Scalable Machine Learning Algorithms to Design Massive MIMO Systems

Dolores García Martí
IMDEA Networks Institute
University Carlos III
Madrid, Spain
dolores.garcia@imdea.org

Damiano Badini
Huawei Technologies Italia S.R.L
Milan, Italy
damiano.badini@huawei.com

Danilo De Donno
Huawei Technologies Italia S.R.L
Milan, Italy
danilo.dedonno@huawei.com

Joerg Widmer
IMDEA Networks Institute
Madrid, Spain
joerg.widmer@imdea.org

ABSTRACT

Machine Learning (ML) is a highly promising tool to design the physical layer of wireless communication systems, but its scaling properties for this purpose have not been widely studied. ML algorithms are typically evaluated to learn SISO communications and low modulation orders, whereas current wireless standards use MIMO and high-order modulation schemes to increase capacity. The memory requirements of current ML algorithms for wireless communications increase exponentially with the number of antennas and thus they cannot be used for advanced physical layers and massive MIMO. In this paper, we study the requirements of end-to-end ML models for large-scale MIMO systems, determine the bottlenecks of the architecture, and design different solutions that vastly reduce overhead and allow training higher MIMO and modulation orders. We show that by training the autoencoder in a bit-wise manner, the memory requirements are reduced by several orders of magnitude, which is a critical step forML-based physical layer design in practical scenarios. Additionally, our design also improves performance over the classical autoencoder for MIMO.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Networks** → **Mobile networks**.

KEYWORDS

Machine Learning, Neural Networks, MIMO, Physical Layer

ACM Reference Format:
Dolores García Martí, Damiano Badini, Danilo De Donno, and Joerg Widmer. 2021. Scalable Machine Learning Algorithms to Design Massive MIMO Systems. In *ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, November 22–26, 2021, Alicante, Spain.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3416010.3423229

MSWiM '21, November 22–26, 2021, Alicante, Spain

ACM ISBN 978-1-4503-8117-8/20/11…$15.00
https://doi.org/10.1145/3416010.3423229

1 INTRODUCTION

The physical layer of wireless communication systems is formed by discrete and highly optimized signal processing blocks with well-defined functions to cope with the multiple impairments of the communication channels. Such designs work well for current wireless networks with links operating close to their theoretical capacity even if they do not take into account possible interactions between non-linearities and residual errors of individual blocks.

However, the complexity of the physical layer of next-generation wireless networks is increasing to meet higher performance demands. 5G and beyond mobile networks include millimeter-wave (mm-wave) and multi-input multi-output (MIMO). As these systems become more complex in terms of hardware design due to the higher frequencies and number of antennas, hardware imperfections and non-linearities become more difficult to model and compensate for explicitly. This is aggravated by the constraints imposed by integration in consumer devices and the mandate of a cost-efficient design. The above considerations inspired the application of ML to wireless physical layer design. At higher layers, ML has been used to reduce the high complexity of current optimization problems or cope with the lack of accurate models. But thus far, no component of 5G has been *designed* by ML. An AI-native interface could allow to optimally deal with imperfections and reduce the heuristic optimization of the parameters of the physical layer [6].

Recently, the first design of an end-to-end communication system using deep learning models was proposed [10]. The authors of [10] discuss how to learn a transmitter-receiver architecture represented as Deep Neural Networks (DNN) for a specific channel model using an autoencoder structure. Most of the follow-up work based on the concepts presented in [10] focuses on more complex channel models for the single-input single-output (SISO) autoencoder [3, 9] and on learning the channel through data [1, 5, 11, 14]. However, few works consider the extension of these concepts to MIMO systems, where the gains could be larger due to the higher number of antennas and the higher complexity of the hardware. In [8, 12], an extension of the autoencoder architecture to MIMO is presented, but both works only consider a simple 2×2 system and very low order modulations. In contrast, large MIMO systems may use hundreds

of antennas thanks to the very small wavelength of mm-wave, and recent wireless standards such as 5G and IEEE 802.11ax support very high modulation orders such as 1024-QAM.

In this paper, we carry out a thorough analysis of the scalability of the autoencoder architecture and conclude that current architectures are not suitable for the physical layer learning of large MIMO systems due to their memory demands that increase exponentially with the number of antennas. For example, for a 4×4 MIMO system and a modulation equivalent to 64-QAM, ~800 GB of vRAM would be needed to train the network, whereas even powerful Graphics Processing Units (GPUs) available in the market only have 40-80 GB of vRAM. This large memory demand comes from the number of possible messages that grow exponentially with the number of antennas and order of the constellation. We evaluate typical approaches of reducing the memory allocation for DNNs and conclude that these can only achieve small reductions. We then design a different embedding for the inputs which has some similarity to the bit-wise autoencoder architecture presented in [2] for SISO systems, but that has not been studied for MIMO and for the purpose of memory reduction. This architecture preserves the soft information of the network by defining a measure of probability for these new outputs. It can reduce the memory requirements by 99.99%. Our work is a highly important step towards practical ML-based physical layer design of MIMO systems. Additionally, we make the code to estimate the memory consumption and the different architectures available to the research community.

The paper is organized as follows. In Section 2 we introduce end-to-end learning concepts. In Section 3, we analyze the GPU memory requirements of MIMO autoencoders. We then evaluate typical Deep Learning (DL) solutions to reduce memory requirements in Section 4, and propose a much more memory-efficiency bit-wise autoencoder in Section 5. The results are presented in Section 6. Finally, in Section 7 we discuss existing literature on end-to-end learning systems and give some concluding remarks in Section 8.

2 AUTOENCODER DESIGN FOR MIMO

We first discuss autoencoder architectures for end-to-end learning of the encoder and decoder structures for MIMO systems. The works in [8, 12] extend the concept of the autoencoder for MIMO communications. There are two kinds of architectures for the MIMO end-to-end learning as described in [8, 12]: an *open-loop* architecture following the description above, and a *closed-loop* architecture in which the Channel State Information (CSI) information is fed back to the transmitter as shown. This step allows the autoencoder to learn improved encodings with the help of channel knowledge. We consider the general architecture presented in [12], as shown in Table 1, where $H_{T/R}$ are the number of units of the transmitter and receiver dense layers, respectively.

3 SCALABILITY ANALYSIS

We now carry out a thorough analysis of the GPU memory consumption needed to train a MIMO end-to-end model of size $N_T \times N_R$ and of modulation order k, using the method proposed in [4]. The GPU memory required to train a deep learning model can be divided into four main categories: weight tensors, in/out tensors, ephemeral tensors, and resident buffer [4]. A key observation is that the total size of the network does not only depend on the parameters of the network but also on its forward and backward propagation. To take this into account, the two dimensions that are important to our problem are weight tensors and in/out tensors. Weight tensors are the learnable parameters and their gradients, computed under backpropagation. The in/out tensors are the inputs/outputs to operators and the forward and gradient outputs during forward and backward propagation respectively. The resident buffer is the memory allocated to the managing information to control the GPU and its size depends on the framework.

Table 1: Autoencoder layers for open-loop MIMO

Number	Layers	Output Dimensions
1	Input	S_M
2	*Dense* (act = relu)	H_T
3	*Dense* (act = relu)	H_T
4	*Dense* (act = linear)	$2NN_T$
5	*Normalization*	$2NN_T$
6	*ComplexMult*	$2NN_R$
7	*Noise*	$2NN_R$
8	*Dense* (act = relu)	H_R
9	*BacthNorm*	H_R
10	*Dense* (act = relu)	H_R
11	*BacthNorm*	H_R
12	*Dense* (act = relu)	H_R
13	*BacthNorm*	H_R
14	*Dense* (act = softmax)	S_M

3.1 MIMO network GPU memory estimation

We calculate the memory allocation for the different operators in the architecture presented in Table. 1 as in [4]. The encoder/decoder memory estimates are

$$M_{Le} = 2p((2^k)^{N_T} H_T + H_T^2 + H_T 2NN_T + 2H_T + 2NN_T) + 4pB(2H_T 24NN_T) \quad (1)$$

$$M_{Ld} = 2p(H_R(N_R + 1) + 3H_R(H_R + 1) + 16H_R + H_R(2^k)^{N_T} + (2^k)^{N_T}) + 4pB(4H_R + (2^k)^{N_T} + 16) \quad (2)$$

where the *Batch Normalization* layers are considered to have 4 parameters (as in the TensorFlow implementation). By p we denote the precision format of the data type (e.g., for *float32* it is 4 bytes).

The results of the GPU memory allocation estimation for the MIMO autoencoder network are presented in Table 2 (without embedding column) for a $N_T \times N_R$ systems of antennas with $N_T = \{2, 4, 16, 64\}$ and $N_R = N_T$. We consider a batch size of $B = 2048$, $H_t = 256$ and $H_R = 2048$ as in the implementation in [12] for $S_M = 4$ and a 2×2 system. However, in order to cope with the increasing number of combinations the network depth would need to be increased for larger systems. As can be observed in Table 2, for a 4×4 system and a 64-QAM like modulation, the estimated GPU memory is ~ 800 GB. The current largest GPU VRAM available in the market is 80GB on the *Nvidia A100* with a cost around $\$10,000$. Even with such high-end hardware and using a multi-GPU system it would be very difficult to train a typical size MIMO system using end-to-end learning.

The problem of the current architectures resides mainly in the input and output layers, resulting in two exponential growths. For a MIMO autoencoder with the one-hot encoding, the input size is $(2^k)^{N_T}$ and the memory allocation grows as $M_L \sim 8(2^k)^{N_T} H_T$ for the encoder and $M_L \sim 16B(2^k)^{N_T}$ for the decoder. Therefore, the growth is exponential with respect to the number of antennas N_T and the modulation order k. In the next section, we discuss different techniques to reduce the memory footprint of end-to-end deep learning.

4 REDUCING MEMORY CONSUMPTION

In this section we discuss three different deep learning techniques to reduce the memory footprint of a model:

• Embeddings are a mappings of a higher dimensional input space to a lower dimensional space. For the autoencoder network, each possible message is labelled as an integer $i \in \{1, 2, \ldots, S_M\}$. This is fed to the embedding layer before the first fully connected layer, which finds a representation of this integer into an embedding of size E_{emb}. The loss function is updated to *sparse categorical cross-entropy loss*. The estimated memory allocation using embeddings can be obtained by modifying equation 1 and the results are presented in Table. 2. This technique provides a maximum memory reduction of ∼10%.

• Mini-batching is the technique of updating the gradients after every mini-batch instead of after seeing the whole dataset. The study of the memory allocation for our particular network in Section 3 allows us to determine the limits of the batch size parameters to optimize performance and maximize the computing resources available.

• Mixed precision training (FP16) is a technique that uses half-precision floating-point numbers to train the network without losing model accuracy or having to modify the model's hyperparameters [7]. This novel technique halves the memory requirements on all GPUs where this feature is available.

The results of the GPU memory allocation estimation for the MIMO autoencoder network with embeddings and mixed precision training are presented in Table. 2, second column. Although we observe some improvements from using these techniques, the gains are insufficient to make the symbol-wise autoencoder practical.

5 BIT-WISE AUTOENCODER FOR MIMO

Even with the performance gains of all the techniques described above, training the end-to-end autoencoder for a 64-QAM like modulation for a 4×4 MIMO system still requires 144 GB of VRAM (using the architecture described in Table 1 and $B = 1$ to minimize memory consumption). To overcome the large memory consumption, we propose the use of a different architecture for the end-to-end learning for MIMO systems, using a custom embedding in the input and the output layers.

5.1 The architecture

In the bit-wise autoencoder, instead of the one-hot encoding of size S_M, the inputs to the network are N_T bit vectors of length k, which are concatenated at the input to form a bit vector of length $k * N_T$ and the output of the network is a vector of the same size in which each value is the probability over the bit. This is achieved by changing the last layer's activation function to a sigmoid so that the outputs are in the $[0, 1]$ range. This proposed embedding is the extension of the bit-wise autoencoder proposed in [2] for a SISO system to optimize constellation shaping and labeling. Additionally, we present a new encoder architecture that improves the performance of the bit-wise MIMO autoencoder as we show in Section 6. We propose an architecture where the encoder layers are followed by hyperbolic tangent (tanh) activation functions. Since the output of the encoder is constrained to the range $[-1, 1]$, no extra normalization layer is necessary. The normalization layer used in previous architectures may result in spurious solutions. For example when the training uses very small modulation values close to zero, the normalization of the batch of symbols can result into modulations with a very large range. Our proposed encoder converges to more stable solutions. Also, using the Tanh-BAE the outputs of the encoder layer are not dependent on the batch size, avoiding different modulations due to different batch sizes. In order to train the network, we consider the binary cross-entropy, BCE, as the loss function. This is possible since the inputs and outputs of the network are normalized. Moreover, the binary cross-entropy loss is minimized, $\frac{\partial BCE(y,p)}{\partial p} = 0$, when the input value equals the predicted value, as can be proved from the definition of the BCE. The soft information of the bits is now obtained as the i-th output of the network as the outputs are in the range $[0, 1]$ due to the sigmoid activation of the last layer.

5.2 Bit-wise network GPU memory estimation

The resulting network's memory consumption can be calculated using (2) and (1) by changing $(2^k)^{N_T}$ to $(k * N_T)$. The results of the memory consumption for this compressed architecture for a $N_T \times N_R$ MIMO system with $N_T = \{2, 4, 16, 64\}$ and $N_R = N_T$ are presented in Table 2. The improvement of using such embedding is substantial and allows to train large complex physical layer systems involving massive MIMO and large modulation orders. These results show that end-to-end optimization of the physical layer is possible even for future high-performance wireless networks.

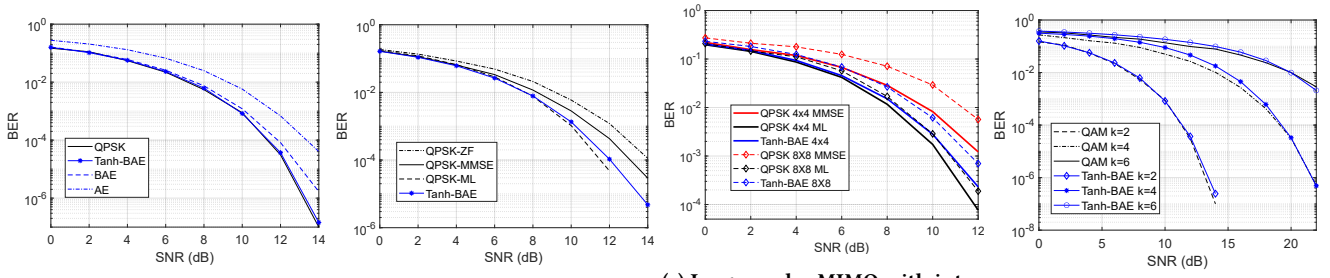
6 SIMULATION RESULTS

We run different experiments to compare the performance of the bit-wise autoencoder proposed in Section 5. The implementation of the different autoencoders is done in a TensorFlow framework. All autoencoders are trained using the Adam optimizer with a learning rate 0.001. We train the different autoencoder models with batch size $B = 2048$, and over 4096000 random channel realizations.

Comparison of the different schemes. We consider the parameters $N_T = 2$, $N_R = 2$, $S_M = 16$. Thus $M = 4$, which corresponds to a modulation equivalent to QPSK for each transmitter. First, we compare the symbol wise autoencoder [8] (AE), the bit-wise autoencoder (BAE), our proposed architecture the Tanh bit-wise autoencoder (BAE-T) and the standard QPSK on a Additive White Gaussian Noise (AWGN) channel (note that since the channel is AWGN no explicit equalization is needed and the baseline is a classical demapper based on LLR [13]). The results are presented in Fig. 1a. We observe that both the standard AE and BAE do not outperform the standard QPSK with hard-decision demapping. The AE

Table 2: Memory allocation for a mimo autoencoder for different constellation sizes and number of antennas N_T.

Constellation (2^k)	Estimated memory (GB)											
	AE				AE with embedding+mixed precision				BAE			
	N_T=2	N_T=4	N_T=16	N_T=64	N_T=2	N_T=4	N_T=16	N_T=64	N_T=2	N_T=4	N_T=16	N_T=64
4	0.7	0.8	$2.0{\cdot}10^{5}$	$1.6{\cdot}10^{34}$	0.6	0.6	98352.6	$7.8{\cdot}10^{33}$	0.7410	0.7414	0.7435	0.7519
16	0.8	3.9	$8.8{\cdot}10^{14}$	$5.2{\cdot}10^{72}$	0.6	2.1	$4.2{\cdot}10^{14}$	$2.7{\cdot}10^{72}$	0.7412	0.7418	0.7450	0.75805
64	0.9	800.9	$3.8{\cdot}10^{24}$	$1.9{\cdot}10^{111}$	0.7	384.7	$1.8{\cdot}10^{24}$	$9.0{\cdot}10^{110}$	0.7414	0.7422	0.7466	0.76415
128	1.5	12802.8	$2.5{\cdot}10^{29}$	$3.5{\cdot}10^{130}$	0.9	6147.6	$1.2{\cdot}10^{29}$	$1.7{\cdot}10^{130}$	0.7415	0.7424	0.7473	0.7672
256	3.9	$2.0{\cdot}10^{5}$	$1.6{\cdot}10^{34}$	$6.4{\cdot}10^{149}$	2.1	98352.6	$7.8{\cdot}10^{33}$	$3.1{\cdot}10^{149}$	0.7416	0.7426	0.7481	0.7702
1024	50.7	$5.2{\cdot}10^{7}$	$6.9{\cdot}10^{43}$	$2.2{\cdot}10^{188}$	24.6	$2.5{\cdot}10^{7}$	$3.3{\cdot}10^{43}$	$1.0{\cdot}10^{188}$	0.7418	0.7429	0.7496	0.7763



(a) Baselines with AWGN **(b) Baselines with interference** **(c) Larger order MIMO with interference** **(d) Higher modulation orders**

Figure 1: Performance evaluation of the different approaches.

results in the worse performance as it has been optimized for symbol error rate (SER) rather than bit error rate (BER). Our approach, the Tanh-BAE improves over the performance of previous works and performs as well as QPSK while keeping the GPU memory consumption low, as shown in Section 5.2. Therefore, an optimal constellation and labelling are learned. For higher MIMO orders similar results are observed when comparing to the baselines.

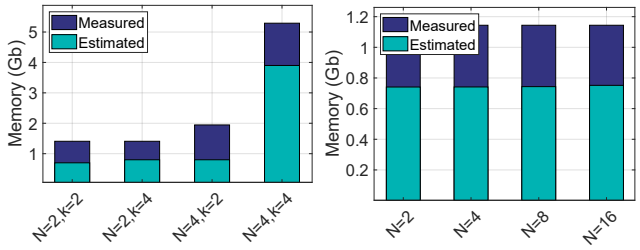
Following the comparison with the classical symbol-wise autoencoder, AE and the BAE, we consider a MIMO channel with strong inter-stream interference. This channel is designed such that the cross channel coefficients have a value of 0.3. The same parameters are considered, $N_T = 2$, $N_R = 2$, $S_M = 16$. Fig. 1b shows the comparison of the BER for different Signal-to-Noise Ratios (SNRs) for our autoencoder approach Tanh-BAE and three conventional baseline schemes based on Zero Forcing (ZF), Minimum Mean Squared Error (MMSE) and Maximum Likelihood (MaxL). We observe that the Tanh-BAE outperforms ZF and MMSE by $\sim$ 2 dB and is close the optimal MaxL equalization. Althought for higher SNRs the performance is slightly worse than MaxL we attribute this to the autoencoder training being SNR dependent (whereas we train using a single SNR, the optimum constellation is in fact SNR dependent e.g as the SNR decreases the constellation symbols are grouped in clusters). This is the first example of the autoencoder outperforming classical equalization schemes. In future research, we aim to study the complexity of this solution and energy efficiency compared to classical decoding.

Comparison for higher order MIMO. We consider a channel with high inter-stream interference as above, with cross channel coefficients of 0.3 and compare the performance of the Tanh-BAE with standard QAM with $k = 2$ (QPSK) and different number of antennas $N_T, N_R = 4, 8$. Fig. 1c shows the BER for different SNRs for the Tanh-BAE and the baselines corresponding to QPSK modulation with MMSE and MaxL equalization for a $4x4$, and $8x8$ system. It can be seen that the Tanh-BAE model outperforms the MMSE equalization for the different number of antennas and comes close to the performance of MaxL. The Tanh-BAE has gains of $\sim$ 1.5 and $\sim$ 2 dB for the $4x4$ and $8x8$ MIMO systems respectively. The gains increase as the size of the system grows.

Comparison for larger order modulations. For the last experiment we consider a $2x2$ MIMO system and different modulation orders $k = 2, 4, 6$ corresponding to QPSK, 16-QAM and 64-QAM like modulation for each antenna. In this scenario we consider an AWGN channel. Fig. 1d shows the BER for different SNRs for the Tanh-BAE and the QAM modulation for the different modulation orders. It can be seen that the autoencoder performs close to the baseline for all modulations, as expected. However, we observe again the effects of the autoencoder being trained for a single SNR, as for $k = 4, 6$ the performance for lower SNRs is slightly worse. This could be addressed by training an autoencoder per SNR.

Memory allocation. Lastly, we compare the estimated memory allocation as obtained in the previous sections and the real measured allocation using a GPU NVIDIA A100. Fig. 2a and Fig. 2b show the estimated and measured memory consumption for different configurations of the AE and BAE respectively. The results show that the measured memory allocation is larger than the estimated one, e.g., it is $\sim$ 0.7 GB larger for the first three configurations of the AE in Fig. 2a and for all configurations in the BAE in Fig. 2b. We assume this is due to this GPU having a different CUDA-context value (pre-allocated memory) than the one measured in [4] as this can vary with the GPU and also with the framework implementation. The

fourth configuration of Fig. 2a presents a larger error in the estimation, however it is inside the bounds of what the authors report in [4] (16% error for Tensorflow implementations). In general, it is hard to analyze the GPU memory usage as it largely depends on the implementation, the framework and APIs but having an estimate allows to study the practicality of our solutions. Our estimate very well follows the trend of the different true allocations.



(a) AE **(b) BAE**

Figure 2: Comparison of the estimated and measured memory consumption during training for a MIMO autoencoder (a) and a MIMO bit-wise autoencoder for k=2 (b).

7 RELATED WORK

In this paper, we focus on ML applied to the end-to-end learning of communication systems using autoencoders [1, 3, 9–11, 14]. The concept of end-to-end learning for communications systems using autoencoder was first introduced in [10].

End-to-end learning for SISO:. Since the field of ML based physical layer design is very recent, most of the works in the literature focus on developing the general architecture and therefore use simple AWGN channels [10, 11, 14]. Some works present more complex channel models [3], considering up-sampling, pulse shaping, constant sample time offset, constant phase offset, Carrier Frequency Offset (CFO) and AWGN.

End-to-end learning for MIMO:. The SISO work of [10] was extended for MIMO systems in [8], with a Rayleigh fading channel as channel model. The authors consider a closed-loop MIMO system and compare it against a standard Singular Value Decomposition (SVD)-based MIMO precoding technique. The authors further compare an open-loop MIMO architecture without any CSI knowledge, as well as MIMO with perfect CSI knowledge and CSI with quantized values. The results demonstrate gains for the 2x2 scheme with perfect CSI at the transmitter. Finally, the work in [12] extends the work in [8] by giving more details about the architecture and benchmarking the gains for a 2×2 systems and low order modulations. However, none of the above-mentioned works study larger systems of antennas; only 2×2 systems have been considered so far. Similarly, modulation schemes are restricted to 4 bits per symbol or less. However, high-order MIMO and modulation schemes are highly interesting for end-to-end learning due to the inherent complexity when implemented using conventional signal processing algorithms.

8 CONCLUSION

In this work, we carry out a deep analysis of the scaling properties of end-to-end learning for communications for MIMO systems and/or high-order modulation schemes. We conclude that the bottleneck is due to the inputs (as one-hot encoded vectors) and outputs (as probability vectors) of the designed network. These large MIMO systems constitute highly demanding scenarios in which training would require unmanageable memory resources. We study typical approaches available in DL to reduce the memory allocation. However, all of these approaches fall short in sufficiently reducing the memory consumption. Finally, we present a bit-wise architecture for MIMO (BAE) and a novel encoder Tanh-BAE, that allow to reduce the memory allocation by several orders of magnitude. Our evaluation shows that the BER performance of Tanh-BAE is better than that of the classical autoencoder for the MIMO system improving the state of the art for end-to-end learning for MIMO communications. Thanks to our analysis and design, it is now possible to study large system of antennas. This study is an important first step towards developing practical end-to-end learning algorithms for large scale MIMO systems.

REFERENCES

[1] Fayçal Ait Aoudia and Jakob Hoydis. 2019. Model-Free Training of End-to-End Communication Systems. *IEEE Journal on Selected Areas in Communications* 37, 11 (2019), 2503–2516.

[2] Sebastian Cammerer, Fayçal Ait Aoudia, Sebastian Dörner, Maximilian Stark, Jakob Hoydis, and Stephan Ten Brink. 2020. Trainable communication systems: Concepts and prototype. *IEEE Transactions on Communications* 68, 9 (2020), 5489–5503.

[3] Sebastian Dörner, Sebastian Cammerer, Jakob Hoydis, and Stephan ten Brink. 2017. Deep Learning Based Communication Over the Air. *IEEE Journal of Selected Topics in Signal Processing* 12, 1 (2017), 132–143.

[4] Yanjie Gao, Yu Liu, Hongyu Zhang, Zhengxian Li, Yonghao Zhu, Haoxiang Lin, and Mao Yang. 2020. Estimating gpu memory consumption of deep learning models. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1342–1352.

[5] Dolores García Martí, Joan Palacios Beltrán, Jesús Omar Lacruz, and Joerg Widmer. 2020. A Mixture Density Channel Model for Deep Learning-Based Wireless Physical Layer Design. In *Proceedings of the 23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. 53–62.

[6] Jakob Hoydis, Fayçal Ait Aoudia, Alvaro Valcarce, and Harish Viswanathan. 2020. Towards a 6G AI-Native Air Interface. *arXiv preprint arXiv:2012.08285* (2020).

[7] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. 2017. Mixed precision training. *preprint arXiv:1710.03740* (2017).

[8] Timothy J O'Shea, Tugba Erpek, and T Charles Clancy. 2017. Deep learning based MIMO communications. *arXiv preprint arXiv:1707.07980* (2017).

[9] Timothy J O'Shea, Tamoghna Roy, Nathan West, and Benjamin C Hilburn. 2018. Physical Layer Communications System Design Over-The-Air Using Adversarial Networks. In *2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE, 529–532.

[10] Timothy O'Shea and Jakob Hoydis. 2017. An Introduction to Deep Learning for the Physical Layer. *IEEE Transactions on Cognitive Communications and Networking* 3, 4 (2017), 563–575.

[11] Timothy J O'Shea, Tamoghna Roy, and Nathan West. 2019. Approximating the Void: Learning Stochastic Channel Models from Observation with Variational Generative Adversarial Networks. In *2019 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 681–686.

[12] Jinxiang Song, Christian Häger, Jochen Schröder, Tim O'Shea, and Henk Wymeersch. 2020. Benchmarking End-to-end Learning of MIMO Physical-Layer Communication. *arXiv preprint arXiv:2005.09718* (2020).

[13] Andrew J. Viterbi. 1998. An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes. *IEEE Journal on Selected Areas in Communications* 16, 2 (1998), 260–264.

[14] Hao Ye, Geoffrey Ye Li, Biing-Hwang Fred Juang, and Kathiravetpillai Sivanesan. 2018. Channel Agnostic End-to-End Learning Based Communication Systems With Conditional GAN. In *2018 IEEE Globecom Workshops (GC Wkshps)*. IEEE, UAE, 1–5.