Computer Networks 218 (2022) 109367

Contents lists available at ScienceDirect

Computer Networks

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

Automated deep learning-based wide-band receiver

Bahar Azari [*,1], Hai Cheng[1], Nasim Soltani[1], Haoqing Li[1], Yanyu Li[1], Mauro Belgiovine[1], Tales Imbiriba, Salvatore D'Oro, Tommaso Melodia, Yanzhi Wang, Pau Closas, Kaushik Chowdhury, Deniz Erdoğmuş

Electrical & Computer Engineering Department, Northeastern University, 360 Huntington Ave, Boston, MA 02115, United States of America

ARTICLE INFO

Keywords:
Deep learning
Neural network pruning
Wireless communications
Channel estimation
Physical layer

ABSTRACT

We propose a modular and full-fledged physical layer receiver design for Orthogonal Frequency Division Multiplexing (OFDM) wireless systems leveraging the advances of deep neural networks (DNN). We adopt a detailed modular design that includes proper and utmost domain knowledge in each element and train it using data collected both via simulations as well as over-the-air and emulated wireless transmissions. We then unify all the modules into an end-to-end automated deep learning-based wide-band receiver and fine-tune it to further improve its accuracy. Our combined pipeline analysis exhibits superior performance by showing bit error rate values up to 8 times lower if compared to the traditional approaches for wireless communications.

1. Introduction

Wireless networking systems heavily rely upon well-defined functionalities and procedures that are implemented and executed at several layers of the protocol stack. At the bottom of the protocol stack, we find the physical layer which interfaces the transceiver with the wireless channel and is in charge of the challenging task of translating wireless waveforms into information bits, and vice versa. This is an essential task, because any errors occurring at the physical layer will inevitably propagate to all upper layers, eventually resulting in erroneous communications and performance degradation. For this reason, the design of the physical layer for wireless communications has been largely built upon rigorous and well-established signal processing (e.g., modulation, multiplexing) and coding (e.g., Forward Error Correction (FEC) or channel coding) algorithms that aim at reducing the risk of such errors to a minimum.

Although these powerful tools have proved themselves to be reliable and effective in many practical applications, they still rely upon several assumptions on the nature of the wireless channel, its noise, the interference, and the features of the hardware components. For example, the noise on the channel is commonly assumed to follow a Gaussian distribution, multipath fading channel is treated as a linear time invariant system [1,2], and there always exists some form of stationarity assumption. Similarly, hardware components (e.g., power amplifiers, mixers, ADC/DAC) often have nonlinear responses [3], which distort electromagnetic signals. Despite these assumptions facilitate the study, modeling, design and implementation of such solutions, the obtained models can only approximate – or completely neglect in some cases – non-stationary and non-linear effects that characterize wireless systems, which eventually results in bottlenecks and sub-optimal performance.

In this paper, we aim at relaxing those assumptions to develop a wireless networking system capable of providing reliable and high-performance communications even in the case where none of the above assumptions hold. Indeed, to capture and compensate for the sources of nonlinear distortion, we need nonlinear expressive functions. Thanks to their inherent nature of being universal function approximators, deep neural networks – and in general machine learning approaches – have found their way into the design and development of wireless communications [4–14]. Being able to adapt to time-varying channel conditions and to capture non-linearities make deep learning one of the most promising technologies to build next-generation wireless networks. Part of what makes these models achieve comparable, and at times superior, performance w.r.t traditional models is the fact that they do not require any *a priori* information regarding the wireless system model. Therefore, deep learning-based solutions are less prone to inaccuracies and approximations baked in the model, and can thus learn their own representation of the model and detect the most relevant features as demonstrated in [15,16].

We advance the field of deep learning-based wireless networking systems by presenting a totally modular and full-fledged physical layer receiver design for OFDM-based wireless networks. OFDM is a popular multicarrier modulation scheme for wireless communications with a

* Corresponding author.
E-mail address: azari.b@northeastern.edu (B. Azari).
[1] Equal contribution.

https://doi.org/10.1016/j.comnet.2022.109367
Received 6 March 2022; Accepted 16 September 2022
Available online 28 September 2022

variety of advantages ranging from robustness to multipath propagation to efficient practical implementation, among others. Similar to traditional solutions, our system embeds modules for IQ symbol demodulation, channel estimation, equalization and FEC decoding. Different from traditional solutions, each component in our system leverages deep neural networks to recover transmitted data bits and mitigate the impact of non-linearities and non-stationarities on the decoding process. Our system leverages a modular design where each module is designed to accomplish a single task. At the same time, all modules are trained in synergy to improve not only the effectiveness of each individual task, but also to achieve superior performance in collaboration with other modules.

With the goal of making our system robust against non-stationarities and non-linearities, as well as encouraging our neural networks to generalize across different – and possibly unseen – channel conditions, we train each module by using a combination of simulated and real data. Specifically, we have collected more than 60 GB of data generated both via simulations as well as by collecting over-the-air data using the Arena testbed [17], and emulated data using Colosseum wireless emulator [18]. Arena is an open-access wireless testing platform based on a grid of antennas mounted on the ceiling of a large office-space environment. Arena allows data collection in a real-world wireless deployment with diverse multipath, fading, scattering and path-loss conditions, which makes it a valuable source of nonlinear phenomena. Colosseum is the world's largest wireless emulator equipped with a large-scale emulation environment employed for experimentation in wireless systems.

In the first part of the paper, we design each module of our receiver based on their inherit characteristics using the well celebrated multilayer perceptrons (MLPs) and recurrent neural networks (RNNs). MLPs are simple and fast-to-train architectures deployed when we either have a relatively low-dimensional input vector, or we do not have temporal dependencies among elements of our input vector. RNNs, on the other hand, operate based on a shared-weight architecture across the time sequences to capture temporal dependencies. We describe the methodology for each module in Section 2. We then provide the performance analysis and bit error rate comparison with baseline solutions in Section 3. Additionally, we combine all the pre-trained modules to form an end-to-end unified framework for automated wide-band receiver and demonstrate how this combined framework improves on module-based training. Finally, we provide an optimized compression technique for FPGA implementation in Section 4 and conclude the paper in Section 5.

2. Receiver modules design

We consider a standard wide-band OFDM system whose receiver processing chain, shown in Fig. 1, is composed of smaller modules including synchronization, channel estimation and equalization, IQ symbol demodulation, and error correction. These blocks use the classical signal processing methods for wireless channel parameters estimation, noise reduction, and eventually efficient recovery of the transmitted symbols and bits. The fundamental assumption in such systems is that the multipath fading channel acts as a linear system and, therefore, the received signal after channel effects can be expressed in the form of a convolution between the transmitted signal $x(t)$ and channel $h(t)$ plus additive white Gaussian noise (AWGN) as $y(t) = h(t) * x(t) + n(t)$. The goal is to recover the transmitted symbols encoded in $x(t)$ from the received signal $y(t)$. We begin by briefly explaining what the standard pipeline accomplishes and then elaborate on each proposed module independently.

We denote a discrete time transmitted OFDM symbol in the time domain as $\boldsymbol{x}$, its sample at discrete time n as $\boldsymbol{x}[n]$, its FFT as $\boldsymbol{X}$, and its FFT at kth frequency bin as $\boldsymbol{X}[k]$. We use a similar notation for received OFDM symbol $\boldsymbol{y}$ and channel impulse response $\boldsymbol{h}$. Throughout the paper the coded bits and source bits are denoted as $\boldsymbol{c}$ and $\boldsymbol{b}$, respectively.

We decompose the receiver chain into four major units depicted in Fig. 2(a): (i) channel estimator, (ii) channel equalizer, (iii) demapper, and (iv) decoder. This pipeline can be described as follows. The received OFDM preamble $\boldsymbol{y}^{pre}$ is transformed into the frequency-domain via FFT. It is then multiplied by inverse of a diagonal matrix containing the transmitted BPSK-modulated preamble to predict the channel frequency response $\hat{H}$. Using the estimated channel, the received OFDM symbol $\boldsymbol{y}$ is equalized, after FFT, by getting multiplied by another diagonal matrix composed of inverses of the elements of $\hat{H}$ to produce noisy IQ symbols $\tilde{X}$. The noisy IQ symbols are then transformed to the most likely sequences of coded bits $\boldsymbol{c}$ (or their probability, or their log-likelihood ratio), and finally to the most likely source bits $\boldsymbol{b}$ through Viterbi algorithm.

We propose to meticulously replace these main modules with their neural network counterparts, as shown in Fig. 2(b). This design extends the traditional solution into a more generalizable framework that alleviates the shortcomings of the linear assumption by using the expressive power of neural networks in approximating nonlinear functions. The rest of this section covers the methodology behind each of the modules and their custom design.

2.1. Channel estimation & equalization

The classical channel estimation and equalization in an OFDM system is based on the least square (LS) estimation and convolution theorem. According to the LS estimation, the channel frequency response for each sub-band is calculated by averaging two channel estimates, each of which is computed by dividing the received preamble $\boldsymbol{Y}^{pre}_{[1,2]}$ by the transmitted preamble $\boldsymbol{X}^{pre}$ in the frequency domain, as:

$$\hat{H}[k] = \frac{1}{2}\sum_{i=1,2}\frac{Y_i^{pre}[k]}{X^{pre}[k]}, \text{ for } k = 1, \dots, 52. \tag{1}$$

where $\hat{H}[k]$ denotes the estimated frequency response of the channel at sub-band k. The BPSK modulated signal, X^{pre}, is replicated twice to compose the transmission preamble. The received preamble comprises Y_1^{pre} and Y_2^{pre} each corresponding to one copy of the X^{pre} going through the multipath fading channel. The estimated channel is then used to equalize the received signal, Y, and retrieve the unknown coded data symbols, X, in the frequency domain by applying the same LS estimation using the estimated channel frequency response, $\hat{H}$, from (1), as:

$$\hat{X}[k] = \frac{Y[k]\hat{H}[k]^*}{\hat{H}[k]\hat{H}[k]^* + \sigma_n^2}, \tag{2}$$

where σ_n^2 is the noise power of the received signal.

2.1.1. Channel estimation neural network

We propose to replace both channel estimation and channel equalization algorithms with neural network-based models that enhance the traditional solution and account for possible non-ideal effects such as non-linearities or non-Gaussianity of the channel. The motivation behind this is that in practice, we may have nonlinear components in the communication system (e.g., nonlinear amplifiers) that affect the channel impulse response and make the linear assumptions and, hence, the convolution theorem invalid when estimating the channel impulse response. We use an MLP architecture, denoted as Net_H that takes as input the received preamble $\boldsymbol{y}^{pre}$ in the time domain and estimates the channel response $\hat{H}$ in the output. Our strategy is to replace the traditional division in the frequency domain with a MLP to account for channel distortions and variations in the estimation process. Specifically, we use two parallel, identical and independent MLP structures modeling the real and imaginary parts of the channel, respectively. Both MLPs are fed with the received preamble signal $y^{\text{pre}} \in \mathbb{C}^{160}$ as inputs (in time domain, i.e., before FFT) and output the channel estimation $\hat{H}^{\nu}_{DNN} \in \mathbb{C}^{56}$, with ν denoting the *real* or *imaginary* part.

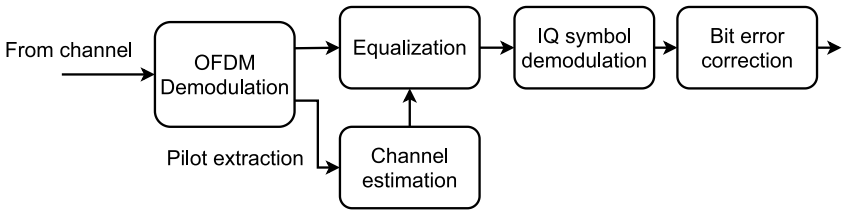


Fig. 1. Receiver chain for IEEE 802.11 Non-HT packet transmission.

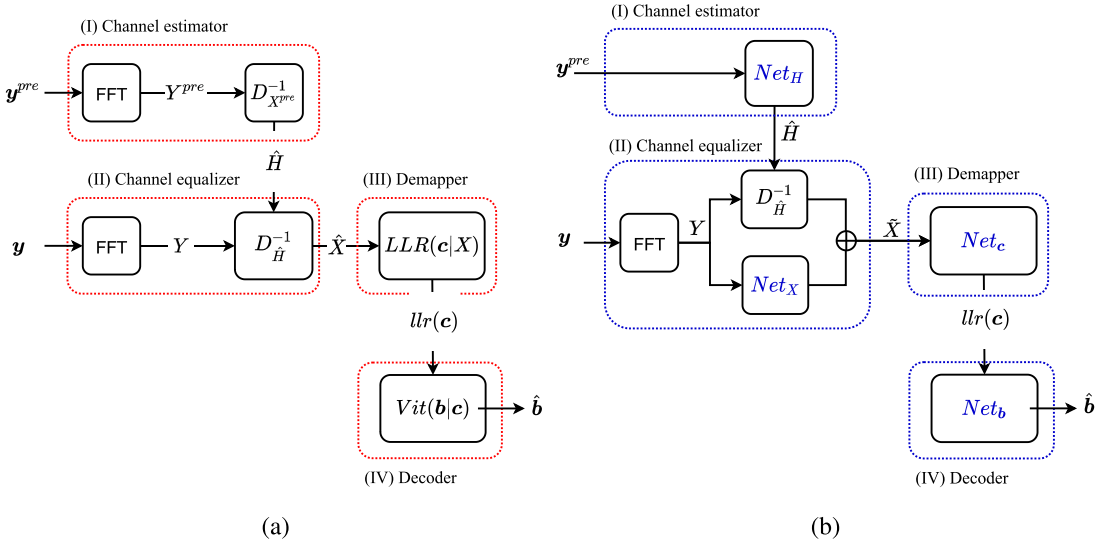


Fig. 2. (a) Traditional receiver pipeline. The FFT blocks denotes fast Fourier transform, the D_X^{-1} represents a diagonal matrix containing the inverses of vector X, the $LLR(.)$ is the log likelihood ratio, and $Vit(.)$ is the output of the Viterbi algorithm. (b) Proposed receiver pipeline. Four main modules are replaced with appropriate deep neural networks.

Our training approach consists of two strategies for making the model robust to (i) noise-plus-interference variations and (ii) signal power level. Specifically, training pilot signals are collected during simulation with noiseless channel instances and traditional LS estimation is used to compute channel state information (CSI), which is known to perform as an optimal minimum mean square error (MMSE) estimation under noiseless conditions. Similar to the noise model in [19], in order to make our model robust against noise variation, we augment the training data samples y^{pre} by adding artificial white Gaussian noise, with noise variance adjusted to produce different signal-to-noise ratio (SNR) levels. We refer to this approach as *de-noising training approach*. Our training data is composed of input–output pairs $\{y_i^{\text{pre}}, \hat{H}_{LS,i}^{v}\}_{i=1}^{N_s}$, with y_i^{pre} being the noiseless preamble signals and H_{LS}^{v} being the LS estimate of the channel in the noiseless case. The power of the Gaussian noise was selected randomly for each training batch to produce SNR values of $\{0, 10, 20, 30, 100\}$ dB.

Furthermore, we perform *RMS-normalization* of each input signal in the training batch. This normalization enforces each input signal to have a nominal power of 0 dBW and makes our framework immune to the power variations in the input signal due to user mobility and multipath propagation effects in real experimental setups. The RMS-normalization is computed as follows:

$$s_n = \frac{s}{\sqrt{\frac{\sum_{i=1}^{N} |s_i^2|}{N}}} \tag{3}$$

where s is the original signal, s_n is the resulting normalized signal, s_i is the ith complex symbol of the signal, and the denominator corresponds to the RMS factor. Once the pilot signals are normalized, the relative channel estimation output must be normalized as well. Therefore, the CSI estimation output from the DNN has to be scaled again using the same RMS factor, which is computed for each specific input signal.

Fig. 3 depicts the power normalization process in relation to the Channel Estimation DNN model at the time of inference. Both MLP models associated with the CSI estimation for real and imaginary domains are trained in a regression fashion. The Loss function adopted for training each MLP is the minimum square error (MSE) loss and is expressed on an input–output sample basis as:

$$L(H_{DNN}^{v}, H_{LS}^{v}) = \frac{1}{K}\sum_{k=1}^{K}\left(H_{DNN}^{v} - H_{LS}^{v}\right)^2, \quad v = \{\text{real, img}\} \tag{4}$$

with

$$H_{DNN}^{v} = \text{MLP}^{v}(y^{\text{pre}}). \tag{5}$$

For a detailed description regarding the architecture of both MLPs refer to Table 1.

2.1.2. Channel estimation correction neural network

The channel estimation correction model is a DNN that takes in both preamble and data signal. The goal of this module is to improve the accuracy of channel estimation by incorporating additional inputs. The structure of this DNN is shown in Fig. 4, where Net_H represents the previously explained channel estimation model, and PE, PC and EQ blocks are the phase estimation (PE), phase correction (PC) and linear equalization (EQ) units, respectively. The cpe unit represents the phase correction estimator and Y_c denotes the unequalized data corrected by the PC block. The NN block is the neural network for channel correction (see Table 1 for the details regarding the neural network architectures). The loss function is defined as:

$$L(\hat{X}, \hat{x}_{\text{pilot}}) = W \log(\mathcal{N}(\hat{x}_{\text{pilot}})) + (1 - W)\log(\mathcal{N}_{MM}(\hat{X})) \tag{6}$$

where $\log(\mathcal{N}(\hat{x}_{\text{pilot}}))$ is the log-likelihood of $\hat{x}_{\text{pilot}}$, which is assumed to have a Gaussian distribution, $\log(\mathcal{N}_{MM}(\hat{X}))$ is the log-likelihood of $\hat{X}$,

Table 1
Neural network architectures for different modules.

Channel estimator				
Layer	Input	Output	# weights	Activation
Dense	2 × 160	2 × 512	2 × 81920	ReLu
Dense	2 × 512	2 × 256	2 × 131072	ReLu
Dense	2 × 256	2 × 52	2 × 13312	–
Channel corrector				
Layer	Input	Output	# weights	Activation
Dense	8464	20	169280	Softplus
Dense	20	104	2080	-
Dense	208	104	21632	–
Channel equalizer				
Layer	Input	Output	# weights	Activation
Dense	96	96	9216	softplus
Dense	96	96	9216	-
Dense	192	96	18432	–
Demapper				
Layer	Input	Output	# weights	Activation
Dense	2	20	20	ReLu
Dense	20	4	80	Sigmoid
FEC Decoder				
Layer	Input	Output	# weights	Activation
RNN (3-layered GRU)	$(35 \times 2 + 20, 2)$	$(35 \times 2 + 20, 512)$	2.8M	Tanh & Sigmoid
Dense	$(20, 512)$	$(20, 16)$	8208	Relu
dense	$(20, 16)$	$(20, 1)$	17	Sigmoid

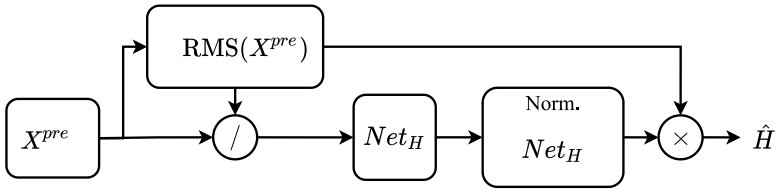


Fig. 3. RMS normalization scheme applied to the input and output of the channel estimation DNN model.

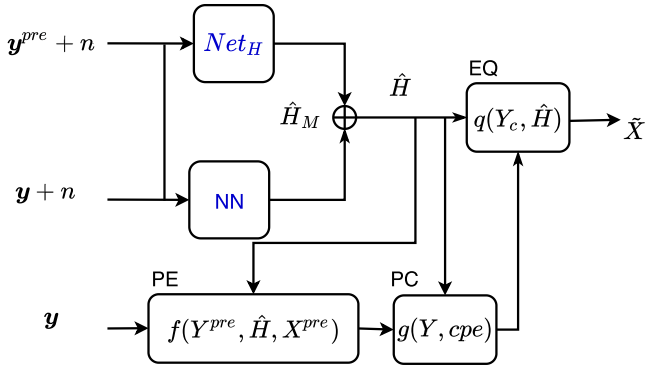


Fig. 4. Training block diagram for channel estimation correction model implemented with MLP.

and is assumed to have a Gaussian mixture distribution since the true value of the equalized signal is unknown, and W is a regularization weight to balance the influence of these two terms in the loss function. Here, $\log(\mathcal{N}_{MM}(\hat{X}))$ keeps the output $\hat{X}$ in the range of the classical estimation, while $\log(\mathcal{N}(\hat{x}_{\text{pilot}}))$ further refines the estimation accuracy of $\hat{X}$. Adopting the loss in equation (6), the equalization block is totally included in the loss function to achieve optimal performance directly on equalized symbol estimations rather than focusing on channel estimation alone.

2.1.3. Channel equalization neural network

The channel equalization neural network maps the OFDM received samples to their corresponding IQ symbols and is composed of a linear trend plus a nonlinear fluctuation. This module uses the estimated channel, provided by the channel estimation module, in its linear part and at the same time, models any nonlinear fluctuation with a parallel neural network structure (see Fig. 2(b)). This NN structure captures possible non-ideal effects such as non-linearities or non-Gaussianity of the channel. The model is initialized with the optimal least square solution under the assumption of linear Gaussian noise, therefore, all biases and weights of non-linear part were initialized with zeros.

2.2. IQ symbol demodulation

In a traditional OFDM system, a demodulator or demapper (see (III) in Fig. 2(a)) estimates the coded bits from equalized IQ symbols $\tilde{\boldsymbol{X}}$. The estimated coded bits can take the form of actual bits $\hat{\boldsymbol{c}}$, in which case we call it a hard demapper, or it can be the probability or the log-likelihood ratio of the coded bits, in which case we call it a soft demapper. In an M-QAM modulation, there are M origins or possible values for the IQ symbols. In the hard demapping approaches, the noise variance of equalized IQ symbols around those origins are calculated. Each of the IQ symbols is then assigned to one of these origins (hard decision) in such a way that the likelihood of them coming from a Gaussian distribution with IQ symbol value as its mean and noise variance as its variance is maximized. These hard decisions are converted back to the corresponding bits. In the soft demapping approach, a Log Likelihood

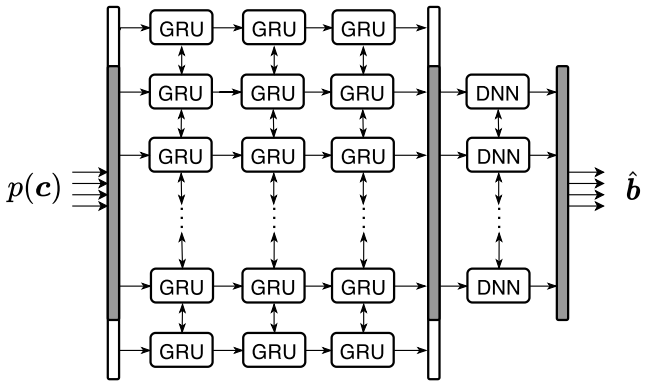


Fig. 5. Network model for RNN-based convolutional code decoder (Net_b) with three GRU layers.

Ratio (LLR) value is calculated for each bit, based on the probability of the bit as:

$$\mathrm{LLR}(\hat{c}) = \log\left(\frac{P(\hat{c}=0|\tilde{X})}{P(\hat{c}=1|\tilde{X})}\right). \tag{7}$$

We aim to replace the traditional demapper block with a neural network (denoted as Net_c in Fig. 2(a)) optimized for performing the demodulation task. The neural network architecture is a dense network with two fully connected layers (see Table 1 for details regarding the architecture). The input to the neural network is an equalized IQ symbol with I and Q (real and imaginary parts) coming in separate channels. For the loss function, we use *binary cross-entropy* which optimizes each output separately for true labels of '0' and '1' representing the true coded bits. Hence, the output of the neural network for each input symbol is a vector of size four corresponding to the probability of each bit to be '1'.

2.3. Forward error correction decoding

The IEEE 802.11 Non-HT transmission standard employs convolutional codes as its forward error correction (FEC) scheme and the well-known Viterbi algorithm for FEC decoding. The Viterbi decoder takes a sequence of coded symbols, either in the form of bits (hard decoding) or in the form of the probability of the bits (soft decoding), and returns a bit sequence as the estimate of source bits.

In the neural network decoder denoted as Net_b in Fig. 2(b), we aim at mimicking the operation of the traditional Viterbi decoding algorithm, and surpassing its performance via an RNN-based architecture. The RNN decoder consists of three RNN layers and two dense layers, as shown in Fig. 5 (see Table 1 for architecture details). The NN decoder takes the sequence of IQ symbol estimates per packet in the form of the posterior probability of coded bits $P(\hat{c}=1|\tilde{X})$ as input, and the probability of source bits $\boldsymbol{b}$ to be '1' (i.e., $P(\hat{b}=1|\tilde{X})$) as output. As we already have access to the log likelihood ratio (LLR) values of coded bits $\mathrm{LLR}(\boldsymbol{c})$ (discussed in the previous subsection), we compute the posterior probability as:

$$P(\hat{c}=1|\tilde{X}) = \frac{\exp(-\mathrm{LLR}(\hat{c}))}{1+\exp(-\mathrm{LLR}(\hat{c}))}, \tag{8}$$

where the minus before LLR(.) comes from the definition of LLR in Eq. (7). For the loss function, we consider the binary cross-entropy, just the same as the IQ demodulation.

The input vector to the RNN is the posterior of coded bits with the size $(2l_{\mathrm{aux}} + l_{\mathrm{data}}, 2)$. The l_{data} is the length of the bit sequence to be decoded (the gray segments in Fig. 5), and it is padded by the heading and tailing l_{aux} auxiliary bits (the white segments in Fig. 5). The output is of size $(2l_{\mathrm{aux}} + l_{\mathrm{data}}, l_{\mathrm{hidden}})$, which partially serves as the input to the following DNN layer (i.e., dense layer). The input vector to the DNN is of size $(l_{\mathrm{data}}, l_{\mathrm{hidden}})$ and the output are the source bit estimates with the size of $(l_{\mathrm{data}}, 1)$. It is worth noting that the DNN is applied to each time-step, which is the first axis of the RNN output. The reason to add padded auxiliary bits is that the source bit $b[m]$ is correlated with coded bits $\boldsymbol{c}[2\ m-k\ :\ 2\ m+k]$ (in case of $\frac{1}{2}$ coding rate and k is an integer related to the constraint length of convolutional codes). In this paper, l_{aux}, l_{data}, and l_{hidden} are given in Table 1.

3. Experimental results

In this section, we describe our data collection platform, provide details on our experimental settings, and demonstrate the performance evaluation results for each proposed module separately. We then introduce a combined-module training procedure and report its corresponding performance results.

3.1. Data simulation and data collection

We trained and tested our NN modules on two sets of data: simulated and real (over-the-air). We generated the simulated data using the implementation of an OFDM-based wireless LAN based on IEEE 802.11 standards by WLAN Toolbox™ in MATLAB. In addition, we collected real data using Arena [17], a software-defined radio (SDR) testbed that covers an indoor office area of 2240 sqft within the Interdisciplinary Science and Engineering Complex (ISEC) located in the Northeastern University's main campus. We collected more than 16k packets via an OFDM transceiver system with a bandwidth of 10 MHz and a modulation scheme of 16-QAM. Specifically, we apply the GNU Radio IEEE 802.11 a/g transceiver [20] that operates on the Arena testbeds. The SDR framework works similarly to the MATLAB framework. It is worthwhile to note that the data collected in the Arena testbed includes real RF transceivers which transmit/receive baseband waveforms over the wireless channel. The collected over-the-air dataset is processed by the MATLAB functions to obtain intermediate-level data. We use the same processing pipeline to compute the baseline performance for both simulated and over-the-air data.

3.2. Channel estimation & equalization

We analyzed the channel estimation and equalization modules independently and then jointly as presented in Fig. 6. For training the channel estimation neural network, we simulated 10k OFDM packets using MATLAB WLAN toolbox. The simulator at this step, comprises a multi-path fading channel with no additive white Gaussian noise (AWGN) so that the LS estimator of the channel yields the exact estimation for the regressand in the model. The received OFDM symbols of the legacy long training field (L-LTF) preamble extracted from these packets together with the exact estimation of the channel (i.e., LS estimation under no noise) constitute our training data. To account for a noisy channel, noise at various power levels with SNRs from 0 to 30 dB is added to the received OFDM symbols. The bit error rate (BER) and MSE performances of the model were tested on 500 packets for each SNR level from 4 dB to 30 dB (see Fig. 6). For channel equalization, the model was trained with the received OFDM payloads from 1k packets of SNR = 4 dB, and tested with 5k wifi packets for each SNR level from 4 dB to 30 dB.

Figs. 6(a) and 6(b) summarize the results in terms of IQ symbol MSE and BER for different experiments. Specifically, we compare four scenarios: (i) Baseline (dashed black line), i.e., LS channel estimation and equalization; (ii) LS channel estimation followed by an NN-based equalization (green line with dot marker); (iii) an NN-based channel estimation followed by traditional LS equalization (blue line with asterisk marker); and (iv) NN-based strategy for both channel estimation and equalization (red dashed line with diamond marker). By analyzing the MSE results depicted in Fig. 6(a) one can note that the main performance gain was obtained when using the NN-based channel

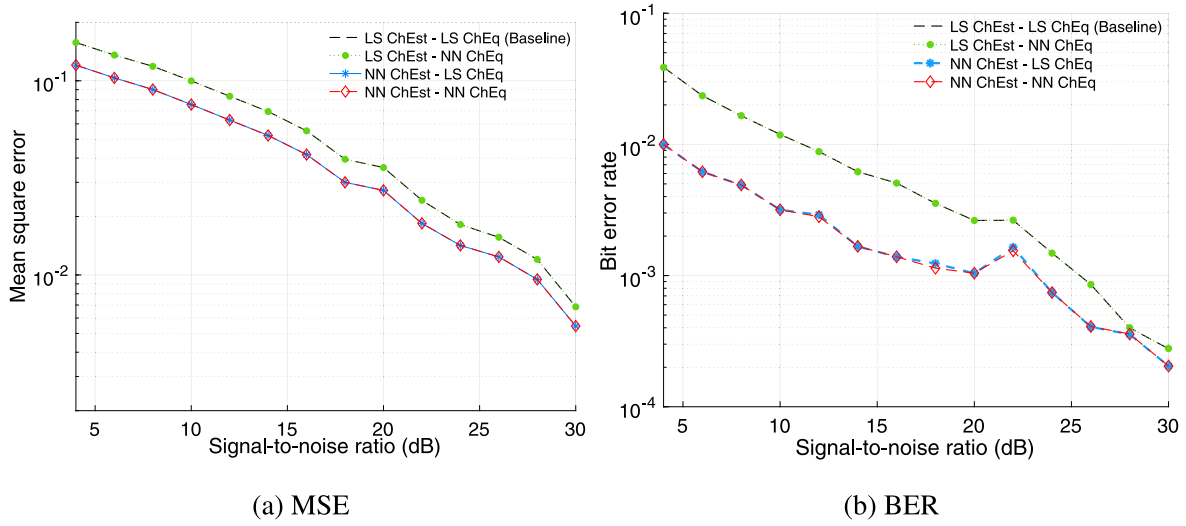


Fig. 6. Comparison of MSE and BER for proposed NN model substituting channel estimation and equalization blocks in classical OFDM receiver pipeline.

estimation strategy, which led to significant improvements, while the influence of the NN-based channel equalization is negligible. A similar behavior is also observed in Fig. 6(b) in terms of BER results. It is important, however, to note that: (i) these results are expected since no non-ideal effects were considered in these experiments and (ii) the proposed NN-based methodologies were able to at least reproduce the results of the traditional methodologies.

3.3. IQ symbol demodulation

We generated 16k packets per SNR level for the range of 2 dB to 30 dB with steps of 2 dB and saved inputs of the mapper at the transmit side, i.e., IQ symbols in the frequency domain, as the labels and equalized symbols after demapper at the receiver side as the input features for the neural network. We used 90% of this dataset for training and kept the remaining 10% for validation. Similarly, we collected 16k packets per SNR for testing the model. To account for the inevitable conversion of IQ symbols to bits at the next stage, the output of the neural network is a vector of four elements for 16QAM modulation whose elements determine the probability of each bit to be '1'. Given this probability at the output of the neural network, we can calculate four LLR values for each equalized symbol that are fed to the following module through Eq. (7). These LLR values are later used in MATLAB for calculating BER with the neural network module as its demapper.

Fig. 7 shows Bit Error rate (BER) for different SNR levels between 2 dB and 30 dB calculated using two methods. The dashed black curve shows the BER calculated using MATLAB WLAN toolbox. The green curve with 'o' marker shows BER where LLRs are calculated using a neural network trained on all the SNR levels and tested on all SNR levels. The green curve with dot marker shows BER where LLRs are calculated using a neural network that has been trained with data only from SNR=2 dB and tested on all SNR levels. The results show up to 36% improvement (decrease in BER) in the neural network method (green curve with dot) compared to the traditional MATLAB method (dashed black curve). It is worth noting that low SNR data provides more challenging instances for the NN model to train over, hence the significant improvement is achieved in comparison with the NN model that partially trained with higher SNR data. We also searched the design space for the optimized model by changing the number of layers and neurons in each layer. According to Fig. 8, the smallest model without compromising the performance has two layers with twenty neurons in the first layer.

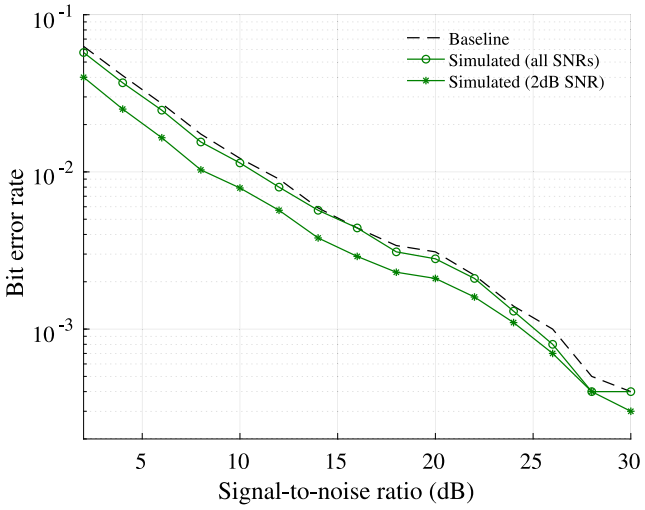


Fig. 7. BER versus SNR calculated using IEEE 802.11 WLAN standard pipeline in MATLAB and our deep learning-based receiver.

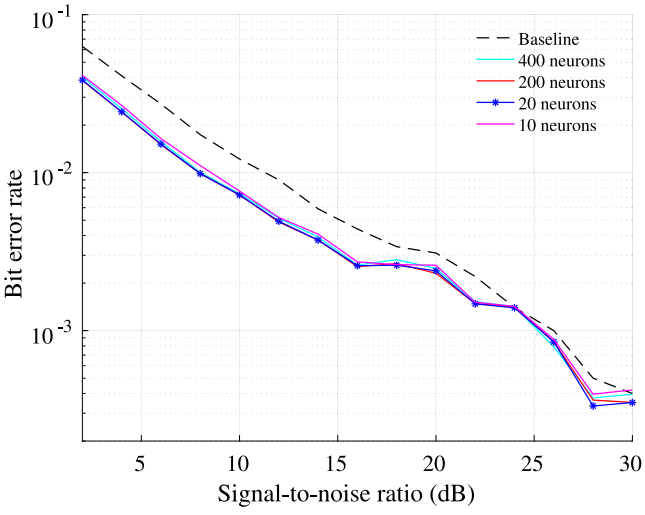


Fig. 8. Model size optimization.

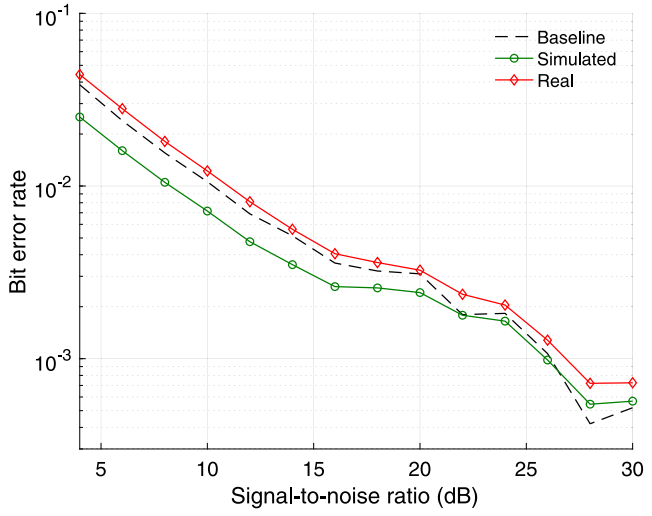


Fig. 9. BER versus SNR calculated using various RNN decoder models, trained on the simulated and real (over-the-air) data separately, and tested on the MATLAB data.

3.4. Forward error correction decoding

To train the RNN decoder, we generated 20k simulated WiFi packets with SNR=2 dB. We also collected nearly 16k the over-the-air data packet from the Arena platform. The input sequences to the RNN decoder are the posterior of coded bits $\{P(\hat{c}_1 = 1|\hat{X}_1), \ldots, P(\hat{c}_M = 1|\hat{X}_N)\}$, and the corresponding outputs are the estimates of source bits $\{b_1, b_2, \ldots, b_L\}$. Since the decoder is a sequential data processing module, we split our long packets into shorter sequences and then feed shorter sequences to RNN model. We first reshaped the length L packets into $[\frac{L}{2}, 2]$ tensor and then the tensor are split into $[2 * 35 + 20, 2]$ tensors. The shorter tensors are finally fed to the RNN model. Fig. 9 shows BER for different SNR levels between 4 dB and 30 dB calculated using the RNN model trained on the simulated and over-the-air data separately and tested on the simulated data. We observe that the model trained on the simulated data achieves better performance because the simulated data contains a wider range of SNR levels and hence provides more complex samples for training.

3.5. Combined NN-based receiver pipeline

As the final step, we combined all the pre-trained modules to train an end-to-end unified framework for automated wide-band receiver. To this end, we implemented all the required intermediate functions such as interleaver and noise power estimator to build a stand-alone receiver that automatically decodes received WiFi packets. We then collect simulated and emulated real data to carry out two sets of fine-tuning on the unified structure comprising all the pre-trained NN models. For fine-tuning with simulated data, we collected 12k simulated data packets per SNR level for training and additional 8k simulated packets for testing the model. Similarly, for fine-tuning with emulated real data, we collected 4k real data packets per SNR level for training and additional 4k packets for testing. The real data was collected from the Colosseum, which is a powerful wireless network emulator located in the Northeastern Burlington campus [18]. To have the advantage of training the model with packets from a challenging real-world scenario, we collected the real data in a non-line-of-sight (NLOS) fashion.

The fine-tuning results with simulated and emulated real data are illustrated in Fig. 10. As before, the dashed black curves represent the baselines. The blue curves with larger 'o' markers represent the performance of the entire pipeline initialized with the pre-trained modules. In other words, we pass the received IQ symbols through each of the pre-trained NN models and calculate the BER to obtain the mentioned curve. The blue curves with larger square marker show the performance of the model after additional training using simulated data (Fig. 10(a)) and emulated real data (Fig. 10(b)). We also visualized the performance of the combined NN model initialized with all the pre-trained modules but the RNN decoder, denoted by the red curves with small 'o' markers. Similarly, the performance of the fine-tuned model containing all the pre-trained modules but the RNN decoder is shown by the red curves with small square markers. For these cases, in which we replaced the RNN decoder with the standard Viterbi algorithm, we observed that the Viterbi decoder combined with the rest of the NN modules also provides a competitive performance. The curves labeled as *initial* represent the BER for the combined model before fine-tuning, and the curves labeled as *trained* shows the performance of the model after fine-tuning with simulated or real data. Interestingly, we observed that the bit error rate of the model before fine-tuning is higher than the corresponding baseline for the real data (Fig. 10(b)). This is because the new Colosseum data collected in an NLOS environment is more challenging, and it has not been seen by the model. However, after fine-tuning by the mentioned real data, we see a considerable improvement in terms of bit error rate over the baseline. We also see that the end-to-end NN-based model achieves a better performance compared with its counterpart where RNN is replaced with the Viterbi decoder.

4. Compression for FPGA implementation

Our final goal is to prepare our architectures for an optimized FPGA implementation. FPGA is not a flexible platform for online adaptation in comparison with mobile CPUs and GPUs which are supported by compiler-level optimization. Therefore, in order to make our modules, which are composed of various architecture (i.e. RNN and MLP), compatible for implementation over FPGA platform, we need to perform pruning and quantization, and co-optimize the joint scheme with FPGA architecture and compilation. We use our proposed framework that consist of two main steps: pruning and quantization. The former reduces the number of weights to be stored, and the latter saves up memory space required to store each weight. In designing of our framework, we have exploited the unique characteristics of FPGA platforms: (i) the capability to support various quantization schemes, including fixed point [21] and power-of-two (Po2) [22] number systems, and (ii) the flexibility in accommodating computations using multiple on-chip computing resources such as DSPs and look-up tables (LUTs). This framework as a whole obtains high parallelism and efficiency level and preserves the accuracy performance of the original NN models.

4.1. BCR pruning scheme

There are a couple of well-established approaches to compress and accelerate deep neural networks in the literature: structured pruning [23–25] and irregular pruning [26]. In structured pruning, the entire filter or channel is removed. Therefore, while it is intuitive that it advantages the hardware acceleration, it decreases the accuracy. In irregular pruning, weights with small magnitudes at arbitrary locations are removed, which preserves accuracy but generally cannot attain acceleration on most hardware platforms. For the purpose of optimizing our multi-architecture decoder network, we use our proposed Block-based Column-Row (BCR) pruning that can serve as the universal, fine-grained structured pruning scheme. BCR pruning is a general framework applicable to both convolutional layers with different kernel sizes and fully connected layers, which are popular in the RNN and Transformer models. As shown in Fig. 11, the weight matrix is divided into a number of blocks, and then independent row and column pruning are applied to each block. Finally, the ADMM-based pruning framework [27] is employed as the BCR pruning algorithm, which can determine the row/column pruning ratio for each block automatically.

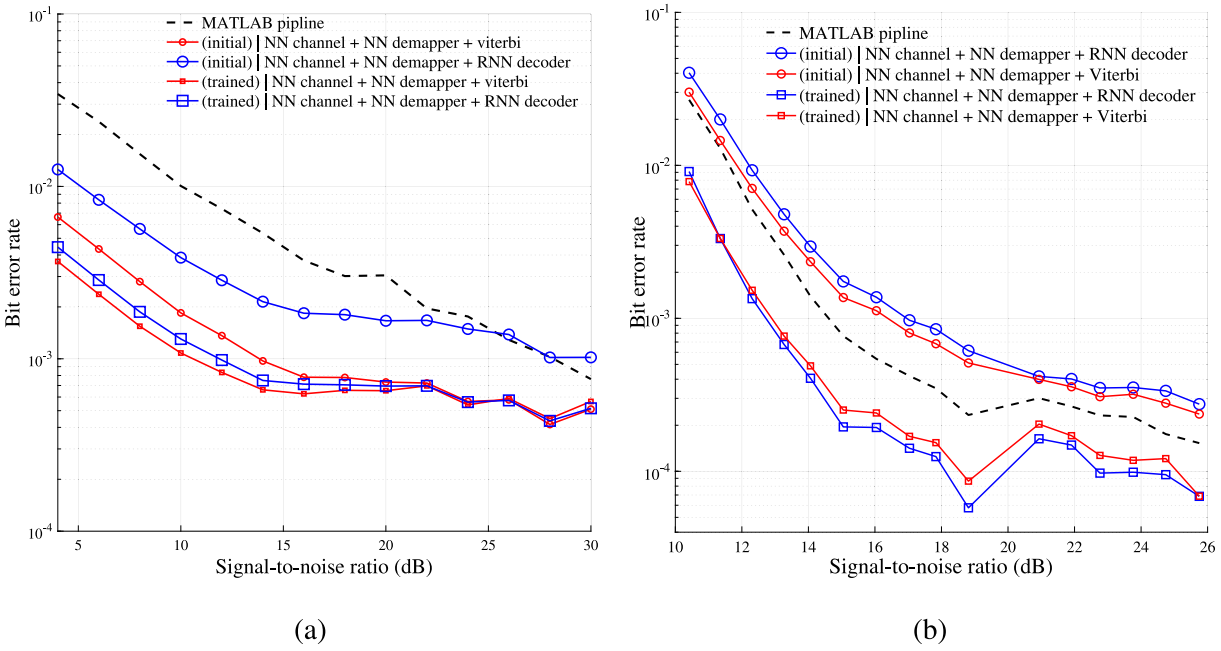


Fig. 10. Combined NN model fine-tuned on (a) the simulated data and (b) the emulated real data. The curves that are labeled with (initial) represent the BER for the combined model before fine-tuning, and the curves that are labeled with (trained) shows the performance of the model after fine-tuning with simulated or emulated real data.

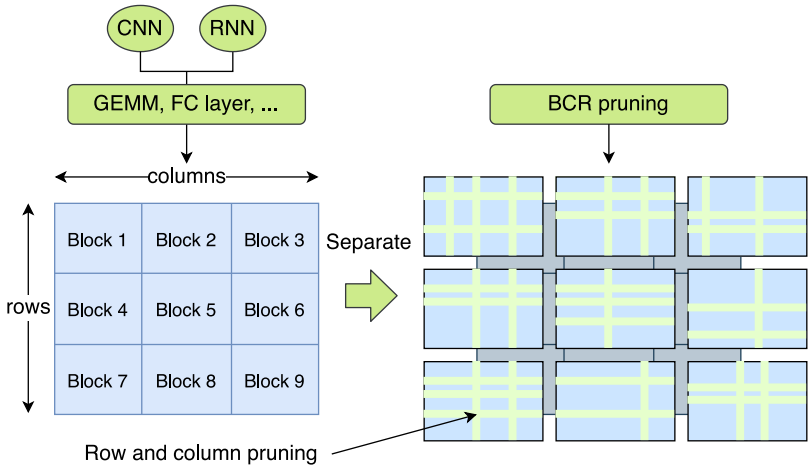


Fig. 11. The proposed Block-based Column-Row (BCR) pruning scheme. Weight matrix is divided into blocks in fixed size, then we apply column and row pruning to each individual block.

4.2. Mixed scheme quantization (MSQ)

Weight quantization is a necessary step for optimized implementation of NN modules on a hardware platform. A naive approach, called *fixed-point quantization* scheme, can be used to prepare the NN module for implementation on a fixed-point hardware, namely FPGA. A more efficient approach is *power-of-two (Po2)* [22] in which multiplication arithmetic is replaced by bit-shifting that can be realized through configurable resources, e.g., LUTs. Both approaches have advantages that we exploit jointly in our hardware-friendly quantization scheme, *mixed scheme quantization* (MSQ). To apply quantization on different rows of the weight matrix, MSQ choose between the two mentioned schemes in the following manner. Since Po2 scheme has higher resolution around the center, it is applied to the rows with lower variance. In contrast, the fixed-point quantization is applied to the rows whose weights are uniformly distributed, i.e., that have higher variance. Furthermore, MSQ facilitates a heterogeneous utilization of FPGA hardware resources by executing different rows in parallel on the corresponding computation module. By representing different weight distributions, MSQ maintains or even achieves higher accuracy performance compared to each of its building components.

4.3. Experimental results

We apply our BCR pruning approach followed by MSQ 8-bit quantization to all of the NN modules, trained on Nvidia RTX 2080Ti GPUs, to verify their effectiveness using different pruning rates. The pruning rate refers to the sparsity of the pruned model with respect to the original one. The compression rate for quantization can be simply derived by the final bit-width. For instance, a 4-bit quantized model is compressed eight times (i.e., 8×) compared to a 32-bit floating point model.

4.3.1. Individual pre-trained model compression

The demodulation model (NN-based demapper) consists of two fully connected layers, which is a very compact architecture. We performed 3-bit and 4-bit quantization to achieve 10.7× and 8× overall compression rate (see Fig. 12). The 4-bit quantization preserves accuracy well,

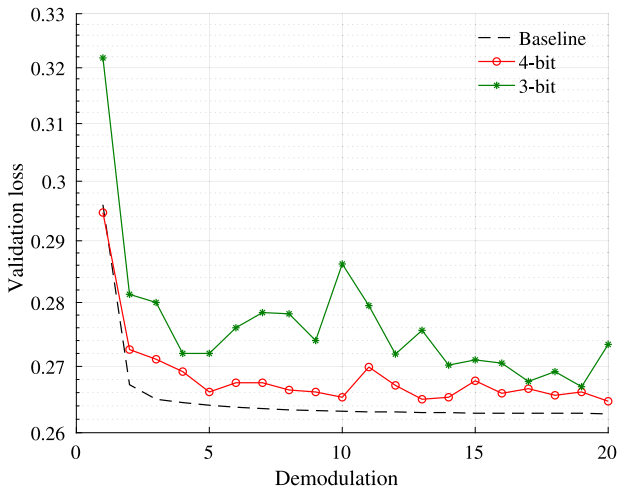


Fig. 12. Accuracy performance of compressed networks.

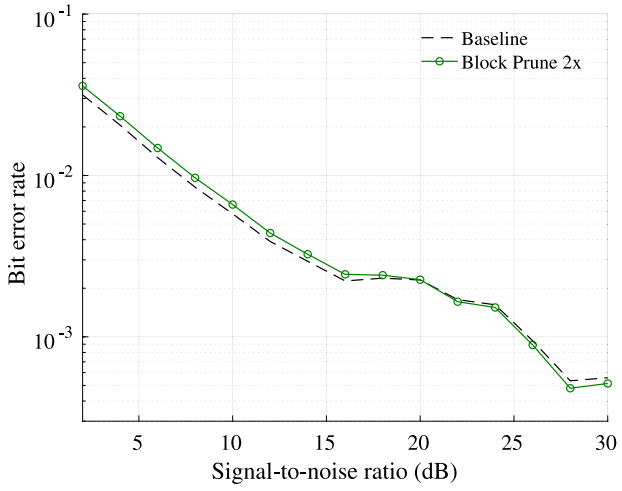


Fig. 13. BCR pruning results on the decoding NN-based model.

Table 2
Compression techniques, overall compression rate and FPGA speedup of our neural networks.

	Demodulation	Decoding
Model type	Dense	RNN+Dense
Architecture	Linear (2, 20) Linear (20, 4)	GRU (2, 256, 3) Linear (512, 16) Linear (16, 1)
MACs	1.2×10^2	5.5×10^6
Pruning Rate	1.0×	2.0×
Weight bit-width	4	8
Overall size compression	8×	8×
Baseline latency (μs)	2.02	204.12
Compressed latency (μs)	1.15	93.56

while the 3-bit quantization results in a slight degradation. For our NN-based bit decoder model, which consists of both RNN and fully connected layers, we apply the proposed BCR pruning to compress the decoding model by 2×, which results in a comparable (in low SNR levels) or even better (in high SNR levels) bit error rates (see Fig. 13).

Finally, as shown in Table 2, for both of the demodulation and decoding modules, we observe that the proposed pruning and quantization approach accelerates inference with negligible accuracy loss by

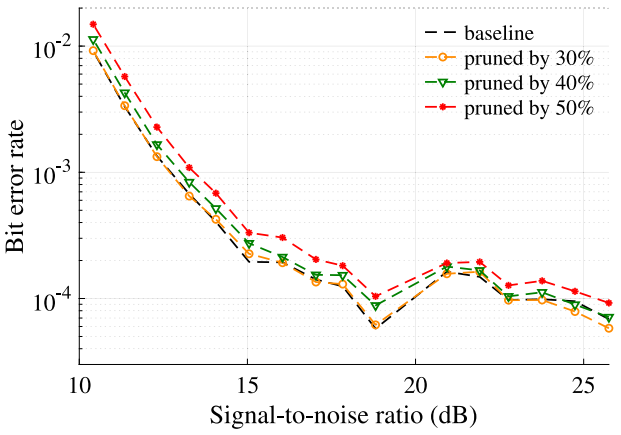


Fig. 14. BCR pruning results on the end-to-end fine-tuned model.

Table 3
Compression latency for the end-to-end fine-tuned model.

	Channel estimation	Demapper	RNN
Frequency (MHz)	100	100	100
Latency Baseline	6.84 ms	8.19 μs	210.04 μs
Latency Compressed	1.22 ms	1.49 μs	38.19 μs

decreasing the latency from 2.02 μs to 1.15 μs, and from 204.12 μs to 93.56 μs, respectively.

4.3.2. Combined fine-tuned model compression

We also applied our compression techniques on the end-to-end fine-tuned model and demonstrated their compression results. We used an 8-bit quantization equivalent to 4× compression with multiple pruning ratios. We achieved lossless performance at about 50% computational complexity, i.e., 8× compression in total with quantization (see Fig. 14). We also verified the inference latency based on Vivado HLS on the USRP X310 platforms which shows a significant acceleration (see Table 3).

5. Conclusion

We proposed a modular and parameter-efficient physical layer receiver for an OFDM-based wireless system by extending classical signal processing techniques for wireless communications to a more modern approach by leveraging deep neural network models. Our detailed modular design includes various NN architectures suited for each wireless module. Furthermore, in our designing, we took into account the proper domain knowledge to avoid designing a mere black-box model, and to prevent over-parameterized models. We trained our model using a variety of simulated and real data. Our results demonstrate that the proposed framework outperforms the classical system in terms of bit error rate over a large range of SNR levels.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Deniz Erdogmus reports financial support was provided by Defense Advanced Research Projects Agency. Kaushik Chowdhury reports financial support was provided by National Science Foundation. Tommaso Melodia reports financial support was provided by National Science Foundation. Pau Closas reports financial support was provided by National Science Foundation.

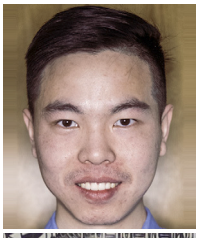
Acknowledgments

This work is supported by Defense Advanced Research Projects Agency, USA (SPiNN HR00112090055, LwLL SC1821301) and National Science Foundation, USA (CNS-1923789, CNS-1925601, ECCS-1845833).

References

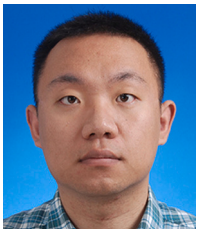
[1] D. Tse, P. Viswanath, Fundamentals of Wireless Communication, Cambridge University Press, 2005.
[2] R.G. Gallager, Principles of Digital Communication, Cambridge University Press, 2008.
[3] E. Yamazaki, N. Farsad, A. Goldsmith, Low noise non-linear equalization using neural networks and belief propagation, 2019, arXiv preprint arXiv:1905.04893.
[4] X. Gao, S. Jin, C.-K. Wen, G.Y. Li, Comnet: Combination of deep learning and expert knowledge in OFDM receivers, IEEE Commun. Lett. 22 (12) (2018) 2627–2630.
[5] H. Ye, L. Liang, G.Y. Li, B.-H. Juang, Deep learning-based end-to-end wireless communication systems with conditional GANs as unknown channels, IEEE Trans. Wireless Commun. 19 (5) (2020) 3133–3143.
[6] P. Jiang, T. Wang, B. Han, X. Gao, J. Zhang, C.-K. Wen, S. Jin, G.Y. Li, AI-aided online adaptive OFDM receiver: Design and experimental results, IEEE Trans. Wireless Commun. 20 (11) (2021) 7655–7668.
[7] N. Shlezinger, N. Farsad, Y.C. Eldar, A.J. Goldsmith, ViterbiNet: A deep learning based viterbi algorithm for symbol detection, IEEE Trans. Wireless Commun. 19 (5) (2020) 3319–3331.
[8] A. Zappone, M. Di Renzo, M. Debbah, Wireless networks design in the era of deep learning: Model-based, AI-based, or both? IEEE Trans. Commun. 67 (10) (2019) 7331–7376.
[9] S. D'Oro, F. Restuccia, T. Melodia, Can you fix my neural network? Real-time adaptive waveform synthesis for resilient wireless signal classification, in: IEEE INFOCOM 2021-IEEE Conference on Computer Communications, IEEE, 2021, pp. 1–10.
[10] H. Ye, G.Y. Li, B.-H. Juang, Power of deep learning for channel estimation and signal detection in OFDM systems, IEEE Wirel. Commun. Lett. 7 (1) (2017) 114–117.
[11] Z. Qin, H. Ye, G.Y. Li, B.-H.F. Juang, Deep learning in physical layer communications, IEEE Wirel. Commun. 26 (2) (2019) 93–99.
[12] Q. Hu, F. Gao, H. Zhang, S. Jin, G.Y. Li, Deep learning for channel estimation: Interpretation, performance, and comparison, IEEE Trans. Wireless Commun. 20 (4) (2021) 2398–2412.
[13] H. Kim, Y. Jiang, R.B. Rana, S. Kannan, S. Oh, P. Viswanath, Communication algorithms via deep learning, in: International Conference on Learning Representations, 2018.
[14] D. Tandler, S. Dörner, S. Cammerer, S. ten Brink, On recurrent neural networks for sequence-based processing in communications, in: 2019 53rd Asilomar Conference on Signals, Systems, and Computers, IEEE, 2019, pp. 537–543.
[15] T. O'shea, J. Hoydis, An introduction to deep learning for the physical layer, IEEE Trans. Cogn. Commun. Netw. 3 (4) (2017) 563–575.
[16] S. Dörner, S. Cammerer, J. Hoydis, S. Ten Brink, Deep learning based communication over the air, IEEE J. Sel. Top. Sign. Proces. 12 (1) (2017) 132–143.
[17] L. Bertizzolo, L. Bonati, E. Demirors, T. Melodia, Arena: A 64-antenna SDR-based ceiling grid testbed for sub-6 GHz radio spectrum research, in: Proceedings of the 13th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization, 2019, pp. 5–12.
[18] L. Bonati, P. Johari, M. Polese, S. D'Oro, S. Mohanti, M. Tehrani-Moayyed, D. Villa, S. Shrivastava, C. Tassie, K. Yoder, A. Bagga, P. Patel, V. Petkov, M. Seltser, F. Restuccia, A. Gosain, K.R. Chowdhury, S. Basagni, T. Melodia, Colosseum: Large-scale wireless experimentation through hardware-in-the-loop network emulation, in: Proc. of IEEE Intl. Symp. on Dynamic Spectrum Access Networks (DySPAN), Virtual Conference, 2021.
[19] N. Soltani, K. Sankhe, J. Dy, S. Ioannidis, K. Chowdhury, More is better: Data augmentation for channel-resilient rf fingerprinting, IEEE Commun. Mag. 58 (10) (2020) 66–72.
[20] B. Bloessl, M. Segata, C. Sommer, F. Dressler, An IEEE 802.11 a/g/p OFDM receiver for GNU radio, in: Proceedings of the Second Workshop on Software Radio Implementation Forum, 2013, pp. 9–16.
[21] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, Y. Zou, Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients, 2016, arXiv preprint arXiv:1606.06160.
[22] D. Miyashita, E.H. Lee, B. Murmann, Convolutional neural networks using logarithmic data representation, 2016, arXiv preprint arXiv:1603.01025.
[23] W. Wen, C. Wu, Y. Wang, Y. Chen, H. Li, Learning structured sparsity in deep neural networks, Adv. Neural Inf. Process. Syst. 29 (2016).
[24] Y. He, X. Zhang, J. Sun, Channel pruning for accelerating very deep neural networks, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 1389–1397.
[25] J.-H. Luo, J. Wu, W. Lin, Thinet: A filter level pruning method for deep neural network compression, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 5058–5066.
[26] S. Han, J. Pool, J. Tran, W.J. Dally, Learning both weights and connections for efficient neural networks, in: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1, NIPS '15, MIT Press, Cambridge, MA, USA, 2015, pp. 1135–1143.
[27] A. Ren, T. Zhang, S. Ye, J. Li, W. Xu, X. Qian, X. Lin, Y. Wang, Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers, in: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, 2019, pp. 925–938.

Bahar Azari is a Ph.D. student at the center for signal processing, imaging, reasoning, and learning (SPIRAL) of Northeastern University. She received her B.Sc. in electrical engineering from the Amirkabir University of Technology in 2011 and her M.Sc. in telecommunications engineering from Politecnico di Milano in 2014. Her research interests include applied signal processing and machine learning with expertise in deep generative models and latent variable models.

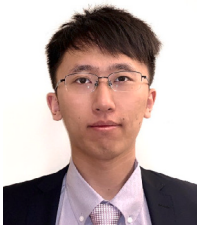


Hai Cheng is a Ph.D. candidate in Computer Engineering at the Institute for Wireless IoT at Northeastern University. He received his B.Eng degree in 2015 from Xidian University, China, and master degree in 2018 from ShanghaiTech University, China. His research interests include machine learning and optimization in wireless network systems.

Nasim Soltani is a Ph.D. candidate at the Institute for Wireless IoT at Northeastern University, advised by professor Chowdhury. Her area of interest is AI-aided algorithms for applications in the physical layer of wireless communications systems.



Haoqing Li is a Ph.D. candidate in Electrical and Computer Engineering at Northeastern University, Boston, MA. got his BS degree in Electrical Engineering from Wuhan University, China and MS degree in Electrical and Computer Engineering at Northeastern University, Boston, MA. His research interests include GNSS signal processing, anti-jamming technology and robust statistics.



Yanyu Li is a Ph.D. candidate at the Department of Electrical and Computer Engineering in Northeastern University, advised by Professor Yanzhi Wang. His research interests include deep learning, neural network architecture search, pruning and quantization.

Mauro Belgiovine is pursuing his Ph.D. at the Electrical and Computer Engineering department at Northeastern University, under the guidance of Professor Kaushik Chowdhury. His current research interests involve deep learning, wireless communications, and heterogeneous computing.

Tales Imbiriba received his Doctorate degree from the Department of Electrical Engineering (DEE) of the Federal University of Santa Catarina (UFSC), Florian\'opolis, Brazil, in 2016. He served as a Postdoctoral Researcher at the DEE--UFSC and is currently a Postdoctoral Researcher at the ECE dept. of the Northeastern University, Boston, MA, USA. His research interests include audio and image processing, pattern recognition, kernel methods, adaptive filtering, and Bayesian Inference.

Salvatore D'Oro is a Research Assistant Professor with the Institute for the Wireless IoT at Northeastern University, USA. He received his Ph.D. from the University of Catania in 2015. He serves on the technical program committee of IEEE INFOCOM and the Elevier Computer Communications journal. His research interests include optimization and learning in NextG systems.

Tommaso Melodia is a Professor at Northeastern University. He has been named William Lincoln Smith Professor in recognition of his significant research contributions and exceptional leadership in the field of electrical and computer engineering. He is the Director of the Institute for the Wireless IoT, and the Director of Research for the PAWR Project Office. He received his Ph.D. degree in Electrical and Computer Engineering from Georgia Institute of Technology in 2007. His research focuses on modeling, optimization, and experimental evaluation of wireless networked systems. He serves as Editor-in-Chief for Computer Networks.

Yanzhi Wang is currently an Assistant Professor at the Department of ECE at Northeastern University, Boston, MA. His research focuses on model compression and platform-specific acceleration of deep learning architectures, maintaining the highest model compression rates on representative DNNs. He received the U.S. Army Young Investigator Program Award (YIP), Massachusetts Acorn Innovation Award, Ming Hsieh Scholar Award, and other research awards from Google, MathWorks, etc. His recent research achievement, CoCoPIE, can achieve real-time performance on almost all deep learning applications using off-the-shelf mobile devices, outperforming competing frameworks by up to 180X acceleration.

Pau Closas is an Assistant Professor at Northeastern University, Boston, MA. He received the MS and Ph.D. degrees in Electrical Engineering from UPC in 2003 and 2009. He also holds a MS in Advanced Mathematics from UPC, 2014. His primary areas of interest include statistical signal processing, robust stochastic filtering, and machine learning, with applications to positioning systems and wireless communications. He is the recipient of the 2014 EURASIP Best Ph.D. Thesis Award, the 9th Duran Farell Award, the 2016 ION Early Achievements Award, and a 2019 NSF CAREER Award.

Kaushik Chowdhury (M'09-SM'15) is a Professor at Northeastern University, Boston, MA. He received his Ph.D. degree from Georgia Institute of Technology in 2009. His current research interests involve systems aspects of networked robotics, machine learning for agile spectrum sensing/access, wireless energy transfer, and large-scale experimental deployment of emerging wireless technologies.

Deniz Erdogmus (Senior Member, IEEE) is a Professor of ECE at Northeastern University, Boston, MA. He received his Ph.D. degree in electrical and computer engineering from the University of Florida, Gainesville, FL, in 2002. He held a postdoctoral position at the University of Florida, until 2004. His research focuses on statistical signal processing and machine learning with applications to contextual signal/image/data analysis with applications in cyber-human systems.