An Introduction to Deep Learning for the Physical Layer

Tim O'Shea, *Senior Member, IEEE,* and Jakob Hoydis, *Member, IEEE*

***Abstract*—We present and discuss several novel applications of deep learning (DL) for the physical layer. By interpreting a communications system as an autoencoder, we develop a fundamental new way to think about communications system design as an end-to-end reconstruction task that seeks to jointly optimize transmitter and receiver components in a single process. We show how this idea can be extended to networks of multiple transmitters and receivers and present the concept of radio transformer networks (RTNs) as a means to incorporate expert domain knowledge in the machine learning (ML) model. Lastly, we demonstrate the application of convolutional neural networks (CNNs) on raw IQ samples for modulation classification which achieves competitive accuracy with respect to traditional schemes relying on expert features. The paper is concluded with a discussion of open challenges and areas for future investigation.**

I. Introduction

Communications is a field of rich expert knowledge about how to model channels of different types [1], [2], compensate for various hardware imperfections [3], [4], and design optimal signaling and detection schemes that ensure a reliable transfer of data [5]. As such, it is a complex and mature engineering field with many distinct areas of investigation which have all seen diminishing returns with regards to performance improvements, in particular on the physical layer. Because of this, there is a high bar of performance over which any machine learning (ML) or deep learning (DL) based approach must pass in order to provide tangible new benefits.

In domains such as computer vision and natural language processing, DL shines because it is difficult to characterize real world images or language with rigid mathematical models. For example, while it is an almost impossible task to write a robust algorithm for detection of handwritten digits or objects in images, it is almost trivial today to implement DL algorithms that learn to accomplish this task beyond human levels of accuracy [6], [7]. In communications, on the other hand, we can *design* transmit signals that enable straightforward algorithms for symbol detection for a variety of channel and system models (e.g., detection of a constellation symbol in additive white Gaussian noise (AWGN)). Thus, as long as such models sufficiently capture real effects we do not expect DL to yield significant improvements on the physical layer.

Nevertheless, we believe that the DL applications which we explore in this paper are a useful and insightful way of fundamentally rethinking the communications system design problem, and hold promise for performance improvements in complex communications scenarios that are difficult to describe with tractable mathematical models. Our main contributions are as follows:

- We demonstrate that it is possible to learn full transmitter and receiver implementations for a given channel model which are optimized for a chosen loss function (e.g., minimizing block error rate (BLER)). Interestingly, such "learned" systems can be competitive with respect to the current state-of-the-art. The key idea here is to represent transmitter, channel, and receiver as one deep neural network (NN) that can be trained as an autoencoder. The beauty of this approach is that it can even be applied to channel models and loss functions for which the optimal solutions are unknown.
- We extend this concept to an adversarial network of multiple transmitter-receiver pairs competing for capacity. This leads to the interference channel for which finding the best signaling scheme is a long-standing research problem. We demonstrate that such a setup can also be represented as an NN with multiple inputs and outputs, and that all transmitter and receiver implementations can be jointly optimized with respect to a common or individual performance metric(s).
- We introduce radio transformer networks (RTNs) as a way to integrate expert knowledge into the DL model. RTNs allow, for example, to carry out predefined correction algorithms ("transformers") at the receiver (e.g., multiplication by a complex-valued number, convolution with a vector) which may be fed with parameters learned by another NN. This NN can be integrated into the end-to-end training process of a task performed on the transformed signal (e.g., symbol detection).
- We study the use of NNs on complex-valued IQ samples for the problem of modulation classification and show that convolutional neural networks (CNNs), which are the cornerstone of most DL systems for computer vision, can outperform traditional classification techniques based on expert features. This result mirrors a relentless trend in DL for various domains, where learned features ultimately outperform and displace long-used expert features, such as the scale-invariant feature transform (SIFT) [8] and Bag-of-words [9].

The ideas presented in this paper provide a multitude of interesting avenues for future research that will be discussed in detail. We hope that these will stimulate wide interest within the research community.

T. O'Shea is with the Bradley Department of Electrical and Computer Engineering, Virginia Tech and DeepSig, Arlington, VA, US (oshea@vt.edu).

J. Hoydis is with Nokia Bell Labs, Route de Villejust, 91620 Nozay, France (jakob.hoydis@nokia-bell-labs.com).

The rest of this article is structured as follows: Section I-A discusses potential benefits of DL for the physical layer. Section I-B presents related work. Background of deep learning is presented in Section II. In Section III, several DL applications for communications are presented. Section IV contains an overview and discussion of open problems and key areas of future investigation. Section V concludes the article.

A. Potential of DL for the physical layer

Apart from the intellectual beauty of a fully "learned" communications system, there are some reasons why DL could provide gains over existing physical layer algorithms.

First, most signal processing algorithms in communications have solid foundations in statistics and information theory and are often provably optimal for tractable mathematically models. These are generally linear, stationary, and have Gaussian statistics. A practical system, however, has many imperfections and non-linearities [4] (e.g., non-linear power amplifiers (PAs), finite resolution quantization) that can only be approximately captured by such models. For this reason, a DL-based communications system (or processing block) that does not require a mathematically tractable model and that can be optimized for a specific hardware configuration and channel might be able to better optimize for such imperfections.

Second, one of the guiding principles of communications systems design is to split the signal processing into a chain of multiple independent blocks; each executing a well defined and isolated function (e.g., source/channel coding, modulation, channel estimation, equalization). Although this approach has led to the efficient, versatile, and controllable systems we have today, it is not clear that individually optimized processing blocks achieve the best possible end-to-end performance. For example, the separation of source and channel coding for many practical channels and short block lengths (see [10] and references therein) as well as separate coding and modulation [11] are known to be sub-optimal. Attempts to jointly optimize each of these components, e.g., based on factor graphs [12], provide gains but lead to unwieldy and computationally complex systems. A learned end-to-end communications system on the other hand is unlikely to have such a rigid modular structure as it is optimized for end-to-end performance.

Third, it has been shown that NNs are universal function approximators [13] and recent work has shown a remarkable capacity for algorithmic learning with recurrent NNs [14] that are known to be Turing-complete [15]. Since the execution of NNs can be highly parallelized on concurrent architectures and easily implemented with low-precision data types [16], there is evidence that "learned" algorithms taking this form could be executed faster and at lower energy cost than their manually "programmed" counterparts.

Fourth, massively parallel processing architectures with distributed memory architectures, such as graphic processing units (GPUs) but also increasingly specialized chips for NN inference (e.g., [17]), have shown to be very energy efficient and capable of impressive computational throughput when fully utilized by concurrent algorithms [18]. The performance of such architectures, however, has been largely limited by the ability of algorithms and higher level programming languages to make efficient use of them. The inherently concurrent nature of computation and memory access across wide and deep NNs has demonstrated a surprising ability to readily achieve high resource utilization on these architectures with minimal application specific tuning or optimization required.

B. Historical context and related work

Applications of ML in communications have a long history covering a wide range of applications. These comprise channel modeling and prediction, localization, equalization, decoding, quantization, compression, demodulation, modulation recognition, and spectrum sensing to name a few [19], [20] (and references therein). However, to the best of our knowledge and due to the reasons mentioned above, few of these applications have been commonly adopted or led to a wide commercial success. It is also interesting that essentially all of these applications focus on individual receiver processing tasks alone, while the consideration of the transmitter or a full end-to-end system is entirely missing in the literature.

The advent of open-source DL libraries (see Section II-B) and readily available specialized hardware along with the astonishing progress of DL in computer vision have stimulated renewed interest in the application of DL for communications and networking [21]. There are currently essentially two different main approaches of applying DL to the physical layer. The goal is to either improve/augment parts of existing algorithms with DL, or to completely replace them.

Among the papers falling into the first category are [22], [23] and [24] that consider improved belief propagation channel decoding and multiple-input multiple-output (MIMO) detection, respectively. These works are inspired by the idea of *deep unfolding* [25] of existing iterative algorithms by essentially interpreting each iteration as a set of NN layers. In a similar manner, [26] aims at improving the solution of sparse linear inverse problems with DL.

In the second category, papers include [27], dealing with blind detection for MIMO systems with low-resolution quantization, and [28], in which detection for molecular communications for which no mathematical channel model exists is studied. The idea of learning to solve complex optimization tasks for wireless resource allocation, such as power control, is investigated in [29]. Some of us have also demonstrated initial results in the area of learned end-to-end communications systems [30] as well as considered the problems of modulation recognition [31], signal compression [32], and channel decoding [33], [34] with state-of-the art DL tools.

Notations: We use boldface upper- and lower-case letters to denote matrices and column vectors, respectively. For a vector $\mathbf{x}$, x_i denotes its ith element, $\|\mathbf{x}\|$ its Euclidean norm, $\mathbf{x}^{\mathsf{T}}$ its transpose, and $\mathbf{x} \odot \mathbf{y}$ the element-wise product with $\mathbf{y}$. For a matrix $\mathbf{X}$, X_{ij} or $[\mathbf{X}]_{ij}$ denotes the (i, j)-element. $\mathbb{R}$ and $\mathbb{C}$ denote the sets of real and complex numbers, respectively. $\mathcal{N}(\mathbf{m}, \mathbf{R})$ and $\mathcal{CN}(\mathbf{m}, \mathbf{R})$ are the multivariate Gaussian and complex Gaussian distributions with mean vector $\mathbf{m}$ and covariance matrix $\mathbf{R}$, respectively. $\mathrm{Bern}(\alpha)$ is the Bernoulli distribution with success probability α and ∇ is the gradient operator.

II. Deep learning basics

A feedforward NN (or multilayer perceptron (MLP)) with L layers describes a mapping $f(\mathbf{r}_0;\boldsymbol{\theta}):\mathbb{R}^{N_0}\mapsto\mathbb{R}^{N_L}$ of an input vector $\mathbf{r}_0\in\mathbb{R}^{N_0}$ to an output vector $\mathbf{r}_L\in\mathbb{R}^{N_L}$ through L iterative processing steps:

$$\mathbf{r}_\ell = f_\ell(\mathbf{r}_{\ell-1};\theta_\ell), \qquad \ell=1,\dots,L \tag{1}$$

where $f_\ell(\mathbf{r}_{\ell-1};\theta_\ell):\mathbb{R}^{N_{\ell-1}}\mapsto\mathbb{R}^{N_\ell}$ is the mapping carried out by the ℓth layer. This mapping depends not only on the output vector $\mathbf{r}_{\ell-1}$ from the previous layer but also on a set of parameters θ_ℓ. Moreover, the mapping can be stochastic, i.e., f_ℓ can be a function of some random variables. We use $\boldsymbol{\theta}=\{\theta_1,\dots,\theta_L\}$ to denote the set of all parameters of the network. The ℓth layer is called *dense* or *fully-connected* if $f_\ell(\mathbf{r}_{\ell-1};\theta_\ell)$ has the form

$$f_\ell(\mathbf{r}_{\ell-1};\theta_\ell)=\sigma\left(\mathbf{W}_\ell\mathbf{r}_{\ell-1}+\mathbf{b}_\ell\right) \tag{2}$$

where $\mathbf{W}_\ell\in\mathbb{R}^{N_\ell\times N_{\ell-1}}$, $\mathbf{b}_\ell\in\mathbb{R}^{N_\ell}$, and $\sigma(\cdot)$ is an *activation* function which we will be defined shortly. The set of parameters for this layer is $\theta_\ell=\{\mathbf{W}_\ell,\mathbf{b}_\ell\}$. Table I lists several other layer types together with their mapping functions and parameters which are used in this manuscript. All layers with stochastic mappings generate a new random mapping each time they are called. For example, the noise layer simply adds a Gaussian noise vector with zero mean and covariance matrix $\beta\mathbf{I}_{N_{\ell-1}}$ to the input. Thus, it generates a different output for the same input each time it is called. The activation function $\sigma(\cdot)$ in (2) introduces a non-linearity which is important for the so-called *expressive power* of the NN. Without this non-linearity there would be not much of an advantage of stacking multiple layers on top of each other. Generally, the activation function is applied individually to each element of its input vector, i.e., $[\sigma(\mathbf{u})]_i=\sigma(u_i)$. Some commonly used activation functions are listed in Table II.[1] NNs are generally trained using labeled training data, i.e., a set of input-output vector pairs $(\mathbf{r}_{0,i},\mathbf{r}^\star_{L,i})$, $i=1,\dots,S$, where $\mathbf{r}^\star_{L,i}$ is the desired output of the neural network when $\mathbf{r}_{0,i}$ is used as input. The goal of the training process is to minimize the loss

$$L(\boldsymbol{\theta})=\frac{1}{S}\sum_{i=1}^{S} l(\mathbf{r}^\star_{L,i},\mathbf{r}_{L,i}) \tag{3}$$

with respect to the parameters in $\boldsymbol{\theta}$, where $l(\mathbf{u},\mathbf{v}):\mathbb{R}^{N_L}\times\mathbb{R}^{N_L}\mapsto\mathbb{R}$ is the loss function and $\mathbf{r}_{L,i}$ is the output of the NN when $\mathbf{r}_{0,i}$ is used as input. Several relevant loss functions are provided in Table III. Different norms (e.g., $L1$, $L2$) of parameters or activations can be added to the loss function to favor solutions with small or sparse values (a form of regularization). The most popular algorithm to find good sets of parameters $\boldsymbol{\theta}$ is stochastic gradient descent (SGD) which starts with some random initial values of $\boldsymbol{\theta}=\boldsymbol{\theta}_0$ and then updates $\boldsymbol{\theta}$ iteratively as

$$\boldsymbol{\theta}_{t+1}=\boldsymbol{\theta}_t-\eta\nabla\tilde{L}(\boldsymbol{\theta}_t) \tag{4}$$

Table I: List of layer types

Name	$f_\ell(\mathbf{r}_{\ell-1};\theta_\ell)$	θ_ℓ
Dense	$\sigma\left(\mathbf{W}_\ell\mathbf{r}_{\ell-1}+\mathbf{b}_\ell\right)$	$\mathbf{W}_\ell,\mathbf{b}_\ell$
Noise	$\mathbf{r}_{\ell-1}+\mathbf{n}$, $\mathbf{n}\sim\mathcal{N}(\mathbf{0},\beta\mathbf{I}_{N_{\ell-1}})$	none
Dropout [36]	$\mathbf{d}\odot\mathbf{r}_{\ell-1}$, $d_i\sim\text{Bern}(\alpha)$	none
Normalization	e.g., $\frac{\sqrt{N_{\ell-1}}\mathbf{r}_{\ell-1}}{\|\mathbf{r}_{\ell-1}\|_2}$	none

Table II: List of activation functions

Name	$[\sigma(\mathbf{u})]_\mathbf{i}$	Range
linear	u_i	$(-\infty,\infty)$
ReLU [37]	$\max(0,u_i)$	$[0,\infty)$
tanh	$\tanh(u_i)$	$(-1,1)$
sigmoid	$\frac{1}{1+e^{-u_i}}$	$(0,1)$
softmax	$\frac{e^{u_i}}{\sum_j e^{u_j}}$	$(0,1)$

Table III: List of loss functions

Name	$l(\mathbf{u},\mathbf{v})$
MSE	$\|\mathbf{u}-\mathbf{v}\|_2^2$
Categorical cross-entropy	$-\sum_j u_j\log(v_j)$

where $\eta>0$ is the learning rate and $\tilde{L}(\boldsymbol{\theta})$ is an approximation of the loss function which is computed for a random *mini-batch* of training examples $\mathcal{S}_t\subset\{1,2,\dots,S\}$ of size S_t at each iteration, i.e.,

$$\tilde{L}(\boldsymbol{\theta})=\frac{1}{S_t}\sum_{i\in\mathcal{S}_t} l(\mathbf{r}^\star_{L,i},\mathbf{r}_{L,i}). \tag{5}$$

By choosing S_t small compared to S, the gradient computation complexity is significantly reduced while still reducing weight update variance. Note that there are many variants of the SGD algorithm which dynamically adapt the learning rate to improve convergence [35, Ch. 8.5]. The gradient in (4) can be very efficiently computed through the backpropagation algorithm [35, Ch. 6.5]. Definition and training of NNs of almost arbitrary shape can be easily done with one of the many existing DL libraries presented in Section II-B.

A. Convolutional layers

Convolutional neural network (CNN) layers were introduced in [6] to provide an efficient learning method for 2D images. By tying adjacent shifts of the same weights together in a way similar to that of a filter sliding across an input vector, convolutional layers are able to force the learning of features with an invariance to shifts in the input vector. In doing so, they also greatly reduce the model complexity (as measured by the number of free parameters in the layer's weight matrix) required to represent equivalent shift-invariant features using fully connected layers, reducing SGD optimization complexity and improving generalization on appropriate datasets.

In general, a convolutional layer consists of a set of F filter weights $\mathbf{Q}^f\in\mathbb{R}^{a\times b}$, $f=1,\dots,F$ (F is called the *depth*), which generate each a so-called *feature map* $\mathbf{Y}^f\in\mathbb{R}^{n'\times m'}$ from an input matrix $\mathbf{X}\in\mathbb{R}^{n\times m}$ [2] according to the following

[1]The linear activation function is typically used at the output layer in the context of regression tasks, i.e., estimation of a real-valued vector.

[2]In image processing, $\mathbf{X}$ is commonly a three-dimensional tensor with the third dimension corresponding to color channels. The filters weights are also three-dimensional and work on each input channels simultaneously.

convolution:

$$Y_{i,j}^{f} = \sum_{k=0}^{a-1} \sum_{\ell=0}^{b-1} Q_{a-k,b-\ell}^{f} X_{1+s(i-1)-k,1+s(j-1)-\ell} \tag{6}$$

where $s \geq 1$ is an integer parameter called *stride*, $n' = 1 + \lfloor \frac{n+a-2}{s} \rfloor$ and $m' = 1 + \lfloor \frac{m+b-2}{s} \rfloor$, and it is assumed that $\mathbf{X}$ is padded with zeros, i.e., $X_{i,j} = 0$ for all $i \notin [1, n]$ and $j \notin [1, m]$. The output dimensions can be reduced by either increasing the stride s or by adding a *pooling* layer. The pooling layer partitions $\mathbf{Y}$ into $p \times p$ regions for each of which it computes a single output value, e.g., maximum or average value, or $L2$-norm.

For example, taking a vectorized grayscale image input consisting of 28×28 pixels and connecting it to a dense layer with the same number of activations, results in a single weight matrix with $784 \times 784 = 614,656$ free parameters. On the other hand, if we use a convolutional feature map containing six filters each sized 5×5 pixels, we obtain a much reduced number of free parameters of $6 \cdot 5 \cdot 5 = 150$. For the right kind of dataset, this technique can be extremely effective. We will see an application of convolutional layers in Section III-D. For more details on CNNs, we refer to [35, Ch. 9].

B. Machine learning libraries

In recent times, numerous tools and algorithms have emerged that make it easy to build and train large NNs. Tools to deploy such training routines from high level language to massively parallel GPU architectures have been key enablers. Among these are Caffe [38], MXNet [39], TensorFlow [40], Theano [41], and Torch [42] (just to name a few), which allow for high level algorithm definition in various programming languages or configuration files, automatic differentiation of training loss functions through arbitrarily large networks, and compilation of the network's forwards and backwards passes into hardware optimized concurrent dense matrix algebra kernels. Keras [43] provides an additional layer of NN primitives with Theano and TensorFlow as its back-end. It has a highly customizable interface to quickly experiment with and deploy deep NNs, and has become our primary tool used to generate the numerical results for this manuscript [44].

C. Network dimensions and training

The term "deep" has become common in recent NN literature, referring to the number of sequential layers within a network (but also more generally to the methods commonly used to train such networks). Depth relates directly to the number of iterative operations performed on input data through sequential layers' transfer functions. While deep networks allow for numerous iterative transforms on the data, a minimum latency network would likely be as shallow as possible. "Width" is used to describe the number of output activations per layer, or for all layers on average, and relates directly to the memory required by each layer.

Best practice training methods have varied over the years, from direct solution techniques over gradient descent to genetic algorithms, each having been favored or considered at one time (see [35, Ch. 1.2] for a short history of DL). Layer-by-layer pre-training [45] was also a recently popular method for scaling training to larger networks where backpropagation once struggled. However, most systems today are able to train networks which are both wide and deep directly using backpropagation and SGD methods with adaptive learning rates (e.g., Adam [46]), regularization methods to prevent overfitting (e.g., Dropout [36]), and activations functions which reduce gradient issues (e.g., ReLU [37]).

III. Examples of machine learning applications for the physical layer

In this section, we will show how to represent an end-to-end communications system as an autoencoder and train it via SGD. This idea is then extended to multiple transmitters and receivers and we study as an example the two-user interference channel. We will then introduce the concept of RTNs to improve performance on fading channels, and demonstrate the application of CNNs to raw radio frequency time-series data for the task of modulation classification.

A. Autoencoders for end-to-end communications systems

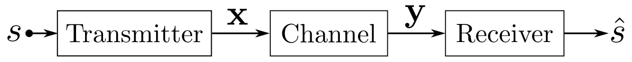


Figure 1: A simple communications system consisting of a transmitter and a receiver connected through a channel

In its simplest form, a communications system consists of a transmitter, a channel, and a receiver, as shown in Fig. 1. The transmitter wants to communicate one out of M possible messages $s \in \mathcal{M} = \{1, 2, ..., M\}$ to the receiver making n discrete uses of the channel. To this end, it applies the transformation $f : \mathcal{M} \mapsto \mathbb{R}^n$ to the message s to generate the transmitted signal $\mathbf{x} = f(s) \in \mathbb{R}^n$.[3] Generally, the hardware of the transmitter imposes certain constraints on $\mathbf{x}$, e.g., an energy constraint $\|\mathbf{x}\|_2^2 \leq n$, an amplitude constraint $|x_i| \leq 1 \, \forall i$, or an average power constraint $\mathbb{E}\left[|x_i|^2\right] \leq 1 \, \forall i$. The communication rate of this communications system is $R = k/n$ [bit/channel use], where $k = \log_2(M)$. In the sequel, the notation (n,k) means that a communications system sends one out of $M = 2^k$ messages (i.e., k bits) through n channel uses. The channel is described by the conditional probability density function $p(\mathbf{y}|\mathbf{x})$, where $\mathbf{y} \in \mathbb{R}^n$ denotes the received signal. Upon reception of $\mathbf{y}$, the receiver applies the transformation $g : \mathbb{R}^n \mapsto \mathcal{M}$ to produce the estimate $\hat{s}$ of the transmitted message s.

From a DL point of view, this simple communications system can be seen as a particular type of *autoencoder* [35, Ch. 14]. Typically, the goal of an autoencoder is to find a low-dimensional representation of its input at some intermediate layer which allows reconstruction at the output with minimal error. In this way, the autoencoder learns to

[3]We focus here on real-valued signals only. Extensions to complex-valued signals are discussed in Section IV. Alternatively, one can consider a mapping to $\mathbb{R}^{2n}$, which can be interpreted as a concatenation of the real and imaginary parts of $\mathbf{x}$. This approach is adopted in Sections III-B, III-C, and III-D.

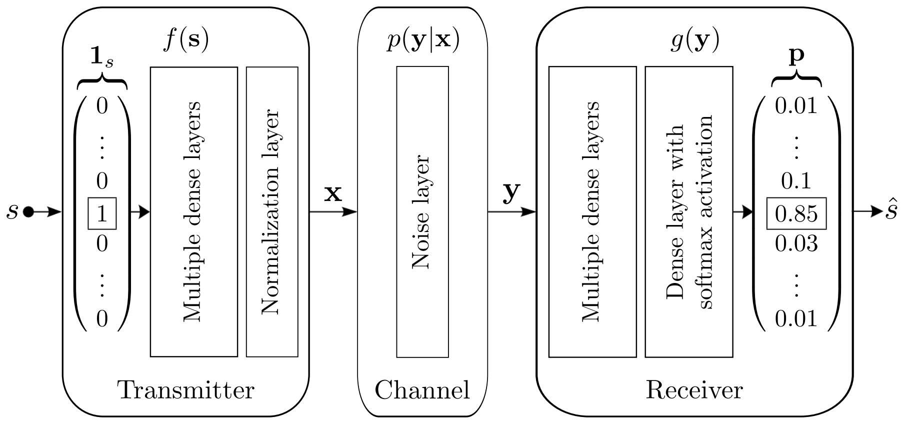


Figure 2: A communications system over an AWGN channel represented as an autoencoder. The input s is encoded as a one-hot vector, the output is a probability distribution over all possible messages from which the most likely is picked as output $\hat{s}$.

non-linearly compress and reconstruct the input. In our case, the purpose of the autoencoder is different. It seeks to learn representations $\mathbf{x}$ of the messages s that are robust with respect to the channel impairments mapping $\mathbf{x}$ to $\mathbf{y}$ (i.e., noise, fading, distortion, etc.), so that the transmitted message can be recovered with small probability of error. In other words, while most autoencoders remove redundancy from input data for compression, this autoencoder (the "channel autoencoder") often adds redundancy, learning an intermediate representation robust to channel perturbations.

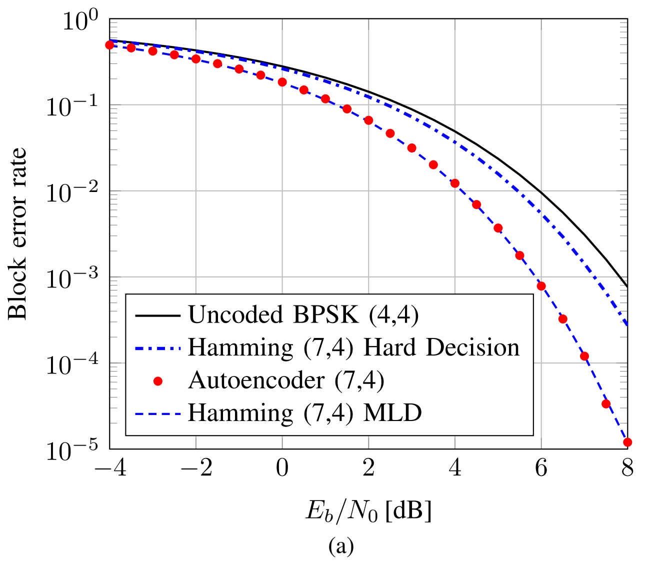
An example of such an autoencoder is shown in Fig. 2. Here, the transmitter consists of a feedforward NN with multiple dense layers followed by a normalization layer that ensures that physical constraints on $\mathbf{x}$ are met. Note that the input s to the transmitter is encoded as a *one-hot vector* $\mathbf{1}_s \in \mathbb{R}^M$, i.e., an M-dimensional vector, the sth element of which is equal to one and zero otherwise. The channel is represented by an additive noise layer with a fixed variance $\beta = (2RE_b/N_0)^{-1}$, where E_b/N_0 denotes the energy per bit (E_b) to noise power spectral density (N_0) ratio. The receiver is also implemented as a feedforward NN. Its last layer uses a softmax activation whose output $\mathbf{p} \in (0,1)^M$ is a probability vector over all possible messages. The decoded message $\hat{s}$ corresponds then to the index of the element of $\mathbf{p}$ with the highest probability. The autoencoder can then be trained end-to-end using SGD on the set of all possible messages $s \in \mathcal{M}$ using the well suited categorical cross-entropy loss function between $\mathbf{1}_s$ and $\mathbf{p}$.[4]

Fig. 3a compares the block error rate (BLER), i.e., $\Pr(\hat{s} \neq s)$, of a communications system employing binary phase-shift keying (BPSK) modulation and a Hamming (7,4) code with either binary hard-decision decoding or maximum likelihood decoding (MLD) against the BLER achieved by the trained autoencoder (7,4) (with fixed energy constraint $\|\mathbf{x}\|_2^2 = n$). Both systems operate at rate $R = 4/7$. For comparison, we also provide the BLER of uncoded BPSK (4,4). This result shows that the autoencoder has learned without any prior knowledge an encoder and decoder function that together achieve the same performance as the Hamming (7,4) code with MLD. The layout of the autoencoder is provided in Table IV. Although a single layer can represent the same mapping from message index to corresponding transmit vector, our experiments have shown that SGD converges to a better global solution using two transmit layers instead of one. This increased dimension parameter search space may actually help to reduce likelihood of convergence to sub-optimal minima by making such solutions more likely to emerge as saddle points during optimization [47]. Training was done at a fixed value of $E_b/N_0 = 7\,\text{dB}$ (cf. Section IV-B) using Adam [46] with learning rate 0.001. We have observed that increasing the batch size during training helps to improve accuracy. For all other implementation details, we refer to the source code [44].

Fig. 3b shows a similar comparison but for an (8,8) and (2,2) communications system, i.e., $R = 1$. Surprisingly, while the autoencoder achieves the same BLER as uncoded BPSK for (2,2), it outperforms the latter for (8,8) over the full range of E_b/N_0. This implies that it has learned some joint coding and modulation scheme, such that a coding gain is achieved. For a truly fair comparison, this result should be compared to a higher-order modulation scheme using a channel code (or the optimal sphere packing in eight dimensions). A detailed performance comparison for various channel types and parameters (n,k) with different baselines is out of the scope of this paper and left to future investigations.

Fig. 4 shows the learned representations $\mathbf{x}$ of all messages for different values of (n,k) as complex constellation points, i.e., the x- and y-axes correspond to the first an second transmitted symbols, respectively. In Fig. 4d for $(7,4)$, we depict the seven-dimensional message representations using a two-dimensional t-distributed stochastic neighbor embedding (t-SNE)[48] of the noisy observations $\mathbf{y}$ instead. Fig. 4a shows the simple $(2,2)$ system which converges rapidly to a classical quadrature phase shift keying (QPSK) constellation with some arbitrary rotation. Similarly, Fig. 4b shows a $(4,2)$ system which leads to a rotated 16-PSK constellation. The impact of the chosen normalization becomes clear from Fig. 4c for the same parameters but with an average power normalization

[4]A more memory-efficient approach to implement this architecture is by replacing the one-hot encoded input and the first dense layer by an *embedding* that turns message indices into vectors. The loss function can then be replaced by the *sparse categorical cross-entropy* that accepts message indices rather than one-hot vectors as labels. This was done in our experiments [44].



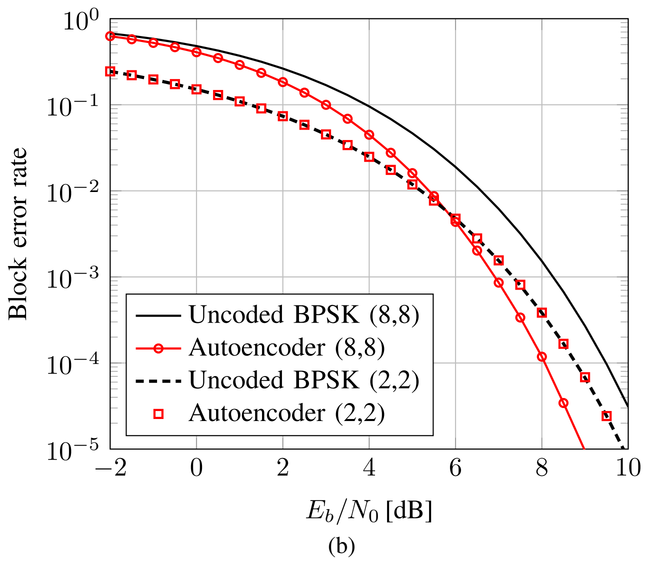


Figure 3: BLER versus E_b/N_0 for the autoencoder and several baseline communication schemes

Table IV: Layout of the autoencoder used in Figs. 3a and 3b. It has $(2M+1)(M+n)+2M$ trainable parameters, resulting in 62, 791, and 135,944 parameters for the (2,2), (7,4), and (8,8) autoencoder, respectively.

Layer	Output dimensions
Input	M
Dense + ReLU	M
Dense + linear	n
Normalization	n
Noise	n
Dense + ReLU	M
Dense + softmax	M

instead of a fixed energy constraint (that forces the symbols to lie on the unit circle). This results in an interesting mixed pentagonal/hexagonal grid arrangement (with indistinguishable BLER performance from 16-QAM). The constellation has a symbol in the origin surrounded by five equally spaced nearest neighbors, each of which has six almost equally spaced neighbors. Fig. 4d for $(7,4)$ shows that the t-SNE embedding of $\mathbf{y}$ leads to a similarly shaped arrangement of clusters.

The examples of this section treat the communication task as a classification problem and the representation of s by an M-dimensional vector becomes quickly impractical for large M. To circumvent this problem, it is possible to use more compact representations of s such as a binary vector with $\log_2(M)$ dimensions. In this case, output activation functions such as sigmoid and loss functions such as MSE or binary cross-entropy are more appropriate. Nevertheless, scaling such an architecture to very large values of M remains challenging due to the size of the training set and model. We recall that a very important property of the autoencoder is also that it can learn to communicate over *any* channel, even for which no information-theoretically optimal scheme is known.

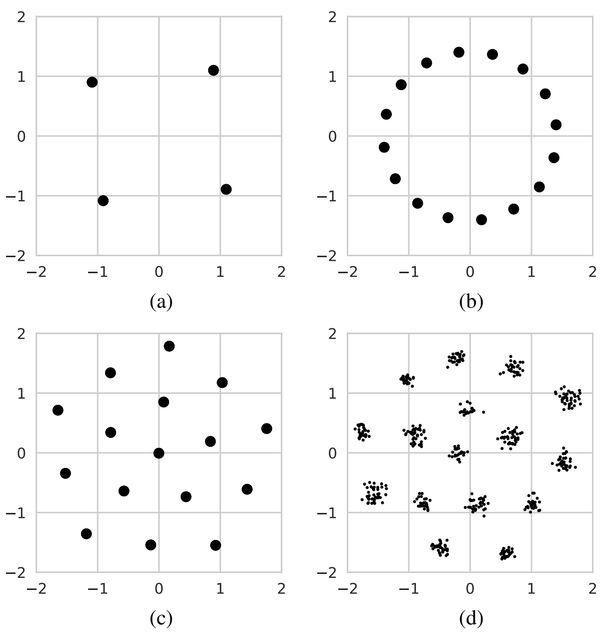


Figure 4: Constellations produced by autoencoders using parameters (n,k): (a) $(2,2)$ (b) $(2,4)$, (c) $(2,4)$ with average power constraint, (d) $(7,4)$ 2-dimensional t-SNE embedding of received symbols.

B. *Autoencoders for multiple transmitters and receivers*

The autoencoder concept from Section III-A, can be readily extended to multiple transmitters and receivers that share a common channel. As an example, we consider here the two-user AWGN interference channel as shown in Fig. 5. Transmitter 1 wants to communicate message $s_1 \in \mathbb{M}$ to Receiver 1 while Transmitter 2 wants to communicate message $s_2 \in \mathbb{M}$ to Receiver 2.[5] Both transmitter-receiver pairs are

[5]Extensions to K users with possibly different rates, i.e., $s_k \in \mathbb{M}_k\,\forall k$, as well as to other channel types are straightforward.

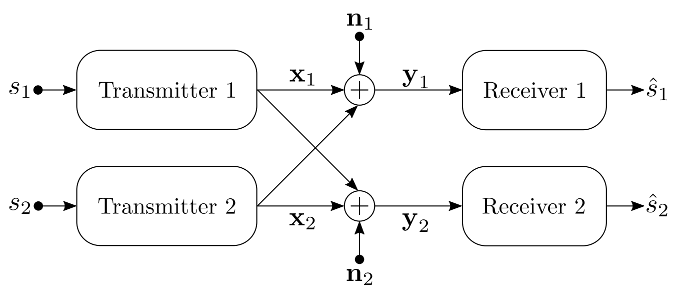


Figure 5: The two-user interference channel seen as a combination of two interfering autoencoders that try to reconstruct their respective messages

implemented as NNs and the only difference with respect to the autoencoder from the last section is that the transmitted messages $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{C}^n$ now interfere at the receivers, resulting in the noisy observations

$$\mathbf{y}_1 = \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{n}_1 \tag{7}$$

$$\mathbf{y}_2 = \mathbf{x}_2 + \mathbf{x}_1 + \mathbf{n}_2 \tag{8}$$

where $\mathbf{n}_1, \mathbf{n}_1 \sim \mathcal{CN}(0, \beta\mathbf{I}_n)$ is Gaussian noise. For simplicity, we have adopted the complex-valued notation, rather than considering real-valued vectors of size $2n$. That is, the notation (n, k) means each of the 2^k messages is transmitted over n complex-valued channel uses. Denote by

$$l_1 = -\log\left([\hat{\mathbf{s}}_1]_{s_1}\right), \quad l_2 = -\log\left([\hat{\mathbf{s}}_2]_{s_2}\right) \tag{9}$$

the individual cross-entropy loss functions of the first and second transmitter-receiver pair, respectively, and by $\tilde{L}_1(\boldsymbol{\theta}_t)$, $\tilde{L}_2(\boldsymbol{\theta}_t)$ the associated losses for mini-batch t (cf. (5)). In such a context, it is less clear how one should train two coupled autoencoders with conflicting goals. One approach consists of minimizing a weighted sum of both losses, i.e., $\tilde{L} = \alpha\tilde{L}_1 + (1-\alpha)\tilde{L}_2$ for some $\alpha \in [0,1]$. If one would minimize $\tilde{L}_1$ alone (i.e., $\alpha = 1$), Transmitter 2 would learn to transmit a constant signal independent of s_2 that Receiver 1 could simply subtract from $\mathbf{y}_1$. The opposite is true for $\alpha = 0$. However, giving equal weight to both losses (i.e., $\alpha = 0.5$) does not necessarily result in equal performance. We have observed in our experiments that it generally leads to highly unfair and suboptimal solutions. For this reason, we have adopted dynamic weights α_t for each mini-batch t:

$$\alpha_{t+1} = \frac{\tilde{L}_1(\boldsymbol{\theta}_t)}{\tilde{L}_1(\boldsymbol{\theta}_t) + \tilde{L}_2(\boldsymbol{\theta}_t)}, \quad t > 0 \tag{10}$$

where $\alpha_0 = 0.5$. Thus, the smaller $\tilde{L}_1(\boldsymbol{\theta}_t)$ is compared to $\tilde{L}_2(\boldsymbol{\theta}_t)$, the smaller is its weight α_{t+1} for the next mini-batch. There are many other possibilities to train such a system and we do not claim any optimality of our approach. However, it has led in our experiments to the desired result of identical BLERs for both transmitter-receiver pairs.

Fig. 6 shows the BLER of one of the autoencoders (denoted by AE) as a function of E_b/N_0 for the sets of parameters $(n,k) = \{(1,1),(2,2),(4,4),(4,8)\}$. The NN-layout for both autoencoders is that provided in Table IV by letting $n = 2n$. We have used an average power constraint to be competitive

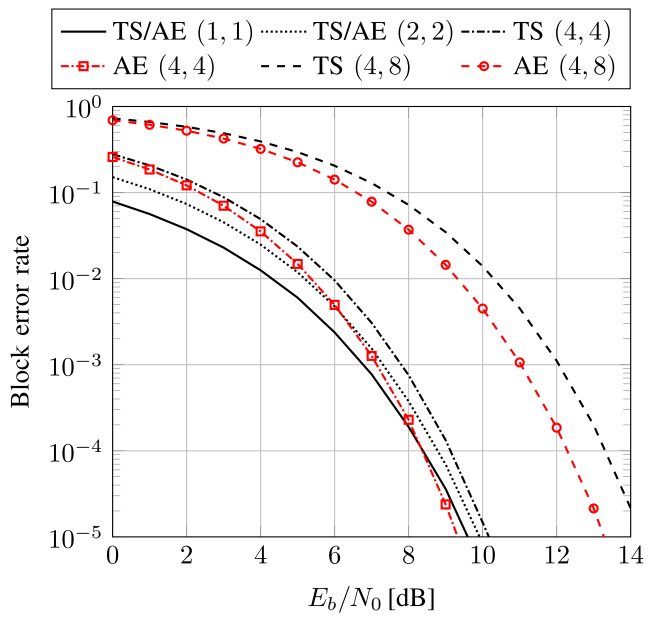


Figure 6: BLER versus E_b/N_0 for the two-user interference channel achieved by the autoencoder (AE) and $2^{2k/n}$-QAM time-sharing (TS) for different parameters (n,k)

with higher-order modulation schemes (cf. Fig. 4c). As a baseline for comparison, we provide the BLER of uncoded $2^{2k/n}$-QAM that has the same rate when used together with time-sharing (TS) between both transmitters.[6] While the autoencoder and time-sharing have identical BLERs for $(1,1)$ and $(2,2)$, the former achieves substantial gains of around 0.7 dB for $(4,4)$ and 1 dB for $(4,8)$ at a BLER of 10^{-3}. The reasons for this are similar to those explained in Section III-A.

It is interesting to have a look at the learned message representations which are shown in Fig. 7. For $(1,1)$, the transmitters have learned to use binary phase shift keying (BPSK)-like constellations in orthogonal directions (with an arbitrary rotation around the origin). This achieves the same performance as QPSK with time-sharing. However, for $(2,2)$, the learned constellations are not orthogonal anymore and can be interpreted as some form of super-position coding. For the first symbol, Transmitter 1 uses high power and Transmitter 2 low power. For the second symbol, the roles are changed. For $(4,4)$ and $(4,8)$, the constellations are more difficult to interpret, but we can see that the constellations of both transmitters resemble ellipses with orthogonal major axes and varying focal distances. This effect is more visible for $(4,8)$ than for $(4,4)$ because of the increased number of constellation points. An in-depth study of learned constellations and how they are impacted by the chosen normalization and NN weight initializations is out of the scope of this paper but a very interesting topic of future investigations.

We would like to point out that one can easily consider other types of multi-transmitter/receiver communications systems with this approach. These comprise, the general multiple access channel (MAC) and broadcast channel (BC), as well as

[6]For $(1,1)$, $(2,2)$, and $(4,4)$, each transmitter sends a 4-QAM (i.e., QPSK) symbol on every other channel use. For $(4,8)$, 16-QAM is used instead.

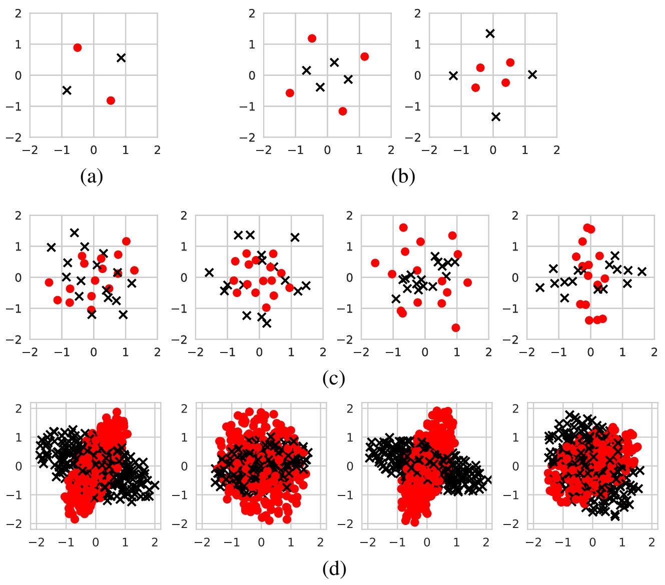


Figure 7: Learned constellations for the two-user interference channel with parameters (a) $(1, 1)$, (b) $(2, 2)$, (c) $(4, 4)$, and (d) $(4, 8)$. The constellation points of Transmitter 1 and 2 are represented by red dots and black crosses, respectively.

systems with jammers and eavesdroppers. As soon as some of the transmitters and receivers are non-cooperative, adversarial training strategies could be adopted (see [49], [50]).

C. Radio transformer networks for augmented signal processing algorithms

Many of the physical phenomena undergone in a communications channel and in transceiver hardware can be inverted using compact parametric models/transformations. Widely used transformations include re-sampling to estimated symbol/clock timing, mixing with an estimated carrier tone, and convolving with an inverse channel impulse response. The estimation processes for parameters to seed these transformations (e.g., frequency offset, symbol timing, impulse response) is often very involved and specialized based on signal specific properties and/or information from pilot tones (see, e.g., [3]).

One way of augmenting DL models with expert propagation domain knowledge but not signal specific assumptions is through the use of an RTN as shown in Fig. 8. An RTN consists of three parts: (i) a learned parameter estimator $g_{\boldsymbol{\omega}} : \mathbb{R}^n \mapsto \mathbb{R}^p$ which computes a parameter vector $\boldsymbol{\omega} \in \mathbb{R}^p$ from its input $\mathbf{y}$, (ii) a parametric transform $t : \mathbb{R}^n \times \mathbb{R}^p \mapsto \mathbb{R}^{n'}$ that applies a deterministic (and differentiable) function to $\mathbf{y}$ which is parametrized by $\boldsymbol{\omega}$ and suited to the propagation phenomena, and (iii) a learned discriminative network $g : \mathbb{R}^{n'} \mapsto \mathcal{M}$ which produces the estimate $\hat{s}$ of the transmitted message (or other label information) from the canonicalized input $\bar{\mathbf{y}} \in \mathbb{R}^{n'}$. By allowing the parameter estimator $g_{\boldsymbol{\omega}}$ to take the form of an NN, we can train the system end-to-end to optimize for a given loss function. Importantly, the training process of such an RTN does not seek to directly improve the parameter estimation itself but rather optimizes the way the parameters are estimated to obtain the best end-to-end performance (e.g., BLER). While the example above describes an RTN for receiver-side processing, it can similarly be used wherever parametric transformations seeded by estimated parameters are needed. RTNs are a form of learned feed-forward attention inspired by Spatial Transformer Networks (STNs) [51] which have worked well for computer vision problems.

The basic functioning of an RTN is best understood from a simple example, such as the problem of phase offset estimation and compensation. Let $\mathbf{y}_c = e^{j\varphi}\tilde{\mathbf{y}}_c \in \mathbb{C}^n$ be a vector of IQ samples that have undergone a phase rotation by the phase offset φ, and let $\mathbf{y} = [\Re\{\mathbf{y}\}^{\mathsf{T}}, \Im\{\mathbf{y}\}^{\mathsf{T}}]^{\mathsf{T}} \in \mathbb{R}^{2n}$. The goal of $g_{\boldsymbol{\omega}}$ is to estimate a scalar $\hat{\varphi} = \boldsymbol{\omega} = g_{\boldsymbol{\omega}}(\mathbf{y})$ that is close to the phase offset φ, which is then used by the parametric transform t to compute $\bar{\mathbf{y}}_c = e^{-j\hat{\varphi}}\mathbf{y}_c$. The canonicalized signal $\bar{\mathbf{y}} = [\Re\{\bar{\mathbf{y}}_c\}^{\mathsf{T}}, \Im\{\bar{\mathbf{y}}_c\}^{\mathsf{T}}]^{\mathsf{T}}$ is thus given by

$$\bar{\mathbf{y}} = t(\hat{\varphi}, \mathbf{y}) = \begin{bmatrix} \cos(\hat{\varphi})\Re\{\bar{\mathbf{y}}_c\} + \sin(\hat{\varphi})\Im\{\bar{\mathbf{y}}_c\} \\ \cos(\hat{\varphi})\Im\{\bar{\mathbf{y}}_c\} - \sin(\hat{\varphi})\Re\{\bar{\mathbf{y}}_c\} \end{bmatrix} \tag{11}$$

and then fed into the discriminative network g for further processing, such as classification.

A compelling example demonstrating the advantages of RTNs is shown in Fig. 9 which compares the BLER of an autoencoder (8,4)[7] with and without RTN over a multipath fading channel with $L = 3$ channel taps. That is, the received signal $\mathbf{y} = [\Re\{\mathbf{y}_c\}^{\mathsf{T}}, \Im\{\mathbf{y}_c\}^{\mathsf{T}}]^{\mathsf{T}} \in \mathbb{R}^{2n}$ is given as

$$y_{c,i} = \sum_{\ell=1}^{L} h_{c,\ell}\, x_{c,i-\ell+1} + n_{c,i} \tag{12}$$

where $\mathbf{h}_c \sim \mathcal{CN}(0, L^{-1}\mathbf{I}_L)$ are i.i.d. Rayleigh fading channel taps, $\mathbf{n}_c \sim \mathcal{CN}(0, (RE_b/N_0)^{-1}\mathbf{I}_n)$ is receiver noise, and $\mathbf{x}_c \in \mathbb{C}^n$ is the transmitted signal, where we assume in (12) $x_{c,i} = 0$ for $i \leq 0$. Here, the goal of the parameter estimator is to predict a complex-valued vector $\boldsymbol{\omega}_c$ (represented by $2L$ real values) that is used in the transformation layer to compute the complex convolution of $\mathbf{y}_c$ with $\boldsymbol{\omega}_c$. Thus, the RTN tries to equalize the channel output through inverse filtering in order to simplify the task of the discriminative network. We have implemented the estimator as an NN with two dense layers with $\tanh$ activations followed by a dense output layer with linear activations.

While the plain autoencoder struggles to meet the performance of differential BPSK (DBPSK) with maximum likelihood sequence estimation (MLE) and a Hamming (7,4) code, the autoencoder with RTN outperforms it. Another advantage of RTNs is faster training convergence which can be seen from Fig. 10 that compares the validation loss of the autoencoder with and without RTN as a function of the training epochs. We have observed in our experiments that the autoencoder with RTN consistently outperforms the plain autoencoder, independently of the chosen hyper-parameters. However, the performance differences diminish when the encoder and decoder networks are made wider and trained for more iterations. Although there is theoretically nothing an RTN-augmented NN can do that a plain NN cannot, the RTN helps by incorporating domain knowledge to simplify the target manifold, similar

[7]We assume complex-valued channel uses, so that transmitter and receiver have $2n$ real-valued inputs and outputs.

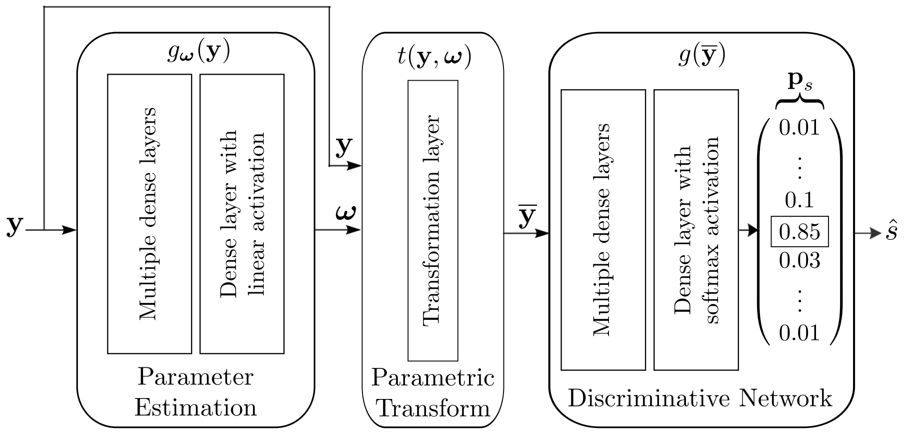


Figure 8: A radio receiver represented as an RTN. The input $\mathbf{y}$ first runs through a parameter estimation network $g_\omega(\mathbf{y})$, has a known transform $t(\mathbf{y}, \boldsymbol{\omega})$ applied to generate the canonicalized signal $\overline{\mathbf{y}}$, and then is classified in the discriminative network $g(\overline{\mathbf{y}})$ to produce the output $\hat{s}$.

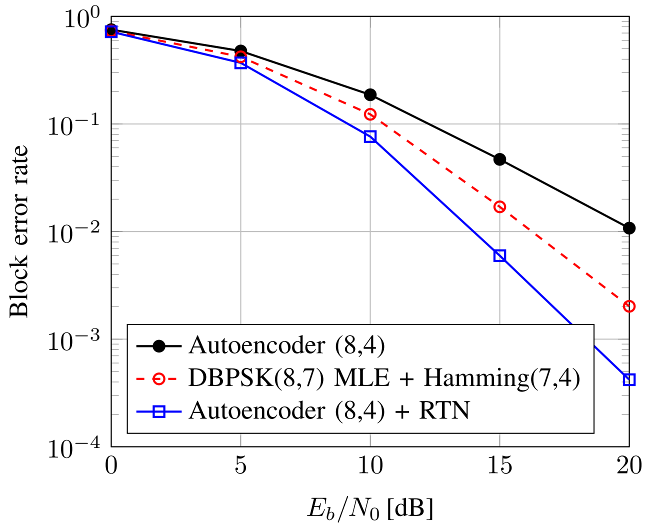


Figure 9: BLER versus E_b/N_0 for various communication schemes over a channel with $L = 3$ Rayleigh fading taps

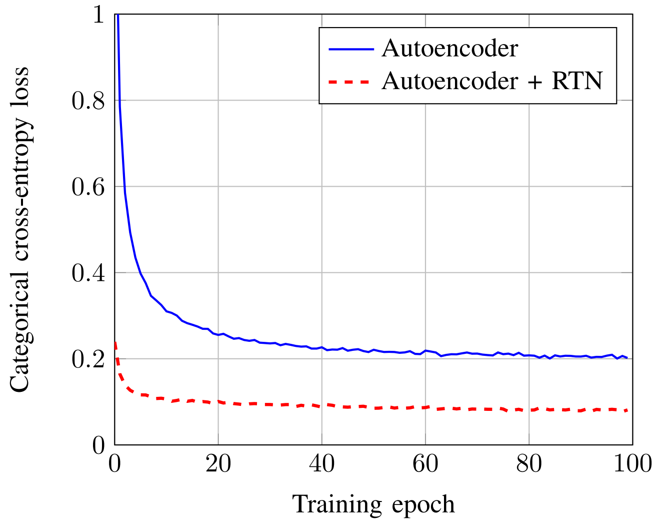


Figure 10: Autoencoder training loss with and without RTN

to the role of convolutional layers in imparting translation invariance where appropriate. This leads to a simpler search space and improved generalization.

The autoencoder and RTN as presented above can be easily extended to operate directly on IQ samples rather than symbols to effectively deal with problems such as pulse shaping, timing-, frequency- and phase-offset compensation. This is an exciting and promising area of research that we leave to future investigations. Interesting applications of this approach could also arise in optical communications dealing with highly non-linear channel impairments that are notoriously difficult to model and compensate for [52].

D. *CNNs for classification tasks*

Many signal processing functions within the physical layer can be learned as either regression or classification tasks. Here we look at the well-known problem of modulation classification of single carrier modulation schemes based on sampled radio frequency time-series data, i.e., IQ samples. This task has been accomplished for years through the approach of expert feature engineering and either analytic decision trees (single trees are widely used in practice) or trained discrimination methods operating on a compact feature space, such as support vector machines, random forests, or small feedforward NNs [53]. Some recent methods take a step beyond this using pattern recognition on expert feature maps, such as the spectral coherence function or α-profile, combined with NN-based classification [54]. However, approaches to this point have not sought to use feature learning on raw time-series data in the radio domain. This is however now the norm in computer vision which motivates our approach here.

As is widely done for image classification, we leverage a series of narrowing convolutional layers followed by dense/fully-connected layers and terminated with a dense softmax layer for our classifier (similar to a VGG architecture [55]). The layout is provided in Table V and we refer to the source code

Table V: Layout of the CNN for modulation classification with 324,330 trainable parameters

Layer	Output dimensions
Input	2×128
Convolution (128 filters, size 2×8) + ReLU	128×121
Max Pooling (size 2, strides 2)	128×60
Convolution (64 filters, size 1×16) + ReLU	64×45
Max Pooling (size 2, strides 2)	64×22
Flatten	1408
Dense + ReLU	128
Dense + ReLU	64
Dense + ReLU	32
Dense + softmax	10

[44] for further implementation details. The dataset[8] for this benchmark consists of 1.2M sequences of 128 complex-valued basedband IQ samples corresponding to ten different digital and analog single-carrier modulation schemes (AM, FM, PSK, QAM, etc.) that have gone through a wireless channel with harsh realistic effects including multipath fading, sample rate and center frequency offset [31]. The samples are taken at 20 different signal-to-noise ratios (SNRs) within the range from -20 dB to 18 dB.

In Fig. 11, we compare the classification accuracy of the CNN against that of extreme gradient boosting[9] with 1000 estimators, as well as a single scikit-learn tree [56], operating on a mix of 16 analog and cumulant expert features as proposed in [53] and [57]. The short-time nature of the examples places this task on difficult end of the modulation classification spectrum since we cannot compute expert features with high stability over long periods of time. We can see that the CNN outperforms the boosted feature-based classifier by around 4 dB in the low to medium SNR range while the performance at high SNR is almost identical. Performance in the single tree case is about 6 dB worse than the CNN at medium SNR and 3.5 % worse at high SNR. Fig. 12 shows the confusion matrix for the CNN at SNR = 10 dB revealing confusing cases for the CNN are between QAM16 and QAM64 and between analog modulations Wideband FM (WBFM) and double-sideband AM (AM-DSB), even at high SNR. The confusion between AM and FM arises during times when the underlying voice signal is idle or does not cary much information. The distinction between QAM16 and QAM64 is very hard with a short-time observation over only a few symbols which share constellation points. The accuracy of the feature-based classifier saturates at high SNR for the same reasons. In [58], the authors report on a successful application of a similar CNN for the detection of black hole mergers in astrophysics from noisy time-series data.

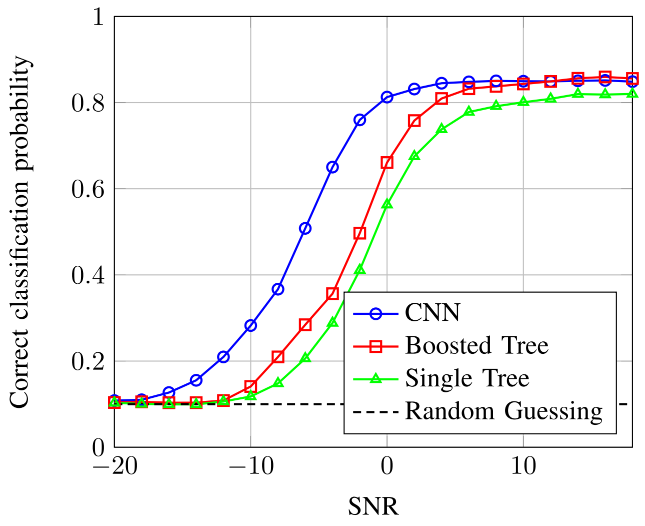


Figure 11: Classifier performance comparison versus SNR

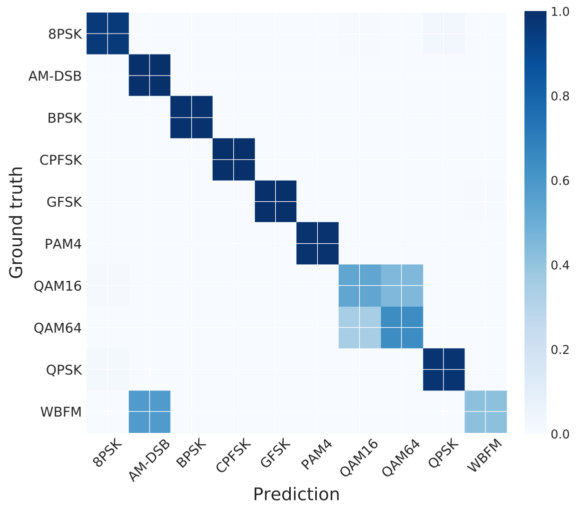


Figure 12: Confusion matrix of the CNN (SNR = 10 dB)

IV. Discussion and Open Research Challenges

A. Data sets and challenges

In order to compare the performance of ML models and algorithms, it is crucial to have common benchmarks and open datasets. While this is the rule in the computer vision, voice recognition, and natural language processing domains (e.g., MNIST[10] or ImageNet[11]), nothing comparable exists for communications. This domain is somewhat different because it deals with inherently man-made signals that can be accurately generated synthetically, allowing the possibility of standardizing data generation routines rather than just data in some cases. It would be also desirable to establish a set of common problems and the corresponding datasets (or data-generating software) on which researchers can benchmark and compare their algorithms. One such example task is modulation classification in Section III-D; others could include mapping of impaired received IQ samples or symbols to codewords or bits. Even "autoencoder competitions" could be held for a standardized set of benchmark impairments, taking the form of canonical "impairment layers" that would need to be made available for some of the major DL libraries (see Section II-B).

[8]RML2016.10b—https://radioml.com/datasets/radioml-2016-10-dataset/

[9]At the time of writing of this document, XGB (http://xgboost.readthedocs.io/) was together with CNNs the ML model that consistently won competions on the data-science platform Kaggle (https://www.kaggle.com/).

[10]http://yann.lecun.com/exdb/mnist/

[11]http://www.image-net.org/

B. Data representation, loss functions, and training SNR

As DL for communications is a new field, little is known about optimal data representations, loss-functions, and training strategies. For example, binary signals can be represented as binary or one-hot vectors, modulated (complex) symbols, or integers, and the optimal representation might depend among other factors of the NN architecture, learning objective, and loss function. In decoding problems, for instance, one would have the choice between plain channel observations or (clipped) log-likelihood ratios. In general, it seems that there is a representation which is most suited to solve a particular task via an NN. Similarly, it is not obvious at which SNR(s) DL processing blocks should be trained. It is clearly desirable that a learned system operates at any SNR, regardless at which SNR or SNR-range it was trained. However, we have observed that this is generally not the case. Training at low SNR for instance does not allow for the discovery of structure important in higher SNR scenarios. Training across wide ranges of SNR can also severely effect training time. The authors of [58] have observed that starting off the training at high SNR and then gradually lowering it with each epoch led to significant performance improvements for their application. A related question is the optimal choice of loss function. In Sections III-A to III-C, we have treated communications as a classification problem for which the categorical cross-entropy is a common choice. However, for alternate output data representations, the choice is less obvious. Applying an inappropriate loss function can lead to poor results.

Choosing the right NN architecture and training parameters for SGD (such as mini-batch size and learning rate) are also important practical questions for which no satisfying hard rules exist. Some guidelines can be found in [35, Ch. 11], but methods for how to select such *hyper-parameters* are currently an active area of research and investigation in the DL world. Examples include architecture search guided by hyper-gradients and differential hyper-parameters [59] as well as genetic algorithm or particle swarm style optimization [60].

C. Complex-valued neural networks

Owing to the widely used complex baseband representation, we typically deal with complex numbers in communications. Most related signal processing algorithms rely on phase rotations, complex conjugates, absolute values, etc. For this reason, it would be desirable to have NNs operate on complex rather than real numbers [61]. However, none of the previously described DL libraries (see Section II-B) currently support this due to several reasons. First, it is possible to represent all mathematical operations in the complex domain with a purely real-valued NN of twice the size, i.e., each complex number is simply represented by two real values. For example, an NN with a scalar complex input and output connected through a single complex weight, i.e., $y = wx$, where $y, w, x \in \mathbb{C}$, can be represented as a real-valued NN $\mathbf{y} = \mathbf{W}\mathbf{x}$, where the vectors $\mathbf{y}, \mathbf{x} \in \mathbb{R}^2$ contain the real and imaginary parts of y and x in each dimension and $\mathbf{W} \in \mathbb{R}^{2\times 2}$ is a weight matrix. Note that the real-valued version of this NN has four parameters while the complex-valued version has only two. Second, a complication arises in complex-valued NNs since traditional loss and activation functions are generally not holomorphic so that their gradients are not defined. A solution to this problem is Wirtinger calculus [62]. Although complex-valued NNs might be easier to train and consume less memory, we currently believe that they do not provide any significant advantage in terms of expressive power. Nevertheless, we keep them as an interesting topic for future research.

D. ML-augmented signal processing

The biggest challenge of learned end-to-end communications systems is that of scalability to large message sets $\mathcal{M}$. Already for $k = 100$ bits, i.e., $M = 2^{100}$ possible messages, the training complexity is prohibitive since the autoencoder must see at least every message once. Also naive neural channel decoders (as studied in [33]) suffer from this "curse of dimensionality" since they need to be trained on all possible codewords. Thus, rather than switching immediately to learned end-to-end communications systems or fully replacing certain algorithms by NNs, one more gradual approach might be that of *augmenting* only specific sub-tasks with DL. A very interesting approach in this context is *deep unfolding* of existing iterative algorithms outlined in [25]. This approach offers the potential to leverage additional side information from training data to improve an existing signal processing algorithm. It has been recently applied in the context of channel decoding and MIMO detection [22], [23], [24]. For instance in [22], it was shown that training with a single codeword is sufficient since the structure of the code is embedded in the NN through the Tanner graph. The concept of RTNs as presented in Section III-C is another way of incorporating both side information from existing models along with information derived from a rich dataset into a DL algorithm to improve performance while reducing model and training complexity.

E. System identification for end-to-end learning

In Sections III-A to III-C, we have tacitly assumed that the transfer function of the channel is known so that the backpropagation algorithm can compute its gradient. For example, for a Rayleigh fading channel, the autoencoder needs to know during the training phase the exact realization of the channel coefficients to compute how a slight change in the transmitted signal $\mathbf{x}$ impacts the received signal $\mathbf{y}$. While this is easily possible for simulated systems, it poses a major challenge for end-to-end learning over real channels and hardware. In essence, the hardware and channel together form a black-box whose input and output can be observed, but for which no exact analytic expression is known a priori. Constructing a model for a black box from data is called system identification [63], which is widely used in control theory. Transfer learning [64] is one appealing candidate for adapting an end-to-end communications system trained on a statistical model to a real-world implementation which has worked well in other domains (e.g., computer vision). An important related question is that of how one can learn a general model for a wide range of communication scenarios and tasks that would avoid retraining from scratch for every individual setting.

F. Learning from CSI and beyond

Accurate channel state information (CSI) is a fundamental requirement for multi-user MIMO communications. For this reason, current cellular communication systems invest significant resources (energy and time) in the acquisition of CSI at the base station and user equipment. This information is generally not used for anything apart from precoding/detection or other tasks directly related to processing of the current data frame. Storing and analyzing large amounts of CSI (or other radio data)—possibly enriched with location information—poses significant potential for revealing novel big-data-driven physical-layer understanding algorithms beyond immidiate radio environment needs. New applications beyond the traditional scope of communications, such as tracking and identification of humans (through walls) [65] as well as gesture and emotion recognition [66], could be achieved using ML on radio signals.

V. Conclusion

We have discussed several promising new applications of DL to the physical layer. Most importantly, we have introduced a new way of thinking about communications as an end-to-end reconstruction optimization task using autoencoders to jointly learn transmitter and receiver implementations as well as signal encodings without any prior knowledge. Comparisons with traditional baselines in various scenarios reveal extremely competitive BLER performance, although the scalability to long block lengths remains a challenge. Apart from potential performance improvements in terms of reliability or latency, our approach can provide interesting insight about the optimal communication schemes (e.g., constellations) in scenarios where the optimal schemes are unknown (e.g., interference channel). We believe that this is the beginning of a wide range of studies into DL and ML for communications and are excited at the possibilities this could lend towards future wireless communications systems as the field matures. For now, there are a great number of open problems to solve and practical gains to be had. We have identified important key areas of future investigation and highlighted the need for benchmark problems and data sets that can be used to compare performance of different ML models and algorithms.

References

[1] T. S. Rappaport, *Wireless communications: Principles and practice*, 2nd ed. Prentice Hall, 2002.

[2] R. M. Gagliardi and S. Karp, *Optical communications*, 2nd ed. Wiley, 1995.

[3] H. Meyr, M. Moeneclaey, and S. A. Fechtel, *Digital communication receivers: Synchronization, channel estimation, and signal processing*. John Wiley & Sons, Inc., 1998.

[4] T. Schenk, *RF imperfections in high-rate wireless systems: Impact and digital compensation*. Springer Science & Business Media, 2008.

[5] J. Proakis and M. Salehi, *Digital Communications*, 5th ed. McGraw-Hill Education, 2007.

[6] Y. LeCun *et al.*, "Generalization and network design strategies," *Connectionism in perspective*, pp. 143–155, 1989.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proc. IEEE Int. Conf. Computer Vision*, 2015, pp. 1026–1034.

[8] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. IEEE Int. Conf. Computer Vision*, 1999, pp. 1150–1157.

[9] Z. S. Harris, "Distributional structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.

[10] A. Goldsmith, "Joint source/channel coding for wireless channels," in *Proc. IEEE Vehicular Technol. Conf.*, vol. 2, 1995, pp. 614–618.

[11] E. Zehavi, "8-PSK trellis codes for a Rayleigh channel," *IEEE Trans. Commun.*, vol. 40, no. 5, pp. 873–884, 1992.

[12] H. Wymeersch, *Iterative receiver design*. Cambridge University Press, 2007, vol. 234.

[13] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[14] S. Reed and N. de Freitas, "Neural programmer-interpreters," *arXiv preprint arXiv:1511.06279*, 2015.

[15] H. T. Siegelmann and E. D. Sontag, "On the computational power of neural nets," in *Proc. 5th Annu. Workshop Computational Learning Theory*. ACM, 1992, pp. 440–449.

[16] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," in *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, 2011.

[17] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[18] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proc. Int. Conf. Mach. Learn. (ICML)*. ACM, 2009, pp. 873–880.

[19] M. Ibnkahla, "Applications of neural networks to digital communications–A survey," *Elsevier Signal Processing*, vol. 80, no. 7, pp. 1185–1215, 2000.

[20] M. Bkassiny, Y. Li, and S. K. Jayaweera, "A survey on machine-learning techniques in cognitive radios," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 3, pp. 1136–1159, 2013.

[21] J. Qadir, K.-L. A. Yau, M. A. Imran, Q. Ni, and A. V. Vasilakos, "IEEE Access Special Section Editorial: Artificial Intelligence Enabled Networking," *IEEE Access*, vol. 3, pp. 3079–3082, 2015.

[22] E. Nachmani, Y. Be'ery, and D. Burshtein, "Learning to decode linear codes using deep learning," in *Proc. IEEE Annu. Allerton Conf. Commun., Control, and Computing (Allerton)*, 2016, pp. 341–346.

[23] E. Nachmani, E. Marciano, D. Burshtein, and Y. Be'ery, "RNN decoding of linear block codes," *arXiv preprint arXiv:1702.07560*, 2017.

[24] N. Samuel, T. Diskin, and A. Wiesel, "Deep MIMO detection," *arXiv preprint arXiv:1706.01151*, 2017.

[25] J. R. Hershey, J. L. Roux, and F. Weninger, "Deep unfolding: Model-based inspiration of novel deep architectures," *arXiv preprint arXiv:1409.2574*, 2014.

[26] M. Borgerding and P. Schniter, "Onsager-corrected deep learning for sparse linear inverse problems," *arXiv preprint arXiv:1607.05966*, 2016.

[27] Y.-S. Jeon, S.-N. Hong, and N. Lee, "Blind detection for MIMO systems with low-resolution ADCs using supervised learning," *arXiv preprint arXiv:1610.07693*, 2016.

[28] N. Farsad and A. Goldsmith, "Detection algorithms for communication systems using deep learning," *arXiv preprint arXiv:1705.08044*, 2017.

[29] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos, "Learning to optimize: Training deep neural networks for wireless resource management," *arXiv preprint arXiv:1705.09412*, 2017.

[30] T. J. O'Shea, K. Karra, and T. C. Clancy, "Learning to communicate: Channel auto-encoders, domain specific regularizers, and attention," in *Proc. IEEE Int. Symp. Signal Process. and Inf. Technol. (ISSPIT)*, 2016, pp. 223–228.

[31] T. J. O'Shea, J. Corgan, and T. C. Clancy, "Convolutional radio modulation recognition networks," in *Proc. Int. Conf. Eng. Applications of Neural Networks*. Springer, 2016, pp. 213–226.

[32] T. J. O'Shea, J. Corgan, and T. C. Clancy, "Unsupervised representation learning of structured radio communication signals," in *Proc. IEEE Int. Workshop Sensing, Processing and Learning for Intelligent Machines (SPLINE)*, 2016, pp. 1–5.

[33] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning-based channel decoding," in *Proc. IEEE 51st Annu. Conf. Inf. Sciences Syst. (CISS)*, 2017, pp. 1–6.

[34] S. Cammerer, T. Gruber, J. Hoydis, and S. t. Brink, "Scaling deep learning-based decoding of polar codes via partitioning," *arXiv preprint arXiv:1702.06901*, 2017.

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[36] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting." *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[37] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2010, pp. 807–814.

[38] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[39] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.

[40] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[41] R. Al-Rfou, G. Alain, A. Almahairi *et al.*, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv preprint arXiv:1605.02688*, 2016.

[42] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, 2011.

[43] F. Chollet, "keras," https://github.com/fchollet/keras, 2015.

[44] T. O'Shea and J. Hoydis, "Source code," https://github.com/-available-after-review, 2017.

[45] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[46] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[47] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," in *Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 2933–2941.

[48] L. v. d. Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[49] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 2672–2680.

[50] M. Abadi and D. G. Andersen, "Learning to protect communications with adversarial neural cryptography," *arXiv preprint arXiv:1610.06918*, 2016.

[51] M. Jaderberg, K. Simonyan, A. Zisserman *et al.*, "Spatial transformer networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 2017–2025.

[52] J. Estaran *et al.*, "Artificial neural networks for linear and non-linear impairment mitigation in high-baudrate IM/DD systems," in *Proc. 42nd European Conf. Optical Commun. (ECOC)*. VDE, 2016, pp. 1–3.

[53] A. K. Nandi and E. E. Azzouz, "Algorithms for automatic modulation recognition of communication signals," *IEEE Trans. Commun.*, vol. 46, no. 4, pp. 431–436, 1998.

[54] A. Fehske, J. Gaeddert, and J. H. Reed, "A new approach to signal classification using spectral correlation and neural networks," in *IEEE Int. Symp. New Frontiers in Dynamic Spectrum Access Networks (DYSPAN)*, 2005, pp. 144–150.

[55] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[56] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[57] A. Abdelmutalab, K. Assaleh, and M. El-Tarhuni, "Automatic modulation classification based on high order cumulants and hierarchical polynomial classifiers," *Physical Communication*, vol. 21, pp. 10–18, 2016.

[58] D. George and E. Huerta, "Deep neural networks to enable real-time multimessenger astrophysics," *arXiv preprint arXiv:1701.00008*, 2016.

[59] D. Maclaurin, D. Duvenaud, and R. P. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *Proc. 32nd Int. Conf. Mach. Learn. (ICML)*, 2015.

[60] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.

[61] A. Hirose, *Complex-valued neural networks*. Springer Science & Business Media, 2006.

[62] M. F. Amin, M. I. Amin, A. Al-Nuaimi, and K. Murase, "Wirtinger calculus based gradient descent and Levenberg-Marquardt learning algorithms in complex-valued neural networks," in *Int. Conf. on Neural Information Processing*. Springer, 2011, pp. 550–559.

[63] G. C. Goodwin and R. L. Payne, *Dynamic system identification: experiment design and data analysis*. Academic press, 1977.

[64] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, 2010.

[65] F. Adib, C.-Y. Hsu, H. Mao, D. Katabi, and F. Durand, "Capturing the human figure through a wall," *ACM Trans. Graphics (TOG)*, vol. 34, no. 6, p. 219, 2015.

[66] M. Zhao, F. Adib, and D. Katabi, "Emotion recognition using wireless signals," in *Proc. ACM Annu. Int. Conf. Mobile Computing and Networking*, 2016, pp. 95–108.