A Mixture Density Channel Model for Deep Learning-Based Wireless Physical Layer Design

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

Joan Palacios Beltrán
IMDEA Networks Institute
University Carlos III
Madrid, Spain
joan.palacios@imdea.org

Jesús Omar Lacruz
IMDEA Networks Institute
Madrid, Spain
jesusomar.lacruz@imdea.org

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

ABSTRACT

Machine learning is a highly promising tool to design the physical layer of wireless communication systems, but it usually requires that a channel model is known. As data rates increase and wireless transceivers become more complex, the wireless channel, hardware imperfections, and their interactions become more difficult to model and compensate explicitly. New machine learning schemes for the physical layer do not require an explicit model but *implicitly learn* the end-to-end link including channel characteristics and non-linearities of the system directly from the training data.

In this paper, we present a novel neural network architecture that provides an *explicit* stochastic channel model, by learning the parameters of a Gaussian mixture distribution from real channel samples. We use this channel model in conjunction with an autoencoder for physical layer design to learn a suitable modulation scheme. Since our system learns an explicit model for the channel, we can use transfer learning to adapt more quickly to changes in the environment. We apply our model to millimeter wave communications with its challenges of phased arrays with a large number of antennas, high carrier frequencies, wide bandwidth and complex channel characteristics. We experimentally validate the system using a 60 GHz FPGA-based testbed and show that it is able to reproduce the channel characteristics with good accuracy.

CCS CONCEPTS

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

KEYWORDS

Machine Learning, Neural Networks, Channel Modelling, Physical Layer, Autoencoder, Gaussian Mixture Networks

MSWiM '20, November 16–20, 2020, Alicante, Spain

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

ACM Reference Format:
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 *ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, November 16–20, 2020, Alicante, Spain.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3416010.3423229

1 INTRODUCTION

Machine Learning (ML) has proven to be a very powerful tool in diverse areas such as language processing and image recognition, where it is difficult to design conventional algorithms. In networking, ML has been widely applied at higher layers, for example for resource allocation optimization and traffic prediction [34]. In contrast, wireless physical layer designs typically use conventional discrete and highly optimized signal processing blocks with well defined functions to cope with the multiple impairments of communication channels. Modulation, coding schemes and equalization techniques are optimized under the assumption of Gaussian noise processes and simple non-linear models [33], with the aim of simplifying hardware implementation. While such designs do not take into account possible interactions between non-linearities and residual errors of individual blocks, they work well for current wireless networks and allow to operate links close to their theoretical capacity. However, hardware imperfections and non-linearities become more and more difficult to model and compensate explicitly as communication environments become more complex and radio hardware is scaled to higher and higher data rates and frequencies. This is exacerbated 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 the physical layer. ML has been used for modulation recognition [3, 21, 25, 29], encoding and decoding [2, 11, 20–22, 32], channel estimation [28] and equalization [6, 12, 15, 31]. The first proposal to design end-to-end communication systems using deep learning models was presented in [21] where the authors discuss how to learn a transmitter-receiver architecture represented as neural networks (NN) for a specific channel model using an autoencoder structure. An autoencoder learns an efficient representation or encoding of data that is subjected to noise, together with a reconstruction mechanism that aims to regenerate the data as close as possible to the original

input. The autoencoder is trained using the Stochastic Gradient Descent (SGD) method [13] to optimize a loss function (e.g., bit error rate (BER)), using a differentiable model for the channel in order to perform backpropagation. Such *model-aware* systems require an analytical function of the channel as input, for example using an Additive White Gaussian Noise (AWGN) channel or simple models for phase offset and Carrier Frequency Offset (CFO) [11]. However, as the channel becomes more complex due to interference, hardware imperfections, and quantization effects, building realistic analytical functions becomes difficult, and they may no longer be differentiable, restricting the use of the backpropagation algorithm. In some cases, models for hardware effects and the channel itself are not even known, such as in molecular communications [19]. Recently, *model-free* approaches have been proposed that do not require a channel model as input. Instead, in [22], a differentiable channel model is learned from channel data using Generative Adversarial Networks (GANs). However, this mechanism is only evaluated for simple simulated AWGN and similar channels, but not for practical real-world channels. Further, it may be difficult to generalize the architecture to more complex channels [10]. Another approach was presented in [2], introducing a different training method in which the end-to-end learning is done by sampling the channel, but no explicit channel model is obtained. One of the benefits of learning an explicit channel model is that transfer learning can be used to speed up the training on time-variant channels using previous knowledge to make the system robust to environment changes. Learning explicit channel models is also interesting in the case of molecular communication to drive the design of better nanoscale networks. Here, we will focus on millimeter-wave (mm-wave) communications, where the increase in carrier frequency and bandwidth is pushing Radio Frequency (RF) hardware to its limits and complex calibration methods are used to ensure good performance. More specifically, Giga-sampling rate converters with limited resolution, as present in mm-wave hardware, introduce non-negligible quantization effects [8]. Besides, power amplifier non-linearities are more harmful and high carrier frequencies introduce critical CFO and phase noise effects [8, 9]. Last but not least, mm-wave communication hardware usually integrates phased antenna arrays with a large number of antenna elements to deal with the high path-loss by means of beam forming and to exploit spatial diversity in multi-input multi-output (MIMO) systems. Such large antenna arrays require complex calibration methods to ensure matching between the phase shifters and amplifiers of antenna elements [5, 27]. While careful optimization of each hardware component independently can partially abate the hardware imperfections, this magnifies the design complexity, and increases time-to-market and costs. Future THz frequency systems further exacerbate this situation and make ML a highly attractive tool to design and optimize end-to-end communication systems.

In this paper, we propose a model-free ML approach to obtain the explicit stochastic channel. The proposed neural network can accurately learn complex stochastic channel models and, together with previous works on synchronization, equalization [2] and continuous transmission [11] improves the current state of the art of end-to-end optimization algorithms. Our main idea is to learn the conditional distribution of each symbol of the constellation proposed by the transmitter (the encoder) using a neural network to estimate the parameters of a mixture of Gaussian distributions, known as Mixture Density Network (MDN) [4]. This approach allows us to learn the channel model from data, while remaining differentiable when it is introduced in the autoencoder structure, since it remains in the form of a neural network. We prove the convergence of the neural network output to the real channel distribution, since its loss function is convex with a minimum value equal to the distribution's entropy when the output distribution equals the target. In contrast, with GAN-based approaches the convergence during training does not have a direct statistical meaning. We study the capability of the proposed network to approximate simple channels and then extend the analysis to practical real-world mm-wave channels by means of experiments with an FPGA-based 60 GHz platform [16]. We compare the performance of an end-to-end autoencoder trained with our mixture density neural network to that of an autoencoder using a known channel function, and show that they have the same performance over an AWGN channel. In addition, we show that having an explicit channel model is beneficial to rapidly explore the channel and adjust to changes in the environment.

2 RELATED WORK

In recent years, ML has been applied to wireless physical layer design, showing that it is a promising and exciting new field with a range of potential applications. It can be used, for example, for modulation recognition with Deep Learning (DL) using classification techniques [3, 25, 29]. Equalization and sampling synchronization have been explored in [6, 12, 15, 31]. In this paper, we focus on end-to-end training of communication systems by means of autoencoders [2, 11, 20–22, 32]. It is also of interest to molecular communications [19]. For example, [23] provides a physical end-to-end model for the emission, diffusion and reception processes, whereas [7, 18] model them using a machine learning approach.

Autoencoder training methods can be divided into two categories: i) model-aware, in which a channel function is defined analytically and the training evaluates this function and ii) model-free, in which no analytical function of the channel is needed and the channel is instead approximated from the data itself.

2.1 Model-aware approach

In [21], the authors use an additive noise layer to replicate the effects of an AWGN channel, showing that the autoencoder can perform as well as Quadrature Phase Shift Keying (QPSK). In the design proposed in [2], the autoencoder is able to outperforms QPSK by learning a channel coding that encodes multiple symbols together for AWGN and Radial Basis Function (RBF) channels.

A more complex channel model, considering up-sampling, pulse shaping, constant sample time offset, constant phase offset, CFO and AWGN is used in [11]. It mimics the effects of a transmitter and receiver without perfect synchronization and with a mismatch in their oscillator frequencies. After training the autoencoder with these parameters, the receiver network is fine-tuned based on real data. However, when tested with real hardware implementations, the use of inaccurate channel models leads to performance loss compared to conventional modulation schemes such as M-ary Quadrature Amplitude Modulation (M-QAM).

2.2 Model-free approach

The first approaches to learn a differentiable channel model were presented in [22, 32]. They are based on variational GANs to learn the stochastic channel distribution. GANs [14] are formed by a generator network and a discriminator network that are commonly used in image generation. The generator produces channel samples from a random noise input trying to "fool" the discriminator with these samples. The discriminator is optimized to differentiate between real and generated samples. Variational GANs extend this concept by adding a sampling layer to the generator network to sample the latent representation space adding stochasticity to the network. Variational GANs are used in [22] to learn a differentiable channel model from channel data generated from a simulated AWGN and an additive Chi-Squared channel with a Binary Phase Shift Keying (BPSK) system. A review of this work [10] shows that this approach may not be valid for a fading channel, even when the generator degrees of freedom are increased, and the generalization of this approach to more complex channels and non-Gaussian channels may be difficult. Similarly, [32] uses GANs to learn the channel for the autoencoder, but unfortunately the work does not present metrics to evaluate whether the learned distributions match the channel, and the approach is not studied for real channels.

In contrast, in [2] the autoencoder network is split between transmitter and receiver network functions, which are jointly optimized by means of alternate training without a channel function. First, the receiver is trained with the real gradient as it only requires information about the channel output. Then, the transmitter is trained using a gradient approximation obtained by sampling the channel. In order to approximate the gradient at the transmitter, the authors relax the transmitter output/channel input to be a random variable x that follows a distribution $\delta(x - \bar{x})$, where δ is the Dirac distribution. It proves that if $p(y|x)$ is differentiable with respect to x, the gradient approximates well the real gradient of the transmitter. However, this approach does not allow to extract an explicit channel model. Our approach differs from the work above in that we provide a neural network that gives an explicit channel model, which we validate in real channels.

3 SYSTEM MODEL

We first discuss the autoencoder architecture for end-to-end optimization of the encoder and decoder and then present our approach to learn an explicit channel model using a Gaussian mixture network. We show how to train the autoencoder with this Gaussian mixture channel network and finally discuss how to speed up the training using transfer learning for the learned channel model.

3.1 Autoencoder model

Both model-aware and model-free approaches share the autoencoder architecture. As shown in Fig. 1, a simple point-to-point communication model consisting of a transmitter, a channel, and a receiver can be modelled as an autoencoder [21].

The autoencoder [13] aims to obtain a compressed representation of the input data. This is accomplished by means of finding a low dimensional representation of the transmitted bits at an intermediate layer, in this case a symbol representation, which allows

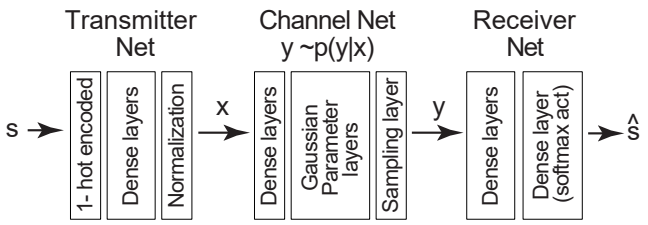


Figure 1: Autoencoder structure with Mixture density channel neural network

the reconstruction at the output with minimal error. The transmitter communicates messages $s \in \mathcal{M} = \{1, 2, \ldots, M\}$ to the receiver, making use of N ($2N$) complex (real) channel uses. The transmitter network T, modulates this message into a symbol $x = T(s)$, such that $x \in \mathbb{R}^{2N}$, with certain constraints, such as an average power constraint, $\mathbb{E}[x^2] < 1$ or amplitude constraint, $|x_i| < 1 \quad \forall i$. The channel takes x as an input variable and its output y follows the conditional probability distribution $y \sim p(y|x)$. The receiver network R decodes y to obtain an estimate of the original message $\hat{s} = R(y)$. T and R are the functions corresponding to the networks of transmitter/encoder and receiver/decoder inside the autoencoder, as shown in Fig. 1.

3.2 Gaussian mixture channel network

We now describe our proposed model-free approach to estimate the channel using a neural network architecture to obtain the Gaussian mixture representation of the conditional distribution of the symbols $p(y|x)$. The Gaussian mixture network is trained using samples of the channel. This allows to replicate any communications channel. It further does not require any specific properties of the conditional probability distribution $p(y|x)$, since the fully connected neural network approximation allows back propagation.

Statistical modeling of a variety of channels has been approached mathematically using finite mixture distributions [26]. Gaussian mixture distributions are proven to approximate any given density with arbitrary accuracy, as can be shown using Weiners approximation theorem [17, 24]. For this reason, a neural network that estimates the Gaussian distribution that best fits the channel is highly useful. The advantage of this approach compared to GAN approaches for learning the channel is that convergence of the neural network can be proven to be related to closeness of approximation to the desired distribution in statistical terms, while with a GAN the loss in training is not directly related to the statistical approximation of the channel distribution.

We make use of MDNs [4], deep neural networks that estimate the parameters of the conditional distribution of a point. Unlike regression approaches, an MDN not only learns the mean value of the distribution at each point x but also the noise distribution, including the case of continuous distributions. Therefore, the conditional probability density of the measured data is represented by

$$p(y|x) = \sum_{i=1}^{k} \pi_i(x)\phi(y|x) \ , \tag{1}$$

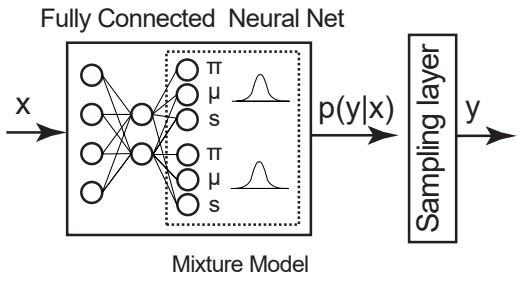


Figure 2: Overview of the proposed channel layer using a mixture density network to obtain the conditional distribution and a sampling layer.

where k is the number of mixture components, π_i are the mixing coefficients of the probabilities of drawing a sample from each mixture component i, and $\phi(y|x)$ are the kernel functions representing the conditional densities of y. The mixing coefficients satisfy $\sum_{i=1}^{k} \pi_i(x) = 1$. The kernel functions are chosen to be Gaussian:

$$\phi_i(y|x) = \frac{1}{\sqrt{2\pi\sigma_i(x)^2}} \exp \frac{-\|y - \mu_i(x)\|^2}{2\sigma_i(x)^2} \ , \tag{2}$$

where μ_i and σ_i^2 are the mean and variance of each mixture, respectively. The parameters that best fit the training data will be the output of the neural network with input x, as shown in Fig. 2. The MDN can approximate any conditional density arbitrarily well by choosing a sufficiently high number of hidden units [4].

The network architecture is presented in Table 1. It has two main hyperparameters: the number of mixtures per conditional distribution and the number of hidden units for layers 2 and 3, N_{hidden}.

Table 1: Channel layers: layers 1-7 are for parameter estimation of the Gaussian mixture and layer 8 is a sampling layer of the estimated distribution

Number	Layers	Output Dimensions
1	Input	$2N$
2	*Dense* (act = relu)	N_{hidden}
3	*Dense* (act = relu)	N_{hidden}
4	$Dense_{\mu}$ (act =linear)	$k \cdot 2N$
5	$Dense_{\sigma}$ (act = custom elu)	$k \cdot 2N$
6	$Dense_{\pi}$ (act= softmax)	$k \cdot 2N$
7	Concatenate [4,5,6]	$k \cdot 2N \cdot 3$
8	Sampling layer	$2N$

The input layer corresponds to the number N of complex channels uses, corresponding to $2N$ input dimensions. Then, layers 1 and 2 are two fully connected layers with universal approximation capabilities with *relu* activation functions. The output of the fully connected layers is the input to three different dense layers, one for each vector of parameters of the Gaussian mixtures μ_i, σ_i, π_i for $i = 1, ..., k$. The output of each layer is the number of mixtures k multiplied by the number of expected outputs, which corresponds to the real number of channel uses $2N$. In order to ensure that the $\pi_i \in [0, 1]$ and they sum to unity, the activation of the layer $Dense_{\pi}$ is a softmax function:

$$\pi_i = \frac{e^{z_i}}{\sum_{j=1}^{J} e^{z_j}} \ , \tag{3}$$

where z_i are the outputs of the previous layer. To avoid small values in the variance, the outputs of the $Dense_{\sigma}$ layer are followed by an exponential linear unit *elu* activation function:

$$R(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha(e^z - 1) & \text{if } z \leq 0 \end{cases} \ , \tag{4}$$

where α is the scale of the negative factor. We customize the *elu* activation function to ensure the positivity of the variance outputs:

$$R_{\text{custom}}(z) = R(z) + 1 + \epsilon \ , \tag{5}$$

where ϵ is a small constant.

Once the parameters of the mixture have been estimated by layers 1 to 7, the sampling layer builds the conditional density function $p(y|x)$ for each input x, taking samples from it to obtain y. We call the resulting deep neural network a GM-Channel network.

3.3 Transmitter and receiver networks

To obtain the full autoencoder structure together with the proposed Gaussian mixture channel layers, we need to implement the transmitter and receiver layers of the autoencoder. A generic architecture of the transmitter and receiver networks is presented in Fig. 1, where the channel network corresponds to the Gaussian mixture layers described in Section 3.2. The input to the transmitter is a one-hot encoded vector, followed by multiple fully connected layers and a normalization layer, that restricts the average energy per symbol or amplitude per channel use.

The receiver architecture is formed by a succession of dense layers and a final dense layer with a *softmax* activation function, ensuring that the output of the autoencoder is a probability vector. Outside of the autoencoder network, the maximum of this vector is chosen as the resulting message. This *argmax* activation is not included in the network as it is not differentiable. The effect of the transmitter and receiver that are unknown or not differentiable are included as channel effects.

3.4 Training methods

In this section, we first show how to train the model-free autoencoder with the channel network. We then discuss how to train the GM-channel network for a specific channel. Further, we propose an improved training for static channels, that allows to train the model-free autoencoder faster. Finally, we discuss how to speed up the training for time-varying channels with a transfer learning approach. This allows to further reduce the overhead of training and adapt faster to time-varying channels.

In general, model-aware training is faster than model-free training. For a model-aware approach where the channel is known and static, the autoencoder is trained once for N_e epochs and a training time of t_a. For our model-free approach, the channel network needs to be trained at each epoch of the autoencoder training, to learn the channel for the chosen modulation of the encoder. Therefore, training the model-free model will take the time to train the autoencoder plus the training time of the channel network at each epoch,

$t_a + N_e \cdot t_c$. However, $t_c \ll t_a$, since the network is significantly less complex than the autoencoder network and Gaussian parameters can be estimated with a small sample size, which makes the channel network converge much faster.

The training of the full autoencoder structure with the GM-channel network is described in Algorithm 1. First, the modulation of the encoder network is obtained by sampling the possible messages $s \in \mathcal{M}$. Then the obtained symbols $\{x^q\}$ are passed by the real channel to obtain training samples for the channel network, $\{y^q\}$. The GM-Channel network is trained with these samples as described in Section 3.4.1. The GM-Channel network is passed to the autoencoder and the weights corresponding to these layers are frozen. The autoencoder network is updated by a single step of SGD. Since the encoder and decoder have been updated, the modulation constellation has changed and therefore the weights of the GM-Channel network need to be re-initialized. This process is done iteratively for every epoch. The loss function for the autoencoder is the *cross-entropy loss*.

Algorithm 1: Training of the autoencoder with the GM-Channel Net

```
for i = 1 : N_Epochs do
    Obtain modulation x^q = Encoder ({s^q})
    y^q=Sample channel ({x^q})
    Train GM-Channel Net({x^q, y^q})
    Freeze weights of GM-Channel Net
    Train autoencoder(GM-Channel Net,{s^q})
    Initialise GM-Channel Net weights
end
```

3.4.1 Training the channel network. We now discuss how the GM-Channel network is trained and the loss function that we use. The network training includes the layers 1-7 described in table 1, since the last layer is a sampling layer that will only be used when training the full autoencoder model.

As described in Algorithm 1, the training data for the GM-Channel network is $\{x^q, y^q\}$, where $\{x^q\}$ are the constellations points obtained from the encoder and $\{y^q\}$ are the result of passing the constellation points trough the channel $h(x)$. Note that the output of the GM-Channel network consists of the parameters of the Gaussian mixtures of the conditional distributions and not the samples of the network. For this reason, the loss is calculated as the probability that the training outputs $\{y^q\}$ are sampled from the predicted distribution, as we explain next.

The training loss is calculated as follows. The predicted value y_{pred} obtained from inputting a symbol x to the GM-Channel network contains the concatenated parameters of the Gaussian mixture as:

$$y_{\text{pred}} = [\mu_1^p, \ldots, \mu_k^p; \sigma_1^p, \ldots, \sigma_k^p; \pi_1^p, \ldots, \pi_k^p] \ . \tag{6}$$

A mixture model is constructed from these components, where p indicates that these are the predicted values from the network. The loss is defined so as to maximize the likelihood $\mathcal{L}$, that given x, the n training output data samples were sampled from the distribution.

$$\mathcal{L} = \prod_{q=1}^{n} p(y^q|x^q) \ , \tag{7}$$

which is equivalent to minimizing the negative logarithm of $\mathcal{L}$. Therefore the loss can be written as:

$$L(p) = -\frac{1}{n}\sum_{q}^{n} \log p(y^q|x^q) = -\frac{1}{n}\sum_{q}^{n} \log\left(\sum_{i=1}^{k} \pi_i^p(x^q)\phi_i^p(y^q|x^q)\right) \ . \tag{8}$$

We denote the actual conditioned probability density function by $\bar{p}(y|x)$.

We now prove that with this loss function, by increasing n and q the solution converges to the real probability with a loss value equal to the distribution's entropy. This is important since it relates the loss function convergence to the closeness of the approximation to the desired distribution.

Theorem 3.1. *The loss function $L(p)$ defined above is convex and it is minimum when $p(y^q|x^q)$ is equal to the actual probability almost everywhere (over its whole domain but for a null-measure set of points).*

Proof. Since the negative logarithm is convex in the field of distributions over $\mathbb{R}^N$ and the sum of convex functions is convex, also our defined loss function $L(p) = -\frac{1}{n}\sum_q^n \log p(y^q|x^q)$ is convex over the space of possible distributions $p(y^q|x^q)$. To prove the minimum point, note that when increasing n, the loss function converges uniformly to $\mathbb{E}(-\log(p(y|x)))$ because the numerical mean converges to the statistical mean. We will prove that for every distribution $p(y|x)$ that is different from the real distribution $\bar{p}(y|x)$ in a non-null measure set, the inequality $h(X) - \mathbb{E}(-\log(p(y|x))) < 0$ holds and since for $p(y|x) = \bar{p}(y|x)$ the equality is satisfied because $h(X) = \mathbb{E}(-\log(\bar{p}(y|x)))$ is the differential entropy definition, the proof will be complete.

$$h(X) - \mathbb{E}(-\log(p(y|x))) = \mathbb{E}(\log(\frac{p(y|x)}{\bar{p}(y|x)})) \ . \tag{9}$$

Since the logarithm is a strictly concave function, if $p(y|x)$ and $\bar{p}(y|x)$ are different in a measurable set, then $\frac{p(y|x)}{\bar{p}(y|x)}$ will not be constant in a non-null measure set. Thus, applying Jensen's inequality for strictly concave functions we have that

$$\begin{aligned}\mathbb{E}(\log(\tfrac{p(y|x)}{\bar{p}(y|x)})) < \log(\mathbb{E}(\tfrac{p(y|x)}{\bar{p}(y|x)})) = \\ \log(\textstyle\int \frac{p(y|x)}{\bar{p}(y|x)}\bar{p}(y|x)) = \log(\int p(y|x))) = \log(1) = 0 \ .\end{aligned} \tag{10}$$

Note that the inequality is strict because Jensen's inequality only holds as equal when the values it is applied to are constant almost everywhere (except for a null measure set, which is not the case). □

For each epoch of the autoencoder network the gradient is updated and the transmitter weights are updated, changing the modulation obtained at the transmitter. Therefore, the constellation changes and the GM-Channel network would have to be retrained. If training time is not an issue or the channel is not static, we retrain the network as presented in Algorithm 1. Under the assumption of a relatively static channel, however, we can take advantage of the extrapolation capabilities of neural networks and learn a generalized channel by changing the training to a generalized constellation. This way, the channel layer does not have to be retrained as explained in the next section.

3.4.2 Generalized training of the GM-Channel network. In cases where the channel does not change significantly over the time scale that training takes, or we are interested in optimizing the hardware imperfections that remain unchanged with environment changes, we can speed-up the training by training a generalised GM-Channel network. In this case, the GM-Channel network is trained only once prior to choosing the modulation and thus has to include a sufficiently large set of points sampled from the channel. This avoids retraining the GM-Channel network for every update of the autoencoder by learning the distribution across the whole channel and not only at the chosen constellation points. This reduces the total training time of the model-free approach to $t_a + t_c$.

In systems with quadrature phase keying, the quadrature and phase values are limited in range, say $x \in [-1, 1]$. With sufficient training samples, the neural network can learn the distribution of the channel for the range of values of amplitude and phase, and extrapolate the conditional distribution of the unknown values.

For the different options of sampling the space for training points, we compare sampling uniformly at random, using grid sampling, and with a random constellation initialization of the encoder network. We use the same number of training points sampled and the same channel model to obtain the GM-Channel networks that are then used to train autoencoders with a 4-point constellation. As shown in Fig. 3, the autoencoder loss is lowest for the GM-Channel network trained with grid sampling.

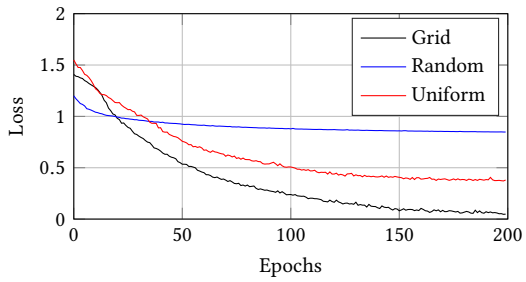


Figure 3: Loss of autoencoder trained with the generalized channel layer with different channel sampling options.

3.4.3 Reducing training overhead in time varying channels. In the case the channel varies due to environment changes, the channel network needs to be retrained. Having an explicit channel model in the form of a neural network allows to use transfer learning approaches to more rapidly adapt to these changes. Current literature on autoencoder end-to-end optimization has been focused on offline training, which cannot be applied to time-varying channels. However, model-aware approaches could be adapted to channel changes within the same channel model by estimating the parameters of the channel functions and retraining the autoencoder model. Similarly, for our model-free approach, the channel network needs to be trained on new data whenever the channel changes, and then the autoencoder is retrained with this updated channel network. However, training takes place on longer time scales than the duration of a frame, and additional measurements need to be obtained from the channel in order to train the GM-Channel network.

Therefore, the required number of channel samples and the time to train the channel network are critical, as this affects how often the model can be updated. This time can be reduced by using transfer learning to learn the new channel with a minimum number of samples. Having learned an initial channel, the new channel can be approximated by the previous channel network trained with the new channel's samples. This approach is known in transfer learning as model reuse. It avoids training the model from scratch with a larger dataset, which would take longer to obtain. In the following evaluation, we show results for per-epoch training, training with a generalized channel model, and training with model reuse.

4 EVALUATION OF THE CHANNEL NETWORK

We implement our GM-Channel network and the autoencoder using the Tensorflow framework [1]. We first evaluate the accuracy of the GM-Channel network for a simple QPSK AWGN channel, where the expected parameters of the Gaussian mixtures are known and can be evaluated. We then show the benefits of having an explicit channel model for the case of time-varying channels. Finally, We compare the end-to-end autoencoder trained with the GM-Channel network to the corresponding model-aware approach and QPSK.

4.1 Simple AWGN channel

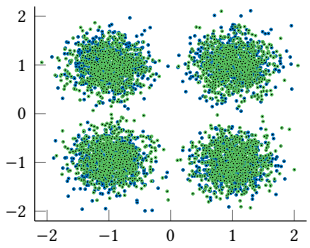
Let us first examine a simple example using an AWGN channel for a quadrature and phase QPSK modulation where the symbols are $x \in \{[-1, 1], [1, 1], [-1, -1], [1, -1]\}$, and $p(x)$ is a discrete uniform random variable over these values. We consider that the channel function is given by $p(y|x)$ such that

$$y = x + n \tag{11}$$

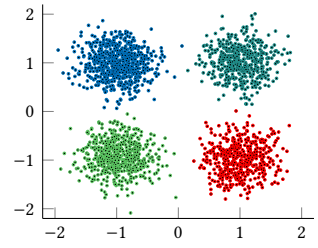
We have $n \sim N(0, \sigma_h)$, with σ_h corresponding to an SNR of 7db. For simplicity, we fix the symbols in this example (but they could be updated at every epoch by the autoencoder as described in section 3.4, using the learned function for the channel). The GM-Channel network is trained as described in section 3.4.1, using $N_{\text{hidden}} = 20$ and one Gaussian mixture, $k = 1$. The result of the training is shown in Fig. 4a, where the training points from the measured channel ground truth for all symbols are shown in blue and the sampled points from the predicted distributions in green. The channel network approximation accurately reflects both the mean and variance of the channel distribution. Fig. 4b shows the conditional distribution $p(y|x)$ of the symbols in a different color. Our algorithm both learns the conditional distribution of the channel for each symbol and captures very well the signal to noise ratio.

Since for this simple channel the true parameters of the distribution $\bar{p}(y|x)$ are known, we study the error between the estimated parameters and the real parameters for different SNR values ranging from -4 dB to 10 dB. The results are presented in Fig. 4c and Fig. 4d. We observe that the error does not depend on the noise level. The error of the mean value for the different SNRs of the mixture distributions is $\mu_{\text{error}} = 2.06 \pm 0.56\%$. For the sigma parameter of the mixtures the error of the variance is $\sigma_{\text{error}} = 1.33 \pm 0.41\%$. Such small error magnitudes show that our network approximates well the parameters of the conditional QPSK-AWGN channel response.

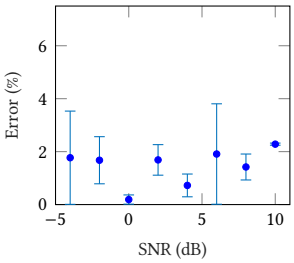
In addition, Fig. 4e shows the probability mass function of a single symbol where it can be seen that the actual marginal probability mass function $\bar{p}(y|x = [1, 1])$ matches the one predicted by the



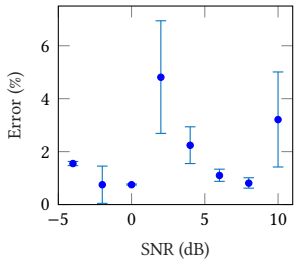
(a) Measured (blue) and sampled symbols using the GM-Channel network (green)



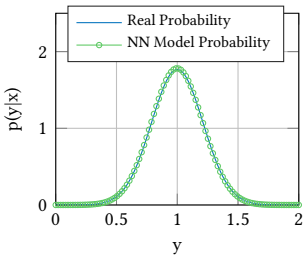
(b) Conditional channel distribution $p(y|x)$, learned with the GM-Channel network for each symbol



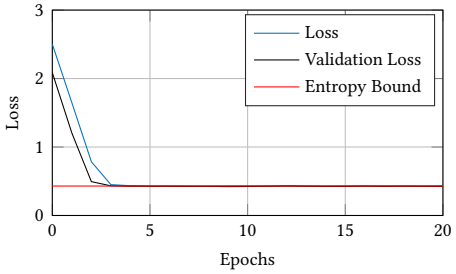
(c) μ error distribution



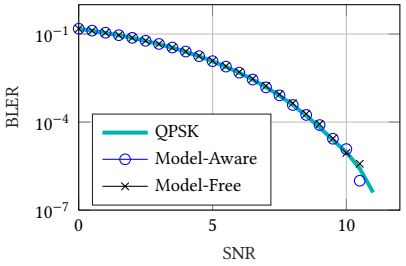
(d) σ error distribution



(e) Marginal mass function of the conditional channel distribution



(f) Loss function of the GM-Channel network converges to the entropy of the channel.



(g) Block error rate of QPSK, model-aware and our model-free autoencoder

Figure 4: Parameter estimation for a AWGN channel for different Signal-to-Noise Ratio (SNR) values.

GM-Channel network when x_1 is given as input. This shows that our model also approximates well the probability distribution of the channel. The convergence of the loss function of the GM-Channel network to the entropy of the actual channel distribution is shown in Fig. 4f, with entropy $H(x) = 0.42993$ and the $Loss = 0.43366$. From the result presented in theorem 3.1, when the loss function converges to the entropy of the actual distribution $\tilde{p}(y|x)$, the estimated distribution is equal to the actual probability distribution.

Finally, we use the full end-to-end training together with the encoder and decoder networks as presented in Fig. 1. We compare the modulation obtained with our model-free approach to a model-aware autoencoder with a Gaussian noise channel and to conventional QPSK modulation. The training was done for an SNR of 7 dB. The results in Fig. 4g show that both the model-free and model-aware networks achieve a block error rate (BLER) close to the one achieved by conventional QPSK encoding. This result is expected when the actual channel is known by the model-aware approach and other effects that could degrade the performance of QPSK are not present.

4.2 Time-varying channel

To validate the model reuse training explained in section 3.4.3, we consider a channel in which the SNR changes, and show that the original channel network can be used to reduce training overhead. We use a channel with AWGN noise with $\sigma = 0.05$ using a grid of symbols in the $[-1, 1] \times [-1, 1]$ range. The GM-channel network is trained for this channel during 100 epochs, which we refer to as the offline phase. Then, the channel changes to a AWGN with $\sigma = 0.30$. We consider a small number of 50 samples per symbol of the new channel for training. The GM-Channel network is retrained with these samples over 50 epochs. The results of training are shown in Fig. 5. Since the GM-Channel network had already learnt a similar channel during the offline phase (in blue), the retrained network continues learning having a validation loss equal to the loss on the new data during training (red). We compare the retrained network to a new channel network trained only with the samples obtained from the new channel. The loss of the network trained from scratch during 150 epochs is shown in black. The validation loss is worse for this network as the channel network suffers from being trained on a small dataset. This result shows that transfer learning can help train on a new channel faster as it can use previous knowledge to adapt to the new channel.

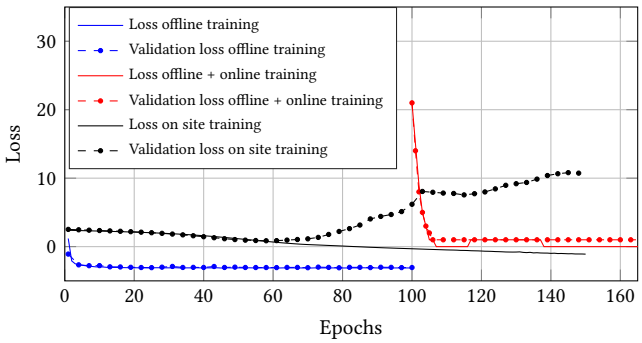


Figure 5: Offline plus online training versus on site training

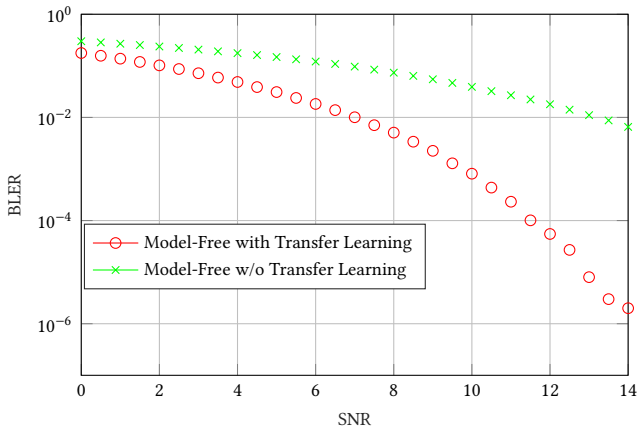


Figure 6: Block error rate for the model-free approach with and without transfer learning in a time-varying channel

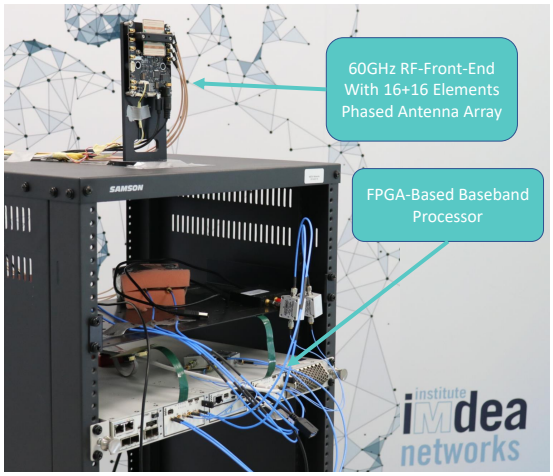


Figure 7: Equipment used for the channel measurements.

We further show in Fig. 6 the BLER results for the time-varying channel where we train an autoencoder once with the retrained channel network (in red) and once learning the channel network from scratch (in green). The autoencoder trained with the retrained channel network outperforms the autoencoder that has been trained with a less accurate channel network using only a few channel samples. Therefore, learning a better model of the channel quickly by using transfer learning helps to maintain performance when the channel changes and there is only a small data set to train on.

As shown in the previous sections, the GM-Channel network is able to learn the properties of the channel and obtain an explicit channel model. The application of our model-free approach is primarily of importance for more complex channel effects that include non-linearities. We show this in the next section, where we use our model to approximate the channel distribution of a real mm-wave channel in an experimental testbed.

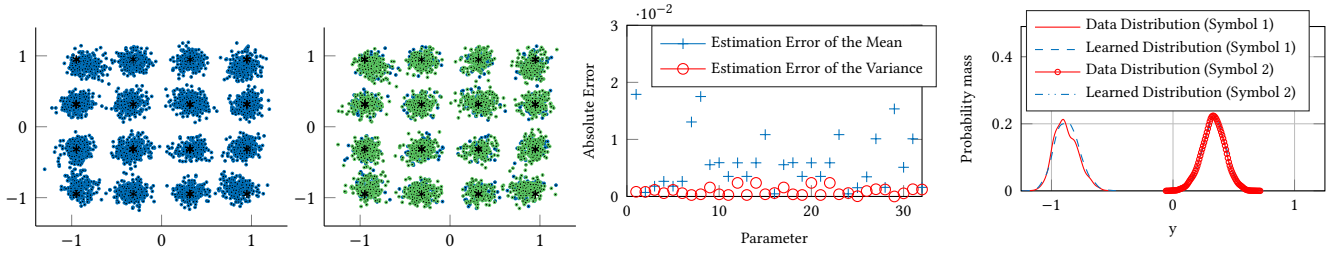
5 EXPERIMENTAL VALIDATION

One of the main advantages of our proposal is that it is able to obtain the conditional distribution of the channel from experimental data, where the combination of wireless channel effects and transmitter/receiver imperfections result in a more complex channel that is harder to model. Real channels also include non-linearities and imperfections due to hardware limitations, that may not be fully corrected by conventional signal processing algorithms or for which an analytic expression may be unknown.

To test our proposal with real mm-wave channels, we present a practical implementation of the channel learning using a custom ultra wideband experimental platform composed of an FPGA-based baseband processor operating with 2 GHz of bandwidth, connected to a Sivers 60 GHz RF front-end with a 16+16 element phased antenna array [16]. Fig. 7 shows the experimental platform and its main components. We set up the testbed in an indoor laboratory environment, with the transmitter phased antenna array front-end at 4 m distance from the receiver antenna array. Transmitter and receiver are in Line-of-Sight (LOS). The experiments are performed by continuously sending blocks of symbols, which are then captured and stored in DDR memory on the FPGA platform to be used as training data to obtain the channel model.

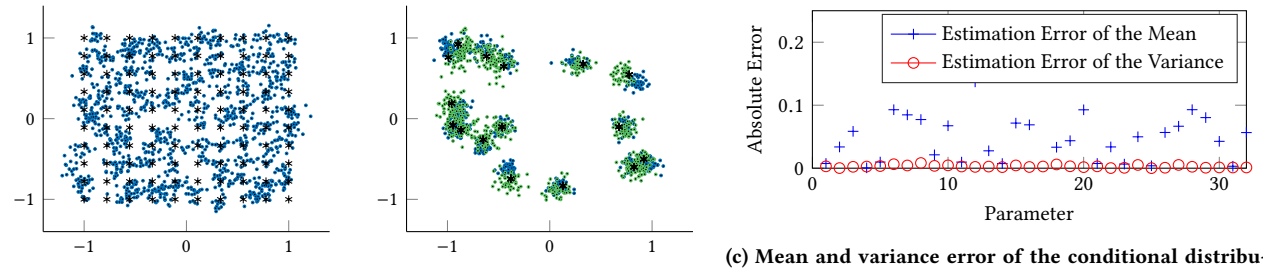
5.1 16-QAM model from real experimental data

To perform this experiment, we generate random bits which are 16-QAM modulated and grouped in blocks of 448 symbols, preceded by a guard interval of 64 symbols, according to the IEEE 802.11ad PHY specification [30]. We group 9 such blocks in a packet, and include a packet preamble which follows the structure of IEEE 802.11ad single carrier frames [30]. The purpose of the preamble is to perform time synchronization, estimate and compensate CFO and phase offsets, and perform channel estimation and equalization. The packets are transmitted over one of the 2.16GHz channels defined by the IEEE 802.11ad standard. In Fig. 8a we show the IQ constellation of the received data which is used to train the GM-Channel network. We observe that despite the corrections made based on the packet preamble, the constellation points around the corners are distributed towards the center, in contrast to the remaining points in the constellation. Such effects are produced by saturation of the power amplifier and non-linear gains, among others, which elongate the distributions of those symbols more in one dimension than others. The learned distribution is presented in Fig. 8b, where the sampled points obtained from the GM-channel network are shown in green and the real distribution is shown in black We observe that each symbol's distribution is accurately approximated and the elongations of the symbol's distributions in the corners of the IQ constellations together with more complex channel effects are well captured. This result shows that the GM-Channel network is able to reproduce the stochastic channel even for the symbols in the corners, which would not be captured with a simple AWGN channel model. To obtain a measure of accuracy of the learned distributions, the mean and variance of the measured distributions are compared to the parameters estimated from the GM-Channel network. In total there are 32 parameters, corresponding to the 16 symbols and 2 dimensions. We compare the parameters of the real samples to the mean and variance parameters by sampling the learned distribution of each symbol. The results of the absolute



(a) 16-QAM real data (b) Learned channel data (c) Mean and variance error of the symbol conditional distribution (d) Probability mass distribution of two symbols (real and sampled)

Figure 8: Results of the 16-QAM channel measurements obtained using the experimental setup and learned with the GM-Channel network.



(a) Training grid samples for GM-Channel net (b) Sampled symbols from predicted distributions (c) Mean and variance error of the conditional distribution of each symbol for uniformly sampled constellation.

Figure 9: Results for the generalized training of the GM-Channel network on experimental data.

errors are presented in Fig. 8c. Both, the variance and mean have small errors considering the scale of y, showing that our model is able to accurately reproduce the characteristics of the real channel. Finally, we estimate the probability mass function of the distributions of two symbols from the constellation to show that they are well approximated by the channel network. The results are shown in Fig. 8d, where we can observe that different distributions are obtained for different symbols and that the GM-Channel network captures both very well.

5.2 16-point constellation with generalized training

In this section we study how the generalized training presented in Section 3.4.2 can estimate unseen constellations in the case of static channels. Generalized training allows to reduce the training time of the end-to-end autoencoder as the channel network does not need to be updated after each autoencoder epoch. We use our millimeter-wave experimental platform to send blocks of symbols of an IQ constellation grid shown as black dots in Fig. 9a. The received symbols are also shown in Fig. 9a in blue, where it can be seen that a denser IQ constellation reveals more harmful joint effects from the channel and the transmitter/receiver system. This is a well-known effect for mm-wave single carrier systems, mainly due to the non-ideal behaviour of hardware components present in the system. It is the main reason to restrict practical implementations to low-order modulation schemes.

We use this data from the experimental setup to train the GM-Channel network. In order to test how well a new constellation is estimated, we consider 16 symbols uniformly sampled in the range $[-1, 1]$ and obtain the ground truth of their channel distribution using the experimental set up. The results are shown in Fig. 9b, where the green symbols are obtained using the trained GM-Channel network with the sampling layer, the blue points correspond to the real channel response of the 16-symbol constellation sent multiple times through the 60 GHz wireless channel and captured with the testbed. The distribution obtained from the channel network is close to the real symbol distribution. However, some small differences are observed. The errors of the mean and variance for the conditional distribution of the uniformly sampled symbols are plotted in Fig. 9c. The errors are larger for the mean parameters, while staying small for the variance. Remarkably, the trained channel is able to reproduce the statistics of the 16 uniformly distributed symbols without having been explicitly trained for this data, but extrapolating it from the dense grid training data. Higher accuracy could be achieved on unseen symbols by training the GM-Channel network using a denser grid. The generalized training therefore allows to reduce the autoencoder training time as the channel does not need to be queried for new data at every epoch.

6 CONCLUSIONS

In this paper, we introduced a new model-free approach for optimizing end-to-end communication channels using an autoencoder. This new tool, together with previous works on continuous transmission

blocks, equalization and synchronisation with an end-to-end learning approach bring us one step closer to real-time application of machine learning optimization of the wireless physical layer. Specifically, the GM-channel network we design allows learning complex stochastic channels without a parametric model. Our model can be paired with an autoencoder approach to optimize communication for any real-world channel. Furthermore, we obtain an explicit channel model, which makes it amenable to transfer learning and allows faster retraining during changing channel conditions. This reduces the overhead introduced in the communication to obtain the channel data needed to train the models. Finally, we show that our proposed channel network approximates well simulated data and experimental data obtained from a 60 GHz FPGA-based testbed.

For future work, we intend to investigate the very interesting problem to find a trade-off between exploration of the channel and an increase in overhead for the communication, using a classifier derived from the frame channel estimation to take into account how much the channel has changed. This will allow to further reduce channel training overhead without sacrificing accuracy. We further intend to join the GM-Channel network with recurrent neural networks, which can capture temporal dependencies between symbols, e.g. from inter-symbol interference and fading. We believe these end-to-end learning methods will become increasingly important in future generations of wireless networks, where the system complexity will increase due to a higher number of antennas, higher frequencies, wider bandwidths, and more complex hardware.

7 ACKNOWLEDGMENTS

This work is partially supported by the the Madrid Regional Government through the TAPIR-CM program (S2018/TCS4496) and by Ministerio de Ciencia, Innovacion y Universidades (MICIU) grant RTI2018-094313-B-I00 (PinPoint5G+).

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, g, et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, *http://tensorflow.org/*. Software available from tensorflow.org.

[2] 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.

[3] Elsayed Elsayed Azzouz and Asoke Kumar Nandi. 1996. Modulation Recognition Using Artificial Neural Networks. In *Automatic Modulation Recognition of Communication Signals*. Springer, Boston, MA, 132–176.

[4] Christopher M Bishop. 1994. *Mixture Density Networks*. Technical Report NCRG/94/004. Neural Computing Research Group, Aston University.

[5] Steve Blandino, Claude Desset, Giovanni Mangraviti, André Bourdoux, and Sofie Pollin. 2018. Phase-Noise Mitigation at 60 GHz with a Novel Hybrid MIMO Architecture. In *Proceedings of the 2nd ACM Workshop on Millimeter Wave Networks and Sensing Systems* (New Delhi, India) *(mmNets '18)*. Association for Computing Machinery, New York, NY, USA, 39–44. https://doi.org/10.1145/3264492.3264499

[6] S Chen, GJ Gibson, CFN Cowan, and PM Grant. 1990. Adaptive Equalization of Finite Non-Linear Channels Using Multilayer Perceptrons. *Signal processing* 20, 2 (1990), 107–119.

[7] Wei Chen and Andrew L Ferguson. 2018. Molecular Enhanced Sampling With Autoencoders: On-the-Fly Collective Variable Discovery and Accelerated Free Energy Landscape Exploration. *Journal of computational chemistry* 39, 25 (2018), 2079–2102.

[8] Chang-Soon Choi, Maxim Piz, and Eckhard Grass. 2010. *Non-Ideal Radio Frequency Front-End Models in 60GHz Systems*. John Wiley and Sons, Ltd, Chapter 3, 63–87. https://doi.org/10.1002/9780470972946.ch3 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470972946.ch3

[9] C.-S. Choi, Y. Shoji, H. Harada, R. Funada, S. Kato, K. Maruhashi, and I. Toyoda. 2006. RF impairment models for 60 GHz band system simulation. *IEEE 802.15–06–0396–01–003c, IEEE 802.15.3c* (2006).

[10] Mason del Rosario. 2019. Approximating the Void: A Paper Review. http://mdelrosario.com/2019/06/28/fading-gans.html Accessed: 2020-05-25.

[11] 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.

[12] Alexander Felix, Sebastian Cammerer, Sebastian Dörner, Jakob Hoydis, and Stephan Ten Brink. 2018. OFDM-Autoencoder for End-to-End Learning of Communications Systems. In *2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, 1–5.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*. MIT press, 2672–2680.

[15] Aldebaro Klautau, Nuria González-Prelcic, Amine Mezghani, and Robert W Heath. 2018. Detection and Channel Equalization With Deep Learning for Low Resolution MIMO Systems. In *2018 52nd Asilomar Conference on Signals, Systems, and Computers*. IEEE, 1836–1840.

[16] Jesús O. Lacruz, Dolores Garcia, Pablo Jimenez, Joan Palacios, and Joerg Widmer. 2020. mm-FLEX: An Open Platform for Millimeter-Wave Mobile Full-Bandwidth Experimentation). In *18th Annual International Conference on Mobile Systems, Applications, and Services (MOBISYS)* (Toronto, Canada). Association for Computing Machinery, USA.

[17] Geoffrey J McLachlan and Kaye E Basford. 1988. *Mixture Models: Inference and Applications to Clustering*. Vol. 38. New York: M. Dekker.

[18] Soha Mohamed, Jian Dong, AR Junejo, et al. 2019. Model-Based: End-to-End Molecular Communication System Through Deep Reinforcement Learning Auto Encoder. *IEEE Access* 7 (2019), 70279–70286.

[19] Tadashi Nakano, Andrew W Eckford, and Tokuko Haraguchi. 2013. *Molecular Communication*. Cambridge University Press.

[20] 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.

[21] 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.

[22] 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.

[23] Massimiliano Pierobon and Ian F Akyildiz. 2010. A Physical End-to-End Model for Molecular Communication in Nanonetworks. *IEEE Journal on Selected Areas in Communications* 28, 4 (2010), 602–611.

[24] Kostantinos N Plataniotis and Dimitris Hatzinakos. 2017. Gaussian Mixtures and Their Applications to Signal Processing. In *Advanced Signal Processing Handbook*. CRC Press, 89–124.

[25] Sharan Ramjee, Shengtai Ju, Diyu Yang, Xiaoyu Liu, Aly El Gamal, and Yonina C Eldar. 2019. Fast Deep Learning for Automatic Modulation Classification. *arXiv preprint arXiv:1901.05850* (2019).

[26] Bassant Selim, Omar Alhussein, Sami Muhaidat, George K Karagiannidis, and Jie Liang. 2015. Modeling and Analysis of Wireless Channels Via the Mixture of Gaussian Distribution. *IEEE Transactions on Vehicular technology* 65, 10 (2015), 8309–8321.

[27] Joao Vieira, Fredrik Rusek, and Fredrik Tufvesson. 2014. Reciprocity Calibration Methods for Massive MIMO Based on Antenna Coupling. In *2014 IEEE Global Communications Conference*. IEEE, 3708–3712.

[28] Chao-Kai Wen, Shi Jin, Kai-Kit Wong, Jung-Chieh Chen, and Pangan Ting. 2014. Channel Estimation for Massive MIMO Using Gaussian-Mixture Bayesian Learning. *IEEE Transactions on Wireless Communications* 14, 3 (2014), 1356–1368.

[29] Nathan E West and Tim O'Shea. 2017. Deep Architectures for Modulation Recognition. In *2017 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*. IEEE, 1–6.

[30] IEEE 802.11 working group. 2012. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band. *IEEE Standard 802.11ad* (2012).

[31] Weihong Xu, Zhiwei Zhong, Yair Be'ery, Xiaohu You, and Chuan Zhang. 2018. Joint Neural Network Equalizer and Decoder. In *2018 15th International Symposium on Wireless Communication Systems (ISWCS)*. IEEE, Portugal, 1–5.

[32] 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.

[33] A. Zaidi, F. Athley, J. Medbo, U. Gustavsson, G. Durisi, and X. Chen. 2018. *5G Physical Layer: Principles, Models and Technology Components*. Elsevier Science.

[34] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. 2019. Deep Learning in Mobile and Wireless Networking: A Survey. *IEEE Communications Surveys & Tutorials* 21, 3 (2019), 2224–2287.