in previous cases, an activation function is applied to this output to account for nonlinearities. At the training stage, the error is propagated through time back to a certain moment. This may result in infinite or extremely small gradient values. The former problem can be solved via gradient clipping [49]. The latter one requires introducing specific gate structures which control the internal state updates. The most popular gate-based RNN layers include long short-term memory (LSTM) [50] and gated recurrent units (GRUs) [51]. Another RNN architecture is echo state network (ECN) which addresses computational expensiveness of gated RNNs [52]. In ECN, the weights of the hidden reservoir layer are randomly assigned and they are not trainable. The same applies to the weights between the input and the hidden layer. The only trainable part of the network consists of the weights of the output neurons. It is learned in a supervised way so that the entire framework reproduces specific temporal patterns. It is worth mentioning that convolutional layers can also be employed to extract features from temporal data. For example, DeepMind's Q-network that teaches itself to play Atari games, stacks the last four frames of the historical data to produce an input for the CNN [53].

Recent studies propose a new architecture that replaces RNNs with attention mechanisms [54]. As a rule, an attention function maps a set of key-value pairs and a query to some output. The attention weight for each value in this mapping is computed as a com-

patibility function of the query with the corresponding key. The sum of the values each of which is multiplied by the corresponding weight acts as the output of the attention function. Transformer, the neural network architecture proposed in the aforementioned study [54], is based on the attention mechanism entirely. The transformer allows one to achieve a new state-of-the-art accuracy in certain language processing tasks. At the same time, it can be trained significantly faster than architectures based on recurrent or convolutional layers.

The last but not the least deep learning mechanism worth mentioning in the introduction is residual neural networks [55]. Such networks aim to alleviate the problem of vanishing gradients that is typical in deep learning architectures: as the gradient is back-propagated to earlier layers, repeated multiplication makes the gradient infinitely small. The core idea of residual networks is introducing an identity shortcut connection that skips one or several layers. Stacking the resulting residual blocks on top of each other allows one to build a very deep network architecture which does not suffer from vanishing gradients. The original paper that introduced residual networks used two different blocks: identity block and convolutional block. The difference is the latter has additional convolution, batch normalisation and activation layers in the main path and convolution accompanied by the batch normalisation in the shortcut.

There are several supervised algorithms not based on deep learning which are still popular among AI/ML researchers. For example, a support vector machine (SVM) constructs a hyperplane or a set of hyperplanes in a high- or infinite-dimensional space to separate samples in a dataset. Such hyperplanes are usually found by solving the corresponding constrained optimization problem. Linear regression is the simplest regression method which is essentially a single-layer neural network with linear activation function. Support vector regression (SVR) is closely related to both SVM and linear regression algorithms. The SVR algorithm aims to find the best fit hyperplane such that the maximum number of the training samples can be placed within some small distance from it. K-nearest neighbours (k-NN) algorithm is another classification algorithm. It predicts the label of a new sample based on several closest samples from the training set. In regression tasks, the output value is calculated as the average of the nearest training samples. Some studies still rely on employing the naive Bayes algorithm which is a classification technique which assumes that the presence of a certain feature in a class does not depend on other features and predicts the label using Bayes theorem. According to this theorem the probability of an event is calculated using prior knowledge of conditions connected to this event. The next category of algorithms is based on decision trees. A decision tree classifier can be trained using a variety of algorithms. Probably the most popular one, called ID3, is based on measuring entropy values of the features and selecting the one with the smallest value at each iteration to build the tree. Decision tree classifiers are often combined into ensembles. For example, the random forest algorithm constructs multiple decision trees at the training stage and outputs the mean of the labels returned by those trees during the inference. Finally, the gradient boosting decision tree is an algorithm which builds a prediction model in the form of an ensemble of decision trees in an iterative fashion: each tree in the ensemble attempts to correct the errors of its predecessor [56].

**Table 2.** AI/ML algorithms used by researchers in the mobile networking domain.

| Category | Algorithm | Mobile Network Applications |
|---|---|---|
| Supervised | Fully-connected neural network | Channel estimation [28,57], symbol detection [58], automatic modulation classification [59], channel coding [60–62], beamforming [34,63–65], activation control [66,67], power allocation [37,68–71], scheduling [72], routing [73], security [41,43,47,74] |
| | CNN | Channel estimation [57,75–77], automatic modulation classification [31,78], channel coding [61,62,79–81], beamforming [34], power allocation [82], routing [83], localization [84] |
| | RNN (GRU, LSTM and ECN) | Automatic modulation classification [32], channel coding [61], power allocation [85], scheduling [39], routing [83], localization [84], security [45], slicing [86], caching [87] |
| | Attention (transformer) | Channel coding [62,79] |
| | Residual neural network | Beamforming [34] |
| | K-NN | Security [74] |
| | SVM (and SVR) | Beamforming [88], security [47] |
| | Linear regression | Beamforming [88], security [74] |
| | Decision trees (including random forest and gradient boosting trees) | Beamforming [88], security [74] |
| | Naive Bayes | Security [74] |
| Unsupervised | Autoencoder | Channel estimation [77], channel coding [79], scheduling [39], caching [89], security [43] |
| | GAN | Channel estimation [76], beamforming [90] |
| | Denoising CNN (DIP, LDAMP and DnCNN) | Channel estimation [91,92] |
| Reinforcement | Q-learning | Channel coding [60], beamforming [90], power allocation [93], caching [94] |
| | DQN | Activation control [67], power allocation [70,82,85], scheduling [95], routing [83], slicing [96–98] |
| | DDPG | Activation control [66], scheduling [72], routing [99] |
| | SARSA | Activation control [36] |
| | Policy gradients | Channel coding [62], scheduling [39,95] |
| | TRPO | Routing [73] |
| | PPO | Slicing [100] |

### *3.2. Unsupervised Learning*

Speaking of the unsupervised learning, autoencoder is probably the most popular deep learning architecture in this category. In general, an autoencoder includes an input layer, several hidden layers, and an output layer. The model aims to adjust its parameters in such a way that the output layer is equal to the input one despite the information bottleneck caused by the hidden layers. The role of the loss function is often played by the reconstruction error which is the difference between the input and the output. There are several autoencoder variations depending on the properties of the hidden layers and additional terms in the loss function. For example, in variational autoencoders (VAEs) the encoder returns a distribution over the latent space instead of a single point. Another autoencoder architecture variation is called sparse autoencoder. To train this architecture, the loss function includes not only the reconstruction error but also a special penalty function. This penalty is the bigger the greater the number of units which fire in the bottleneck layer. Autoencoders can be employed for anomaly detection. This process

usually involves two main steps. First, the training data is fed to an autoencoder until it is well trained to reconstruct the expected output with minimum error. Second, the same data is fed again to the trained autoencoder and the error term of each reconstructed data point is measured. In theory, a well-trained autoencoder learns how to reconstruct an input that follows a certain format. A badly formatted data point fed to such an autoencoder model results in something that is quite different from the expected output, and therefore a large error term. This mechanism can for example be used to classify anomalies by comparing the reconstruction error of an unknown sample to the error threshold observed during the training.

One interesting unsupervised CNN-based network architecture which is close to the autoencoder is deep image prior (DIP) model, which fits the parameters of a CNN on the fly instead of training beforehand [101]. In this model, the input is a codeword and the output is an image sized tensor. The model weights are updated iteratively to minimise the difference between the network output and the real image given. This learning-free model can be used for image restoration problems, where the information lost during the degradation process is expected to be supplemented with the image prior. Another CNN-based deep learning architecture employed for image recovery is learned denoising-based approximate message passing (LDAMP) neural network [102]. This network aims to solve the problem of finding the image given the low-dimensional codeword by exploiting the fact that the image is sampled from the set of all natural images. Another denoising network based on CNN is proposed in [103]. The input to such a denoising CNN (DnCNN) is a noisy observation, during the training the model learns to minimise the difference between the output of the network and the noise tensor.

Another deep learning framework which shares a similar idea with autoencoders is the restricted Boltzmann machine (RBM). In distinction to autoencoders, it uses stochastic units with a particular distribution instead of deterministic distribution. An RBM consists of only two layers: visible and hidden, both of the layers have binary-valued neurons. The RBM training process is different from the one used for traditional neural network models. It starts with taking a training sample, computing the probability of each hidden unit and using this probability distribution to sample a hidden activation vector. After that, the outer product of these two vectors is calculated and called the positive gradient. Next, a reconstruction vector of the visible units is sampled from the hidden distribution and it is used to resample the hidden activations via Gibbs sampling [104]. This routine is repeated several times. The last outer product of the two resampled vectors is called the negative gradient. The weight matrix update is calculated as the positive gradient minus the negative gradient times some learning rate. RBMs are usually stacked together to build a deep belief network (DBN). During the DBN training, each RBM layer is trained separately in an unsupervised way. Given a labelled dataset, the entire DBN can be then fine-tuned in a supervised way.

The next unsupervised deep learning approach worth to be mentioned is generative adversarial networks (GANs) [105]. A GAN consists of two neural networks: a generator and a discriminator. The former transforms a random vector into a sample from the input domain, whereas the latter learns to discriminate between real samples and the ones produced by the generator. The error between the actual label and the discriminator output is measured with the cross-entropy loss. The generator's weights are adjusted based on how well the samples produced by the generator fool the discriminator. A conditional extension of the GAN algorithm is proposed in [106]. Such conditioning is usually carried out by extending both the generator and discriminator with an additional input layer to feed some extra information.

Outside of the deep learning area, K-means is probably the most popular unsupervised algorithm. It partitions a dataset of objects into a predefined number of clusters. The algorithm attempts to minimise the sum of distances between each sample and the mean value of the cluster into which this sample falls. As a rule, it uses an iterative refinement technique. First, cluster means are initiated, e.g., by picking samples from the dataset at

random. Next, each sample is assigned to the cluster which corresponds to the least distant mean. New means are then calculated for the new clusters built. These two steps are repeated until there are no longer changes in clusters during the assignment step.

### *3.3. Reinforcement Learning*

Speaking of reinforcement learning, there are two main approaches: value-based and policy-based. The goal of the former is to maximise a function which evaluates the total reward expected to be accumulated in the future iterations starting at a particular state. Once such a value function has been learned, it is used by the agent at each step to pick such an action which is believed to maximise the value of this function. Contrary to the value-based RL approach, policy-based agents attempt to optimise the policy function directly without learning the value function. The policy function usually outputs a probability for each action the agent can select at the given state.

Deep Q-network (DQN) proposed in [53] presents the first deep value-based RL model to learn control policies directly from high-dimensional sensory input. In particular, the original DQN algorithm uses the images shown on the Atari emulator as the input and a convolution neural network - to process the image data. The role of this neural network is to approximate Q-function used to estimate the value of the action taken by an agent at a certain time step. The real reward value observed in the environment is used to calculate the target Q-function value using the Bellman Equation [107]. The neural network weights are then updated to minimise the difference between the target and current Q-function values. The Q-learning algorithm is proven to converge to an optimal value [108].

There are two main issues with using deep Q-networks. First, deep learning assumes data samples to be independent, however, the training data for the Q-network are collected by the sequence correlated states which are led out by actions chosen. Second, the collected data distributions are non-stationary, since the agent keeps learning new strategies at every iteration. To overcome these issues, two following mechanisms can be employed: experience replay and freezing the target. The latter requires constructing two neural networks with the same structure, but different weight values. One of these networks is updated online at each training step, while weights of the other one are kept frozen for a fixed amount of iterations. To calculate the value of the loss function during training, Q-function value is predicted with the first neural network, while the target is evaluated using the second one. Weights of the target network are periodically updated with weight values of the online network. Since the evaluation of the target value uses the old parameters whereas the predicted value uses the current parameters, this can break the data correlation efficiently. Moreover, this mechanism allows one to avoid policy oscillations caused by rapid changes of the Q-function. The second mechanism, experience replay, is the method which aims to break correlations between data samples by accumulating a buffer of experiences from many previous episodes and training the agent by sampling mini-batches of experiences from this buffer uniformly at random.

DQN can handle the problems with discrete low-dimensional observations and actions. However, in real world engineering applications, not only the dimensionality of both states and actions can be high, both of them are often continuous. If the number of discrete actions is finite, Q-function maximisation poses no problem, since Q-values can be computed for each action separately and then they can be simply compared to each other. However, in case the action space is continuous, solving the resulting optimization problem is non-trivial. Deep deterministic policy gradient (DDPG) has been developed specifically for dealing with environments that operate in continuous action spaces [109]. Similarly to DQN, DDPG learns the Q-function with the help of the Bellman equation. The Q-function is then used to derive and learn an optimal policy. In addition to the value-function in DDPG, the second neural network that represents the agent's policy is employed to learn a deterministic policy which for every given state of the environment returns the action that maximises the Q-function. Assuming the Q-function is differentiable with respect to the action, a gradient ascent is performed with respect to policy parameters only to find the

action that maximises the Q-value. Both DQN tricks, experience replay and freezing the target, can be also used with DDPG.

The DQN and DDPG algorithms are off-policy meaning that experiences collected in an environment are appended to a data buffer and then data from this buffer is used to train an updated new policy. In general, off-policy algorithms are more sample-efficient, however, they may be biassed due to the fact that past policies are quite different from the current ones and therefore in the old data does not suit well to calculate updates for the current policies. This may cause the algorithms to become unstable and difficult to tune [110]. State-action-reward-state-action (SARSA) algorithm [111] is an on-policy variation of DQN. It uses the action performed by the current policy to learn the Q-value. This difference is visible during the approximation of neural network parameter updates.

Another on-policy RL approach is called policy gradients. An agent trained using this method observes the total cumulative reward at the end of each training episode. If this reward is positive, the agent starts taking each action in the sequence that has led to this reward more frequently. The policy network weights are updated by minimising the negative log likelihood between the actions taken and the network outputs. The main issue with the policy gradients algorithm is the fact that, since trajectories during the training stage can deviate significantly from each other, updating the policy network weights by sampling a random action may result in a high variability in cumulative reward values and action probabilities. This may lead to the policy distribution being skewed to a non-optimal direction. In order to increase stability and reduce variance, the value function can be subtracted from the cumulative reward. This allows for evaluating the action taken in terms of the average action return. For this purpose, another neural network is built to estimate the value function. The resulting two-layered architecture is named an advantageous actor-critic (A2C), in which the value function is estimated by the critic and the policy distribution is updated by the actor in the direction pointed by the critic [112].

By employing trust region policy optimization (TRPO) algorithm, stability of the learning can be even further improved [113]. This algorithm uses average Kullback–Leibler divergence between the old and the current policies to find a region around the current policy within which the model is trusted to be an accurate estimation of the objective function. After that, a step inside this region is chosen to be an approximate minimizer of the model. Despite the fact that this method achieves consistently great performance, its implementation is extremely complicated and computationally heavy. Proximal policy optimization (PPO) algorithm reduces complexity of TRPO by limiting the impact of each action via clipping the ratio of the probability of the action under the current policy to the probability of the action under the previous policy [114]. Such a technique allows the agent to avoid having too large policy updates.

## 4. AI/ML in 5G

In this section, we discuss successful applications of AI/ML in mobile networks found in scientific literature. We focus on the recent studies published in the last five years and attempt to find use cases of employing AI/ML techniques that would benefit 5G mobile networking in high degree. We divide the use cases found into several categories based on the network component in which AI/ML is employed. For each use case, we provide information about the statistical features extracted from the data during the preprocessing stage, technical details of the learning algorithm used and modus operandi of the resulting intelligent framework. All the use cases found are summarised in Table 3.

**Table 3.** AI/ML applications in mobile networks.

| Category | Ref. | Description | Data and Features | Algorithm | Modus Operandi |
|---|---|---|---|---|---|
| Channel estimation | [75] | Estimates CSI of the whole time-frequency using the transmitted pilots | Received pilot signals transmitted in lattice formation | Supervised: CNN | Transmit pilots; train a CNN and freeze its parameters; train another CNN; in both cases, MSE between the estimated and the actual channel responses is used as the loss function |
| | [57] | Estimates CSI at the uplink for mixed ADCs massive MIMO systems | The least square channel estimations of pilot signals | Supervised: fully-connected neural network and CNN | Transmit pilots; estimate the channel with the least squares; either train one neural network for all antennas or two networks: one for high- and another for low-resolution ADCs and combine the outputs; in both cases, MSE between the estimated and the actual channel responses is used as the loss function |
| | [91] | Estimates CSI for high-dimensional signals in massive MIMO-OFDM | Received signal powers of OFDM symbols | Unsupervised: CNN-based DIP neural network | Initiate the DIP network; transmit pilots and measure the received signal; optimise the network parameters to minimise $L_2$ loss between the signal received and the output of the network given the input is a randomly chosen input tensor; use the resulting network to estimate the signal; estimate the channel using the least squares method |
| | [76] | Predicts CSI in DL based on the past UL measurements in OFDM FDD | Time-frequency UL-CSI | Supervised: CNN and GAN | Transmit UL pilots; measure the received signal; use the CNN to predict DL-CSI to minimise MSE between the real and predicted values; alternatively, train the GAN to generate UL- and DL-CSI, initialise a random vector and update it using the gradient descent so that the generated and the real UL part become similar with regards to the loss function equal to the weighted sum of the difference between the real and predicted UL CSIs and discriminator loss |
| | [28] | Maps CSI at one set of antennas and frequency band to CSI at another set of antennas and frequency band | CSI matrix for a subset of antennas in the uplink | Supervised: fully-connected neural network | Estimate channel matrix for a subset of antennas in the uplink; preprocess the data: normalise, zero-mask, flatten; train the fully-connected network to predict the channel matrix for all antennas in the downlink; normalised MSE averaged over mini-batches is used as the loss function |
| | [92] | Estimates CSI in beamspace mmWave massive MIMO | Received pilot signals | Supervised: CNN-based LDAMP neural network with DnCNN denoiser | Transmit pilots and measure the received signal; train an LDAMP network with DnCNN denoiser to minimise MSE between the real and predicted channel matrix; use the network trained to estimate CSI |
| | [77] | Recovers CSI on the BS through feedback links in OFDM | CSI matrix estimated at UE | Unsupervised: CNN-based autoencoder | Transmit pilots and measure the received signal at the UE; evaluate CSI at the downlink; train an autoencoder to minimise MSE between the real CSI matrix and its reconstruction; deploy the encoder at the UE and the decoder - at the BS to reduce CSI feedback overhead |
| Symbol detection | [58] | Recovers the transmitted symbols in OFDM | The received data of the pilot block and one data block | Supervised: fully-connected neural network | Transmit pilots and one data block, use the data received as the input to a neural network; train the network to minimise the $L_2$ loss between its output and the data transmitted; use the network trained to recover the transmitted symbols |
| | [115] | Estimates the signal transmitted given the signal received and the CSI matrix | Received signal and the CSI matrix | Supervised: neural network MMNet | Transmit a signal and measure the signal received; estimate the CSI at the current time interval; train a neural network for 1000 iterations for the 1-st subcarrier and fine-tune using only 3 training iterations per subcarrier for subsequent subcarriers; use the models trained to detect the signals received in the current time interval on the corresponding subcarriers; the training algorithm is repeated at each time interval |

**Table 3.** *Cont.*

| Category | Ref. | Description | Data and Features | Algorithm | Modus Operandi |
|---|---|---|---|---|---|
| | [116] | Estimates the signal transmitted given the signal received and the CSI matrix | Received signal and the channel state matrix | Supervised: neural network ScNet | Transmit a signal and measure the signal received; estimate the channel state at the current time interval; train an ScNet to minimise the Euclidean distance between the real signal sent and the one predicted by the neural network; use the network trained to estimate the original signal transmitted given the signal received and the CSI matrix |
| AMC | [31] | Performs AMC based on sequences of the received signals and SNR | Sequences of the signals received, SNR, noise values | Supervised: CNN | Collect received signals, SNR and noise values; train a network to predict the modulation type and noise level; replace the output layer with the new one without the noise output; fine-tune the network using only modulation types; in both cases use multi-class cross-entropy as the loss function |
| | [32] | Performs AMC using the received signal | Received signal in I/Q format, FFT magnitudes | Supervised: LSTM | Measure the received signal or retrieve averaged FFT magnitude; train an LSTM network to predict the modulation type; the loss function is softmax cross-entropy with logits |
| | [59] | Performs AMC based on manually crafted features | Amplitude var., max PSD of the amplitude, in-band spectral var. and deviation from unit circle, cumulants, SNR | Supervised: fully-connected neural network | Process the received signal; extract a specific set of manually designed features; train a neural network which predicts the modulation type; the loss function is multi-class cross-entropy between the real modulation expressed as a one-hot vector and the softmax output of the network |
| | [78] | Performs AMC using constellation diagrams | Constellation diagrams of the modulated signals | Supervised: CNN | Measure the received signal; transform the signal into constellation diagrams; train an AlexNet to predict the modulation type; softmax cross-entropy with logits is used as the loss function |
| Channel coding | [79] | Interprets end-to-end communication systems as autoencoders | A message transformed into one-hot vector | Unsupervised: autoencoder, attention CNN-based RTN | Select and transmit messages; train an autoencoder neural network to learn representations of the messages that are robust with respect to noise, fading and distortion; use the categorical cross-entropy between the input message and the reconstruction as the loss function; the trained autoencoder can be used for both encoding and decoding so that the transmitted message can be recovered with small probability of error |
| | [60] | Searches for an optimal decoding strategy for binary linear codes | PC matrix, hard-decisions vector | Reinforcement: Q-table or Q-learning with a fully-connected network | Initiate a deep neural network; receive a codeword; initiate the state as PC matrix multiplied by the hard decisions vector; perform an action by flipping a bit in the received word; evaluate the reward based on whether the codeword is decoded or not; calculate Q-function value based on Bellman equation; repeat until the terminal state is obtained, i.e., the codeword is decoded; explore the environment with either selecting a random action or an action which flips one of the incorrect bits |
| | [61] | Views the decoding problem as a classification problem | Received symbol vector after channel encoding, BPSK mapping and simulated channel noise | Supervised: fully-connected neural network, CNN, LSTM | Transmit and receive a codeword; train a fully-connected neural network, CNN or LSTM to minimise MSE between the original codeword sent and the output of the network; estimate information bits from the new received symbol using the network trained |
| | [80] | Solves the decoding problem for convolutional and LDPC codes | Received symbol vector after channel encoding, BPSK mapping and channel noise | Supervised: CNN | Transmit and receive a codeword; train a CNN to minimise MSE between the original codeword sent and the output of the network; estimate information bits from the new received symbol using the network trained |

**Table 3.** *Cont.*

| Category | Ref. | Description | Data and Features | Algorithm | Modus Operandi |
|---|---|---|---|---|---|
| | [81] | Solves the decoding problem for LDPC codes | Received symbol vector after channel encoding, BPSK mapping and channel noise | Supervised: CNN | Transmit and receive a codeword; train a CNN to minimise the weighted sum of the residual noise power (squared $L_2$ loss) and the normality test; use BP decoder to estimate transmit symbols and calculate; reconstruct the noise with the CNN trained; construct the new version of the received vector the noise reconstructed and feed it back to the BP decoder to perform another round of BP decoding |
| | [62] | Learns to communicate over an unknown channel without a reliable link | Transmitter: message sent, receiver: received signal | Reinforcement: policy gradients, supervised: fully-connected neural network, attention CNN-based RTN | Transmit messages over the channel; build the receiver network to estimate the conditional probability of the message sent given the signal received; build the transmitter using policy gradients to minimise the loss obtained at the receiver and sent back to the transmitter over an unreliable channel; train both the transmitter and the receiver iteratively to minimise the MSE between the message sent and the receiver's output |
| Beamforming | [88] | Predicts mmWave beam power based on positions of the RSU and neighbouring cars | Position of the RSU and the cars in different lanes with regards to the receiver | Supervised: linear, support vector, random forest and gradient boosting regression models | Measure and encode positions of the RSU and surrounding cars; calculate received power for each beam pair using the channel model provided; train one of the regression models mentioned to predict the maximum beam power or all beam powers based on the feature values using root MSE between the real and predicted values as the loss function |
| | [63] | Predicts achievable rate with every beamforming codeword based on the OFDM omni-received sequences | The OFDM omni-received sequences collected from several BSs | Supervised: fully-connected neural network | Receive omni uplink pilots from multiple UEs on multiple BSs; evaluate the achievable rate of every beamforming vector; train a network to minimise the MSE between the desired normalised achievable rate and the network output given the OFDM omni-received sequences collected from all the BSs; use the network trained for coordinated beamforming |
| | [64] | Predicts the best beam based on a subset of RSS measurements | RSS values from a subset of beams | Supervised: fully-connected neural network | Measure RSS values for a given subset of beams; calculate the optimal beam by measuring the angle between the receiver and the transmitter; train a neural network to retrieve the best beam based on the subset of RSS values; use the network to map this subset and to the best beam overall; the subset of beams used as the input is selected by cycling through the combinations of beams and then selecting the beam subset with the best accuracy |
| | [34] | Performs beam selection using context-awareness of the UE | GNSS and lidar data | Supervised: fully-connected and residual CNN | Measure GNSS positions, collect lidar image; train a neural network to predict the optimal beam based on the aforementioned data; use categorical cross-entropy as the loss function; use the network trained to determine the beam direction with the use of context information |
| | [90] | Calculates optimal antenna diagrams for massive MIMO | User distribution, channel information | GAN + reinforcement learning: Q-learning | Use the first generator to produce a sample of users' locations; use the second generator to produce an antenna diagram; use the discriminator to evaluate the reward in terms of the total aggregate throughput; update the discriminator using Bellman equation; repeat until the convergence criteria is met, namely, all users have the highest possible spectral efficiency values |

**Table 3.** *Cont.*

| Category | Ref. | Description | Data and Features | Algorithm | Modus Operandi |
|---|---|---|---|---|---|
| | [65] | Performs beam training based on measurements for a subset of codewords | RSS values for a subset of BS codewords and the user codewords combinations | Supervised: fully-connected neural network | Send a subset of all possible combinations of the BS codewords and the user codewords; measure the received signal power; train a network to predict the optimal beam combination based on the RSS measurements; use the network trained to select the optimal combination of the combiner and the precoder |
| Activation control | [66] | Searches for an optimal SBS activation strategy | Traffic rates, SBS on/off modes | Reinforcement: DDPG, supervised: fully-connected neural networks | Initialise networks; observe historical data rates; predict the future data rate using a traffic rate prediction network; observer SBS on/off modes; calculate continuous action using a policy network; transform the action into a set of discrete actions and select the best using a cost-estimation network; observe actual traffic rates and costs; update all the network parameters |
| | [67] | Controls BS sleeping | Traffic volumes | Reinforcement: DQN with fully-connected neural network | Initialise the agent and IPP model; observe traffic volumes; filter the traffic information with IPP model; feed the filtered observation to a neural network; calculate and scale the reward based on the number of requests served and not served; store the experience in a replay buffer; sample an experience from the buffer and update the network parameters; update the reward scaling factor; use the network trained to decide whether the BS should sleep or not at the given time interval |
| | [36] | Selects sensor power mode for indoor localisation | Sensor mode: low-power or enhanced | Reinforcement: SARSA | Initialize an RL agent; observe sensor modes; infer location using HMM model; estimate the localisation error depending on the sensor mode; update the agent's Q-function using Bellman equation; use the algorithm for continuous learning of energy-efficient location sensing |
| Power allocation | [37] | Maps positions of UEs to the optimal power allocation policies | UE positions | Supervised: fully-connected neural network | Obtain UE positions; calculate an optimal power allocation policy vector; train a network using UE positions as the input and the MSE between the network output and the optimal power allocation policy vector as the loss function; the resulting network can be used to calculate the optimal power allocation policy for a new set of UEs' positions |
| | [68] | Learns to optimise interference management | Channel coefficients | Supervised: fully-connected neural network | Measure channel coefficients; calculate optimal power allocation values using WMMSE algorithm; train a neural network to minimise MSE between the network output and the signal power given the channel coefficient values; use the network trained to estimate the power allocation policy for new channel coefficient values to reduce real-time processing |
| | [69] | Develops a framework for energy-efficient power control | Channel realisations | Supervised: fully-connected neural network | Use the BB algorithm to generate the training set containing optimal power allocations for many different channel realisations; train a neural network to minimise MSE between the network output and the optimal power allocation policy; use the model trained as an effective and low-complexity online power allocation method |
| | [93] | Selects power level to mitigate smart jamming | Estimated jamming power and all the users' SNR at the current time slot | Reinforcement: Q-learning | Initialize the agent; estimate jamming power and users' SNR at the current time slot; sample transmission power using the model; calculate the transmission utility as the weighted sum of the users' SNR values and the transmission costs; update the model using Bellman Equation |

**Table 3.** *Cont.*

| Category | Ref. | Description | Data and Features | Algorithm | Modus Operandi |
|---|---|---|---|---|---|
| | [82] | Selects optimal relay signal power on an UAV during a jamming attack | BER of the previous user messages, the jamming power received by the UAV, gains of the channels between the BSs, UAV, user and jammer | Reinforcement: DQN with CNN | Initialise the agent; estimate feature values; sample the relay power; calculate the utility based on the BER of the user message received by the serving BS, the BER of the weaker message signal received by UAV or the relay BS, and the relay cost; store the experience in a replay buffer; sample an experience from the buffer and update the network parameters |
| | [70] | Develops a distributively executed dynamic power allocation scheme | Tx power, contribution to the network objective, direct DL channel, SNR, interference and its contribution to the objective from the interferer and interfered sets | Reinforcement: DQN with fully-connected neural network | Initialize RL agents; for each agent determine interferer and interfered sets; measure feature values; sample the discrete power levels; calculate the reward for each agent based on its direct contribution to the network objective and the penalty due to its interference to all interfered neighbours; store the experience in a replay buffer; sample an experience from the buffer and update the network parameters |
| | [85] | Performs jointly dynamic channel access and power control | the channel selected, its power level, the feedback signal, the indicator of no transmission | Reinforcement: DQN with LSTM | Initialise RL agents; for each agent determine the feature values; sample the discrete power levels for the channels selected or no transmission; calculate the reward which can be either individual transmission rate, sum-rate or proportional fairness; store the experience in a replay buffer; sample an experience from the buffer and update the network parameters |
| | [71] | Studies the power allocation problem based on channel gains | Channel gains | Supervised: fully-connected neural network | Transmit pilot signals from the BS from each subcarrier; estimate the channel gains at the UE and report them to the BS; train a network to return each subcarrier power by minimising the error between the network output and the optimal power allocation vector calculated using the interior point method; use the network trained to estimate power allocation vectors for all the UEs |
| Scheduling | [39] | Formulates the resource allocation problem in LTE-LAA as a non-cooperative game | Traffic load history | Reinforcement: policy gradients, autoencoder, LSTM | Initialise the LSTM autoencoder; feed the past traffic observations for each SBS and WLAN into LSTM traffic encoders; compute the actions for all SBSs; sample an action for each SBS depending on the actions expected to be taken by other SBSs; use policy gradients algorithm to update model parameters; train the model until all coupled constraints are satisfied |
| | [117] | Infers free slots in MF-TDMA networks | Last MF-TDMA frames | Supervised: fully-connected neural network | Observe few last frames and traffic information; use a neural network to predict its probability matrix of free slots in the next frame; forward the result to the root node; receive the full schedule from the root; transmit according to the schedule; update the network weights using the total number of collisions as the loss function |
| | [95] | Formulates the cooperative localization problem as multi-agent RL | Estimated node positions; number of neighbours; local covariance matrix | Reinforcement: DQN, policy gradients | Initialise an RL agent; observe estimated node positions; calculate and execute measurement action; measure the reward function; update the agent's network parameters according to the algorithm selected; the process is terminated when each node's uncertainty falls below the threshold |

**Table 3.** *Cont.*

| Category | Ref. | Description | Data and Features | Algorithm | Modus Operandi |
|---|---|---|---|---|---|
| | [72] | Schedules HVFT application traffic | KPIs: cell congestion, number of sessions, cell efficiency | Reinforcement: DDPG with fully-connected neural network | Initialize an RL agent; measure the KPIs; sample an action; estimate the average throughput using an RF model trained on the real data; use the throughput estimation to calculate the reward; update the agent's network parameters |
| Routing | [83] | Selects AP to maximise the user throughput | RSSIs of uplink user traffic data for each AP | Reinforcement: DQN with CNN and RNN | Initialize a policy network; observe RSSIs for each AP; select the AP to which the user should be allocated using the network; evaluate the user's throughput as the reward function; store the experience in a replay buffer; sample an experience from the buffer and update the network parameters; use the network trained to select the optimal AP |
| | [99] | Optimizes routing in SDN | Traffic matrix | Reinforcement: DDPG | Initialize an RL agent; observe the traffic matrix; sample link-weights for each source-destination pair of nodes using the neural network; evaluate the mean network delay as the reward function; update the network parameters; use the network trained to perform the routing |
| | [73] | Optimizes online routing in OTNs | Statistics of the several candidate paths of all the sourcedestination pairs in the network | Reinforcement: TRPO with fully-connected neural network | Initialize an RL agent; collect statistics of the candidate paths for each source-destination pair; use the policy network to sample an end-to-end path from the list of candidate paths that connect the source and the destination given and provide the bandwidth required; evaluate the bandwidth of the current traffic demand; update the network parameters; use the network trained to route new source–destination traffic |
| Localization | [84] | Estimates the travel time | GPS trajectory points, travel distance, information about weather, driver and time | Supervised: CNN+ LSTM | Initialise attribute, spatio-temporal and multi-task learning components of the network; embed information about weather, driver and time into a low-dimensional feature vector; feed this vector and the travel distance to the attribute component; feed the output of the attribute network to the spatio-temporal learning network; feed the outputs of the attribute and spatio-temporal learning components to the multi-task learning network; train the entire framework end-to-end in a supervised way by minimising the error between the travel time of the entire path and the output of the multi-task network |
| Slicing | [118] | Predicts REVA metric for RAN slice broker to provision network slices with RAN resource | REVA metric historical values | Supervised: LSTM | Calculate REVA for last time intervals; train a network using these values to minimise the loss function between the REVA predictions and the actual values; use the network trained to estimate future REVA values; use these estimations coupled with CSI to derive wireless link throughput and RAN resources required for each slice |
| | [86] | Proposes proactive dynamic network slicing scheme | Network traffic historical data | Supervised: GRU | Collect the standardised traffic load data; train multiple neural networks to predict future network loads for each slice; select the best model for each slice; use the data predictions to adjust and schedule traffic in the future; periodically retrain the models |
| | [96] | Formulates the radio resource allocation problem as an MDP | Number of arrived packets in each slice within specific time window | Reinforcement: DQN | Initialize an RL agent; observe numbers of arrived packets in the network slices under consideration; sample the bandwidth allocation vector using the Q-network; measure the weighted sum of SE and QoE as the reward function; store the experience in a replay buffer; sample an experience from the buffer and update the agent network parameters |

**Table 3.** *Cont.*

| Category | Ref. | Description | Data and Features | Algorithm | Modus Operandi |
|---|---|---|---|---|---|
| | [100] | Formulates the network slice resource allocation as an MDP | Numbers of users using different slices | Reinforcement: PPO | Initialise an RL agent; observe numbers of users using the network slices under consideration; sample the bandwidth allocation policy using the agent's policy network; measure the network throughput; update the agent network parameters |
| | [97] | Formulates the network slicing resource allocation problem at the network edge as an MDP | Allocated resources, resources in use, the serving node, the task priority, the resources required, holding time | Reinforcement: DQN | Initialize an RL agent; observe values of the features; select the fog node for the task or reject the task using the Q-network; calculate the reward function; store the experience in a replay buffer; sample an experience from the buffer and update the agent network parameters |
| | [98] | Proposes a dynamic resource allocation scheme for radio access network slicing | Frequency-time blocks, CPU usage, and transmit power available at the time slot | Reinforcement: DQN | Initialise an RL agent; observe available resources at the current time slot; select the requests to serve using the Q-network; calculate the weighted sum of requests satisfied which acts as the reward function; store the experience in a replay buffer; sample an experience from the buffer and update the agent network parameters |
| Caching | [87] | Proposes a proactive caching framework for CRANs | Content request time, week, gender, occupation, age, and device type | Supervised: ECN | Collect content request data; train an ECN model; use the ECN to predict the distribution of content requests and user mobility patterns; determine which content to cache based on the content request percentage; based on the content request distribution, cluster and sample the content request distributions to calculate the percentage of each content; use it to select the contents to cache |
| | [119] | Learns content cache priorities | Historical sequence of video content requests | Supervised: LSTM | Observe sequence of video content requests, train a network to predict cache priority values; use the network trained to decide which content should be evicted from the cache when a new content request in one base station arrives |
| | [89] | Predicts content popularity class in evolved packet core | Historical traffic volumes of each content type | Supervised: sparse autoencoder | Collect historical data on content popularity; train a neural network to predict content popularity using the historical patterns; train each autoencoder in the network in unsupervised way to minimise the reconstruction error; fine-tune the network to minimise the prediction error; use the network trained to evaluate content popularity |
| | [94] | Finds optimal caching policy | local and global user request profiles | Reinforcement: Q-learning | Initialize the agent; observe user request profiles; calculate binary caching policy vector; calculate the reward based on the cost of refreshing the cache contents, the operational cost and the cost of mismatch between the caching action vector and the global popularity profile; update the agent parameters using Bellman equation |
| Security | [41] | Classifies network flows, classifies attack symptoms | Network traffic flow features, timestamps, attack class | Supervised: fully-connected neural network, DBN, LSTM | Train one neural network to classify feature vectors extracted from network flows as either normal or attack symptoms; use the feature vector, timestamp and symptom type as the input to another neural network; train the second neural network to classify symptom sequences as attacks |
| | [43] | Detects intrusion in WiFi networks | MAC layer, radio and general frame information | Unsupervised: autoencoder, supervised: fully-connected neural network | Extract and preprocess features; train an autoencoder to minimise the reconstruction error; use the output of the autoencoder pre trained as the input to a fully-connected network; train the second network to minimise the categorical cross-entropy between its output and the attack class label |

**Table 3.** *Cont.*

| Category | Ref. | Description | Data and Features | Algorithm | Modus Operandi |
|---|---|---|---|---|---|
| | [45] | Looks for an optimal CNN architecture and detects network intrusions | Network traffic flow features | Supervised: RNN, CNN | Collect traffic flows from backhaul and core network links; transform feature vectors into images; select an optimal CNN architecture using RNN; train the resulting CNN model; use the model trained to classify network traffic flows |
| | [47] | Classifies jamming attacks | Node state information | Supervised: fully-connected neural network, SVM | Collect traffic; extract features; train a neural network to classify samples as either normal or corresponding to a particular class of jamming; feed the samples classified as normal to an SVM; train the SVM to classify the samples as either normal or malicious; use the model trained to identify and classify jammers |
| | [74] | Detects and classifies jamming attacks | Subcarrier spacing, symbol time, subcarrier length, cyclic prefix length, average received power, threshold, average signal power, average noise power, and SNR | Supervised: linear regression, fully-connected neural network, k-NN, decision tree, random forest, naive Bayes | Extract features from the signals; train a model to detect and/or classify samples as either normal or corresponding to a particular class of jamming; use the model trained to identify and classify jammers |

One potential application of AI/ML in 5G is improving channel state information (CSI) estimation procedure. For example, studies [28,76] suggest to reduce the pilot training and feedback overhead in an orthogonal frequency-division multiplexing (OFDM) system by training an AI/ML model to calculate CSI in the downlink (DL) based on the past uplink (UL) measurements. The reasoning behind the idea is that since DL and UL signals are propagated in the same environment, the former can be estimated with some level of accuracy given the latter. Alternatively, AI/ML can be used to reduce dimensionality of the CSI matrices and therefore decrease the transmission overhead as it is proposed in [77]. In [57], AI/ML is used for estimating UL channels for massive MIMO systems with mixed analog-to-digital converters (ADCs).

AI/ML is also proposed to be deployed for automatic modulation classification (AMC). This allows transceivers to use the channel conditions to select the modulation coding scheme (MCS) without the need for a feedback channel between the transmitter and the receiver. For example, study [59] aims to develop a robust AMC algorithm with the help of a fully-connected neural network. Input features used for the prediction are extracted from the received signal and they include amplitude variance, maximum value of the power spectral density (PSD) of the normalised centred-instantaneous amplitude, in-band spectral variation and deviation from unit circle, and higher order statistics including cumulants and estimated signal-to-noise (SNR) values. In [31], the deep learning approach is used to extract features from symbol sequences automatically. Similarly, study [78] employs a deep learning architecture, namely AlexNet [120], to process the signal samples and retrieve the MCS used. In [32], another data-driven model for automatic modulation classification is proposed. The data input is either the received signal represented in I/Q format which includes two components: in-phase and quadrature or the averaged fast Fourier transform (FFT) magnitude information.

Another interesting application of AI/ML in mobile networking is related to channel coding. For example, study [79] interprets the entire end-to-end communication system as an autoencoder which learns robust representation of the messages transmitted with respect to the channel distortions allowing for its recovery with a small probability of error. The study also proposes using radio transformer network (RTN) which is a modified attention-based spatial transformer applied in computer vision [121]. Further studies [61,80] use AI/ML for the signal decoding. To generate training samples, the authors of both studies pick an information vector from codebook set randomly and then perform channel

encoding, apply the binary phase shift keying (BPSK) mapping and simulate channel noise in order to obtain the received vector. After that, the received signal acts as the input to the neural network whereas the decoded signal as the output. Study [60] uses RL to find effective decoding strategies for binary linear codes. The study focuses on bit-flipping (BF) decoding which is based on constructing such a metric that allows the decoder to rank the bits based on their reliability depending on the code constraints given. As a rule, BF uses the hard-decision output and iteratively looks for the bit that, after flipping it, would maximally reduce the number of currently violated parity-check (PC) equations. The decoding method proposed is mapped to an MDP and it is solved with Q-learning. In [81], a decoding iterative neural network based architecture for linear codes is proposed. It concatenates the CNN trained with a belief propagation (BP) decoder and iterates between them. A block of uniformly distributed bits is encoded to a binary codeword through a linear channel encoder. The codeword is then mapped to a symbol vector through the BPSK modulation which in turn is passed through a channel with additive Gaussian noise. A BP decoder is then used to decode the transmitted information bit vector from the received vector to estimate transmit symbols. A CNN is trained to estimate the channel noise. The procedure is iterative: the new version of the received vector is constructed using the noise estimated with the CNN and fed back to the BP decoder and another round of BP decoding is performed.

AI/ML can also be employed for beamforming. For example, study [64] aims to improve the initial access (IA) procedure with the help of machine learning. As a rule, the IA includes two main components: beam sweeping during which the received signal strength (RSS) is measured for each beam, and beam selection which essentially is picking an optimal beam for the transmitter-receiver pair given. Usually the beam sweeping procedure takes much longer time than the beam selection, and therefore the total IA time can be improved significantly by reducing the beam sweep time via utilising fewer beams. The study attempts to solve this problem with deep learning. In particular, it proposes to pick a small subset of the beams available and measure RSS only for the beams in this subset. After that, the best beam overall is selected based only on these few RSS values measured with the help of an AI/ML model. Study [63] proposes to train an AI/ML model to find the best beam based on uplink training pilots collected at several neighbouring BSs, whereas the AI/ML-driven beam selection framework proposed in [34] uses the data from the Global Navigation Satellite System (GNSS), lidars and cameras.

Employing AI/ML models can also allow for more energy-efficient power allocation. Despite there are several existing optimization methods for solving the power allocation problem such as weighted minimum mean squared error (WMMSE) algorithm, using neural networks allows for learning solution approximations which may be beneficial at the inference stage as it requires less computing resources [68,71]. Furthermore, study [37] proposes an interesting AI/ML-based power allocation scheme which calculates the optimal power allocation policies based on the UE positions. In [67], a data-driven algorithm for dynamic BS sleeping control is proposed. The problem is formulated as an MDP and it is solved with an RL approach. Each state is traffic belief calculated with the help of a interrupted Poisson process (IPP) [122] using the Baum–Welch algorithm [123], each action is either sleep or not, the agent is awarded for each served request and penalised for each queued, retransmitted and failed one. In [82], an unmanned aerial vehicle (UAV) aided cellular framework against jamming is proposed. The study focuses on the scenario in which a jammer is located close to the current serving BS of the user and sends jamming signals to prevent the BS from receiving messages from the user. This scheme enables an UAV to optimise its relay power to help the cellular system to resist jamming without knowing the cellular topology, the message generation model, the server computation model, and the jamming model. The UAV chooses its relay power in order to maximise the utility function which depends on the bit error rate (BER) of the user message received by the serving BS, the BER of the weaker message signal received by UAV or the relay BS, and the relay cost.

Another potential AI/ML application in 5G is dynamic resource allocation to radio access network (RAN) slices. The studies devoted to this problem can be divided into two main categories. The studies in the first category use the supervised learning approach to adjust the resource distribution by predicting the usage of each RAN slice [86,118]. The alternative approach involves formulating the resource allocation problem as a Markov decision process (MDP) and solving it with a reinforcement learning algorithm [96–98,100]. The former approach would most likely be easier to execute assuming a labeled dataset is available. However, the resulting prediction model would not be able to take into account the effect of the resource allocation decisions on the network environment. The RL approach is therefore probably more suited for such a dynamically complicated task even though the model training in this case requires implementing a simulation software and shaping an efficient reward function as mentioned in Section 3.

AI/ML can also be a good alternative for intelligent scheduling. For instance, study [39] formulates the resource allocation problem in cellular long term evolution (LTE) communications in unlicensed spectrum using licensed assisted access LTE (LTE-LAA) as a non-cooperative game. The players in this game are the small BSs (SBSs) which learn autonomously which unlicensed channels to use. The framework proposed by the authors solves this game problem using policy-based reinforcement learning with the LSTM-based model as the policy function. The solution allows the SBSs to learn each channel access probability taking into account future environmental changes caused by LTE-LAA traffic loads and additional WLAN transmissions. The authors in [117] propose a scheduler to infer the free slots in a multiple frequency time division multiple access (MF-TDMA) network to avoid congestion and high packet loss. Study [72] focuses on the problem of scheduling high volume flexible time (HVFT) application traffic in future computer networks. A common property of HVFT applications is that the mobile network operator must serve a large volume of traffic during the day but has significant flexibility in scheduling this traffic, i.e., these applications can tolerate delays on the order of a few hours up-to a day. The problem is formulated as an MDP and it is solved with DDPG.

In addition, AI/ML can also be a promising feature for routing optimization. For example, a framework for smart wireless network management is proposed in [83]. In particular, the study focuses on the scenario when a user moves in an area with multiple access-points (APs), and only associates with one AP at a time. APs can monitor the received signal strength indicator (RSSI) of uplink user traffic data. The target is to find an optimal access policy for the user while maximising the user's throughput. The experiments in the study are conducted using a real-time software-defined networking (SDN) based heterogeneous wireless network testbed [124]. In [73], a framework for online routing in optical transport networks (OTNs) is proposed. The authors consider the scenario in which an agent operates over a logical topology composed of reconfigurable optical add-drop multiplexer (ROADM) nodes and some predefined lightpaths connecting them. The role of the agent is to route incoming traffic demands through particular sequences of lightpaths. The problem is modelled as an MDP and it is solved with TRPO.

Furthermore, cognitive radio capabilities empowered by machine learning allow for performing spectrum awareness and spectrum sharing. For example, in study [9], an AI/ML model at an environmental sensing capability (ESC) station detects citizens broadband radio service (CBRS) as an incumbent user. If such service is not observed in the channel under consideration, the ESC allows a BS to communicate to UEs. Otherwise, the BS cannot use this channel and it is reconfigured to vacate this channel to avoid interference with transmissions of the incumbent user.

Finally, security of mobile networks can be enhanced significantly by employing state-of-the-art AI/ML models for anomaly detection [41,45]. The resulting intrusion detection systems (IDSs) can be used to classify the vectors of features extracted from network traffic flows as either benign or malicious. In the RAN domain, the main security application of AI/ML is jamming detection [47,74]. In this case, an AI/ML model is trained to distinguish between the signals transmitted by normal devices and the one broadcast by the jammer.

The input features may include the location of the device sending the signal, SNR value, traffic volume, frequency of transmissions and several others.

## 5. Attacks

In this section, we summarise both white-box and black-box attack algorithms against several AI/ML models summarised in the previous section, namely k-NN, decision tree ensembles and deep neural networks.

### *5.1. Attacks against k-NN*

In the most straight-forward white-box attack scenario against a k-NN model with $k$ equal to one and Euclidean distance being the distance metric, in order to transform a new unlabeled sample in such a way that it is misclassified as a given target class, it is sufficient for an adversary to select a sample from the target class, and move the new sample towards the target sample along the straight line joining them. Assuming that all data samples are distinct, as the amount of perturbation increases, the perturbed sample would eventually find itself with the target sample as its nearest neighbour [125]. In the case, the target class is not given, i.e., the adversary aims to transform the unlabeled sample in such a way that it is misclassified, no matter which class the perturbed sample belongs to. As mentioned above, such a straight-forward attack is white-box as the adversary would require to know both the training dataset and the distance metric. This attack vector acts as the basis for a naive attack proposed in [126].

When $k = 1$ this strategy is able to easily find an optimal adversarial example. However, it is unclear how to do this efficiently for bigger values of $k$. There is no guarantee of an optimal solution when using the algorithm proposed above for each set of $k$ training samples. Furthermore, it is extremely inefficient and the solution complexity increases exponentially with $k$. To resolve these issues, study [126] proposes the following attack algorithm. First, the nearest neighbour $\hat{x}$ from any class $\hat{y}$ other than $y$ is found. This sample is added to an empty set $S$. After that, out of all samples with class $\hat{y}$, the nearest sample to the mean of $S$ is found and added to $S$. This step is repeated until $|S| = \left\lceil \frac{k}{2} \right\rceil$. Finally, $x$ is moved towards the mean of $S$ until the classifier's prediction differs from $y$. It is worth noticing that the attack is non-targeted, however it can be transformed into the targeted one by selecting $\hat{x}$ as the nearest to $x$ which has the given target class.

Authors in [126] also propose a gradient-based attack. First, a target label $\hat{y}$ is selected as the class which minimises the distance from an original sample $x$ to the mean of all samples of some class $y' \neq y$, where $y$ is the class label of $x$. After that, $m$ samples from the same class $\hat{y}$ that are closest to $x$ are picked. Next, $x$ is moved further from the samples with the original class $y$ and closer to the ones with the target class $\hat{y}$ using a gradient-based optimization algorithm. In other words, to craft such an adversarial perturbation $\delta$, the following optimization problem should be solved:

$$\begin{aligned} &\min_{\delta} \sum_{i=1}^{m} \omega_i \sigma(||\bar{x}_i - (x + \delta)|| - \eta) \\ &\text{subject to } ||\delta|| \leq \epsilon, \end{aligned} \tag{1}$$

where $\bar{x}_1, \ldots, \bar{x}_m$ are the $m$ training samples selected by the algorithm, $\omega_i = 1$ if the label of $\bar{x}_i$ is $\hat{y}$, otherwise $\omega_i = -1$, $\sigma$ is a sigmoid $\sigma(z) = 1/(1 + e^{-\alpha z})$ in which "steepness" of the function is controlled with the help of hyperparameter $\alpha$, $\eta$ is the average distance between a sample in the training dataset and its $k$-th nearest neighbour, and $\epsilon$ is the maximal perturbation size. This attack is non-targeted, but it can be transformed to the targeted one by picking a certain target class $\hat{y}$.

Another gradient-based attack against k-NN is proposed in [127]. As in the previous case, it starts with generating a subset of $m$ guide samples $\bar{X} = \{\bar{x}_1, \ldots, \bar{x}_m\}$. The first half of these samples is generated in the same manner as in the previous attack case:

$$\begin{aligned} &\{\bar{x}_1, \ldots, \bar{x}_{m/2}\} = \arg\min_{X'} \sum_{x' \in X'} ||x' - x|| \\ &\text{subject to } \forall x' \in X', f(x') = y', y' \neq y. \end{aligned} \tag{2}$$

The second half is generated as follows:

$$\begin{aligned} &\{\bar{x}_{m/2+1}, \ldots, \bar{x}_m\} = \arg\min_{X'} \sum_{x' \in X'} ||x' - x|| \\ &\text{subject to } \forall x' \in X', f(x') = y. \end{aligned} \tag{3}$$

Once the guide samples have been found, the following optimization problem is supposed to be solved by the adversary in order to generate an adversarial perturbation $\delta$:

$$\min_{\delta} \sum_{i=1}^{m} \max\left\{\omega_i\left(||\bar{x}_i - (x + \delta)|| - \eta^2\right) + \Delta, 0\right\} + c||\delta||^2, \tag{4}$$

where $\Delta$ a small constant greater than zero, e.g., $1 \times 10^{-5}$, $c$ is a balancing constant, and other notations are the same as in (1). To find $c$ a binary search is used: in case of the successful attack, $c$ is increased and, in the opposite case, $c$ is decreased. The attack is executed multiple times: one for each training sample nearest to $x$ whose class is different from $y$.

Study [128] proposes an adversarial example attack against k-NN algorithm via solving the following optimization problem with regards to $\lambda$:

$$\max_{\lambda \geq 0}\left\{-\frac{1}{2}\lambda^T A A^T \lambda - \lambda^T b\right\}, \tag{5}$$

where matrix $A = \{a_i\}$ and vector $b = \{b_i\}$. If $f(\bar{x}_i) = y$, $a_i = \hat{x} - \bar{x}_i$ and $b_i = \frac{||x - \bar{x}_i||^2 - ||x - \hat{x}||^2}{2}$, otherwise they are equal to zeros. The adversarial perturbation can then be crafted as $\delta = A^T \lambda$. This optimization problem can be solved via a greedy coordinate ascent algorithm. The solving time can be decreased by applying a screening rule to remove variables in each such problem.

A region-based attack against a k-NN classifier is proposed in [129]. The attack is based on $(s, m)$-decomposition which is a partition of the training samples into convex polyhedra $P_1, \ldots, P_s$. Each of these polyhedras can be defined by $m$ linear constraints. If there is such an $(s, m)$-decomposition exists for a classifier $f$ that $f(x)$ is constant $\forall \bar{x} \in P_i \; \forall i$, then classifier $f$ is $(s, m)$-decomposable. Since k-NN is $(s, m)$-decomposable, an adversarial example for $x$ can be found by first determining all polyhedra $P_i$ with the corresponding class label $\hat{y}_i$ such that $f(x) \neq \hat{y}_i$. After that, the following minimization problem is supposed to be solved by the adversary:

$$\min_{i: f(x) \neq \hat{y}_i} \min_{\hat{x} \in P_i} ||x - \hat{x}||, \tag{6}$$

where $P_i$ is described by $m$ linear constraints and the norm objective is convex. Thus, each inner minimization problem in (6) can be solved separately via solving a convex program. Solving the inner problem results in candidates $\hat{x}^i \in P_i \; \forall i$. Taking the outer minimum over $i$ with $f(x) \neq \hat{y}_i$ leads to the optimal adversarial example $\arg\min_{\hat{x}^i} ||x - \hat{x}^i||$. When $k = 1$, this is efficiently solvable, but, unfortunately, for $k > 1$, this attack does not scale well. The attack approach proposed is similar to the one proposed in [128].

In [130], an adversarial example attack against k-NN classifiers based on higher-order Voronoi diagrams is proposed. Assuming $L_i^{(k)}$ is a subset of $k$ samples from the training dataset $X$, order-k Voronoi cell associated with $L_i^{(k)}$ includes the points from that are closer

to $L_i^{(k)}$ than any other $k$ points of $X$. Voronoi facet is a boundary of an order-k Voronoi cell. In the algorithm, finding the set of facets that comprise an unvisited cell can be carried out by finding bisectors which are active with respect to this cell. A bisector between two samples is a set of points that are equidistant from those samples. A bisector is active with respect to a cell if it includes a facet of this cell. It is also worth noticing that the distance between $x$ and a facet can be calculated by solving (5).

All the attacks described in this section assume the adversary has perfect knowledge of the input sample and its label. However, it is hard to classify the attacks described in this section as either white-box or black-box ones as k-NN algorithm does not really have a model as a new unlabeled sample is compared to all the samples in the training dataset, i.e., essentially the "model" consists of all the training samples. On the ond hand, since the attacks presented assume the adversary has knowledge of the dataset, the attacks are rather white-box. On the other hand, if the adversary manages to obtain a substitute of the training dataset or its subset, the attack becomes applicable to a black-box scenario.

### *5.2. Attacks against Tree Ensembles*

The framework proposed in [129] can also be applied to tree-based classifiers since any decision tree is $(L, D)$-decomposable where $D$ is the tree depth and $L$ is the number of the tree leaves. If a classifier is an ensemble of $T$ trees, then it is $(L^T, DT)$-decomposable assuming that each tree in the ensemble has depth $D$ and $L$ leaves. Therefore, the region-based attack is based on solving optimization problem (6). In [131], a linear attack against a single decision tree is proposed. The main idea of the attack algorithm is to determine such a box for each leaf node that any example in this box is labelled according to this leaf. The node $i$'s box can be described as the Cartesian product $B^i = (l_1^i, r_1^i] \times \ldots \times (l_d^i, r_d^i]$ of $d$ intervals on the real line. Here $d$ is the dimension of the samples. By definition, the box of the root node is defined as $(-\infty, +\infty) \times \ldots \times (-\infty, +\infty)$. Boxes of an internal node $i$ children are calculated by splitting one of the box's intervals. If $p$ and $q$ are the $i$-th node's child on the left and on the right, respectively, their boxes $B^p = (l_1^p, r_1^p] \times \ldots \times (l_d^p, r_d^p]$ and $B^q = (l_1^q, r_1^q] \times \ldots \times (l_d^q, r_d^q]$ can be found as follows:

$$\begin{aligned}(l_t^p, r_t^p] &= \begin{cases}(l_t^i, r_t^i], \text{ if } t \neq t_i, \\ (l_t^i, \min\{r_t^i, \eta_i\}], \text{ if } t = t_i\end{cases} \\ (l_t^q, r_t^q] &= \begin{cases}(l_t^i, r_t^i], \text{ if } t \neq t_i, \\ (\max\{l_t^i, \eta_i\}, r_t^i], \text{ if } t = t_i.\end{cases}\end{aligned} \tag{7}$$

Once the boxes for internal nodes have been computed, the boxes for leaf nodes can also be obtained using (7). Once the box for each leaf node has been computed, the minimal perturbation which changes a new unlabeled sample $x$ to move towards a leaf node $i$ can be found as follows:

$$\delta(x, B^i) = \begin{cases}0, \text{ if } \tilde{x} \in (l_t^i, r_t^i], \\ \tilde{x} - r_t^i, \text{ if } \tilde{x} > r_t^i, \\ l_t^i - \tilde{x}, \text{ if } \tilde{x} < l_t^i.\end{cases} \tag{8}$$

After that, the minimal perturbation is supposed to be computed as $\tilde{\delta} = \min_{i:v_i \neq y} ||\delta(x, B^i)||$. Here $v_i$ is the label of leaf node $i$. Such a perturbation can be found by testing boxes $B^i$ for all the leaves. After that, the minimal perturbation is selected. This algorithm is linear-time and it allows for exact verification of a single decision tree robustness. Unfortunately, finding an exact optimal adversarial perturbation for an ensemble of trees requires exponential time and it is not scalable when the size of the ensemble and the depth of the trees grow [132].

In [133], a simple attack algorithm against a decision tree classifier is proposed. It follows the procedure described in the previous section. In particular, to find an adversarial sample, given a sample and a tree, an adversary simply searches for leaves with a class label that is different from the class of the unperturbed sample in the neighbourhood of the

leaf corresponding to the decision tree's original prediction. After that, the path from the original leaf to the adversarial leaf is determined. Next, the sample is perturbed according to the conditions on this path in order to force the decision tree to misclassify the sample as the one belonging to the adversarial class.

The shortest path based attack algorithm against a tree-based ensemble is proposed in [134]. The algorithm also focuses on attacking a binary tree-based classifier. Without loss of generality it assumes that an adversary attempts to flip the label for a new sample $x$ labelled by the ensemble as 1. The shortest path in the algorithm is defined as the one requiring the minimum number of changes over features, i.e., minimising $||x - \tilde{x}||$ to change the decision of a classification tree. The search for the shortest path starts by finding all internal nodes of tree $f_i$ of the ensemble for which $f_i(x) > 0$. After that, all the paths from each node $n$ of the tree to leaf $-1$ are found excluding the paths which include other node $n' \neq n$. The shortest path $P_i$ is selected from the paths needed to be modified to flip the label of $x$. This algorithm essentially manipulates a vote casted by an individual decision tree via a local solution. For a tree ensemble, more steps are needed to find the features of $x$ which are supposed to be modified. For this reason, a weight is assigned to each selected feature in the shortest path. The first feature at a parent node is assumed to be the most influential over a decision whereas impacts of other features in the shortest path are significantly smaller. It is worth noting that there may be multiple shortest paths with the same length. To avoid the situation when the same feature is overweighted because of this, only one shortest path is selected randomly. Once an optimal feature $x_k$ has been found, the corresponding features in the shortest path are supposed to be removed according to the following rules: if length of $P_i = 1$, delete $P_i$; if length of $P_i > 1$ and $x_k \in P_{ij}$, delete $x_k$ from $P_{ij}$ if length of $P_i > 1$ and $x_k \notin P_{ij}$, delete $P_{ij}$, where $P_{ij}$ is the $j$-th path in the shortest path $P_i$ in case there are several paths of the same length. In the black-box settings, i.e., when the attacker does not know the original model parameters, the search for adversarial examples can be carried out for a substitute set of trees.

In [135], an iterative leaf tuple attack approach is proposed. The algorithm starts with a sample $\tilde{x}$ which is already adversarial, i.e., $f(\tilde{x}) \neq y$. At each step, a new sample $\tilde{x}'$ which has the minimum distance to $x$ is selected within a small neighborhood around $\tilde{x}$. The algorithm stops when $\tilde{x}'$ does not provide a smaller perturbation than $\tilde{x}$. The algorithm uses the following notations. $C(x) = (i^{(1)}(x), \ldots, i^{(K)}(x))$ denotes the index tuple of $K$ prediction leaves from input $x$. As previously, $B^i = (l_1^i, r_1^i] \times \ldots \times (l_d^i, r_d^i]$ is the box of boundaries around the $i$-th node, such that each sample inside this box falls into this node. $B(C(x)) = \bigcap_{i \in C(x)} B^i = \bigcap_{i \in C(x)} (l_1^i, r_1^i] \times \ldots \times \bigcap_{i \in C(x)} (l_d^i, r_d^i]$ denotes the Cartesian product of the intersection of $K$ such bounding boxes in the ensemble. If $\tilde{x} \in B(\tilde{C})$ is the sample which minimizes the distance between $\tilde{C}$ and $x$ the indexes of trees that bound $\tilde{x}$ are denoted as $T_{bound}(\tilde{C}) = \{t |\ \text{OnEdge}(\tilde{x}, B^{\tilde{C}^{(t)}}) \forall t \in \{1, \ldots, K\}\}$, where $\text{OnEdge}(x, B)$ is equal to 1 if $x$ corresponds to the right or the left bound of $B$ for at least one dimension. The attack proposed is white-box as the adversary is required to have perfect knowledge of all the trees in the ensemble.

Study [136] proposes a simple black-box cube attack against an ensemble of boosted trees. The attack algorithm proposed aims to craft an adversarial perturbation $\delta : ||\delta|| \leq \epsilon$ against binary tree-based classifier $f(x)$, i.e., label $y = 1$ if $f(x) > 0$ and $y = -1$ if $f(x) \leq 0$. Thus, an unlabeled sample $x$ with label $y$ is classified correctly if $yf(x) > 0$. Projecting a perturbed sample $\tilde{x} + \delta$ onto ball $B(x, \epsilon)$ means that if $\tilde{x} + \delta$ is outside of that ball it is moved to the closest point inside the ball.

### *5.3. White-Box Attacks against Neural Networks*

The last but not the least group of AI/ML algorithms against which adversarial example attacks are common consists of neural network based models. It is worth noticing that the vast majority of the research in the domain of adversarial machine learning focuses on attacking deep learning algorithms [19,20,137,138]. For this reason, we divide those into

three parts: white-box, score-based and decision-based attacks. In this subsection, only white-box attack algorithms are presented.

In distinction to the mentioned above algorithms which are discrete and non-differentiable, a deep neural network most of the time has a differentiable loss function and uses a gradient-based optimizer during the training. This enables gradient-based adversarial example generation by modifying an input sample in the direction of the loss function gradient [137]. In case an adversary aims to transform a new unlabeled sample in such a way that it belongs to a given target class, the gradient step is computed in the direction of the negative gradient with respect to the target class [138]. The adversarial perturbation $\delta$ can therefore be crafted as follows:

$$\delta = -\epsilon \operatorname{sign}(\nabla_x J(x, \hat{y})), \tag{9}$$

where $\epsilon$ is the perturbation size, $J$ is the network loss function, $x$ is the new sample, and $\hat{y}$ is the target class. The adversarial example $\tilde{x}$ as previously is calculated as $\tilde{x} = x + \delta$. In case, the adversary aims to modify the new sample in such a way that its class is simply different from the real one, the perturbation can be crafted as follows:

$$\delta = \epsilon \operatorname{sign}(\nabla_x J(x, y)), \tag{10}$$

where $y$ is the label of the original input sample $x$. In any case, the adversary would require full access to the model parameters and the loss function to calculate the gradient $\nabla_x J(x, \hat{y})$ to carry out these white-box attacks.

Another gradient-based attack named basic iterative method (BIM) is introduced in [138]. It extends the one-shot fast gradient sign method (FGSM) described above by applying it more than one time with some step size below $\epsilon$. After each step, the resulting value is clipped to guarantee that it is in an $\epsilon$-neighbourhood of the original input. A projected gradient descent (PGD) attack against neural network classifiers based on the BIM algorithm is proposed in [139]. It starts from a random perturbation in the $\epsilon$-ball around a sample $x$. After that, a gradient step is taken in the direction of greatest loss. If necessary, the resulting perturbation is projected into the ball to satisfy the budget restrictions. Study [140] proposes momentum iterative methods (MIMs) to improve BIM and PGD algorithms mentioned above. This method accumulates a velocity vector in the gradient direction of the loss function found at each algorithm iteration. This allows the gradient descent algorithm to avoid poor local minimums or maximums which in theory should lead to more efficient adversarial perturbation being generated.

In [19], an adversarial perturbation for a neural network is generated by solving the following problem with the help of limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) nonlinear gradient-based numerical optimization algorithm:

$$\begin{aligned} &\min_{\delta} c||\delta|| + J(x + \delta, \hat{y}) \\ &\text{subject to } x + \delta \text{ is feasible,} \end{aligned} \tag{11}$$

where $\hat{y}$ is the attack target class label and line search is used to find the minimum constant $c > 0$. The line search approach first finds a descent direction along which the objective function is reduced and then computes a step size that determines how far $c$ should move along that direction.

Study [141] proposes an adversarial example attack targeting internal representations of the samples. In particular, it focuses on solving the following optimization problem:

$$\begin{aligned} &\min_{\delta} ||\phi_k(x + \delta) - \phi_k(\hat{x})|| \\ &\text{subject to } ||\delta|| < \epsilon, \end{aligned} \tag{12}$$

where $\hat{x}$ is the target sample, $\phi_k$ is the mapping from $x$ to its internal representation at the $k$-th layer and $\epsilon$ is as previously the maximum perturbation size. This problem can also

be solved with the help of the L-BFGS algorithm for each $k$ with the loss function $J$ being equal to $||\phi_k(x+\delta) - \phi_k(\hat{x})||$.

In [142], an algorithm to compute adversarial examples based on an iterative linearization of the classifier is proposed. The algorithm generates minimal perturbations that are sufficient to change classification labels. Specifically, at each iteration, classifier $f$ is linearized around the current point $x_i$ and the minimal perturbation of the linearized classifier is computed as

$$\begin{aligned} &\arg\min_{\delta_i} ||\delta_i|| \\ &\text{subject to } f(x_i) + \nabla f(x_i)\delta_i = 0 \end{aligned} \tag{13}$$

Once the perturbation $\delta_i$ at iteration i of the algorithm has been computed, $x_{i+1}$ is updated as $x_i + \delta_i$. The algorithm stops when $x_{i+1}$ changes the label $f(x)$ returned by the classifier for sample $x$.

A non-targeted attack which attempts to perform gradient descent to decrease the probability $f(x)_y$ of the original class $y = f(x)$ is proposed in [143]. Assuming that the number of classes is $K$, the descent step size $\alpha$ is determined as follows. When $f(\tilde{x})_y$ is larger than $1/K$, which is the case if $f(\tilde{x}) = y$, then $\alpha = f(\tilde{x})_y - 1/K$. Once $f(\tilde{x})_y$ approaches or falls below $1/K$, $\alpha = \eta||x||||\nabla f(\tilde{x})_y||$, where the hyperparameter $\eta$ defines the level of aggressiveness with which the gradient descent tries to minimise the probability of class $y$.

Study [20] crafts adversarial samples via Jacobian-based saliency map attack (JSMA) algorithm. The approach proposed constructs an adversarial saliency map by evaluating forward derivatives of the network. The purpose of this map is to identify the set of input features which the adversary has to perturb in order to achieve its goal. If the adversary wants to perturb a sample $x$ in such way that it is assigned to a target class $\hat{y} \neq y$, the probability of the target class $\hat{y}$ returned by $f$, i.e., $f(x)_{\hat{y}}$, should be increased, At the same time, the probabilities $f(x)_k$ of all other classes $k \neq \hat{y}$ must be decreased. The misclassification goal is achieved when $\hat{y} = \arg\max_k f(x)_k$. This can be accomplished by the adversary via increasing features of an input sample $x$ using the following saliency map $S(x, \hat{y})$:

$$S(x,\hat{y}) = \begin{cases} 0 \text{ if } \frac{\partial f(x)_{\hat{y}}}{\partial x_i} < 0 \text{ or } \sum_{k\neq\hat{y}} \frac{\partial f(x)_k}{\partial x_i} > 0 \\ \frac{\partial f(x)_{\hat{y}}}{\partial x_i} \left| \sum_{k\neq\hat{y}} \frac{\partial f(x)_k}{\partial x_i} \right| \text{ otherwise,} \end{cases} \tag{14}$$

where $i$ is the input feature. Alternatively, the adversary may try to decrease some of the input features to achieve the misclassification required.

In [144], a set of white-box attacks against neural networks is proposed. The Carlini and Wagner (C&W) attack approach is similar to [19]. It is based on solving the following optimization problem:

$$\min_w ||\delta(w)|| + cF(x + \delta(w)) \tag{15}$$

where $w$ is a new variable such that once $\delta$ has been expressed in the form of a function of this new variable $w$, perturbed sample $x + \delta$ is always feasible, and $F$ can be one of the following functions:

$$
\begin{aligned}
F(x) &= -J(x) + 1 \\
F(x) &= (\max_{k \neq \hat{y}}(f(x)_k - f(x)_{\hat{y}}))^+ \\
F(x) &= \log(1 + \exp \max_{k \neq \hat{y}}(f(x)_k - f(x)_{\hat{y}})) - \log 2 \\
F(x) &= (0.5 - f(x)_{\hat{y}})^+ \\
F(x) &= -\log(2f(x)_{\hat{y}} - 2) \\
F(x) &= (\max_{k \neq \hat{y}}(z(x)_k - z(x)_{\hat{y}}))^+ \\
F(x) &= \log(1 + \exp \max_{k \neq \hat{y}}(z(x)_k - z(x)_{\hat{y}})) - \log 2
\end{aligned}
\tag{16}
$$

where $J$ is the loss function of the network $f$ given the correct classification and $z$ is the logit, i.e., the unnormalized raw probability prediction of the model for each class. Constant $c$ can be found via binary search.

Study [145] employs the approach similar to [144] by utilising the following loss function:

$$
F(x) = \max\{\max_{k \neq \hat{y}}(z(x))_k - (z(x))_{\hat{y}}, -K\}, \tag{17}
$$

where $K$ is a hyperparameter which guarantees a constant gap between $\max_{k \neq \hat{y}}(z(x))_k$ and $(z(x))_{\hat{y}}$. The optimization problem is also modified by adding a regularisation parameter $\beta$:

$$
\begin{aligned}
&\min_{\delta} g(x) + \beta|\delta|_1 \\
&\text{subject to } x + \delta \text{ is feasible,}
\end{aligned}
\tag{18}
$$

where $g(x) = ||\delta||^2 + cF(x + \delta)$. To craft an adversarial perturbation an element-wise projected shrinkage thresholding function is defined as follows:

$$
(S_\beta(z))_i = \begin{cases} \min\{z_i - \beta, 1\} \text{ if } z_i - x_i \\ x_i \text{ if } |z_i - x_i| \leq \beta \\ \max\{z_i + \beta, 0\} \text{ if } z_i - x_i < -\beta \end{cases} \tag{19}
$$

Study [146] applies a recurrent neural network (RNN) with a two-step training process to generate adversarial examples targeting a keyword spotting (KWS) system. In particular, the framework proposed utilises long-short term memory (LSTM) neural network [50]. It is assumed that an adversary has perfect knowledge of the training dataset and the target model. First, for each input sample $x$ from the dataset, the adversary generates an adversarial example $\tilde{x}$ using the BIM algorithm [138]. Each resulting perturbation $\delta(x) = \tilde{x} - x$ is used to pre-train the LSTM model $g_\theta$ with weights $\theta$ by minimising the following loss function:

$$
L_1(\theta) = \sum_x ||g_\theta(x) - \delta(x)|| \tag{20}
$$

Once the network has been pre-trained, it is directly optimised against the KWS network by minimising the following loss function:

$$
\begin{aligned}
L_2(\theta) &= \sum_x ||g_\theta(x)|| + \\
&+ c(\max_{k \neq \hat{y}} z(x + g_\theta(x))_k - z(x + g_\theta(x))_{\hat{y}} + R)^+,
\end{aligned}
\tag{21}
$$

where $z$ is the logit score returned by the KWS, $c > 0$ is a regularisation parameter and $R \geq 0$ is a constant which controls the confidence score of the target label $\hat{y}$.

In [147], an algorithm for crafting universal input-agnostic adversarial perturbations is proposed. The algorithm seeks a universal perturbation $\delta$, such that $||\delta|| \leq \epsilon$, while fooling at least percentage $\alpha$ of data points in $X = \{x_1, \ldots, x_m\}$, i.e.,:

$$\frac{1}{m}\sum_{i=1}^{m} \mathbb{I}\{f(x_i + \delta) \neq f(x_i)\} \geq \alpha \tag{22}$$

The algorithm proposed gradually builds the universal perturbation by iterating over the data points in $X$. At each such iteration, the minimal perturbation $\Delta\delta_i$ that sends the current perturbed point $x_i + \delta$ to the classifier's decision boundary is computed and aggregated into a universal perturbation $\delta$. If $\delta$ does not fool data point $x_i$, the algorithm seeks for an extra perturbation $\Delta\delta_i$ with minimal norm that allows to fool data point $x_i$ by solving the following optimization problem:

$$\begin{aligned} &\min_{\Delta\delta_i} ||\Delta\delta_i|| \\ &\text{subject to } f(x_i + \delta + \Delta\delta_i) \neq f(x_i) \end{aligned} \tag{23}$$

To ensure that the constraint $||\delta|| \leq \epsilon$ is satisfied, the updated universal perturbation is further projected on the ball of radius $\epsilon$ and centred at 0.

It is worth noticing that all the attacks presented by this point generate adversarial examples for AI/ML models deployed for classification tasks. In fact, by definition, an adversarial example is specialised input created with the purpose of confusing a model, resulting in the misclassification of a given input. Concerning adversarial learning in regression settings, there are no natural margins as in the case of classification tasks, and, therefore, it is hindered with difficulties to define the adversarial attacks, its success, and evaluation metrics. However, many of the attack algorithms surveyed are still applicable for attacking regression models. For example, in [148], FGSM and BIM attacks are successfully performed against a deep learning model deployed for time series regression. Since both of these methods introduce such a perturbation that moves an input sample in the direction of the gradients to maximise the loss value, the difference between the original and the adversarial example predictions will increase.

Study [149] a simple method for generating adversarial attacks for regression tasks derived from the algebraic properties of the Jacobian of the network. The attack problem is formulated as follows:

$$\begin{aligned} &\max_{\delta} ||f(x + \delta) - y|| \\ &\text{subject to } ||S^{-1/2}\delta|| \leq \epsilon, \end{aligned} \tag{24}$$

where $S$ a weighting matrix typically corresponding to the covariance matrix of the inputs. The authors perform a linearization of $f$ based on the first-order Taylor expansion with Jacobian matrix $J(x)$ calculated at point $x$ and then solve the resulting optimization problem for standard norm values. For example, in the case of the Euclidean norm, the optimal perturbation can be computed by performing a singular value decomposition (SVD) of $J(x)S^{1/2}$.

### *5.4. Score-Based Attacks against Neural Networks*

In this subsection, we survey attacks that can be carried out in the black-box settings, i.e., the target model can remain unknown to the adversary. However, it is required that the adversary has an ability to query an input sample to the target classifier which returns a vector of scores, each of which is essentially a probability that the input belongs to a certain class. Two such black-box score-based attacks against neural networks are proposed in [150]. Both of the attacks focus on convolutional neural network based models and aim to modify input images in such a way that they are misclassified by the models. The first attack simply modifies the value of one randomly selected pixel $(\tilde{i}, \tilde{j})$ of an input

image $x = \{x_{ijk}\}$ normalised in such a way that $l_{ijk} \leq x_{ijk} \leq u_{ijk}$ where $l_{ijk} < 0$ and $u_{ijk} > 0$ $\forall i, j, k$ as follows:

$$\tilde{x}_{ijk}(\tilde{i}, \tilde{j}) = \begin{cases} x_{ijk} \text{ if } i \neq \tilde{i} \text{ or } j \neq \tilde{j}, \\ p \times sign(x_{ijk}) \text{ otherwise,} \end{cases} \tag{25}$$

where $k$ represents the colour channel index and $p$ is the perturbation factor.

One shortcoming of this algorithm is that when building an adversarial image, the perturbation applied by the adversary to a single pixel is often quite large. Hence, in the adversarial image, there might exist such a pixel coordinate value of which lies outside the valid range $[l, u]$. The search procedure is therefore required to be redesigned in order to ensure that the adversarial image generated still belongs to the original image space. For this purpose, a local-search procedure is proposed that aims to minimise the probability $z_y(x)$ assigned by the neural network model that an input image $x$ belongs to class $y$.

Study [151] proposes a gradient-free optimization method for finding an adversarial perturbation. This method approximates gradients by estimating finite differences calculated in random directions. The method allows for crafting adversarial perturbations in case the directions the gradients point to are not useful or the model itself is non-differentiable.

In [152], a black-box attack based on a genetic algorithm is proposed. In order to estimate each population members, the following fitness function $F(x)$ is introduced:

$$F(x) = \log f(x)_{\hat{y}} - \log \sum_{k \neq \hat{y}} f(x)_k, \tag{26}$$

where $f(x)_k$ is the $k$-th class score returned by the neural network $f$ for input sample $x$.

A simple method for the construction of adversarial samples in black-box settings is proposed in [153]. The attack aims to decrease the confidence scores $f(x + \delta)_y$ for the class label predicted by the target classifier $f$ without the perturbation. For this purpose, the algorithm randomly samples a direction $q$ from a set $Q$ of search directions. Set $Q$ can be the standard or the discrete cosine basis.

An extension of the SimBA method [153] for black-box UAP generation is proposed in [154]. It combines the SimBA method with white-box UAP attack [147]. As previously, the attack aims to decrease the confidence scores $f(x + \delta)_y$ for the labels predicted by the target classifier $f$ without the UAP of any sample $x \in X = \{x_1, \ldots, x_m\}$ with label $y \in Y = \{y_1, \ldots, y_m\}$. For this purpose, the algorithm randomly samples a direction $q$ from a set $Q$ of search directions. Set $Q$ can be the standard or the discrete cosine basis.

### *5.5. Decision-Based Attacks against Neural Networks*

In this subsection, we summarise practical adversarial example generation attacks that can be carried out by an adversary in fully black-box settings, i.e., when the adversary does not have access to the AI/ML model deployed. In addition, we assume that the target model output includes only hard labels, i.e., no score values can be derived. Assuming the adversary has access to samples from the training dataset, the most straight-forward approach is to train a substitute model and then use one of the aforementioned white-box algorithms to craft a perturbation. Such an attack approach is based on the transferability property of adversarial examples which states that the perturbation crafted to fool a specific model can also fool another model with high probability. In [155], this attack approach is employed. The target model in this case is used by the adversary to provide labels for the training samples. Even though the adversary does not need to have full knowledge of the target model, it must at least have some partial knowledge of the model input and expected output. The adversary also should have access to a small set of training data samples that can act as an initial set for training the substitute model. The algorithm efficiency can be improved via periodical step size selection and reservoir sampling [133].

Study [156] proposes another attack approach which does not involve training a substitute model. The attack starts from an adversarial point $\tilde{x}$ calculated for an input sample $x$ and tries to keep it in the adversarial region while reducing the distance to $x$ by adjusting

the step size using a trust region method. The proposal distribution $P$ may significantly affect the efficiency of the algorithm. This distribution defines random directions the algorithm explores at each iteration. The optimal proposal distribution depends on the target neural network. The following is the algorithm procedure in general. In the $t$-th step a perturbation $\delta_t$ is drawn for an adversarial sample $\tilde{x}_{t-1}$ from such a maximum entropy distribution that the following constraints are satisfied: $\tilde{x}_{t-1} + \delta_t$ is a feasible sample, $||\delta_t|| = \alpha||\tilde{x} - \tilde{x}_{t-1}||$, and $||\tilde{x} - \tilde{x}_{t-1}|| - ||\tilde{x} - (\tilde{x}_{t-1} + \delta_t)|| = \beta||\tilde{x}, \tilde{x}_{t-1}||$. In practice, the first two constraints are satisfied by sampling from a Gaussian distribution and then rescaling and clipping the sample accordingly. After that, the sample is projected on the ball with the centre being equal to the original adversarial sample $\tilde{x}$ such that the third and the first constraints hold. The resulting perturbation is denoted as the orthogonal perturbation. A half of such orthogonal perturbations is expected to still be adversarial. If the percentage of adversarial samples is less than half, the length of the perturbation is reduced, otherwise, if more than half of perturbations are adversarial, the length is increased. Similarly, if the success rate is too small the step size is reduced, if it is too large, the size is increased.

Although the boundary attack can find adversarial examples with distortion comparable to white-box attacks, it suffers from exponential search time and lacks convergence guarantees. More query-efficient algorithm for attacking a classifier $f$ by generating an adversarial perturbation for a given sample $x$ with label $y$ is proposed in [157]. Instead of searching for an adversarial example, the approach proposed searches such a direction $\theta$ that minimises the distortion $F(\theta)$ which in case of an non-targeted attack can be obtained as follows:

$$F(\theta) = \arg\min_{\lambda>0}(f(x + \lambda\frac{\theta}{||\theta||} \neq y)) \tag{27}$$

The adversarial example can be found as $\tilde{x} = x + F(\tilde{\theta})\frac{\tilde{\theta}}{||\tilde{\theta}||}$, where $\tilde{\theta} = \arg\min_\theta F(\theta)$.

In [158], another black-box attack based on estimating gradient direction at the decision boundary based solely on access to model decisions is proposed. It is also essentially an advanced version of the boundary attack described earlier. At each iteration of the algorithm, the following three steps are carried out: estimation of the gradient direction, step-size search via geometric progression, and boundary search via a binary search.

Study [159] proposes zeroth order optimization (ZOO) attacks against a black-box classifier. The attack approach is based on C&W attack [144], similarly it uses the following loss function:

$$F(x) = \max\{\max_{k\neq\hat{y}}(z(x))_k - (z(x))_{\hat{y}}, -K\}, \tag{28}$$

where $z(x)$ $K$ is a hyperparameter that guarantees a constant gap between $\max_{k\neq\hat{y}}(z(x))_k$ and $(z(x))_{\hat{y}}$. The gradient $g_i = \frac{\partial F(x)}{\partial x_i}$ is estimated as follows:

$$g_i = \frac{F(x + he_i) - F(x - he_i)}{2h}, \tag{29}$$

where $h$ is a small constant and $e_i$ is a standard basis vector with only the $i$-th component as 1. Similarly, Hessian $h_i = \frac{\partial^2 F(x)}{\partial x_{ii}^2}$ can be estimated as follows:

$$h_i = \frac{F(x + he_i) - 2F(x) + F(x - he_i)}{h^2}. \tag{30}$$

In [160], adversarial examples are generated in black-box settings with the help of a geometric framework under the restriction that only a small number of queries is allowed. The resulting GeoDA attack approach is based on the fact that the mean curvature of the decision boundaries in the neighbourhood of an input sample is quite low for many frequently used deep learning models [161]. As a result, it can be locally approximated by a hyperplane passing through a boundary point $b$ close to data sample $x$, with a normal

vector $w$. In this case, an adversarial perturbation $\delta$ can be derived by solving the following optimization problem:

$$\begin{aligned} &\min_{\delta} ||\delta|| \\ &\text{subject to } w^T(x+\delta) - w^T b = 0 \end{aligned} \tag{31}$$

Assuming $\tilde{x}$ is a boundary point, value of $w$ can be derived by querying the target classifier $f$ $N_t$ times using samples $\tilde{x} + \eta_i$ where $i = 1, \ldots, N_t$. After that, $w$ can be estimated as follows:

$$w = \frac{\frac{1}{N_t}\sum_{i=1}^{N_t} \rho_i \eta_i}{||\frac{1}{N_t}\sum_{i=1}^{N_t} \rho_i \eta_i||}, \tag{32}$$

where $\rho_i = 1$ if $f(\tilde{x} + \eta_i) \neq f(\tilde{x})$ and $\rho_i = -1$ otherwise.

Study [162] attacks a black-box classifier in a query-limited partial-information setting. The former means that the adversary has a limited number of queries to the classifier, and the latter means the adversary has access only to a list of $k$ returned labels sorted according to their probabilities predicted. The attack approach is based on estimating the gradient with natural estimation strategies (NES). The authors introduce the following function to estimate the score of a sample:

$$S(x) = \frac{1}{n}\sum_{i=1}^{n} R(x + \mu u_i), \tag{33}$$

where $n$ is a number of samples, $\sigma$ is a search variance, $u_i$ is a random perturbation, and function $R(x) = k - \text{rank}(\hat{y}|x)$. There are several studies that propose to use GANs [105] to generate adversarial perturbations. For example, study [163] focuses on bypassing a machine learning based malware detection model by training a GAN. The target model $f(x)$ classifies each sample as either benign $y$ or malicious $\hat{y}$. The GAN's generator network $G_{\theta_g}$ with parameters $\theta_g$ transforms an input sample from the malicious class into its adversarial version, whereas the substitute discriminator $D_{\theta_d}$ with parameters $\theta_d$ is used to fit the target model and at the same time it provides gradient information to train the generator. By utilising the transitivity of the adversarial samples, the resulting framework constructs a substitute model that can simulate the target model and generates adversarial samples to bypass detection engines. Training the generator network pushes the substitute detector to misclassify adversarial samples. Since the substitute detector tries to fit the target model, the training of the generator further fools that model. The loss functions for the discriminator and the generator can, respectively, be calculated as:

$$\begin{aligned} L_d = &-E_{x:f(x)=y} \log(1 - D_{\theta_d}(x)) - \\ &- E_{x:f(x)=\hat{y}} \log D_{\theta_d}(x) \end{aligned} \tag{34}$$

and

$$L_g = E_{x \in \hat{X}} \log D_{\theta_d}(x), \tag{35}$$

where $\hat{X}$ is the actual malware dataset, i.e., the samples that actually belong to the target class $\hat{y}$, which may be different from the ones classified as $\hat{y}$ by the target model.

Slightly different adversarial example generation approach that also uses GAN is proposed in [164]. In that study, the generator network $G_{\theta_g}$ takes the original instance $x$ as its input and generates a perturbation $G_{\theta_g}(x)$. The resulting sample $G_{\theta_g}(x)$ is sent to the discriminator $D_{\theta_d}$ which is used to distinguish the generated data and the original instance $x$. Assuming that the target classifier $f$ is known to the adversary, its loss for the perturbed sample $L_f(x + G_{\theta_g}(x), \hat{y})$ can be determined. This loss is essentially either the distance between the prediction and the target class $\hat{y}$ in case of a targeted attack or the opposite of the distance between the prediction and the ground truth class in case of an non-targeted attack. The GAN's generator is trained by minimising the following loss function:

$$
\begin{aligned}
L_g = {} & E_{x \in X} L_f(x + G_{\theta_g}(x), \hat{y}) + \\
& + \alpha(E_{x \in X} \log D_{\theta_d}(x + G_{\theta_g}(x)) + \\
& + E_{x \in X} \log(1 - D_{\theta_d}(x + G_{\theta_g}(x)))) + \\
& + \beta E_{x \in X}(||G_{\theta_g}(x)|| - c)^+,
\end{aligned} \tag{36}
$$

where $x$ is sampled from $X$, i.e., the set of the samples which belong to the original class $y$, $c$ denotes a user-specified bound. Similarly, the discriminator is trained to minimise:

$$
\begin{aligned}
L_d = {} & -E_{x \in X} \log D_{\theta_d}(x + G_{\theta_g}(x)) - \\
& - E_{x \in X} \log(1 - D_{\theta_d}(x + G_{\theta_g}(x)))
\end{aligned} \tag{37}
$$

In case, the target model is unknown to the adversary, the study proposes to build a distilled [165] network $F$ based on the output of the target model $f$ by minimising the cross-entropy $H$ between the distilled and target model outputs, i.e.

$$
L_f = E_x H(F(x), f(x)) \tag{38}
$$

Finally, several studies formulate an adversarial example generation problem as a Markov decision process which consists of three following components: a set of adversarial samples as an environment state space, adversarial sample modifications as an intelligent agent action space and a reward function that is calculated based on whether or not the resulting samples have been correctly classified by the target model. The agent aims to discover a policy that maximises a cumulative future reward signal received, which in case of the decision-based adversarial machine learning means to learn how to transform an adversarial sample in such a way that the target model classifies the sample as a legitimate one. In order to solve the resulting problem, an RL approach is employed. For example, study [166] demonstrates viability of RL approach by attacking machine-learning-based anti-malware engines. In the framework proposed, the attacker learns to find an optimal sequence of functionality-preserving malware code modifications to evade the supervised detector. The RL algorithm employed to search for adversarial examples is DQN [53]. Similarly, study [167] uses DQN to effectively generate adversarial traffic flows to deceive an intrusion detection model by automatically adding perturbations to flow samples. As in the previous example, the attacker looks for such modifications to the sample, which does neither change its original function nor break the sample format. It is worth noticing that other RL algorithms can be employed for the task as well, e.g., since DQN does not support continuous action spaces, DDPG [109] or PPO [114] can be used.

In addition to such complicated attacks, the target model can be tested against simple perturbations generated by adding a noise signal distributed in a certain way to the original input sample. This noise signal can for example be sampled from a uniform or a Gaussian distribution. Another popular choice for the noise signal, especially in case of the target model being an image classifier, is salt-and-pepper noise which presents itself as sparsely occurring white and black pixels. In order to find the minimal size of the adversarial perturbation generated with one of the noise signals mentioned, a line-search algorithm is used. Alternatively, one can iterate over all the pixels perturbed, resetting them to the ones from the original image if the perturbed image still stays adversarial as proposed in [168].

In the rest of this subsection, we summarise black-box hard-label attack algorithms which look for universal perturbations which fool the target model on all or almost all potential input samples $X = \{x_1, \ldots, x_m\}$. An example of such an approach [147] has already been discussed in the previous section. In this part of the report, we focus on algorithms for crafting universal adversarial perturbations in black-box settings. One such attack algorithm is proposed in [169]. The attack approach is similar to the RGF attack introduced in [157], i.e., instead of searching for an adversarial example, the approach proposed searches such a direction $\theta$ that minimises the distortion $F(\theta)$ which is either:

$$F(\theta) = ||(\arg\min_{\lambda>0} f(x_1 + \lambda \frac{\theta}{||\theta||} \neq y_1), \ldots, \arg\min_{\lambda>0} f(x_m + \lambda \frac{\theta}{||\theta||} \neq y_m)|| \tag{39}$$

or

$$F(\theta) = \arg\min_{\lambda>0} \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}\{f(x_i + \lambda \frac{\theta}{||\theta||}) \neq f(x_i)\} \geq \alpha, \tag{40}$$

where $y_1, \ldots, y_m$ are the correct class labels of the samples $x_1, \ldots, x_m$ and $\alpha$ is the amount of samples that are supposed to be misclassified. The resulting algorithm is quite similar to the one described in [157].

Two algorithms for UAP generation are proposed in [170]. Both of them are based on training a substitute neural network using data samples available and then generating perturbations for this substitute model. After that, the attack relies on the transferability property of adversarial attacks. Once perturbations are generated for the data samples given, a universal perturbation is constructed using either employing principal component analysis (PCA) or via training a variational autoencoder (VAE). In the former case, perturbations generated are stacked into a matrix, and PCA is used to obtain the principal component that has the largest variance. The resulting vector is then scaled in order to satisfy the perturbation size constraints. In the latter case, a VAE is trained using the perturbations generated with the data samples available. After that, the mean of the resulting encodings in the latent space is calculated and decoded into a mean perturbation, which is also supposed to be scaled in order to be contained inside of the perturbation limits. Similarly, study [171] uses SVD to decompose the matrix constructed from the normalised perturbations generated for data samples given and then use the first component as the universal perturbation.

## 6. Fuzzing Frameworks

There are several open-source frameworks available to test an AI/ML against existing adversarial example generation attack algorithms. These frameworks offer robust implementations of several standardised, state-of-the-art adversarial attacks, defence and detection schemes, robustness certifications, metrics, and formal verifications. As it was mentioned in the introduction, crafting adversarial examples using one of such frameworks as a rule requires specifying a target model that takes an input and makes a prediction, a criterion that defines what an adversarial perturbation is, a distance measure of the perturbation size, and an attack algorithm that takes the input and its label as well as the model, the adversarial criterion and the distance measure to generate an adversarial perturbation. All the frameworks we managed to find are implemented in Python which is the most popular programming language for developing AI/ML applications. As with most of the Python libraries, one of the aforementioned adversarial frameworks can be easily installed via pip which is a standard package installer for Python. Most of the frameworks found are able to interface with most popular AI/ML libraries such as PyTorch, Keras, TensorFlow, Theano and several others. Furthermore, many frameworks also provide a straight-forward way to add support for other AI/ML libraries.

Study [172] describes Cleverhans which is a Python library that provides standard adversarial example generation technology and reference implementations for adversarial training, which can be used to develop more robust AI/ML models and provide standard model performance benchmarks for confrontation settings. The target model can be either a Keras Sequential or a raw TensorFlow model. A custom target model can also be implemented by inheriting from the certain model class which includes the minimum amount of properties and methods the custom target model should implement. The attacks implemented include L-BFGS attack [19], FGSM attack [137], C&W attack [144], Elastic-Net attack [145], BIM attack [138], PGD attack [139], MIM attack [140], JSMA attack [20], DeepFool attack [142], fast feature attack [141], and SPSA attack [151]. As one can notice, all of the attacks implemented are white-box adversarial example generation based attacks

against neural networks. In addition to the attack implementations, the framework also implements adversarial training which is essentially adding adversarial examples to the training data to allow the target model to generalise more efficiently [137]. The target model is then suggested to be tested using the dataset that includes both the legitimate and adversarial samples. The resulting accuracy value can be used as a metric to benchmark the target model robustness. There are however no metrics implemented to benchmark the attack algorithms themselves.

Adversarial Robustness Toolbox (ART) is another Python library which aims to defend AI/ML models against adversarial threats [173]. In addition to the support of Keras and Tensorflow, ART contains a functional API enabling the integration of models from various ML libraries such as Pytorch, MXNet, Sklearn, XGBoost, LightGBM, CatBoost, and GPy. Combining several classifier models into an ensemble is also supported. In order to test black-box attacks the most basic classifier properties and methods should be implemented using the corresponding abstract class provided. In addition, there are specific interface implementations that should be employed when implementing a custom neural network or a decision tree classifier. In addition to the attack algorithms implemented in Cleverhans, the attack list in ART includes UAP attack [147], NewtonFool attack [143], spatial transformation attack [174], ZOO attacks [159], boundary attack [156], SimBA attack [153], GeoDA attack [160], adversarial patch [175], decision tree attack [133], high confidence low uncertainty (HCLU) attack [176] and HopSkipJump attack [158]. The spatial temporal attack aims to generate an adversarial sample via performing a combination of one rotation and one translation of the input image. The HCLU attack focuses on such adversarial perturbations that are classified with confidence above 95% and with the uncertainty level being lower than when classifying the original unperturbed sample. The adversarial patch is a sample crafted in such a way that pictures of a natural scene into which this sample is inserted are misclassified. The rest of the attacks were covered in two previous sections of this report. As one can notice, in addition to the well-studied white-box attacks, the framework includes implementations for multiple black-box attack approaches. There is also support for decision tree classifiers including an implementation of the simple attack proposed in [133]. As in the previous case, ART allows one to perform adversarial training. It also provides several transformation methods that can be used for hardening an image classifier: spatial smoothing [177], JPEG compression [178], Gaussian augmentation [179] and several others. Other model hardening techniques include feature squeezing [177], label smoothing [180] and thermometer encoding [181]. ART also includes a module which provides runtime detection methods for adversarial examples. Finally, ART provides several metrics which allow researchers to evaluate the classifier robustness against various attack algorithms. The metric list includes: the average sensitivity of the model's logits with respect to changes in the inputs [182], the average sensitivity of the target model's loss function with respect to changes in the inputs [183] and the average minimal perturbation which is required to get an input misclassified [142]. The sensitivity metrics however are not directly related to adversarial example attacks. There is also a decision-tree-specific metric based on clique method [131].

Study [184] introduces Foolbox which is another Python toolbox which allows one to benchmark the robustness of AI/ML models. It also supports the most popular AI/ML libraries Keras and Tensorflow as well as Pytorch, Theano and MXNet. Additionally, Foolbox allows one to combine the predictions of one model with the gradient of another. This makes it possible to attack non-differentiable models using gradient-based attacks and allows transfer-based attacks described in [155]. The attack list includes multiple white-box attacks all of which have already been implemented in ART [173]. Speaking of the black-box attacks, there are implementations of the single pixel and local search attacks [150], GenAttack [152], pointwise attack [168] and already mentioned boundary [156] and HopSkipJump [158] attacks. There are also functions for generating straight-forward perturbations via adding a uniform, Gaussian, or salt-and-pepper noise signal. When performing the attack in the framework, there is a possibility to define a criterion under

which a sample is considered an adversarial. The list of such criteria include targeted and non-targeted misclassification, top-k misclassification, original class probability being below a given threshold, target class probability being above a given threshold. Furthermore, custom adversarial criteria can be defined and employed.

In [185], the design, implementation, and evaluation of DeepSec are presented. DeepSec focuses on the analysis of adversarial attacks and defenses for deep learning models, and it is implemented using Pytorch. The framework considers only white-box attack scenarios, where the adversary has full knowledge of the model under attack but is not aware of defenses that might be deployed. The attack list includes the most popular white-box algorithms such as L-BFGS [19], FGSM [137], C&W [144], ElasticNet [145], BIM [138], PGD [139], MIM [140], JSMA [20], DeepFool [142], and UAP [147]. The only attack implemented in DeepSec that has not been mentioned by this point is OptMargin attack proposed in [186]. This attack is essentially C&W attack applied to classifier ensembles. The defenses implemented in DeepSec include adversarial training [19,187], gradient masking and regularization [159,188,189], input transformation [190–192] and region-based classification [193]. In addition, several adversarial example detection methods are mentioned in the study [177,194,195], but those do not appear to have been yet implemented in DeepSec. It is also worth mentioning that several attacks and defense implementations in DeepSec are reported to be incorrect with the results being flawed and misleading [196]. Finally, DeepSec offers vast variety of attack utility metrics that can be classified into four categories: misclassification, imperceptibility, robustness and computation cost. Misclassification metrics include average confidence of the original class, average confidence of the target adversarial class and misclassification ratio. Imperceptibility metrics are as a rule employed for adversarial examples in computer vision tasks and they include average norm distortion, average structural similarity, perturbation structural sensitivity, and perturbation sensitivity distance. Robustness metrics in DeepSec include robustness to image compression, robustness to Gaussian blur and noise tolerance. The last metric, computation cost, is simply average runtime required by the adversary to generate a successful adversarial example using a certain algorithm.

Another Pytorch-based toolbox for adversarial robustness research called Advertorch is introduced in in [197]. Similarly to DeepSec, it focuses solely on generating adversarial examples for deep neural networks. The attack list consists mostly of popular white-box algorithms: L-BFGS [19], FGSM [137], C&W [144], BIM [138], PGD [139], MIM [140], JSMA [20], as well as fast feature [141] and spatial transformation [198] attacks. The last attack in this list has not yet been mentioned in the report, but it is essentially C&W attack against an image classifier with an adversarial example generated by modifying an original image with a per-pixel displacement field and applying bilinear interpolation to the resulting image in order to guarantee that the pixel coordinates lie on the integer grid. Advertorch also implements backward pass differentiable approximation (BPDA) wrapper proposed in [199]. This attack technique allows one to carry out gradient-based attacks against models which have non-differentiable or gradient-obfuscating components. It basically replaces a non-differentiable layer of a target neural network with a differentiable approximation when performing the backward pass during the gradient estimation in order to generate an adversarial example by one of the aforementioned white-box gradient-based attack algorithms. Black-box attack methods implemented in Advertorch only include single pixel and local search algorithms [150]. The list of defences is limited to a few preprocessing techniques such as JPEG compression [178] as well as feature squeezing and smoothing [177]. No attack utility metrics are mentioned in [197].

Advbox, which is another toolbox for generating adversarial examples against deep learning models, is proposed in [200]. It supports all the major deep learning libraries including PaddlePaddle, PyTorch, MxNet, and, as usual, Keras and TensorFlow. Advbox users can also conduct black-box attacks on model files generated by Caffe2, CNTK, MATLAB and Chainer platforms. The attack list is however limited and includes only a few of the most popular white-box algorithms. The algorithm implementations are based on

Foolbox [184], and as in the original framework, Advbox allows users to test a target model against perturbations generated by adding a uniform, Gaussian, salt-and-pepper and other noise signals. The defence algorithms include feature squeezing and smoothing [177], Gaussian augmentation [179], adversarial training [139] and thermometer encoding [181]. As in the previous case, no information on any attack utility metrics is provided in the study.

Table 4 summarises the adversarial example generation frameworks described in this section. It is worth mentioning that there are no algorithms for generating universal adversarial perturbations in black-box settings implemented in any of the aforementioned frameworks. Speaking of the white-box attack algorithms, implementations for the majority of them can be found in each framework specified. Furthermore, we are interested in generating adversarial examples not only for neural networks, but also other AI/ML models such as k-NN and decision trees. However, none of the frameworks surveyed has support for k-NN, whereas attacks against decision trees are only mentioned in ART, although in almost all of the frameworks a custom model can be implemented using the base model class.

**Table 4.** Frameworks for adversarial example generation.

| Module | Component | Cleverhans | ART | Foolbox | DeepSec | Advertorch | Advbox |
|---|---|---|---|---|---|---|---|
| Target model | Tensorflow or Pytorch | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Sklearn, XGBoost, LightGBM or CatBoost | | ✓ | | | | |
| | Custom model | ✓ | ✓ | ✓ | | | ✓ |
| Whitebox attacks | Attacks against k-NN or k-means | | | | | | |
| | Attacks against decision trees | | ✓ | | | | |
| | Attacks against neural networks | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Scorebased attacks | Single pixel [150] | | | ✓ | | | ✓ |
| | Local search [150] | | | ✓ | | | ✓ |
| | GenAttack [152] | | | ✓ | | | |
| | SimBA attack [153] | | ✓ | | | | |
| Decisionbased attacks | Substitute model attack [155] | ✓ | | | | | |
| | Boundary attack [156] | | ✓ | ✓ | | | |
| | HopSkipJump attack [158] | | ✓ | ✓ | | | |
| | ZOO attack [159] | | ✓ | | | | |
| | GeoDA attack [160] | | ✓ | | | | |
| | NES attack [162] | | ✓ | | | | |
| | Pointwise attack [168] | | | ✓ | | | |
| Attack utility metrics | Minimal perturbation size | | ✓ | | | | |
| | Misclassification rate | | | | ✓ | | |
| | Adv. class confidence | | | | ✓ | | |
| | True class confidence | | | | ✓ | | |
| | Adversarial example robustness | | | | ✓ | | |
| | Imperceptibility | | | | ✓ | | |
| | Computation cost | | | | ✓ | | |
| Defences | Adversarial training [19,137,187] | ✓ | ✓ | | ✓ | | ✓ |
| | Preprocessing [177,178,180,181,191,192] | | ✓ | | ✓ | ✓ | ✓ |
| | Postprocessing [193,201] | | ✓ | | ✓ | | |
| | Model transformation [189] | | ✓ | | ✓ | | |
| | Adversarial example detection [177,194,195] | | ✓ | | | | |

## 7. Adversarial ML in 5G

In this section, we describe how an adversary may try to attack AI/ML-based network components described above. In the channel estimation cases [28,76], the adversary can aim crafting such a perturbation that would maximise the error between the real DL CSI matrix and the one predicted by the target model, whereas in the case of the framework described in [58], the adversary would try to maximise the difference between the transmitted symbols and the output of the target model. In all of these attack scenarios, the adversary would be required to have access to a dataset in order to train a substitute model [133]. The attack approaches in which it is required to query the target model with intelligently crafted samples would be hard to carry out since the outputs of the target models mentioned appear to be used internally by the BS for efficient use of frequency bands and energy by performing various techniques, such as water-filling, appropriate precoding and beamforming [76]. Another option would be to craft a universal adversarial perturbation (UAP) [147].

Study [202] proposes an adversarial example generation attack against the autoencoder based framework for CSI feedback described in [77]. In particular, the attack is performed against the neural network which acts as the decoder. The adversary aims to maximise the error between the real CSI matrix and the one predicted by the decoder model. The attack is white-box, i.e., the adversary is required to know the target decoder network. Moreover, in the attack scheme described, the adversary somehow has access to the input codeword which is essentially the DL CSI matrix encoded at the UE. The former problem can be mitigated via training a substitute autoencoder model [133]: since the target model is unsupervised, the adversary would only need to get access to a dataset of DL CSI matrices. The latter would require the adversary to be able to eavesdrop the signal that contains the CSI matrix encoded and then jam this signal in such an intelligent way that a necessary perturbation is added to the decoder input. Crafting a UAP would also be possible. As in the previous case, the attacks that rely on querying the target model's public API do not look applicable in this scenario, as the output of the decoder is not returned to the UE, but it is used internally by the BS.

Attacks against AI/ML-based modulation recognition models is probably the most well-studied category of the attacks against wireless communication systems based on adversarial example generation [170,203–205]. In a real world scenario, the adversary would have access to neither the exact input of the receiver nor the modulation type selected by the target model. It would also be fair to assume that the adversary does not know the exact channel between the adversary and the receiver, but several realisations of that channel are available to the adversary [170,203]. It can also be assumed that the information available to the adversary includes an arbitrary dataset of the received signals with their corresponding modulation types. In such settings, the adversary will be able to first train a substitute model [133] and craft adversarial perturbations using one of the white-box attack methods described in details in [170,203,205]. After that, a universal perturbation can be generated.

To attack intelligent channel decoding frameworks, the adversary can try to generate a perturbation that causes decoding errors at the receiver [171,204]. In white-box settings, this perturbation can be carried out, e.g., by employing FGSM and then projecting the resulting optimal perturbation on the ball with the centre at the original sample and the radius equal to the power budget available to the attacker [171]. In the black-box settings, the attack can be carried out by training a substitute model using the dataset generated for a similar task since again there is no possibility to query the target model with various test samples. Alternatively, a UAP can then be crafted as it is proposed in [171].

In the case of AI/ML-driven beamforming, the adversary may search for either a perturbation that causes any misclassification at the receiver's classifier or such a perturbation such that it not only causes a misclassification at the receiver's classifier, but also tries to change the beam to one of the worst beams. In [206], such adversarial perturbations are crafted in white-box settings with the FGSM algorithm. In black-box settings, the adversary would again most likely train a substitute model or search for a universal perturbation

since querying the target model does not appear to be a feasible option due to the nature of the attack. An adversarial example generation attack which targets the deep learning model introduced in [63] is demonstrated in [207]. The adversary aims to maximise the error between the real achievable rate and the one predicted by the model. The attack implemented in [207] is white-box. Furthermore, the study assumes the adversary has perfect knowledge of the input feature vector for which the prediction is made. In other words, the adversary most likely needs to have access to the aforementioned cloud processing unit during the inference stage to be able to perform this attack. In black-box settings, a substitute model should be trained by the adversary and then used during the inference to craft an input-agnostic adversarial perturbation.

In order to attack the power allocation model proposed in [71], the adversary can try to jam the channels between the BS and the UEs in order to make the neural network at the BS output a non-optimal power allocation vector. In [71], this attack is performed in white-box settings using FGSM algorithm. In a realistic use case scenario, the adversary would first need to train a substitute model and then search for a universal perturbation during the attack. In order to attack the power allocation system described in [37], the adversary perturbs the input that is fed to the target model by employing one of the GNSS spoofing techniques [208]. The objective of the attacker is to compute the adversarial perturbation of UEs positions in the direction of the gradient to increase the loss function such that the power allocation system outputs non-optimal power allocation vector. In [209], such an attack is implemented in white-box settings using FGSM and PGD algorithms. In black-box settings, a substitute model is suggested to be trained. In this use case, the UE positions are assumed to be known to the adversary. Otherwise, an input-agnostic adversarial perturbation can be crafted as it was done in the previous scenarios.

The adversary may also attempt to attack the network slicing models proposed in [96–98,100] by generating such requests that would force the target RL model to make incorrect resource distribution decisions. Study [210] proposes such an attack against the RL-based resource allocation model presented in [98] in black-box settings. In particular, the adversary aims to determine resources to be specified by fake requests for the most efficient flooding attack. If these fake requests are selected and network resources are allocated to them, fewer resources will be left for real requests from legitimate users. The attack is based on Q-learning algorithm with each state being the number of available PRBs, the action being equal to the number of PRBs assigned for each fake request, and the reward being calculated as the number of served fake requests. It is assumed in the study that the adversary may sense the spectrum in order to detect available PRBs. It is also assumed that the adversary has information whether the request sent has been served or not. Other black-box algorithms also appear to be applicable in this use case scenario: the adversary may query the target model by sending fake requests with various requirements in order to find an optimal solution.

In [9], the adversary aims to compromise the integrity of the target AI/ML model deployed for intelligent spectrum sharing during the sensing periods to force the ESC into making wrong transmit decisions. In particular, the adversary attempts to fool the ESC to allow the BS to transmit when an incumbent user is present, and vice versa, to fool the ESC to stop the BS transmissions even though there are no CBRS users. The attack proposed is black-box: the adversary trains a substitute model by monitoring both CBRS radar signals and whether the BS transmits to its UEs. The former acts as the input to the substitute model, whereas the latter is used to provide ground truth labels for the model. However, the adversary is still required to know the input to the ESC and the channel between the adversary and the ESC for crafting correct perturbations. As previously, algorithms for crafting an input-agnostic perturbation can be employed in order to resolve the aforementioned issues.

An adversarial example generation based attack can also be carried out to craft adversarial network traffic flows that would deceive the detection models proposed in [41,45]. For example, study [167] attempts to craft such a perturbation to a botnet related traffic

flow that it is classified as a legitimate one. To achieve this goal in black-box settings, DQN algorithm for crafting adversarial perturbations is employed, as it allows the adversary to operate in the scenario when its feature space is different from the one employed by the target model, as the latter is assumed to be unknown to the attacker. In theory, however, any black-box algorithm that queries the target model with intelligently crafted input samples can be used in this use case, since both the input sample and the model output can be derived by the attacker assuming any malicious flow will be blocked by the IDS. In the case of jamming attack detection, the adversary can try to manipulate the signal parameters in order to make the target model misclassify it as a legitimate UE. Under the same assumption that the target model output, i.e., whether the device is classified as normal or malicious, is known to the adversary, any of the black-box adversarial perturbation generation algorithms described previously in the study can be used. All the attack use cases found are summarised in Table 5.

**Table 5.** Adversarial example generation attacks against 5G components.

| Cat. | Attack | Target | Motivation | Access | Data and Capabilities |
|---|---|---|---|---|---|
| AMC | Black-box [170,203]; white-box [204,205] | Supervised: CNN [31,78], LSTM [32], fully-connected neural network [59] | Targeted modulation misclassification; any misclassification independent of the target label | Broadcast signal over the air | Black-box: some channels between the attacker and the receiver, ability to retrieve the modulation type predicted by the target classifier; white-box: the same as in the black-box attack plus the target model, the exact input at the receiver, the channel between the attacker and the receiver |
| Channel coding | Black-box [171,204]; white-box [204] | Unsupervised: fully-connected autoencoder [171,204]; RL: policy gradients [204] | Any error in decoding the message transmitted | Broadcast signal over the air | Black-box: distribution of channel realisations between the attacker and the receiver, the set of all possible messages; white-box: the same as in the black-box attack plus the target model |
| Beamforming | White-box [206] | Supervised: fully-connected neural network [64] | Any misclassification; targeted misclassification to select one of the worst beams | Broadcast signal over the air | RSS values over the subset of beams selected, the target model, sensing the target model predictions |
| | White-box [207] | Supervised: fully-connected neural network [63] | Maximise the error between the real achievable rate and the target model output | Perturb the model input at the cloud | The combined feature vector of omni-received sequences collected from all the BSs, the target model |
| Channel estimation | White-box [202] | Unsupervised: CNN autoencoder | Maximise the error between the real CSI matrix and the target model output | Broadcast signal over the air | The target decoder model, the channel between the attacker and the BS, the codeword (the CSI in the latent space) returned by the encoder |
| Power allocation | Black-box, white-box [209] | Supervised: fully-connected neural network [37] | Make the sum of the target model outputs greater than the total power available to the BS | Spoof GNSS UE positions | Black-box: UE positions, sensing the target model outputs; white-box: the same as in the black-box attack plus the target model |
| | White-box [71] | Supervised: fully-connected neural network [71] | Minimise the maximum achievable rate of the UEs | Broadcast over the air | The target model, having perfect knowledge of the channels between the BS and UEs |
| | Black-box [211] | Reinforcement: DQN [70,85] | Minimise the achievable sum-rate of all the transmitters | Broadcast over the air | SNR levels at different channels, the sum-rate returned by the target model once an action is carried out by the adversary |

**Table 5.** *Cont.*

| Cat. | Attack | Target | Motivation | Access | Data and Capabilities |
|---|---|---|---|---|---|
| Scheduling | Black-box [9] | Supervised: fully-connected neural network [9] | Any misclassification independent of the target label | Broadcast signal over the air | The exact input to the target model, the channel between the ESC and the adversary, sensing whether the UEs transmit or not |
| Slicing | Black-box [9] | Supervised: fully-connected neural network [9] | Targeted misclassification: fool the target model to classify the adversary's signal as a authenticated one | Broadcast signal over the air | UE signals, the target model decisions |
| | Black-box [210] | Reinforcement: DQN [98] | Maximise the number of fake requests served | Transmit over the network | The number of available PRBs, obtaining the number of the fake requests served |
| | Black-box [212] | Reinforcement: DQN [98] | Minimise the number of legitimate requests served | Transmit over the network | The number of available PRBs, sensing whether the legitimate UE requests have been served or not |
| Security | Black-box [167] | Supervised: CNN, RNN [41,45] | Targeted misclassification: fool the target model to classify each malicious flow as a legitimate one | Transmit over the network | Botnet flows, allowed flow perturbations that do not brake flow format, capability to determine whether the flow is classified as malicious or legitimate one |
| Localization | Black-box, white-box [213] | Supervised: CNN+RNN [84] | Maximise the error between the real travel time and the target model output | Spoof UE GPS locations | Black-box: GPS trajectory points, output of the target model; white-box: the same as in the black-box attack plus the parameters and gradient information of the target model |

## 8. Discussion

In this section, we discuss the main points of our survey devoted to attacking the next generation mobile networks with adversarial examples and outline existing challenges in this research area. Since an AI/ML model is only as good as the data used for its training, obtaining a high quality dataset is an essential part of the research process in case of both training target AI/ML models as well as attacking these models with adversarial examples. Unfortunately, acquiring such a dataset can be a challenging task as mobile operators and network providers rarely share any real network data due to obvious privacy and security reasons. In the case a group of researchers conducts a study in cooperation with a mobile network operator and as a result manages to get access to such a dataset, it will still most likely not become available for others [72]. For this reason, the majority of the publicly available datasets related to mobile networking are synthetic. For example, RadioML dataset [29] used by many researchers for machine learning based modulation recognition [30] contains modulated real voice and text data with radio channel effects being simulated by employing various robust models including time varying multi-path fading of the channel impulse response, random walk drifting of carrier frequency oscillator and sample time clocks, and additive Gaussian white noise. Generating data using network simulators allows a researcher to obtain more realistic data. However, this may be time consuming as the researcher has to get familiar with the process of writing simulation scripts to generate and collect the data required. Moreover, many advanced network simulators are not openly available or may be extremely expensive to acquire which also hinders the research process. Finally, as mentioned in Section 2, the data collected using real equipment allows researchers to carry out somewhat realistic experiments, however, such an approach is rarely scalable in the lab environment: the scenario is often limited to very few transmitters and receivers [74]. We therefore can conclude that generating realistic

datasets for training AI/ML models which can potentially be deployed in mobile networks would accelerate the research process in this domain tremendously.

Concerning the AI/ML algorithms used in the mobile networking domain, as it has already been mentioned in Section 3, the vast majority of those rely on deep learning which has gained increasing popularity in recent years due to its supremacy in terms of accuracy when trained with huge amounts of data [8]. As a result, mobile networking researchers are also beginning to recognize the power and importance of deep learning, and are exploring its potential to solve various optimization and resource allocation problems. It is also worth noticing that the major part of the deep learning models proposed to be deployed in the next generation wireless networks is trained in a supervised way due to its supreme convergence speed and high accuracy compared to unsupervised and reinforcement learning approaches. Speaking of the deep learning architectures, the most of the models used are comprised of fully-connected layers which have few trainable parameters which allows them to learn faster compared to other structures. Fully-connected layers are often used even in the case of time-series data although using recurrent neural networks would probably allow for lower error and higher accuracy values [61]. In many AI/ML applications in the RAN domain, channel state matrices act as the input to the target models [28]. These matrices can often be interpreted as images with height and width being equal to the number of antennas and the number of OFDM subcarriers, respectively, whereas the real and imaginary part of the signal can be interpreted as the colour channels. Since convolutional layers are usually employed in image related problems, CNNs are frequently used in studies which focus on the analysis of channel state matrices. Although the attention mechanism provides state-of-the-art accuracy results in certain time-series processing tasks, the neural networks which rely on this technique, such as transformers, are rarely employed by mobile network researchers. Therefore, implementing and testing these deep learning architectures for solving various problems in the mobile network domain can be considered as one of the potential future research directions.

In Section 4, we have provided an extensive review of machine learning driven frameworks found in recent scientific papers. As one can notice, AI/ML is often proposed to be deployed in mobile networks for various channel estimation related problems. The traditional way to learn the transmission channels is to have the receiver perform the measurements and send those back to the transmitter. Alternatively, an AI/ML model can be trained to achieve this task by using only a part of the channel state matrix which allows to reduce the feedback transmission overhead [28,64]. Another category of AI/ML research focuses on substituting conventional algorithms used for mobile networking with machine learning ones aiming to reduce the processing time. The models that follow this approach are trained using the datasets in which the output labels are calculated using the corresponding conventional algorithm which might be computationally expensive to use in real time [68]. During the inference, the resulting AI/ML models trained allow for an accurate and time-efficient estimation of the optimal solution without occupying large amounts of computing resources. Finally, supreme predictive capabilities of state-of-the-art AI/ML models allow for efficient dynamic resource allocation in the mobile networking domain. For example, AI/ML can be employed to predict resource block usage in network slicing problems [86], estimate the probability of a certain content being requested in order to implement a proactive caching scheme [87], and discover energy-efficient power management strategies by analysing historical traffic data patterns [95]. It is however worth noticing that there are still not many studies in the 5G domain which employ the reinforcement learning approach for achieving the task given despite the recent progress in this field of study. As mentioned in Section 3, this can be partially explained by the fact that training an RL agent in a real world environment is often not feasible while implementing a simulation software may require some additional resources. Thus, developing a simulation framework which can be used for implementing and training RL agents in realistic network use case scenarios would benefit both mobile networking researchers as well as service providers interested in this topic.

Speaking of the adversarial examples, as it is mentioned in Section 5, the vast majority of the research articles in this domain focuses on attacking deep neural networks. In white-box settings, an adversarial example for such a model can be generated by either modifying an input sample in the direction of the gradient of the loss function [137] or solving a nonlinear optimization problem with the objective function being equal to the weighted sum of the adversarial perturbation norm and the target model loss calculated for the perturbed sample [19]. In black-box settings, the attack as a rule involves querying the target model's public API and using the resulting outputs to estimate the gradient direction at the decision boundary. Alternatively, the adversary may try to learn a substitute for the target model using a synthetic dataset labelled by observing the target model output. Other adversarial perturbation crafting algorithms rely on applying generative adversarial networks and even deep reinforcement learning. Speaking of the other AI/ML models such as k-nearest neighbours and decision trees, finding an adversarial example is a quite straight-forward procedure when the number of the neighbours and the tree depth are small. However, the procedure complexity grows exponentially with the increase of the aforementioned hyperparameters. For this reason, various heuristics are proposed in order to generate an adversarial example in a reasonable amount of time. Despite the variety of existing white-box and black-box algorithms for crafting adversarial examples, unless there is a serious flaw in the 5G network component security, the adversary should be able to neither have access to the exact inputs of the target model, due to the different channel and interference conditions, nor obtain the output label, since it is most of the time used internally by the model and it is not available to any other wireless node outside of the network. For this reason, the adversary has the best chance to fool the target model by crafting a universal input-agnostic adversarial perturbation. Thus, developing novel algorithms for calculating universal adversarial examples is of great interest nowadays [170].

In Section 6, we have overviewed several frameworks which allow one to evaluate various adversarial example generation algorithms against state-of-the-art AI/ML models. These frameworks offer implementations of both white-box and black-box techniques as well as multiple defence and detection schemes. Some of the frameworks found also provide implementations of various attack utility and classifier robustness metrics. All the frameworks discussed are implemented in Python, they are open-source and they can be easily downloaded, installed and used by mobile networking researchers. In addition, most of the frameworks overviewed are able to interface with most popular deep learning libraries such as TensorFlow and PyTorch. Unfortunately, none of the frameworks surveyed has support for k-nearest neighbours algorithms, whereas attacks against decision trees are mentioned in only one of the frameworks. However, many frameworks provide a straightforward way to add support for classifiers implemented in AI/ML libraries. It is also worth mentioning that there are no algorithms for generating universal adversarial perturbations in black-box settings implemented in any of the aforementioned frameworks. As mentioned above, input-agnostic adversarial examples can be employed when the information about neither the user inputs to the model nor the resulting outputs is available to the attacker, and, therefore, they pose the most realistic threat to AI/ML-driven components of the future generation mobile networks.

Finally, concerning the attacks against mobile network components, there are still not so many studies devoted to this research topic despite the number is growing as AI/ML becomes widely adopted in mobile communication systems and as a result novel attack approaches are being constantly developed. There are multiple research papers which focus on the problems of attacking AI/ML-assisted automatic modulation recognition [170], intelligent beamforming [206] and power allocation [71] frameworks. However, the adversarial example generation algorithms tested in those studies are limited to straightforward FGSM, the attack using a substitute model and some universal adversarial perturbation generation techniques with the help of PCA and VAE. Furthermore, the datasets used in those studies are somewhat synthetic which further reduces the quality of the research work. Therefore, one potential future research direction is extending results obtained in

these use case scenarios by employing various white-box and black-box attack algorithms using more realistic simulation environments. Another direction is implementing and evaluating adversarial examples in the use cases for which this attack vector has not yet been studied to the best of our knowledge. These include attacks against channel estimation [28], channel decoding [61,80] and jamming detection [74] models. Furthermore, the recent progress in developing sample efficient RL methods makes this machine learning approach an attractive option for service providers to use for efficient and reliable network resource allocation. For this reason, the number of RL-driven frameworks is expected to grow significantly in the future. Thus, developing attack algorithms which target AI/ML models trained with the help of the reinforcement learning approach is also of the great importance in this field of study.

## 9. Conclusions

As mentioned in the introduction, the purpose of this survey is to provide detailed introduction and references for each of the issues related to the problem of generating adversarial examples against AI/ML systems which might be present in future 5G networks. First, datasets used for training and evaluating AI/ML models in mobile networks are overviewed and various supervised, unsupervised and reinforcement learning algorithms are briefly summarised. Next, multiple AI/ML applications in the next-generation mobile networks found in recent scientific papers are described. After that, we survey existing white-box and black-box adversarial example generation based attack algorithms and list several frameworks which employ these algorithms for fuzzing state-of-the-art AI/ML models. Finally, several adversarial example generation attacks against mobile network applications are presented.

In order to assess vulnerability of AI/ML models to adversarial examples, the minimal perturbation value which causes a misclassification, is supposed to be estimated. Unfortunately, finding the global minimum is close to impossible in any practical settings, and, therefore, heuristic attacks are often employed to find a suitable approximation. The minimal perturbation value found by testing the target model against various attack algorithms can help mobile operators to evaluate how sensitive the target model outputs to changes in its inputs. Such approach cannot guarantee protection of the model against novel attack techniques. However, it reduces the number of ways the adversary can perform attacks which makes it easier to protect the target model via runtime detection and rejection.

We currently employ this research approach in order to evaluate the efficiency of the adversarial examples when applied to various AI/ML-based 5G components and frameworks. For this purpose, we select several attack scenarios against AI/ML components of 5G networks from the ones presented in this survey. Next, we generate data and train multiple target models with the help of the datasets and various AI/ML algorithms mentioned in this study. After that, we evaluate multiple white-box and black-box attacks implemented for the aforementioned use cases based on deterioration of the objective functions optimised by the target models. The results of the experiments conducted are expected to be published in the near future.

**Author Contributions:** Writing—original draft preparation, M.Z. and D.Z.; writing—review and editing, M.Z. and P.M.; supervision, T.H. All authors have read and agreed to the published version of the manuscript.

**Funding:** The research did not receive specific funding but was performed as a part of the employment of the authors in Magister Solutions Ltd., Jyväskylä, Finland. The APC is also expected to be funded by Magister Solutions Ltd.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References


1. *ITU-R M.2410-0*; Minimum Requirements Related to Technical Performance for IMT2020 Radio Interface(s). ITU: Geneva, Switzerland, 2017.
2. Marcus, M.J. 5G and "IMT for 2020 and beyond" [Spectrum Policy and Regulatory Issues]. *IEEE Wirel. Commun.* **2015**, *22*, 2–3. [CrossRef]
3. Elijah, O.; Leow, C.Y.; Rahman, T.A.; Nunoo, S.; Iliya, S.Z. A Comprehensive Survey of Pilot Contamination in Massive MIMO—5G System. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 905–923. [CrossRef]
4. Song, M.; Shan, H.; Yang, H.H.; Quek, T.Q.S. Joint Optimization of Fractional Frequency Reuse and Cell Clustering for Dynamic TDD Small Cell Networks. *IEEE Trans. Wirel. Commun.* **2021**, *21*, 398–412 . [CrossRef]
5. *TR37.817, G*; Study on Enhancement for Data Collection for NR and EN-DC. 3GPP: Sophia Antipolis, France, 2021.
6. Jahangiri, A.; Rakha, H.A. Applying Machine Learning Techniques to Transportation Mode Recognition Using Mobile Phone Sensor Data. *IEEE Trans. Intell. Transp. Syst.* **2015**, *16*, 2406–2417. [CrossRef]
7. Li, H.; Ota, K.; Dong, M. Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing. *IEEE Netw.* **2018**, *32*, 96–101. [CrossRef]
8. Haidine, A.; Salmam, F.Z.; Aqqal, A.; Dahbi, A. Artificial intelligence and machine learning in 5G and beyond: A survey and perspectives. In *Moving Broadband Mobile Communications Forward: Intelligent Technologies for 5G and Beyond*; IntechOpen: London, UK, 2021; p. 47.
9. Sagduyu, Y.E.; Erpek, T.; Shi, Y. Adversarial Machine Learning for 5G Communications Security. *arXiv* **2021**, arXiv:2101.02656.
10. GSMA. *FS.30—Security Manual*; GSMA: London, UK, 2021.
11. GSMA. *FS.31—Baseline Security Controls*; GSMA: London, UK, 2020.
12. GSMA. *IR.77 InterOperator IP Backbone Security Req. For Service and Inter-Operator IP Backbone Providers*; GSMA: London, UK, 2019.
13. GSMA. FF.21 Fraud Manual; GSMA: London, UK, 2021.
14. Steinhardt, J.; Koh, P.W.; Liang, P. Certified Defenses for Data Poisoning Attacks. *arXiv* **2017**, arXiv:1706.03691.
15. Gu, T.; Liu, K.; Dolan-Gavitt, B.; Garg, S. BadNets: Evaluating Backdooring Attacks on Deep Neural Networks. *IEEE Access* **2019**, *7*, 47230–47244. [CrossRef]
16. Schwarzmann, S.; Marquezan, C.C.; Trivisonno, R.; Nakajima, S.; Zinner, T. Accuracy vs. Cost Trade-off for Machine Learning Based QoE Estimation in 5G Networks. In Proceedings of the ICC 2020—2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020; pp. 1–6. [CrossRef]
17. Masri, A.; Veijalainen, T.; Martikainen, H.; Mwanje, S.; Ali-Tolppa, J.; Kajó, M. Machine-Learning-Based Predictive Handover. In Proceedings of the 2021 IFIP/IEEE International Symposium on Integrated Network Management (IM), Bordeaux, France, 17–21 March 2021; pp. 648–652.
18. Minovski, D.; Ogren, N.; Ahlund, C.; Mitra, K. Throughput Prediction using Machine Learning in LTE and 5G Networks. *IEEE Trans. Mob. Comput.* **2021**. [CrossRef]
19. Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; Fergus, R. Intriguing properties of neural networks. *arXiv* **2014** arXiv:1312.6199.
20. Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Celik, Z.B.; Swami, A. The Limitations of Deep Learning in Adversarial Settings. *arXiv* **2015**, arXiv:1511.07528.
21. *TS33.501, G*; Security Architecture and Procedures for 5G System. 3GPP: Sophia Antipolis, France, 2020.
22. Zhang, C.; Patras, P.; Haddadi, H. Deep Learning in Mobile and Wireless Networking: A Survey. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 2224–2287. [CrossRef]
23. Cheng, X.; Fang, L.; Hong, X.; Yang, L. Exploiting Mobile Big Data: Sources, Features, and Applications. *IEEE Netw.* **2017**, *31*, 72–79. [CrossRef]
24. Mehlführer, C.; Ikuno, J.C.; Simko, M.; Schwarz, S.; Wrulich, M.; Rupp, M. The Vienna LTE simulators—Enabling reproducibility in wireless communications research. *EURASIP J. Adv. Signal Process.* **2011**, *2011*, 29. [CrossRef]
25. Palattella, M.R.; Watteyne, T.; Wang, Q.; Muraoka, K.; Accettura, N.; Dujovne, D.; Grieco, L.A.; Engel, T. On-the-Fly Bandwidth Reservation for 6TiSCH Wireless Industrial Networks. *IEEE Sensors J.* **2016**, *16*, 550–560. [CrossRef]
26. Varga, A.; Hornig, R. An overview of the OMNeT++ simulation environment. In Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, Marseille, France, 3–7 March 2008; pp. 1–10.
27. Alkhateeb, A. DeepMIMO: A Generic Deep Learning Dataset for Millimeter Wave and Massive MIMO Applications. *arXiv* **2019**, arXiv:1902.06435.
28. Alrabeiah, M.; Alkhateeb, A. Deep learning for TDD and FDD massive MIMO: Mapping channels in space and frequency. In Proceedings of the 2019 53rd Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 3–6 November 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1465–1470.
29. O'Shea, T.; West, N. Radio Machine Learning Dataset Generation with GNU Radio. In Proceedings of the GNU Radio Conference, Boulder, CO, USA, 12–16 September 2016.
30. O'Shea, T.J.; Corgan, J.; Clancy, T.C. Convolutional radio modulation recognition networks. In Proceedings of the International Conference on Engineering Applications of Neural Networks, Aberdeen, UK, 2–5 September 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 213–226.

31. Meng, F.; Chen, P.; Wu, L.; Wang, X. Automatic Modulation Classification: A Deep Learning Enabled Approach. *IEEE Trans. Veh. Technol.* **2018**, *67*, 10760–10772. [CrossRef]
32. Rajendran, S.; Meert, W.; Giustiniano, D.; Lenders, V.; Pollin, S. Deep Learning Models for Wireless Signal Classification with Distributed Low-Cost Spectrum Sensors. *IEEE Trans. Cogn. Commun. Netw.* **2018**, *4*, 433–445. [CrossRef]
33. Klautau, A.; Batista, P.; González-Prelcic, N.; Wang, Y.; Heath, R.W. 5G MIMO Data for Machine Learning: Application to Beam-Selection Using Deep Learning. In Proceedings of the 2018 Information Theory and Applications Workshop (ITA), San Diego, CA, USA, 11–16 February 2018; pp. 1–9. [CrossRef]
34. Ruseckas, J.; Molis, G.; Bogucka, H. MIMO beam selection in 5G using neural networks. *Int. J. Electron. Telecommun.* **2021**, *67*, 693–698.
35. Twomey, N.; Diethe, T.; Kull, M.; Song, H.; Camplani, M.; Hannuna, S.; Fafoutis, X.; Zhu, N.; Woznowski, P.; Flach, P.; et al. The SPHERE Challenge: Activity Recognition with Multimodal Sensor Data. *arXiv* **2016**, arXiv:1603.00797.
36. Kozlowski, M.; McConville, R.; Santos-Rodriguez, R.; Piechocki, R. Energy Efficiency in Reinforcement Learning for Wireless Sensor Networks. *arXiv* **2018**, arXiv:1812.02538.
37. Sanguinetti, L.; Zappone, A.; Debbah, M. Deep Learning Power Allocation in Massive MIMO. *arXiv* **2019**, arXiv:1812.03640.
38. Balazinska, M.; Castro, P. CRAWDAD Dataset Ibm/Watson (v. 2003-02-19). 2003. Available online: https://crawdad.org/ibm/watson/20030219 (accessed on 11 April 2022 ).
39. Challita, U.; Dong, L.; Saad, W. Proactive Resource Management for LTE in Unlicensed Spectrum: A Deep Learning Perspective. *IEEE Trans. Wirel. Commun.* **2018**, *17*, 4674–4689. [CrossRef]
40. Garciá, S.; Grill, M.; Stiborek, J.; Zunino, A. An empirical comparison of botnet detection methods. *Comput. Secur.* **2014**, *45*, 100–123. [CrossRef]
41. Fernández Maimó, L.; Perales Gómez, A.L.; Garcia Clemente, F.J.; Gil Pérez, M.; Martínez Pérez, G. A Self-Adaptive Deep Learning-Based System for Anomaly Detection in 5G Networks. *IEEE Access* **2018**, *6*, 7700–7712. [CrossRef]
42. Kolias, C.; Kambourakis, G.; Stavrou, A.; Gritzalis, S. Intrusion detection in 802.11 networks: Empirical evaluation of threats and a public dataset. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 184–208. [CrossRef]
43. Rezvy, S.; Luo, Y.; Petridis, M.; Lasebae, A.; Zebin, T. An efficient deep learning model for intrusion classification and prediction in 5G and IoT networks. In Proceedings of the 2019 53rd Annual Conference on Information Sciences and Systems (CISS), Baltimore, MD, USA, 20–22 March 2019; pp. 1–6. [CrossRef]
44. Sharafaldin, I.; Lashkari, A.H.; Ghorbani, A.A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp* **2018**, *1*, 108–116.
45. Lam, J.; Abbas, R. Machine Learning based Anomaly Detection for 5G Networks. *arXiv* **2020**, arXiv:2003.03474.
46. Almomani, I.; Al-Kasasbeh, B.; Al-Akhras, M. WSN-DS: A dataset for intrusion detection systems in wireless sensor networks. *J. Sensors* **2016**, *2016*, 4731953. [CrossRef]
47. Hachimi, M.; Kaddoum, G.; Gagnon, G.; Illy, P. Multi-stage Jamming Attacks Detection using Deep Learning Combined with Kernelized Support Vector Machine in 5G Cloud Radio Access Networks. In Proceedings of the 2020 International Symposium on Networks, Computers and Communications (ISNCC), Montreal, QC, Canada, 20–22 October 2020; pp. 1–5. [CrossRef]
48. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems, Proceedings of the 26th Annual Conference on Neural Information Processing Systems, Lake Tahoe, Nevada, 3–6 December 2012*; Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2012; Volume 25.
49. Pascanu, R.; Mikolov, T.; Bengio, Y. On the difficulty of training recurrent neural networks. In Proceedings of the International Conference on Machine Learning, Atlanta, GA, USA, 16–21 June 2013; pp. 1310–1318.
50. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
51. Cho, K.; van Merrienboer, B.; Bahdanau, D.; Bengio, Y. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *arXiv* **2014**, arXiv:1409.1259.
52. Jaeger, H.; Haas, H. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science* **2004**, *304*, 78–80. [CrossRef] [PubMed]
53. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing Atari with Deep Reinforcement Learning. *arXiv* **2013**, arXiv:1312.5602.
54. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention Is All You Need. *arXiv* **2017**, arXiv:1706.03762.
55. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. *arXiv* **2015**, arXiv:1512.03385.
56. Friedman, J.H. Stochastic gradient boosting. *Comput. Stat. Data Anal.* **2002**, *38*, 367–378. [CrossRef]
57. Gao, S.; Dong, P.; Pan, Z.; Li, G.Y. Deep Learning Based Channel Estimation for Massive MIMO with Mixed-Resolution ADCs. *IEEE Commun. Lett.* **2019**, *23*, 1989–1993. [CrossRef]
58. Ye, H.; Li, G.Y.; Juang, B.H. Power of Deep Learning for Channel Estimation and Signal Detection in OFDM Systems. *IEEE Wirel. Commun. Lett.* **2018**, *7*, 114–117. [CrossRef]
59. Jagannath, J.; Polosky, N.; O'Connor, D.; Theagarajan, L.N.; Sheaffer, B.; Foulke, S.; Varshney, P.K. Artificial Neural Network Based Automatic Modulation Classification over a Software Defined Radio Testbed. In Proceedings of the 2018 IEEE International Conference on Communications (ICC), Kansas City, MO, USA, 20–24 May 2018; pp. 1–6. [CrossRef]

60. Carpi, F.; Hager, C.; Martalo, M.; Raheli, R.; Pfister, H.D. Reinforcement Learning for Channel Coding: Learned Bit-Flipping Decoding. In Proceedings of the 2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton), Monticello, IL, USA, 24–27 September 2019. [CrossRef]
61. Lyu, W.; Zhang, Z.; Jiao, C.; Qin, K.; Zhang, H. Performance Evaluation of Channel Decoding with Deep Neural Networks. In Proceedings of the 2018 IEEE International Conference on Communications (ICC), Kansas City, MO, USA, 20–24 May 2018; pp. 1–6. [CrossRef]
62. Goutay, M.; Aoudia, F.A.; Hoydis, J. Deep Reinforcement Learning Autoencoder with Noisy Feedback. *arXiv* **2019**, arXiv1810.05419.
63. Alkhateeb, A.; Alex, S.; Varkey, P.; Li, Y.; Qu, Q.; Tujkovic, D. Deep Learning Coordinated Beamforming for Highly-Mobile Millimeter Wave Systems. *IEEE Access* **2018**, *6*, 37328–37348. [CrossRef]
64. Cousik, T.S.; Shah, V.K.; Erpek, T.; Sagduyu, Y.E.; Reed, J.H. Deep Learning for Fast and Reliable Initial Access in AI-Driven 6G mmWave Networks. *arXiv* **2021**, arXiv:2101.01847.
65. Qi, C.; Wang, Y.; Li, G.Y. Deep Learning for Beam Training in Millimeter Wave Massive MIMO Systems. *IEEE Trans. Wirel. Commun.* **2020**, 1. [CrossRef]
66. Ye, J.; Zhang, Y.J.A. DRAG: Deep Reinforcement Learning Based Base Station Activation in Heterogeneous Networks. *arXiv* **2018**, arXiv:1809.02159.
67. Liu, J.; Krishnamachari, B.; Zhou, S.; Niu, Z. DeepNap: Data-Driven Base Station Sleeping Operations Through Deep Reinforcement Learning. *IEEE Internet Things J.* **2018**, *5*, 4273–4282. [CrossRef]
68. Sun, H.; Chen, X.; Shi, Q.; Hong, M.; Fu, X.; Sidiropoulos, N.D. Learning to Optimize: Training Deep Neural Networks for Interference Management. *IEEE Trans. Signal Process.* **2018**, *66*, 5438–5453. [CrossRef]
69. Matthiesen, B.; Zappone, A.; Jorswieck, E.A.; Debbah, M. Deep learning for optimal energy-efficient power control in wireless interference networks. *arXiv* **2018**, arXiv:1812.06920.
70. Nasir, Y.S.; Guo, D. Multi-Agent Deep Reinforcement Learning for Dynamic Power Allocation in Wireless Networks. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 2239–2250. [CrossRef]
71. Kim, B.; Shi, Y.; Sagduyu, Y.E.; Erpek, T.; Ulukus, S. Adversarial Attacks against Deep Learning Based Power Control in Wireless Communications. *arXiv* 2021, arXiv:2109.08139.
72. Chinchali, S.; Hu, P.; Chu, T.; Sharma, M.; Bansal, M.; Misra, R.; Pavone, M.; Katti, S. Cellular Network Traffic Scheduling with Deep Reinforcement Learning. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32.
73. Suarez-Varela, J.; Mestres, A.; Yu, J.; Kuang, L.; Feng, H.; Cabellos-Aparicio, A.; Barlet-Ros, P. Routing in optical transport networks with deep reinforcement learning. *J. Opt. Commun. Netw.* **2019**, *11*, 547–558. [CrossRef]
74. Pawlak, J.; Li, Y.; Price, J.; Wright, M.; Al Shamaileh, K.; Niyaz, Q.; Devabhaktuni, V. A Machine Learning Approach for Detecting and Classifying Jamming Attacks Against OFDM-based UAVs. In Proceedings of the 3rd ACM Workshop on Wireless Security and Machine Learning, Abu Dhabi, United Arab Emirates, 2 July 2021; pp. 1–6.
75. Soltani, M.; Pourahmadi, V.; Mirzaei, A.; Sheikhzadeh, H. Deep Learning-Based Channel Estimation. *arXiv* **2019**, arXiv1810.05893.
76. Safari, M.S.; Pourahmadi, V.; Sodagari, S. Deep UL2DL: Data-Driven Channel Knowledge Transfer From Uplink to Downlink. *IEEE Open J. Veh. Technol.* **2020**, *1*, 29–44. [CrossRef]
77. Wen, C.K.; Shih, W.T.; Jin, S. Deep Learning for Massive MIMO CSI Feedback. *IEEE Wirel. Commun. Lett.* **2018**, *7*, 748–751. [CrossRef]
78. Peng, S.; Jiang, H.; Wang, H.; Alwageed, H.; Yao, Y.D. Modulation classification using convolutional Neural Network based deep learning model. In Proceedings of the 2017 26th Wireless and Optical Communication Conference (WOCC), Newark, NJ, USA, 7–8 April 2017; pp. 1–5. [CrossRef]
79. O'Shea, T.; Hoydis, J. An Introduction to Deep Learning for the Physical Layer. *IEEE Trans. Cogn. Commun. Netw.* **2017**, *3*, 563–575. [CrossRef]
80. Yashashwi, K.; Anand, D.; Pillai, S.R.B.; Chaporkar, P.; Ganesh, K. MIST: A Novel Training Strategy for Low-latency Scalable Neural Net Decoders. *arXiv* **2019**, arXiv:1905.08990.
81. Liang, F.; Shen, C.; Wu, F. An Iterative BP-CNN Architecture for Channel Decoding. *IEEE J. Sel. Top. Signal Process.* **2018**, *12*, 144–159. [CrossRef]
82. Lu, X.; Xiao, L.; Dai, C.; Dai, H. UAV-Aided Cellular Communications with Deep Reinforcement Learning Against Jamming. *IEEE Wirel. Commun.* **2020**, *27*, 48–53. [CrossRef]
83. Cao, G.; Lu, Z.; Wen, X.; Lei, T.; Hu, Z. AIF: An Artificial Intelligence Framework for Smart Wireless Network Management. *IEEE Commun. Lett.* **2018**, *22*, 400–403. [CrossRef]
84. Wang, D.; Zhang, J.; Cao, W.; Li, J.; Zheng, Y. When will you arrive? estimating travel time based on deep neural networks. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018.
85. Lu, Z.; Gursoy, M.C. Dynamic Channel Access and Power Control via Deep Reinforcement Learning. In Proceedings of the 2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall), Honolulu, HI, USA, 22–25 September 2019; pp. 1–5. [CrossRef]
86. Guo, Q.; Gu, R.; Wang, Z.; Zhao, T.; Ji, Y.; Kong, J.; Gour, R.; Jue, J.P. Proactive Dynamic Network Slicing with Deep Learning Based Short-Term Traffic Prediction for 5G Transport Network. In Proceedings of the 2019 Optical Fiber Communications Conference and Exhibition (OFC), San Diego, CA, USA, 7–9 March 2019; pp. 1–3.

87. Chen, M.; Saad, W.; Yin, C.; Debbah, M. Echo State Networks for Proactive Caching in Cloud-Based Radio Access Networks With Mobile Users. *IEEE Trans. Wirel. Commun.* **2017**, *16*, 3520–3535. [CrossRef]
88. Wang, Y.; Narasimha, M.; Heath, R.W. MmWave Beam Prediction with Situational Awareness: A Machine Learning Approach. In Proceedings of the 2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), Kalamata, Greece, 25–28 June 2018; pp. 1–5. [CrossRef]
89. Lei, F.; Dai, Q.; Cai, J.; Zhao, H.; Liu, X.; Liu, Y. A Proactive Caching Strategy Based on Deep Learning in EPC of 5G. In *Advances in Brain Inspired Cognitive Systems, Proceedings of the 9th International Conference, BICS 2018, Xi'an, China, 7–8 July 2018*; Lecture Notes in Computer Science; Ren, J., Hussain, A., Zheng, J., Liu, C., Luo, B., Zhao, H., Zhao, X., Eds.; Springer: Berlin/Heidelberg, Germany, 2018; Volume 10989, pp. 738–747. [CrossRef]
90. Maksymyuk, T.; Gazda, J.; Yaremko, O.; Nevinskiy, D. Deep Learning Based Massive MIMO Beamforming for 5G Mobile Network. In Proceedings of the 2018 IEEE 4th International Symposium on Wireless Systems within the International Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS), Lviv, Ukraine, 20–21 September 2018; pp. 241–244. [CrossRef]
91. Balevi, E.; Andrews, J.G. Deep Learning-Based Channel Estimation for High-Dimensional Signals. *arXiv* **2019**, arXiv:1904.09346.
92. He, H.; Wen, C.K.; Jin, S.; Li, G.Y. Deep Learning-Based Channel Estimation for Beamspace mmWave Massive MIMO Systems. *IEEE Wirel. Commun. Lett.* **2018**, *7*, 852–855. [CrossRef]
93. Xiao, Z.; Gao, B.; Liu, S.; Xiao, L. Learning Based Power Control for mmWave Massive MIMO against Jamming. In Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–6. [CrossRef]
94. Sadeghi, A.; Sheikholeslami, F.; Giannakis, G.B. Optimal and Scalable Caching for 5G Using Reinforcement Learning of Space-Time Popularities. *IEEE J. Sel. Top. Signal Process.* **2018**, *12*, 180–190. [CrossRef]
95. Peng, B.; Seco-Granados, G.; Steinmetz, E.; Fröhle, M.; Wymeersch, H. Decentralized Scheduling for Cooperative Localization With Deep Reinforcement Learning. *IEEE Trans. Veh. Technol.* **2019**, *68*, 4295–4305. [CrossRef]
96. Li, R.; Zhao, Z.; Sun, Q.; I, C.L.; Yang, C.; Chen, X.; Zhao, M.; Zhang, H. Deep Reinforcement Learning for Resource Management in Network Slicing. *IEEE Access* **2018**, *6*, 74429–74441. [CrossRef]
97. Nassar, A.; Yilmaz, Y. Deep Reinforcement Learning for Adaptive Network Slicing in 5G for Intelligent Vehicular Systems and Smart Cities. *arXiv* **2020**, arXiv:2010.09916.
98. Shi, Y.; Sagduyu, Y.E.; Erpek, T. Reinforcement Learning for Dynamic Resource Optimization in 5G Radio Access Network Slicing. In Proceedings of the 2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), Pisa, Italy, 14–16 September 2020; pp. 1–6. [CrossRef]
99. Stampa, G.; Arias, M.; Sanchez-Charles, D.; Muntes-Mulero, V.; Cabellos, A. A Deep-Reinforcement Learning Approach for Software-Defined Networking Routing Optimization. *arXiv* **2017**, arXiv:1709.07080.
100. Liu, Y.; Ding, J.; Liu, X. Resource Allocation Method for Network Slicing Using Constrained Reinforcement Learning. In Proceedings of the 2021 IFIP Networking Conference (IFIP Networking), Espoo, Finland, 21–24 June 2021; pp. 1–3. [CrossRef]
101. Ulyanov, D.; Vedaldi, A.; Lempitsky, V. Deep Image Prior. *Int. J. Comput. Vis.* **2020**, *128*, 1867–1888. [CrossRef]
102. Metzler, C.A.; Mousavi, A.; Baraniuk, R.G. Learned D-AMP: Principled Neural Network based Compressive Image Recovery. *arXiv* **2017**, arXiv:1704.06625.
103. Zhang, K.; Zuo, W.; Chen, Y.; Meng, D.; Zhang, L. Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising. *IEEE Trans. Image Process.* **2017**, *26*, 3142–3155. [CrossRef]
104. Hinton, G.E. A practical guide to training restricted Boltzmann machines. In *Neural Networks: Tricks of the Trade*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 599–619.
105. Goodfellow, I.J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative Adversarial Networks. *arXiv* **2014**, arXiv:1406.2661.
106. Mirza, M.; Osindero, S. Conditional Generative Adversarial Nets. *arXiv* **2014**, arXiv:1411.1784.
107. Bellman, R. *Dynamic Programming*; Dover Publications: Mineola, NY, USA, 1957.
108. Watkins, C.J.C.H.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
109. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2019**, arXiv:1509.02971.
110. Fujimoto, S.; van Hoof, H.; Meger, D. Addressing Function Approximation Error in Actor-Critic Methods. *arXiv* **2018**, arXiv:1802.09477.
111. Rummery, G.A.; Niranjan, M. *On-Line Q-Learning Using Connectionist Systems*; Citeseer: London, UK, 1994; Volume 37.
112. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.P.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous Methods for Deep Reinforcement Learning. *arXiv* **2016**, arXiv:1602.01783.
113. Schulman, J.; Levine, S.; Moritz, P.; Jordan, M.I.; Abbeel, P. Trust Region Policy Optimization. *arXiv* **2017**, arXiv:1502.05477.
114. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347.
115. Khani, M.; Alizadeh, M.; Hoydis, J.; Fleming, P. Adaptive Neural Signal Detection for Massive MIMO. *arXiv* **2019**, arXiv:1906.04610.

116. Gao, G.; Dong, C.; Niu, K. Sparsely Connected Neural Network for Massive MIMO Detection. In Proceedings of the 2018 IEEE 4th International Conference on Computer and Communications (ICCC), Hangzhou, China, 30 July–2 August 2018; pp. 397–402. [CrossRef]
117. Mennes, R.; Camelo, M.; Claeys, M.; Latré, S. A neural-network-based MF-TDMA MAC scheduler for collaborative wireless networks. In Proceedings of the 2018 IEEE Wireless Communications and Networking Conference (WCNC), Barcelona, Spain, 15–18 April 2018; pp. 1–6. [CrossRef]
118. Gutterman, C.; Grinshpun, E.; Sharma, S.; Zussman, G. RAN Resource Usage Prediction for a 5G Slice Broker. In Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing, Catania, Italy, 2–5 July 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 231–240. [CrossRef]
119. Pang, H.; Liu, J.; Fan, X.; Sun, L. Toward smart and cooperative edge caching for 5G networks: A deep learning based approach. In Proceedings of the 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), Banff, AB, Canada, 4–6 June 2018; pp. 1–6.
120. Krizhevsky, A. One weird trick for parallelizing convolutional neural networks. *arXiv* **2014**, arXiv:1404.5997.
121. Jaderberg, M.; Simonyan, K.; Zisserman, A.; Kavukcuoglu, K. Spatial Transformer Networks. *arXiv* **2016**, arXiv:1506.02025.
122. Fischer, W.; Meier-Hellstern, K. The Markov-modulated Poisson process (MMPP) cookbook. *Perform. Eval.* **1993**, *18*, 149–171. [CrossRef]
123. Baum, L.E.; Petrie, T.; Soules, G.; Weiss, N. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *Ann. Math. Stat.* **1970**, *41*, 164–171. [CrossRef]
124. Cao, G.; Lu, Z.; Lei, T.; Wen, X.; Wang, L.; Yang, Y. Demo: SDNbased seamless handover in WLAN and 3GPP cellular with CAPWAN. In Proceedings of the 13th International Symposium on Wireless Communication Systems, Poznań, Poland, 20–23 September 2016; pp. 1–3.
125. Amsaleg, L.; Bailey, J.; Barbe, D.; Erfani, S.; Houle, M.E.; Nguyen, V.; Radovanović, M. The vulnerability of learning to adversarial perturbation increases with intrinsic dimensionality. In Proceedings of the 2017 IEEE Workshop on Information Forensics and Security (WIFS), Rennes, France, 4–7 December 2017; pp. 1–6. [CrossRef]
126. Sitawarin, C.; Wagner, D. On the Robustness of Deep K-Nearest Neighbors. *arXiv* **2019**, arXiv:1903.08333.
127. Sitawarin, C.; Wagner, D. Minimum-Norm Adversarial Examples on KNN and KNN based Models. In Proceedings of the 2020 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 21 May 2020; pp. 34–40.
128. Wang, L.; Liu, X.; Yi, J.; Zhou, Z.H.; Hsieh, C.J. Evaluating the Robustness of Nearest Neighbor Classifiers: A Primal-Dual Perspective. *arXiv* **2019**, arXiv:1906.03972.
129. Yang, Y.Y.; Rashtchian, C.; Wang, Y.; Chaudhuri, K. Robustness for Non-Parametric Classification: A Generic Attack and Defense. *arXiv* **2020**, arXiv:1906.03310.
130. Sitawarin, C.; Kornaropoulos, E.M.; Song, D.; Wagner, D. Adversarial Examples for $k$-Nearest Neighbor Classifiers Based on Higher-Order Voronoi Diagrams. *arXiv* **2021**, arXiv:2011.09719.
131. Chen, H.; Zhang, H.; Si, S.; Li, Y.; Boning, D.; Hsieh, C.J. Robustness Verification of Tree-based Models. *arXiv* **2019**, arXiv:1906.03849.
132. Kantchelian, A.; Tygar, J.D.; Joseph, A.D. Evasion and Hardening of Tree Ensemble Classifiers. *arXiv* **2016**, arXiv:1509.07892.
133. Papernot, N.; McDaniel, P.; Goodfellow, I. Transferability in Machine Learning: From Phenomena to Black-Box Attacks using Adversarial Samples. *arXiv* **2016**, arXiv:1605.07277.
134. Zhang, F.; Wang, Y.; Liu, S.; Wang, H. Decision-based evasion attacks on tree ensemble classifiers. *World Wide Web* **2020**, *23*, 2957–2977. [CrossRef]
135. Zhang, C.; Zhang, H.; Hsieh, C.J. An Efficient Adversarial Attack for Tree Ensembles. *arXiv* **2020**, arXiv:2010.11598.
136. Andriushchenko, M.; Hein, M. Provably Robust Boosted Decision Stumps and Trees against Adversarial Attacks. *arXiv* **2019**, arXiv:1906.03526.
137. Goodfellow, I.J.; Shlens, J.; Szegedy, C. Explaining and Harnessing Adversarial Examples. *arXiv* **2015**, arXiv:1412.6572.
138. Kurakin, A.; Goodfellow, I.; Bengio, S. Adversarial examples in the physical world. *arXiv* **2017**, arXiv:1607.02533.
139. Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; Vladu, A. Towards Deep Learning Models Resistant to Adversarial Attacks. *arXiv* **2019**, arXiv:1706.06083.
140. Dong, Y.; Liao, F.; Pang, T.; Su, H.; Zhu, J.; Hu, X.; Li, J. Boosting Adversarial Attacks with Momentum. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 9185–9193. [CrossRef]
141. Sabour, S.; Cao, Y.; Faghri, F.; Fleet, D.J. Adversarial Manipulation of Deep Representations. *arXiv* **2016**, arXiv:1511.05122.
142. Moosavi-Dezfooli, S.M.; Fawzi, A.; Frossard, P. DeepFool: A simple and accurate method to fool deep neural networks. *arXiv* **2016**, arXiv:1511.04599.
143. Jang, U.; Wu, X.; Jha, S. Objective metrics and gradient descent algorithms for adversarial examples in machine learning. In Proceedings of the 33rd Annual Computer Security Applications Conference, Orlando, FL, USA, 4–8 December 2017; pp. 262–277.
144. Carlini, N.; Wagner, D. Towards Evaluating the Robustness of Neural Networks. *arXiv* **2017**, arXiv:1608.04644.
145. Chen, P.Y.; Sharma, Y.; Zhang, H.; Yi, J.; Hsieh, C.J. EAD: Elastic-Net Attacks to Deep Neural Networks via Adversarial Examples. *arXiv* **2018**, arXiv:1709.04114.

146. Chang, K.H.; Huang, P.H.; Yu, H.; Jin, Y.; Wang, T.C. Audio Adversarial Examples Generation with Recurrent Neural Networks. In Proceedings of the 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC), Beijing, China, 13–16 January 2020; pp. 488–493. [CrossRef]
147. Moosavi-Dezfooli, S.M.; Fawzi, A.; Fawzi, O.; Frossard, P. Universal adversarial perturbations. *arXiv* **2017**, arXiv:1610.08401.
148. Mode, G.; Hoque, K. Adversarial Examples in Deep Learning for Multivariate Time Series Regression. *arXiv* **2020**, arXiv:2009.11911.
149. Gupta, K.; Pesquet, J.C.; Pesquet-Popescu, B.; Kaakai, F.; Malliaros, F. An Adversarial Attacker for Neural Networks in Regression Problems. In Proceedings of the IJCAI Workshop on Artificial Intelligence Safety (AI Safety), Macau, China, 10–16 August 2021.
150. Narodytska, N.; Kasiviswanathan, S.P. Simple Black-Box Adversarial Perturbations for Deep Networks. *arXiv* **2016**, arXiv:1612.06299.
151. Uesato, J.; O'Donoghue, B.; van den Oord, A.; Kohli, P. Adversarial Risk and the Dangers of Evaluating Against Weak Attacks. *arXiv* **2018**, arXiv:1802.05666.
152. Alzantot, M.; Sharma, Y.; Chakraborty, S.; Zhang, H.; Hsieh, C.J.; Srivastava, M.B. Genattack: Practical black-box attacks with gradient-free optimization. In Proceedings of the Genetic and Evolutionary Computation Conference, Prague, Czech Republic, 13–17 July 2019; pp. 1111–1119.
153. Guo, C.; Gardner, J.; You, Y.; Wilson, A.G.; Weinberger, K. Simple black-box adversarial attacks. In Proceedings of the International Conference on Machine Learning, Taipei, Taiwan, 2–4 December 2019; pp. 2484–2493.
154. Koga, K.; Takemoto, K. Simple black-box universal adversarial attacks on medical image classification based on deep neural networks. *arXiv* **2021**, arXiv:2108.04979.
155. Papernot, N.; McDaniel, P.; Goodfellow, I.; Jha, S.; Celik, Z.B.; Swami, A. Practical Black-Box Attacks against Machine Learning. *arXiv* **2017**, arXiv:1602.02697.
156. Brendel, W.; Rauber, J.; Bethge, M. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv* **2017**, arXiv:1712.04248.
157. Cheng, M.; Le, T.; Chen, P.Y.; Yi, J.; Zhang, H.; Hsieh, C.J. Query-efficient hard-label black-box attack: An optimization-based approach. *arXiv* **2018**, arXiv:1807.04457.
158. Chen, J.; Jordan, M.I.; Wainwright, M.J. Hopskipjumpattack: A query-efficient decision-based attack. In Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 18–21 May 2020; pp. 1277–1294.
159. Chen, P.Y.; Zhang, H.; Sharma, Y.; Yi, J.; Hsieh, C.J. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, Dallas, TX, USA, 3 November 2017; pp. 15–26.
160. Rahmati, A.; Moosavi-Dezfooli, S.M.; Frossard, P.; Dai, H. Geoda: A geometric framework for black-box adversarial attacks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 8446–8455.
161. Fawzi, A.; Moosavi-Dezfooli, S.M.; Frossard, P. Robustness of classifiers: From adversarial to random noise. *Adv. Neural Inf. Process. Syst.* **2016**, *29*, 1632–1640.
162. Ilyas, A.; Engstrom, L.; Athalye, A.; Lin, J. Black-box adversarial attacks with limited queries and information. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 2137–2146.
163. Hu, W.; Tan, Y. Generating adversarial malware examples for black-box attacks based on GAN. *arXiv* **2017**, arXiv:1702.05983.
164. Xiao, C.; Li, B.; Zhu, J.Y.; He, W.; Liu, M.; Song, D. Generating adversarial examples with adversarial networks. *arXiv* **2018**, arXiv:1801.02610.
165. Hinton, G.; Vinyals, O.; Dean, J. Distilling the knowledge in a neural network (2015). *arXiv* **2015**, arXiv:1503.02531.
166. Anderson, H.S.; Kharkar, A.; Filar, B.; Roth, P. Evading machine learning malware detection. In Proceedings of the Black Hat 2017, Las Vegas, NV, USA, 22–27 July 2017.
167. Wu, D.; Fang, B.; Wang, J.; Liu, Q.; Cui, X. Evading Machine Learning Botnet Detection Models via Deep Reinforcement Learning. In Proceedings of the ICC 2019—2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; pp. 1–6. [CrossRef]
168. Schott, L.; Rauber, J.; Bethge, M.; Brendel, W. Towards the first adversarially robust neural network model on MNIST. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
169. Hogan, T.A.; Kailkhura, B. Universal decision-based black-box perturbations: Breaking security-through-obscurity defenses. *arXiv* **2018**, arXiv:1811.03733.
170. Kim, B.; Sagduyu, Y.E.; Davaslioglu, K.; Erpek, T.; Ulukus, S. Channel-Aware Adversarial Attacks Against Deep Learning-Based Wireless Signal Classifiers. *arXiv* **2021**, arXiv:2005.05321.
171. Sadeghi, M.; Larsson, E.G. Physical Adversarial Attacks Against End-to-End Autoencoder Communication Systems. *arXiv* **2019**, arXiv:1902.08391.
172. Papernot, N.; Faghri, F.; Carlini, N.; Goodfellow, I.; Feinman, R.; Kurakin, A.; Xie, C.; Sharma, Y.; Brown, T.; Roy, A.; et al. Technical report on the cleverhans v2. 1.0 adversarial examples library. *arXiv* **2016**, arXiv:1610.00768.
173. Nicolae, M.I.; Sinn, M.; Tran, M.N.; Buesser, B.; Rawat, A.; Wistuba, M.; Zantedeschi, V.; Baracaldo, N.; Chen, B.; Ludwig, H.; et al. Adversarial Robustness Toolbox v1.0.0. *arXiv* **2019**, arXiv:1807.01069.

174. Engstrom, L.; Tran, B.; Tsipras, D.; Schmidt, L.; Madry, A. Exploring the landscape of spatial robustness. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 10–15 June 2019; pp. 1802–1811.
175. Brown, T.B.; Mané, D.; Roy, A.; Abadi, M.; Gilmer, J. Adversarial patch. *arXiv* **2017**, arXiv:1712.09665.
176. Grosse, K.; Pfaff, D.; Smith, M.T.; Backes, M. The limitations of model uncertainty in adversarial settings. *arXiv* **2018**, arXiv:1812.02606.
177. Xu, W.; Evans, D.; Qi, Y. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. In Proceedings of the 2018 Network and Distributed System Security Symposium, San Diego, CA, USA, 18–21 February 2018. [CrossRef]
178. Dziugaite, G.K.; Ghahramani, Z.; Roy, D.M. A study of the effect of jpg compression on adversarial images. *arXiv* **2016**, arXiv:1608.00853.
179. Zantedeschi, V.; Nicolae, M.I.; Rawat, A., Efficient Defenses Against Adversarial Attacks. In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, Dallas, TX, USA, 3 November 2017; Association for Computing Machinery: New York, NY, USA, 2017; pp. 39–49.
180. Warde-Farley, D.; Goodfellow, I. 11 adversarial perturbations of deep neural networks. *Perturbations Optim. Stat.* **2016**, *311*, 5.
181. Buckman, J.; Roy, A.; Raffel, C.; Goodfellow, I. Thermometer encoding: One hot way to resist adversarial examples. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
182. Weng, T.W.; Zhang, H.; Chen, P.Y.; Yi, J.; Su, D.; Gao, Y.; Hsieh, C.J.; Daniel, L. Evaluating the robustness of neural networks: An extreme value theory approach. *arXiv* **2018**, arXiv:1801.10578.
183. Arpit, D.; Jastrzebski, S.; Ballas, N.; Krueger, D.; Bengio, E.; Kanwal, M.S.; Maharaj, T.; Fischer, A.; Courville, A.; Bengio, Y.; et al. A closer look at memorization in deep networks. In Proceedings of the International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; pp. 233–242.
184. Rauber, J.; Brendel, W.; Bethge, M. Foolbox: A Python toolbox to benchmark the robustness of machine learning models. *arXiv* **2018**, arXiv:1707.04131.
185. Ling, X.; Ji, S.; Zou, J.; Wang, J.; Wu, C.; Li, B.; Wang, T. DEEPSEC: A Uniform Platform for Security Analysis of Deep Learning Model. In Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2019; pp. 673–690. [CrossRef]
186. He, W.; Li, B.; Song, D. Decision boundary analysis of adversarial examples. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
187. Kurakin, A.; Goodfellow, I.; Bengio, S. Adversarial machine learning at scale. *arXiv* **2016**, arXiv:1611.01236.
188. Ross, A.; Doshi-Velez, F. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32.
189. Papernot, N.; McDaniel, P.; Wu, X.; Jha, S.; Swami, A. Distillation as a defense to adversarial perturbations against deep neural networks. In Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 582–597.
190. Xie, C.; Wang, J.; Zhang, Z.; Ren, Z.; Yuille, A. Mitigating adversarial effects through randomization. *arXiv* **2017**, arXiv:1711.01991.
191. Song, Y.; Kim, T.; Nowozin, S.; Ermon, S.; Kushman, N. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. *arXiv* **2017**, arXiv:1710.10766.
192. Guo, C.; Rana, M.; Cisse, M.; van der Maaten, L. Countering Adversarial Images using Input Transformations. *arXiv* **2018**, arXiv:1711.00117.
193. Cao, X.; Gong, N.Z. Mitigating evasion attacks to deep neural networks via region-based classification. In Proceedings of the 33rd Annual Computer Security Applications Conference, Orlando, FL, USA, 4–8 December 2017; pp. 278–287.
194. Ma, X.; Li, B.; Wang, Y.; Erfani, S.M.; Wijewickrema, S.; Schoenebeck, G.; Song, D.; Houle, M.E.; Bailey, J. Characterizing adversarial subspaces using local intrinsic dimensionality. *arXiv* **2018**, arXiv:1801.02613.
195. Meng, D.; Chen, H. MagNet: A Two-Pronged Defense against Adversarial Examples. *arXiv* **2017**, arXiv:1705.09064.
196. Carlini, N. A critique of the deepsec platform for security analysis of deep learning models. *arXiv* **2019**, arXiv:1905.07112.
197. Ding, G.W.; Wang, L.; Jin, X. advertorch v0.1: An Adversarial Robustness Toolbox based on PyTorch. *arXiv* **2019**, arXiv:1902.07623.
198. Xiao, C.; Zhu, J.Y.; Li, B.; He, W.; Liu, M.; Song, D. Spatially transformed adversarial examples. *arXiv* **2018**, arXiv:1801.02612.
199. Athalye, A.; Carlini, N.; Wagner, D. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 274–283.
200. Goodman, D.; Xin, H.; Yang, W.; Yuesheng, W.; Junfeng, X.; Huan, Z. Advbox: A toolbox to generate adversarial examples that fool neural networks. *arXiv* **2020**, arXiv:2001.05574.
201. Tramèr, F.; Zhang, F.; Juels, A.; Reiter, M.K.; Ristenpart, T. Stealing Machine Learning Models via Prediction APIs. *arXiv* **2016**, arXiv:1609.02943.
202. Liu, Q.; Guo, J.; Wen, C.K.; Jin, S. Adversarial attack on DL-based massive MIMO CSI feedback. *J. Commun. Netw.* **2020**, *22*, 230–235. [CrossRef]
203. Kim, B.; Sagduyu, Y.E.; Davaslioglu, K.; Erpek, T.; Ulukus, S. Over-the-Air Adversarial Attacks on Deep Learning Based Modulation Classifier over Wireless Channels. *arXiv* **2020**, arXiv:2002.02400.
204. Usama, M.; Mitra, R.N.; Ilahi, I.; Qadir, J.; Marina, M.K. Examining Machine Learning for 5G and Beyond through an Adversarial Lens. *arXiv* **2020**, arXiv:2009.02473.

205. Kim, B.; Sagduyu, Y.E.; Erpek, T.; Davaslioglu, K.; Ulukus, S. Adversarial Attacks with Multiple Antennas Against Deep Learning-Based Modulation Classifiers. *arXiv* **2020**, arXiv:2007.16204.
206. Kim, B.; Sagduyu, Y.E.; Erpek, T.; Ulukus, S. Adversarial Attacks on Deep Learning Based mmWave Beam Prediction in 5G and Beyond. *arXiv* **2021**, arXiv:2103.13989.
207. Catak, E.; Catak, F.O.; Moldsvor, A. Adversarial Machine Learning Security Problems for 6G: MmWave Beam Prediction Use-Case. *arXiv* **2021**, arXiv:2103.07268.
208. Psiaki, M.L.; Humphreys, T.E. GNSS Spoofing and Detection. *Proc. IEEE* **2016**, *104*, 1258–1270. [CrossRef]
209. Manoj, B.R.; Sadeghi, M.; Larsson, E.G. Adversarial Attacks on Deep Learning Based Power Allocation in a Massive MIMO Network. *arXiv* **2021**, arXiv:2101.12090.
210. Shi, Y.; Sagduyu, Y.E. Adversarial Machine Learning for Flooding Attacks on 5G Radio Access Network Slicing. *arXiv* **2021**, arXiv:2101.08724.
211. Wang, F.; Gursoy, M.C.; Velipasalar, S. Adversarial Reinforcement Learning in Dynamic Channel Access and Power Control. *arXiv* **2021**, arXiv:2105.05817.
212. Shi, Y.; Sagduyu, Y.E.; Erpek, T.; Gursoy, M.C. How to Attack and Defend 5G Radio Access Network Slicing with Reinforcement Learning. *arXiv* **2021**, arXiv:2101.05768.
213. Qiu, J.; Du, L.; Chen, Y.; Tian, Z.; Du, X.; Guizani, M. Artificial Intelligence Security in 5G Networks: Adversarial Examples for Estimating a Travel Time Task. *IEEE Veh. Technol. Mag.* **2020**, *15*, 95–100. [CrossRef]