A Deep Learning Wireless Transceiver with Fully Learned Modulation and Synchronization

Johannes Schmitz, Caspar von Lengerke, Nikita Airee,
Arash Behboodi, Rudolf Mathar
Institute for Theoretical Information Technology
RWTH Aachen University
52074 Aachen, Germany

***Abstract*—In this paper, we present a deep learning based wireless transceiver. We describe in detail the corresponding artificial neural network architecture, the training process, and report on excessive over-the-air measurement results. We employ the end-to-end training approach with an autoencoder model that includes a channel model in the middle layers as previously proposed in the literature. In contrast to other state-of-the-art results, our architecture supports learning time synchronization without any manually designed signal processing operations. Moreover, the neural transceiver has been tested over the air with an implementation in software defined radio. Our experimental results for the implemented single antenna system demonstrate a raw bit-rate of 0.5 million bits per second. This exceeds results from comparable systems presented in the literature and suggests the feasibility of high throughput deep learning transceivers.**

***Index Terms*—Deep Learning, Transceiver, Wireless Communication, Synchronization**

I. Introduction

Deep learning techniques had a tremendous impact on numerous research areas over the last decade. While groundbreaking results in computer vision have been reported already at the beginning of the decade, deep learning methods for communication systems have gained attraction only recently. Although many works already proposed machine learning solutions to communication system problems, for example, modulation detection in [1], the first end-to-end deep learning communication systems appeared in [2] and [3]. Previously the design of communication systems relied on information theoretic models, classical optimization techniques and signal processing algorithms that were optimal for certain systems. These approaches, however, are limited to only mathematically tractable channel models and can in most cases only be applied to isolated components of the digital communication chain. In contrast, the novel deep learning approach promises to overcome these limitations by applying a global end-to-end optimization to the problem at hand. Learning based approaches do not need a tractable channel model and optimize the system globally. A deep neural network (DNN) autoencoder was proposed in [2] as a first attempt to end-to-end learning of communication systems. The autoencoder structure, a multilayer neural network, consists of an encoder-decoder pair that models the transmission of bits on the physical layer with raw data bits as the autoencoder's input and output.

A channel model is included in the middle layers, and the modulation and demodulation are implemented by the first and last layers of the autoencoder. After training, the first layers are separately used as the transmitter. The final layers constitute the receiver of the system. The middle layers, which model the effect of channels, are omitted after training. In order to adapt the communication system to arbitrary real-world channels lacking tractable mathematical models, some extensions of autoencoders have been proposed in [4]–[6].

In this paper we model the effect of channel impairments through customized middle layers of the autoencoder. For achieving practical applicability it is assumed that the autoencoder incorporates a digital complex baseband model, where signal samples generated by the transmitter DNN can directly be fed to an radio frequency (RF)-frontend for up-conversion. Down-converted signal samples from the receiver RF-frontend can be directly fed into the receiver DNN without any manual pre-processing.

For such a complete end-to-end deep learning signal processing approach the problem of synchronization arises. The authors of [7] and [3] discuss radio transformer networks for this purpose. Signal parameters are estimated with a separate DNNs and the signal is then processed by manually programmed operations. The authors in [8] and [9] propose to learn merely the receiver side of the system and afterwards apply manual signal processing to support synchronization.

We present instead a new full end-to-end approach where synchronization is completely learned as part of a single DNN structure. This is achieved by pilot symbols with a learned waveform and by feeding longer signals to the decoder. This paper is structured as follows. In Sec. II we describe the architecture of the autoencoder. The training process is described explicitly in Sec. II-B. The details of the software defined radio (SDR) implementation are elaborated in Sec. III. The paper concludes with the presentation and discussion of the test results in Sec. IV.

II. A Deep Neural Network Transceiver

In this section we describe the structure of the DNN autoencoder, dimensions and parameters are given in Tab. I. The transmitter is represented by the encoder and the receiver by the decoder, both connected by the channel. The autoencoder operates in discrete passes $i \in \mathbb{N}$. In each pass a source symbol $s = (b_1, \dots, b_k) \in \{0,1\}^k$ of k bits is fed into the transmitter. The outputs of the transmitter are n complex samples comprised in a vector $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{C}^n$.

These samples are transformed by a digital to analog converter (DAC) to a baseband waveform and transmitted after up-conversion. Down-conversion and sampling yields the noisy samples $\mathbf{y} = (y_1, \dots, y_n) \in \mathbb{C}^n$ as the output of the channel. The decoder receives a sequence of W output samples as input and then decides which original sequence was sent. The decoder input length W is larger than n to combat synchronization errors. We will discuss the choice of W below.

The autoencoder *encodes* a sequence s of k bits into a sequence of n complex samples and *decodes* W complex samples back into a sequence $\hat{s}$ of bits. There are $M = 2^k$ possible symbols s and estimates $\hat{s}$ so that the autoencoder represents a classification network. We use categorical cross entropy as error function during training.

A. Encoder

Each input symbol s is represented by an integer $1, 2, \dots, M = 2^k$ and fed into an initial embedding layer. The following two layers of M neurons are fully connected and use the ReLU activation function to implement modulation. The encoder is supposed to learn a representation $\mathbf{x} \in \mathbb{C}^n$ for each s. Mapping symbols s to sequences of n complex numbers is carried out by a third dense layer without any activation function. The width of this layer is $2n$, representing the real and imaginary part of n complex samples. In the next layer all samples are normalized to fall into the unit circle of the complex plane, which is necessary to match the DAC input specifications and essentially also due to the limited output power of the RF frontend. The final encoder layer then combines two real values into one complex number, thus generating the complex baseband samples.

TABLE I: Layout of AE-7/16

Encoder:	Parameters	Output Dimensions
Input	0	1 (integer s $\in [0, 127]$)
Embedding	16384	128
Dense (ReLU)	16512	128
Dense (ReLU)	4128	128
Dense (Linear)	1056	32
Normalization	0	32
Real2Complex	0	16
Channel:		
Serialize	0	80
Time Shift	0	47
Phase Noise	0	47
Gaussian Noise	0	47
Multiply	0	47
Receiver:		
Complex2Real	0	94
Synchronization Feature Estimator (SFE):		
Reshape	0	(47,2)
Convolution (ReLU)	896	(45,128)
MaxPool	0	(45,128)
Convolution (ReLU)	131136	(30,64)
MaxPool	0	(15,64)
Flatten	0	960
Dense (ReLU)	492032	512
Dense (ReLU)	5130	10
Decoder:		
Concatenate	0	104
Dense (ReLU)	53760	512
Dense (ReLU)	262656	512
Dense (ReLU)	131328	256
Dense (ReLU)	65792	256
Dense (ReLU)	32896	128
Dense (Softmax)	16512	128
ArgMax	0	1

B. Channel model and training

The present autoencoder transceiver cannot be trained without a channel model, since backpropagation spans over all layers of the DNN. We use a channel model that incorporates additive white Gaussian noise (AWGN), phase shift, time shift and attenuation. All are modeled to be frequency independent, which is a simplification, sufficient for narrowband channels. The channel model hence extends over the standard memoryless channels to a more challenging setting with asynchronous communication.

In real systems phase offset occurs due to asynchronous clocks or phase noise. Since the autoencoder cannot memorize the phase offset from previous transmissions, we model it as an independent, uniformly distributed random variable in each training step. This is reasonable as the phase offset changes only little for the duration of one symbol transmission. Hence, all samples in the vector $\mathbf{x}$ generated by the encoder are rotated by a uniformly distributed phase

$$\mathbf{u} = e^{-j\varphi} \cdot \mathbf{x}. \tag{1}$$

This operation is implemented as a layer of the channel model, where φ is provided as an additional input to the layer. The assumptions above are harder than a real channel with correlation over time.

Attenuation of signals is modeled by multiplying each sample by a uniformly distributed random factor $a \in [a_{\min}, 1]$ as

$$\mathbf{v} = a \cdot \mathbf{u}. \tag{2}$$

At the next step, the random noise is added to the symbols. Independent random noise variables are added to each complex sample

$$\mathbf{y} = \mathbf{v} + \mathbf{r}. \tag{3}$$

A small value of signal-to-noise ratio (SNR) is chosen, as the autoencoder does not generalize well for SNR less than the training SNR. We observed, moreover, that the autoencoder does not converge to a suitable state if the SNR is too small. The autoencoder, however, converges during training and generalizes well for moderately larger SNR values. Note that the noise is added to each complex sample. Therefore the SNR is, in this sense, defined by E_{sample}/N_0. We can convert the SNR per sample to the SNR per bit, denoted by E_{b}/N_0 using

$$E_{\text{b}}/N_0 = \frac{k}{n} E_{\text{sample}}/N_0.$$

The SNR per bit is used to evaluate the performance independent of the number of complex samples n.

Finally, synchronization effects are introduced. Since the decoder has no memory, synchronization is a difficult problem for DNN based transceivers. In practice, synchronization errors occur gradually due to slightly asynchronous clocks at the transmitter and the receiver. To overcome this problem, in this work, the receiver takes account of a larger window of symbols that includes more than one baseband symbol. The purpose is that the receiver always has available the full set of samples for one of the data symbols within each window.

To introduce the time offset in the channel model, multiple baseband symbols are first considered. We use pilot symbols to facilitate synchronization. The pilot baseband samples are generated by the same encoder that generates the data baseband symbols. Hence, the encoder expects as input an alternating stream of pilot symbols p and source symbols s_i. Fig. 1 depicts an exemplary sequence of data and pilot channel symbols at the training step i given by:

$$(\mathbf{x}_{\text{data}}^{i-1}, \mathbf{x}_{\text{sync}}, \mathbf{x}_{\text{data}}^{i}, \mathbf{x}_{\text{sync}}, \mathbf{x}_{\text{data}}^{i+1}) \in \mathbb{C}^{5n}.$$

The pilot symbol p is chosen to be the integer $0 \in \mathbb{M}$. The same number of bits is used for data and pilot symbols. At each training step, the channel symbols, for data and pilots, are generated by the encoder successively. We use the *TimeDistributed* model of Keras to create 5 identical parallel encoders, as depicted in Fig. 2. The weights of the parallel encoders are shared and updated jointly during training. The transmitter output consists of $5n$ consecutive complex samples during training. The channel impairments are added afterwards. First, a random phase shift φ_i and a random attenuation a_i are chosen. They act on each transmitted baseband sample $x \in \mathbb{C}$. White Gaussian noise $\mathbf{r}_i \in \mathbb{R}^W$, which satisfies SNR E_{sample}/N_0, is added at the end. The channel output at the training step i is therefore given by

$$\mathbf{y}^{(i)} = a_i e^{-j\phi_i}(\mathbf{x}_{\text{data}}^{i-1}, \mathbf{x}_{\text{sync}}, \mathbf{x}_{\text{data}}^{i}, \mathbf{x}_{\text{sync}}, \mathbf{x}_{\text{data}}^{i+1}) + \mathbf{r}_i \in \mathbb{C}^{5n}.$$

The receiver uses a window of length $W = 3n-1$ samples, as to include at least one full data symbol regardless of the actual position of the window. Without synchronization errors, this window is placed at the beginning of the first pilot and spans over the data symbol as in Fig. 2. Synchronization errors introduce a shift of the window. At training step i, the window is shifted by an offset m_i drawn from the uniform distribution over $\{-n+1, \ldots, n\}$. Therefore, the input vector of complex baseband samples to the receiver in step i is given by

$$\mathbf{y}^{(i)}[m_i] = (y_{m_i}^{(i)}, \ldots, y_{m_i+W-1}^{(i)}) \in \mathbb{C}^W. \quad (4)$$

This approach provides a sufficient number of samples to the receiver to account for a set of possible time-shifts of the receiver window. Note that, in contrast to over the air deployment, we do not receive a continuous sequence of complex samples during training. Each time only $5n$ samples are generated over which the receiver chooses a window of W samples. It should be noted that by introducing the synchronization symbol, the communication rate is halved.

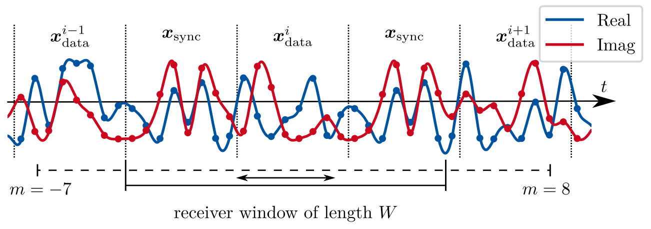


Fig. 1: Visualizing the time shift for the channel and receiver of autoencoder AE-8/8 for a shift m = 0 samples

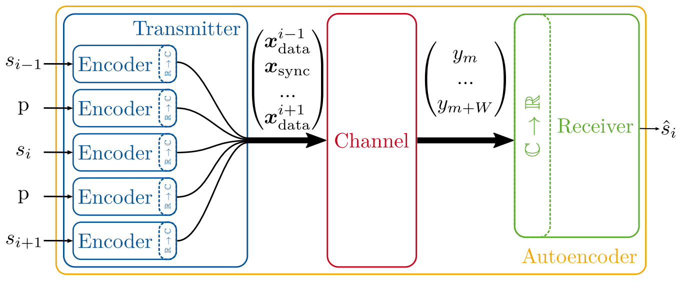


Fig. 2: Autoencoder Layout during Training

C. Decoder

The final step in the transceiver design is the decoder. It receives $2W$ real inputs derived from W complex symbols. At training step i, the decoder shall correctly identify the source symbol $\mathbf{s}_i$ from the input of W complex samples. It is trained to cope with synchronization errors and other channel impairments and, furthermore, includes a particular entity to support the synchronization task called Synchronization Feature Estimator (SFE). SFE extracts a set of features using convolutional layers (see Table I). The decoder then uses these features with $2W$ received inputs to perform the final decoding in the next layers. SFE is similar to correlation filters in conventional transceivers which are used for synchronization.

The final output of the decoder is a probability vector $\hat{\mathbf{s}} \in [0,1]^M$ with components $e^{v_j}/\sum_\ell e^{v_\ell}$, where v_j, $j = 1, \ldots, M$, denotes the real output of the final layer. The symbol with the highest probability is then chosen as the most likely input symbol.

III. Software defined radio implementation

To test the trained transmitter and receiver DNNs in practice, we have implemented a custom signal processing block for the GNU Radio software defined radio (SDR) [10] framework. The Autoencoder is trained with TensorFlow (TF) and the Keras code, written in Python. Thereafter, the DNNs are exported and loaded into the GNU Radio block, which is implemented in C++. This block is able to run trained TF [11] models, i.e. perform the inference, in C++ [12]. GNU Radio provides easy ways to interface with the RF hardware frontends. The autoencoder transceiver operates only in baseband. Hence, for radio transmission over the air, up- and down-conversion to and from the carrier frequency is performed by the RF frontends. We used the 2.4GHz frequency band for transmission. The C++ implementation of the system leads to higher throughput compared to earlier python implementations, e.g., in [3].

In this work, the signal-to-noise ratio of the AWGN is set to $E_{\text{sample}}/N_0 = 5\,dB$. As mentioned above, the autoencoder cannot be trained properly for too small SNR, e.g., 0 dB.

By blockwise processing with one pilot per data block and a bandwidth of 1 MHz, we achieve a throughput of 0.5 Mbit/s on a machine with an Intel Core i7 940 CPU and NVIDIA GeForce GTX 1080 Ti GPU. Two USRP N210 RF-frontends from Ettus were used for the experiments.

In the current setup, the ratio of pilot to data symbols is 1. To improve the throughput, more data symbols per pilot symbol could be transmitted. The optimum ratio of pilot to data symbols will be determined in future experiments.

IV. Experimental Results

The proposed system is evaluated by estimating the symbol error rate of transmissions over simulated and real channels. We compare the results with binary phase shift keying (BPSK) over perfect Gaussian channels. The BPSK transmission differs from our DNN transceiver at least in two different aspects. First, no synchronization error is assumed for BPSK, and secondly, each bit is mapped to one complex channel sample. In contrast, our system considers synchronization errors and maps k bits to n complex samples.

In our benchmark test series, different autoencoders are trained for varying parameters k and n, denoted by AE-k/n. k/n is the ratio between the source symbol bits and the number of complex baseband samples. For example, an AE-7/16 transmits 7 bits using 16 samples. After each data symbol of length n a pilot symbol of the same length is inserted. To evaluate the efficiency of SFE the autoencoder named AE-8/8-2 is trained and tested without this unit.

A. Over-the-air-transmission

To investigate transmission over a real channel, an AE-8/8 was tested over the air at a relative amplitude of 0.61. The error rate is calculated every 200 ms, which corresponds to $200000/8 = 25000$ symbols per evaluated time window. The resulting SER is plotted over time in Fig. 3. Most notably, the error rate fluctuates periodically in intervals of about 2.5 s.

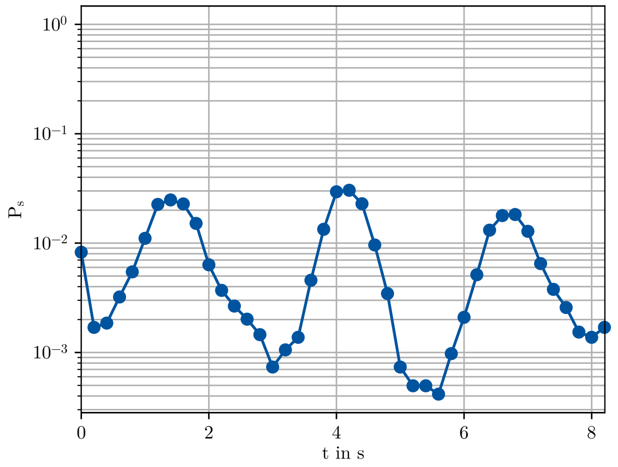


Fig. 3: SER of AE-8/8 tested over the air at amplitude 0.6156, windowed over 200ms for each data point, no overlap, for $1.25 \cdot 10^6$ symbols in total

This is attributed to the slightly different clock speeds of the two USRPs. The error rate changes, as drifting time offsets make it more difficult for the autoencoder to decode certain received sample windows. The minimum at 5.5 s corresponds to an error rate of $4 \cdot 10^{-3}$, which means a single error during a period of 200 ms.

Next, after training AE-7/16 was tested over the air in two independent experiments for different relative amplitudes at the transmitter, each with $N_{\text{test}} = 3.4 \cdot 10^6$ test symbols. The resulting error rates are shown in Fig. 4. For a real radio transmission, the receiver SNR cannot be measured precisely. Hence, the SER is plotted over the the transmit amplitude as x-axis.

Both experiments show a decrease of errors for higher amplitudes, as expected. A higher amplitude causes more energy for the signal and reduces the effect of noise at the receiver. Very low amplitudes correspond to very low values of attenuation a in the channel model, so that the neural receiver decodes increasingly incorrectly for amplitudes below 0.005. This effect was also observed over-the-air, thus reducing performance at low levels of relative amplitude. Both experiments show a bottom floor of symbol error rate (SER) at around 1%.

Notably, the error rates do not decrease monotonically with increasing amplitude. However, the main trend is indicated by dotted lines. The upper green trend curve converges to an error rate of about 2%, the yellow dotted line refers to error rates smaller by approximately a factor of 3. At higher relative amplitudes the error rates increase caused by imperfections of the transmitter. The reason for the two observed distinctly different error rate curves remains unclear at that point and will be investigated in the future.

The observed error rates are significantly worse than the ones found when testing the trained autoencoder over the corresponding simulated channel, results are depicted in Fig. 5. This can be attributed to the channel model that only approx-

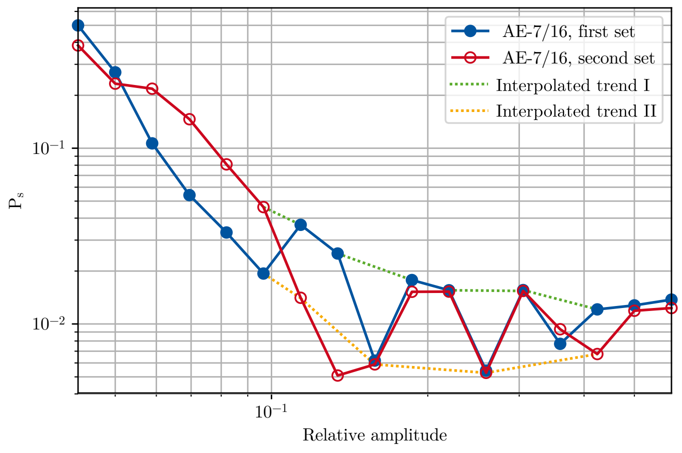


Fig. 4: SER of AE-7/16 tested over the air in two experiments for $3.4{\cdot}10^6$ symbols each. The dotted lines emphasize patterns observed over the two sets.

imates reality, however, is used to train the network. A lot of potential seems to be in applying more accurate channel models or finding ways to train the autoencoder by training even over real channels as has been shown in [13].

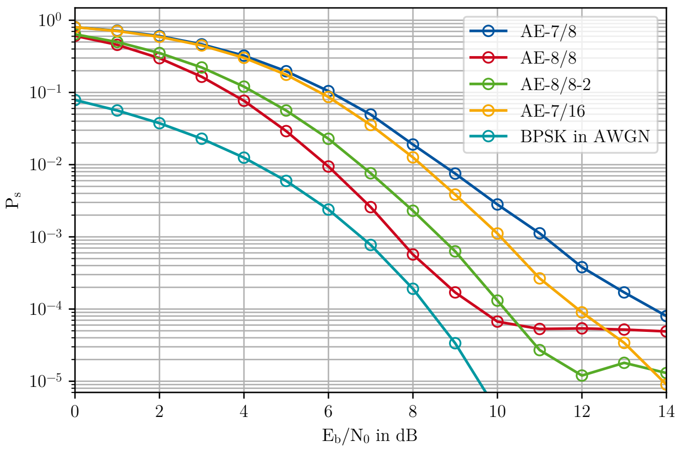
B. *Comparison of the autoencoders*

In this section, the performance of the autoencoders is compared for the simulated channel. The channel attenuation is chosen as $a_{\text{min}} = 0.01$. The phase shift is generated at random uniformly over $[0, 2\pi)$ and the time offset chosen uniformly distributed over $\{-7, \ldots, 8\}$. 10^6 data symbols are sent for each of the four autoencoders and for a range of different SNRs. We use two notions of SNR, namely SNR per sample E_{sample}/N_0 and SNR per bit E_{b}/N_0,

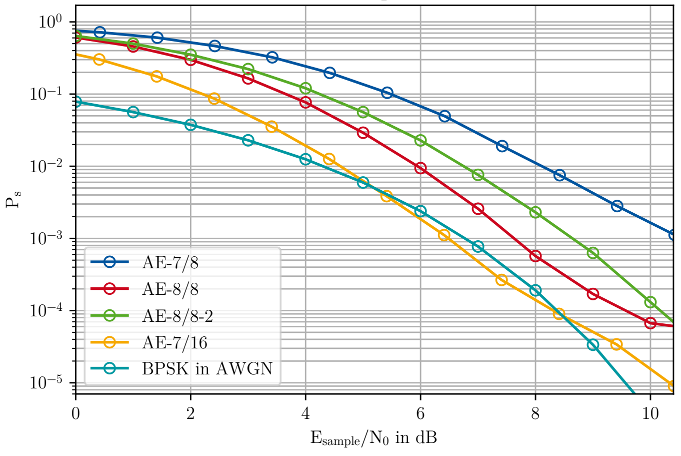
The SERs is plotted versus the SNRs per bit, E_{b}/N_0 in Figure 5a. As previously mentioned, the theoretical error rate of uncoded BPSK is plotted for comparison. The performance of all autoencoders increases with SNR. The AE-7/8 shows the slowest improvement over E_{b}/N_0 and reaches only an SER of 10^{-4} at 14 dB. While the AE-7/16 shows a similarly slow improvement for low SNRs, it improves more quickly and reaches the SER below 10^{-5}. Note that the AE-7/16, however, uses more complex samples than the AE-7/8 to transmit the same amount of data.

At a low SNRs, the best performance is achieved by the AE-8/8 and AE-8/8-2. The improvement of SER with SNR shows an error floor in the high SNR regime. This is in contrast to the AE-7/16 which, at least within the tested range of SNRs, keeps improving and reaches a lower error rate than both the AE-8/8 and the AE-8/8-2. In that regard, the AE-7/16 outperforms other autoencoders in the high SNR regime, however, at the price of using more samples to transmit data.

We also plot the result of the same experiments for the SNR per sample E_{sample}/N_0. The corresponding results for the SERs are shown in Figure 5b. As expected, AE-7/16 performs better than other autoencoders, since it has more samples to transmit the same amount of data and can thus achieve a more robust



(a) Normalized per bit



(b) Normalized per sample

Fig. 5: SER of the four presented autoencoders, tested on 10^6 symbols each over modeled channels, and theoretical BPSK error rate over AWGN as a baseline

representation of the encoded bits within the samples. The AE-8/8 needs about 1.7 dB more SNR to achieve the same error rates as AE-7/16. This number is more than 2 dB for AE-8/8-2. AE-7/8 is significantly worse than the other autoencoders. If SNR per sample is used, the AE-7/16 is able to outperform even uncoded BPSK for SNR values between approximately 5 and 8 dB. This can be attributed to the significantly greater number of complex samples used for transmission.

Uncoded BPSK mostly achieves better error rates than the proposed autoencoders. But as discussed above, this observation should be interpreted with care. The theoretical BPSK error rate is determined for an AWGN channel, disregarding other effects of the channel model. Furthermore, the autoencoders are evaluated by their symbol error rate, which is at most equal to the bit error rate, in most cases, however, significantly lower. Unlike the theoretical results for BPSK, the autoencoders of the present paper are designed to combat more channel impairments, as described above.

For over-the-air experiments, the SER of the four autoencoders is plotted in 6. We use $3.3 \cdot 10^6$ symbols for each

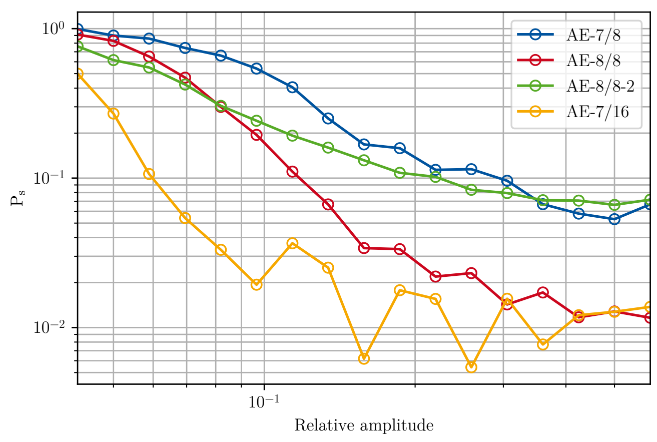


Fig. 6: SER over the air for the four presented autoencoders, tested on $3.3 \cdot 10^6$ symbols each

autoencoder at different relative amplitude levels. Since the noise is added to each complex sample, the SNR per sample is the natural choice for evaluating over-the-air transmission.

At low amplitudes, the AE-7/16 clearly outperforms the other autoencoders, and at amplitudes of about 0.2 it reaches the best error rates of all tested autoencoders with a value of about 0.6%, see Fig. 3. Even for the worst SER values in Figure 3, the AE-7/16 shows better error rates over the whole range of tested amplitudes. The next best performance is obtained by the AE-8/8 which reaches comparable performance only at high amplitudes. This is because its bit rate is more than two times bigger than AE-7/16's, and it thus has less redundancy to mitigate the effects of low amplitudes. Interestingly, AE-8/8-2 performs better than AE-8/8 for low amplitudes, in contrast with what we observed in the simulations. On the other hand, the SER of the AE-8/8-2 decreases very slowly and has an error floor at SER 7%, which is considerably worse than in the simulations. Even the AE-7/8, which was consistently the worst autoencoder in our simulations, reaches lower error rates than the AE-8/8-3 while still showing inferior performance than AE-8/8 and AE-7/16. When ranking the autoencoders with regard to their performance, the results of the over-the-air experiments coincide with the results of simulations, as shown in 5b. Namely, the AE-7/16 performs better than the AE-8/8, followed by the AE-8/8-2 and the AE-7/8, which has the worst performance of the four. It can be seen that the AE-8/8 demonstrates a surprisingly good over-the-air performance, considering its significantly higher data throughput than AE-7/16.

The bad performance of AE-8/8-2 emphasizes the importance of the SFE block to improve decoding. The impact of the SFE on the behavior of the autoencoders should be further investigated. Whereas the AE-8/8-2, without the SFE, performs comparably well in the simulations, it is distinctly inferior to the AE-8/8 over the air.

V. Conclusion and Outlook

In this paper, we presented a fully trainable deep learning transceiver, which addresses in particular synchronization issues. Multiple autoencoders with different architectures are trained. Each has three different components, namely the encoder, the channel and the decoder. The channel is incorporated in the training by using a model, thus enabling back propagation. The performance of the transceivers is evaluated by over the air transmission on an SDR platform. They achieve data rates of about 0.5 Mbps and contribute to high data rate deep learning transceivers. Future work will consider more realistic channel models, for example multipath propagation models or frequency shifts, and training with different SNRs per batch. Recurrent neural networks can potentially learn tracking of synchronization parameters over time and can thereby improve the block based methods presented in this paper. Moreover, an autoencoder structure that transmits multiple data blocks per preamble block will be investigated to improve throughput.

References

[1] Timothy J O'Shea, Johnathan Corgan, and T Charles Clancy, "Convolutional radio modulation recognition networks," in *International conference on engineering applications of neural networks*. Springer, 2016, pp. 213–226.

[2] Timothy O'Shea and Jakob Hoydis, "An introduction to deep learning for the physical layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, 2017.

[3] Sebastian Dörner, Sebastian Cammerer, Jakob Hoydis, and Stephan ten Brink, "Deep learning based communication over the air," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 132–143, 2018.

[4] Fayçal Ait Aoudia and Jakob Hoydis, "End-to-end learning of communications systems without a channel model," *arXiv preprint arXiv:1804.02276*, 2018.

[5] Hao Ye, Geoffrey Ye Li, Biing-Hwang Fred Juang, and Kathiravetpillai Sivanesan, "Channel agnostic end-to-end learning based communication systems with conditional gan," *arXiv preprint arXiv:1807.00447*, 2018.

[6] Vishnu Raj and Sheetal Kalyani, "Backpropagating through the air: Deep learning at physical layer without channel models," *IEEE Communications Letters*, 2018.

[7] Timothy J O'Shea, Latha Pemula, Dhruv Batra, and T Charles Clancy, "Radio transformer networks: Attention models for learning to synchronize in wireless systems," in *Signals, Systems and Computers, 2016 50th Asilomar Conference on*. IEEE, 2016, pp. 662–666.

[8] Zhongyuan Zhao, "Deep-waveform: A learned ofdm receiver based on deep complex convolutional networks," *arXiv preprint arXiv:1810.07181*, 2018.

[9] Alexander Felix, Sebastian Cammerer, Sebastian Dörner, Jakob Hoydis, and Stephan ten Brink, "Ofdm-autoencoder for end-to-end learning of communications systems," *arXiv preprint arXiv:1803.05815*, 2018.

[10] "GNU Radio - The free and open software radio ecosystem," website: http://gnuradio.org/, Accessed: April 15, 2019.

[11] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al., "Tensorflow: a system for large-scale machine learning.," in *OSDI*, 2016, vol. 16, pp. 265–283.

[12] Johannes Schmitz, "gr-tensorflow_cc: TensorFlow GNU Radio blocks with TensorFlow C++ inference API," https://github.com/johschmitz/gr-tensorflow_cc, 2018.

[13] Mathieu Goutay, Fayçal Ait Aoudia, and Jakob Hoydis, "Deep reinforcement learning autoencoder with noisy feedback," *arXiv preprint arXiv:1810.05419*, 2018.