An Introduction to Machine Learning Communications Systems

Article · February 2017

CITATIONS
33

READS
3,019

2 authors, including:

Tim O'Shea
Virginia Polytechnic Institute and State University
32 PUBLICATIONS **552** CITATIONS

An Introduction to Machine Learning Communications Systems

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

***Abstract*—We introduce and motivate machine learning (ML) communications systems that aim to improve on and to even replace the vast expert knowledge in the field of communications using modern machine learning techniques. These have recently achieved breakthroughs in many different domains, but not yet in communications. By interpreting a communications system as an autoencoder, we develop a fundamental new way to think about radio communications system design as an end-to-end reconstruction optimization task that seeks to jointly optimize transmitter and receiver components in a single process. We further present the concept of Radio Transformer Networks (RTNs) as a means to incorporate expert domain knowledge in the ML model and study the application of convolutional neural networks (CNNs) on raw IQ time-series data for modulation classification. We conclude the paper with a deep discussion of open challenges and areas for future investigation.**

I. Introduction

A. Motivation

Since the groundbreaking work of Shannon [1], we know the ultimate limit of communications in terms of reliably achievable data rates over noisy channels. The challenge of achieving this limit has been driving radio engineers for the last 70 years: find specific algorithms that attain an efficient transfer of information (e.g., low error probability, high spectral efficiency, low latency) over a variety of channels. Although many of the algorithms communications engineers have come up with achieve remarkable performance—often close to the Shannon limit—we believe that machine learning (ML) communications systems hold the potential to improve on some of these algorithms in terms of reliability, generality, latency, and energy efficiency. The main reasons for this are:

Inadequate system models: Most signal processing algorithms in communications have solid foundations in statistics and information theory. However, these algorithms are often optimized for mathematically convenient models (which are linear, stationary, and have Gaussian statistics), but not for real systems with many imperfections and non-linearities. An ML based communications system does not require such a rigidly defined model for representation and transformation of information and could be optimized in an end-to-end manner for a real system with harsh realistic effects.

Limiting functional block-structure: Engineers have learned to represent communications systems through a chain of multiple independent processing blocks; each executing a well defined and isolated function (e.g., coding, modulation, channel estimation, equalization). However, it is not clear that individually optimized processing blocks achieve the best possible end-to-end performance. In fact, it seems likely that we are introducing artificial barriers and constraints to efficiency. For example, we do not necessarily care how well we can estimate the channel with a given scheme, or how well any one independent function works, rather we seek to optimize end-to-end system metrics jointly over all components. A learned end-to-end communications system will likely not possess such a well defined block structure as it is trained to achieve only the best end-to-end performance.

Parallelization gains of neural networks: It is well known that neural networks are universal function approximators [2] and recent work has shown a remarkable capacity for algorithmic learning with recurrent neural networks [3], a construct which has been shown to be Turing-complete [4]. Since the execution of neural networks can be highly parallelized using data and computationally distributed concurrent architectures, and has shown to work well with small data types [5] conducive to efficient wide single-instruction multiple-data (SIMD) operations, there is some hope that "learned" algorithms can be executed significantly faster and at lower energy cost than manually "programmed" counterparts.

Specialized hardware for ML applications: Computational micro-architectures have shown a relentless trend towards higher transistor counts and lower powers and clock rates for achieving optimal energy efficiencies. Massively concurrent architectures with distributed memory architectures, such as graphical processing units (GPUs), have shown to be very energy efficient and capable of impressive computational throughput when fully utilized by concurrent algorithms. The performance of such systems, 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 distribution across wide and deep neural networks holds the promise of achieving highly efficient distributed algorithms which can fully utilize parallel hardware architectures.

Our vision is that, in the future, numerous algorithms in computing and communications will likely not be represented by special purpose expert code, but by learned weights of neural networks optimized for end-to-end loss functions. Neural network primitives and tensor expressions offer well defined algorithmic constructs in which parameters can be readily modified to optimize different tasks on highly concurrent processing architectures. Although this vision is quite appealing,

T. O'Shea is with the Bradley Department of Electrical and Computer Engineering, Virginia Tech, 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).

it is currently still unclear to what extent ML can replace or augment the expert knowledge which has been developed during the last century. The goal of this article is to provide an introduction to ML communications systems, discuss related open challenges, and present interesting directions for future research. We have made the source code for all numerical examples available [6] to allow for reproducible research.

The remainder of this article is structured as follows: Fundamentals of deep learning and related topics, such as training and ML libraries, together with a summary of prior work is presented in Sections I-B to I-G. In Section II, several examples of ML applications to communications are presented. Section III contains an overview and discussion of open problems and key areas of future investigation, while Section IV concludes the article.

B. Background and related work

The use of ML techniques in digital communications has a long history covering a wide range of applications. These comprise (non-linear) channel modeling and prediction, localization, equalization, filtering, decoding, quantization, compression, demodulation, and modulation recognition, to name a few [7] (and references therein). However, to the best of our knowledge, hardly any of these applications have been adopted in a product or led to a 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 combination of ML and communications is frequently associated with the term "cognitive radio" which has been around since 1995 [8] and saw a surge of interest and research in methods using ML for radio systems optimization [9]. The dream of radio systems that could rapidly adapt, self-optimize, self-organize, and operate autonomously emerged. Significant work was invested in realizing expert systems for a range of applications, such as dynamic spectrum access and primary user sensing/avoidance. This work proved to be highly specialized per task, and much of it lacked the ability to generalize and provide any form of general intelligence in wireless devices. Since then, the interest in cognitive radio has dwindled, a phenomenon which we refer to as the cognitive radio winter, which basically mirrored the plateau and lack of funding, advancements, or general major steps forward seen in the first and second artificial intelligence winters [10].

The advent of open-source ML libraries (see Section I-F) and cheaply available specialized hardware together with the astonishing progress of ML in computer vision have stimulated renewed interest in the application of ML for networking [11]. The Defense Advanced Research Projects Agency (DARPA) has recently kicked-off the Spectrum Collaboration Challenge (SC2)[1] with the goal of competing self-optimizing communications systems for various yet to be specified global objective functions, such as throughput maximization and inteference minimization. Several research groups have given a fresh look at ways of how to improve existing signal processing algorithms with ML, such as belief propagation for channel decoding [12], compressed sensing [13], [14], blind detection for multiple-input multiple-output (MIMO) systems with low-resolution analog-to-digital converters (ADCs) [15], and learning of encryption/decryption schemes to achieve privacy in an eavesdropper channel [16]. Some of us have recently made preliminary investigations of learned end-to-end communications systems [17] and revisited the problems of modulation detection [18], compression [19], and channel decoding [20] with state-of-the art ML tools. Several of these ideas will be presented in Section II.

[1] https://spectrumcollaborationchallenge.com/

C. 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} denotes the (i,j)-element. $\mathbb{R}$ and $\mathbb{C}$ denote the sets of real and complex numbers, respectively. $\mathcal{N}(\mathbf{m}, \mathbf{R})$ is the multivariate Gaussian distribution with mean vector $\mathbf{m}$ and covariance matrix $\mathbf{R}$. $\mathrm{Bern}(\alpha)$ is the Bernoulli distribution with success probability α and ∇ is the gradient operator.

D. Deep learning basics

A feedforward neural network with L layers describes a mapping $f(\mathbf{r}_0;\theta) : \mathbb{R}^{N_0} \mapsto \mathbb{R}^{N_L}$ 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:

$$\mathbf{r}_l = f_l(\mathbf{r}_{l-1};\theta_l), \qquad l = 1,\ldots,L \tag{1}$$

where $f_l(\mathbf{r}_{l-1};\theta_l) : \mathbb{R}^{N_{l-1}} \mapsto \mathbb{R}^{N_l}$ is the mapping carried out by the lth layer. This mapping depends not only on the output vector $\mathbf{r}_{l-1}$ 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 $\theta = \{\theta_1,\ldots,\theta_L\}$ to denote the set of all parameters of the network. The lth layer is called *dense* or *fully-connected* if $f_l(\mathbf{r}_{l-1};\theta_l)$ has the form

$$f_l(\mathbf{r}_{l-1};\theta_l) = \sigma\left(\mathbf{W}_l\mathbf{r}_{l-1} + \mathbf{b}_l\right) \tag{2}$$

where $\mathbf{W}_l \in \mathbb{R}^{N_l \times N_{l-1}}$, $\mathbf{b}_l \in \mathbb{R}^{N_l}$, and $\sigma(\cdot)$ is an *activation* function which we will be defined shortly. The set of parameters for this layer is $\theta_l = \{\mathbf{W}_l, \mathbf{b}_l\}$. Table I lists some 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 $\beta\mathbf{I}_{N_{l-1}}$ 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 very important for the so-called *expressive power* of the neural network. 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.

TABLE I: List of layer types

Name	$f_l(\mathbf{r}_{l-1};\theta_l)$	θ_l
Dense	$\sigma\left(\mathbf{W}_l\mathbf{r}_{l-1}+\mathbf{b}_l\right)$	$\mathbf{W}_l, \mathbf{b}_l$
Noise	$\mathbf{r}_{l-1}+\mathbf{n},\ \mathbf{n}\sim\mathcal{N}(\mathbf{0},\beta\mathbf{I}_{N_{l-1}})$	none
Dropout	$\mathbf{d}\odot\mathbf{r}_{l-1},\ d_i\sim\mathrm{Bern}(\alpha)$	none
Normalization	$\frac{\sqrt{N_{l-1}}\mathbf{r}_{l-1}}{\|\mathbf{r}_{l-1}\|_2}$	none

TABLE II: List of activation functions

Name	$[\sigma(\mathbf{u})]_\mathbf{i}$	Range
linear	u_i	$(-\infty,\infty)$
ReLU	$\max(0,u_i)$	$[0,\infty)$
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	$-\sum_j u_j\log(v_j)$

Feedforward neural networks are generally trained using labeled training data, i.e., a set of input-output vector pairs $(\mathbf{r}_{0,i},\mathbf{r}^\star_{L,i})$, $i=1,\ldots,S$, 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

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

with respect to the parameters in θ, 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 neural network when $\mathbf{r}_{0,i}$ is used as input. Several relevant loss functions are provided in Table III. Different norms (e.g., ℓ_1, ℓ_2) of some of the parameters can be added to the loss function to favor solutions with small or sparse parameters. The most popular algorithm to find good sets of parameters θ is stochastic gradient descent (SGD) which starts with some random initial values of $\theta=\theta_0$ and then updates θ iteratively as

$$\theta_{t+1}=\theta_t-\eta\nabla\tilde{L}(\theta_t) \tag{4}$$

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

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

In general, S_t is very small compared to S. Note that there are many variants of the SGD algorithm which dynamically adapt the learning rate to achieve faster convergence [21, Ch. 8.5]. The gradient in (4) can be very efficiently computed through the back-propagation algorithm [21, Ch. 6.5]. Defining and training neural networks of almost arbitrary shape can be very easily done with one of the many existing ML libraries presented in Section I-F.

E. Convolutional Layers

Convolutional neural network (CNN) layers were introduced in [22] 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. Additionally, they reduce the model complexity, as measured by the number of free parameters in the layer's weight matrix, required to represent such a feature at any location in the input layer, as compared with a fully connected weight matrix of all inputs to all outputs between the two layers.

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

$$Y^f_{i,j}=\sum_{k=0}^{a-1}\sum_{l=0}^{b-1} Q^f_{k,l}X_{si-k,sj-l} \tag{6}$$

where $s\geq 1$ is an integer parameter called *stride* and it is assumed that $\mathbf{X}$ is padded with zeros, i.e., $X_{i,j}=0$ for all $i\notin[0,n-1]$ and $j\notin[0,m-1]$. The output dimensions can be reduced by either increasing the stride s or by adding a *pooling* layer. The pooling layer essentially cuts $\mathbf{Y}$ into $p\times p$ regions for each of which it computes a single value, e.g., the maximum or average value, or the ℓ_2-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 $6\times 5\times 5=150$. For the right kind of dataset, this technique can be extremely effective. We will see an application of convolutional layers in Section II-C. For more details on CNNs, we refer to [21, Ch. 9].

F. Machine learning libraries

In recent times, numerous tools and algorithms have emerged that make it easy to build and train large neural networks. Tools to deploy such training routines from high level language to massively concurrent GPU architectures have been key enablers. Among these are Theano [23] and TensorFlow [24], which allow for high level algorithm definition in the Python programming language, automatic differentiation of training loss functions through very large networks, and compilation of the network's forwards and backwards passes into hardware optimized concurrent dense matrix algebra kernels. Keras [25] provides an additional layer of neural network primitives with Theano and TensorFlow as its back-end. It has a highly customizable interface to quickly experiment with and deploy deep neural networks, and has become our primary tool used to generate the numerical results for this manuscript [6].

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

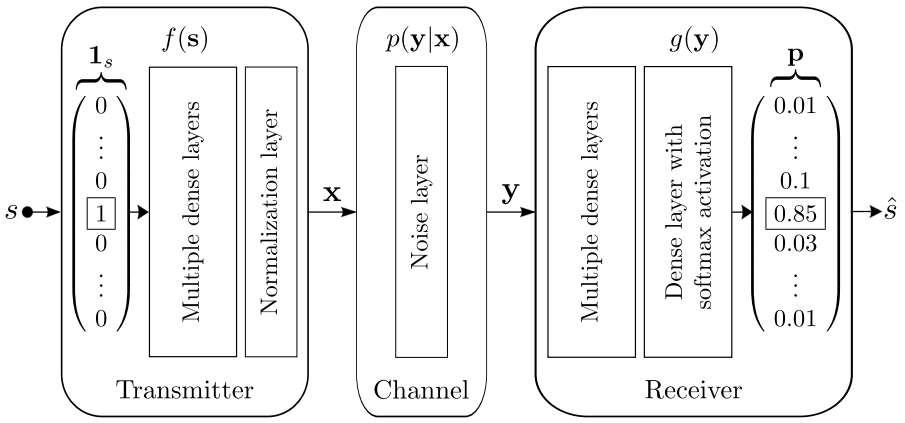

Fig. 2: A communications system over an AWGN channel represented as an autoencoder. The input s is encoded as a one-hot vector, the output is a probability distribution over all possible messages from which the most likely is picked as output $\hat{s}$.

G. Network Dimensions and Training

The term "deep" has become common in recent neural network literature, referring typically to the number of layers used within a single network. It can be related to the number of iterative operations that are performed on the input data through sequential layers' transfer functions. While deep networks may 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 activations in the output of one given layer, or all layers on average. It can also be thought of as an approximate measure for the memory required at each layer to represent its state.

Conventional wisdom in training methods has varied over the years, from direct solution techniques, gradient descent techniques, genetic algorithm search techniques each having been favored or considered at one time (see [21, Ch. 1.2] for a short history of deep learning). Layer-by-layer pre-training [26] was also a recently popular method for scaling weight training to larger networks where back-propagation once struggled. However, most systems today are able to train networks which are both wide and deep by using SGD methods with adaptive learning rates, regularization methods, such as Dropout, to prevent overfitting, and activations functions, such as ReLU, to reduce gradient issues.

II. 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. We will then introduce the concept of Radio Transformer Networks (RTNs) to deal with fading channels, and demonstrate an application of CNNs to raw radio frequency time-series data for the task of modulation classification.

A. Autoencoders for end-to-end communications systems

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 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 $\|\mathbf{x}\|_2^2 \leq n$ or an amplitude constraint $|x_i| \leq 1 \, \forall i$. We assume in this work an energy constraint as it leaves more freedom to the signal design. Extensions to arbitrary constraints are straightforward but might change some of the results. 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 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.

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

From an ML point of view, this simple communications system can be seen as an *autoencoder* [21, Ch. 14]. The goal of the autoencoder is to find a representation $\mathbf{x}$ of s, which is robust with respect to the perturbations created by the channel, so that it can reconstruct s based on $\mathbf{y}$. An example of such an autoencoder is provided in Fig. 2. Here, the transmitter consists of a feedforward network with multiple dense layers, followed by a normalization layer that ensures that all constraints on $\mathbf{x}$ are met. Note that the input s to the transmitter is first 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 all other elements are equal to zero. The channel is represented by a simple noise layer with a fixed variance $\beta = (2RE_b/N_0)^{-1}$. Similar to the transmitter, the receiver is implemented as

[3] We focus here on real-valued signals only. Extensions to complex-valued signals are discussed in Section III. 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 II-B and II-C.

another feedforward network with multiple dense layers. The last layer has a softmax activation whose output $\mathbf{p} \in (0,1)^M$ can be interpreted as a probability distribution over all possible messages, i.e., the elements of $\mathbf{p}$ are positive and sum to one. The decoded message $\hat{s}$ corresponds then to the index of $\mathbf{p}$ with the highest probability. The autoencoder can be trained end-to-end using SGD. The training data is the set of all possible messages $s \in \mathcal{M}$. A suitable loss function is the categorical cross-entropy between $\mathbf{1}_s$ and $\mathbf{p}$.[4]

Fig. 3 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). Both systems operate hence 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. In all of our experiments, training was done at a fixed value of E_b/N_0 (cf. Section III-B). For all other implementation details, we refer to the source code [6].

Fig. 4 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.

The examples of this section treat the communication task as a classification problem and the representation of $\mathbf{s}$ by an M-dimensional vector becomes quickly impractical for large M. To circumvent this problem, it is possible to use more compact representations of $\mathbf{s}$ 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 sheer size of the training set. A very important property of the autoencoder is that it can learn to communicate over *any* channel, even for which no information-theoretically optimal schemes are known.

B. Radio transformer networks for augmented signal processing algorithms

A number of the phenomena undergone in the wireless channel and the transceiver hardware can be removed through the application of parametric transformations. These include re-sampling to adjust time offset, time scale, and clock recovery; mixing with a carrier to correct phase and frequency offset, and convolution with an inverted channel response to

[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 [6].

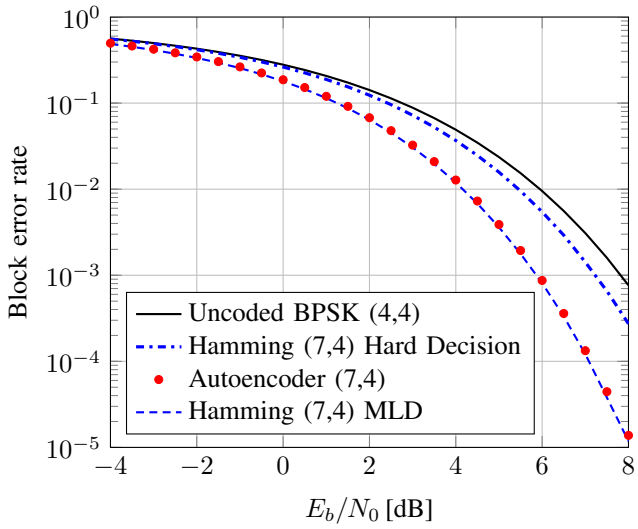

Fig. 3: BLER versus E_b/N_0 for the autoencoder and several baseline communication schemes.

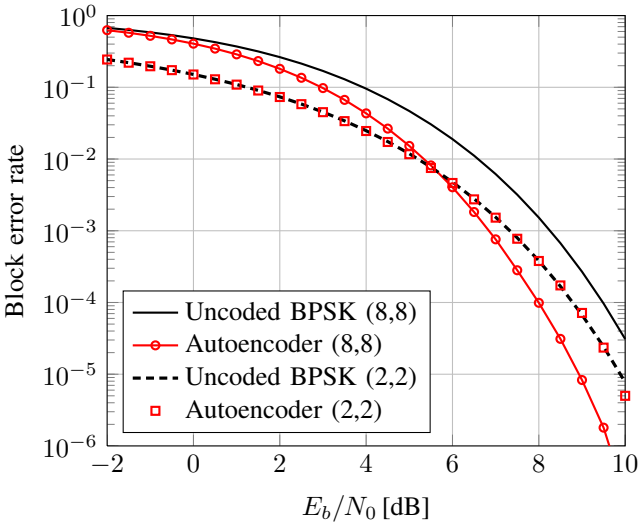

Fig. 4: BLER versus E_b/N_0 for the autoencoder and BSPK.

TABLE IV: Layout of the autoencoder used in Figs. 3 and 4. 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

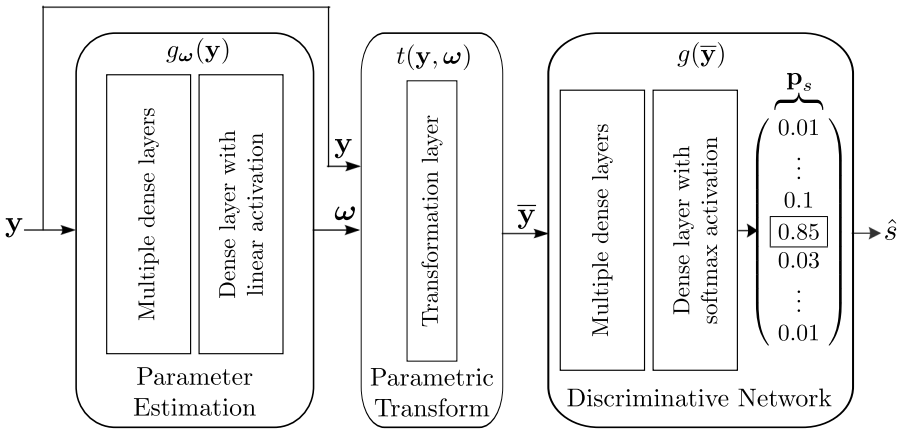

Fig. 5: A radio receiver represented as an RTN. The input $\mathbf{y}$ first runs through a parameter estimation network $g_{\omega}(\mathbf{y})$, has a known transform $t(\mathbf{y}, \boldsymbol{\omega})$ applied to generate the canonicalized signal $\overline{\mathbf{y}}$, and then is classified in the discriminative network $g(\overline{\mathbf{y}})$ to produce the output $\hat{s}$.

remove multipath fading effects. However, the estimation of the parameters to seed these inverse transforms (e.g., frequency offset, channel impulse response) is often a very involved process based on signal specific properties and/or information from pilot tones.

One way of augmenting pure ML models with our expert knowledge of the radio domain's behavior is the use of an RTN as shown in Fig. 5. 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 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 from the canonicalized input $\overline{\mathbf{y}} \in \mathbb{R}^{n'}$. By allowing the parameter estimator $g_{\boldsymbol{\omega}}$ to take the form of a neural network, we can train our system end-to-end to optimize for classification, reconstruction, or regression tasks. 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. 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. The idea of RTNs is inspired from Spatial Transformer Networks [27] that have worked well in computer vision. However, the parametric transforms used in this domain are not well suited for radio domain tasks which leads to our modified approach.

A compelling example demonstrating the advantages of RTNs is shown in Fig. 6 which compares the BLER of an autoencoder (8,4) with and without RTN over a multipath fading channel with $L = 3$ channel taps. That is, the received signal $\mathbf{y} = [\Re\{y(0)\}, \ldots, \Re\{y(n-1)\}, \Im\{y(0)\}, \ldots, \Im\{y(n-1)\}]^{\mathsf{T}}$ is given as

$$y(t) = \sum_{l=0}^{L-1} h(l)x(t-l) + n(t) \tag{7}$$

where $h(l) \sim \mathcal{CN}(0, 1/L)$ are i.i.d. Rayleigh fading channel taps, $n(t) \sim \mathcal{CN}(0, (RE_b/N_0)^{-1})$ is receiver noise, and $x(t) = 0$ for $t < 0$. We implement the complex-valued representation in (7) by having $2n$ real outputs and inputs at the encoder and receiver, respectively, which are interpreted as a concatenation of the real and imaginary parts of $\mathbf{x}$ and $\mathbf{y}$. In this example, the goal of the parameter estimator is to predict a complex-valued vector $\boldsymbol{\omega}$ (represented by $2L$ real values) that is used in the transformation layer to compute the complex convolution of $\mathbf{y}$ with $\boldsymbol{\omega}$. 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 a regression network using two dense layers with tanh activations followed by a dense layer with linear activation.

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. 7 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 become smaller when the encoder and decoder networks are made wider and trained for more iterations. Although there is theoretically nothing an RTN-augmented network can do that a plain feedforward network cannot, the RTN helps the training process by incorporating expert domain knowledge and simplifying the complexity of the target function, similar to the role of convolutional layers in imparting translation invariance where appropriate.

The autoencoder and RTN as presented above could 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

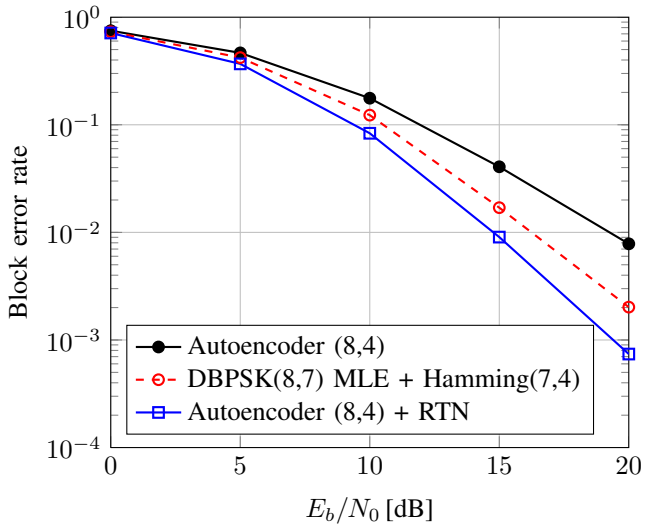

Fig. 6: BLER versus E_b/N_0 for various communication schemes over a channel with $L = 3$ Rayleigh fading taps.

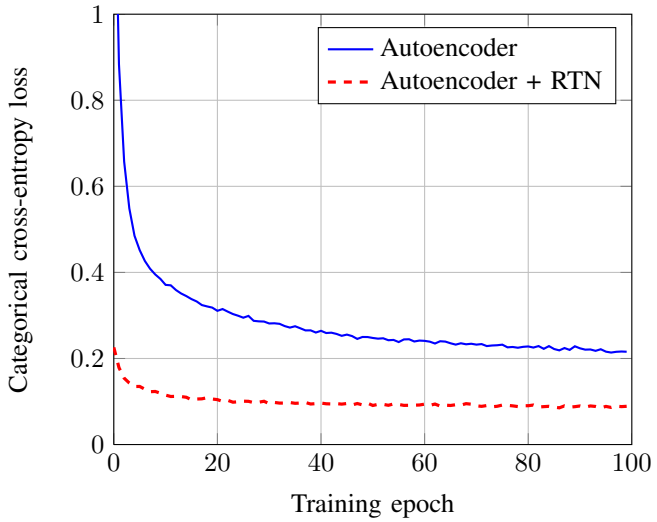

Fig. 7: Autoencoder training loss with and without RTN

could also arise in optical communications dealing with highly non-linear channel impairments that are notoriously difficult to model and compensate for [28].

C. CNNs for classification tasks

Many signal processing functions within a radio 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. This task has been accomplished for years through the approach of expert feature engineering and either analytic decision trees or trained discrimination methods operating within feature space, such as support vector machines, random forests, or small feedforward neural networks [29]. Some recent methods take a step beyond this using pattern recognition on simplified expert feature maps, such as the spectral coherence function or

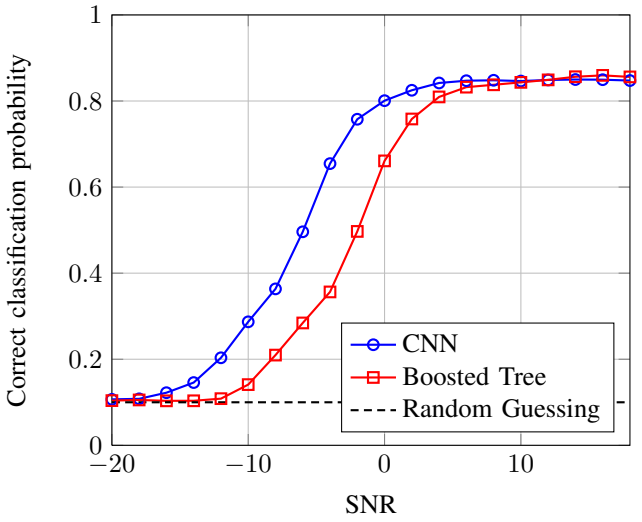

Fig. 8: Classifier performance comparison versus SNR

α-profile, combined with neural network based classification [30]. 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 widely done for image classification problems, we leverage a series of convolutional layers followed by a series of fully-connected layers terminated with a softmax activation in order to form our classifier. The layout is provided in Table V and we refer to the source code [6] for further implementation details. The dataset[5] 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, such as multipath fading, sample rate and center frequency offset [18]. The samples are taken at 20 different SNRs within the range from -20 dB to 18 dB.

In Fig. 8, we compare the classification accuracy of the CNN against that of extreme gradient boosting[6] with 1000 estimators, operating on a mix of 16 analog and cumulant expert features as proposed in [29] and [31]. The short-time nature of the examples places this task on the more difficult end of the modulation classification spectrum since we cannot integrate expert features for high stability over long periods of time. We can see that the CNN outperforms the feature-based classifier by around 4 dB in the low to medium SNR range while the performance at high SNR is almost identical. In [32], the authors report on a successful application of a similar CNN layout for the detection of black hole mergers in astrophysics from noisy time-series data.

[5]RML2016.10b– https://radioml.com/datasets/radioml-2016-10-dataset/
[6]http://xgboost.readthedocs.io/

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

III. 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 boosting a recent surge in ML research progress (e.g., MNIST[7] or ImageNet[8]), nothing comparable exists for ML communications systems. Communications is somewhat different because we are dealing 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 desirable (as is common in many traditional ML domains) 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 II-C.

B. Data representation, loss functions, and training SNR

As ML 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 on the neural network architecture, the learning objective, as well as the 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 a neural network. Similarly, it is not obvious at which SNR(s) an ML communications system should be trained. It is clearly desirable that a learned system operates over any SNR, i.e., generalizing to arbitrary SNR values at test time regardless of training. However, we have observed that training at certain SNRs or SNR ranges does not translate obviously to test performance. Training at low SNR for instance does not allow for the discovery of higher structure important in higher SNR scenarios. The authors of [32] 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 Section II-A, 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 neural network 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 [21, Ch. 11], but methods for how to select such *hyper-parameters* are currently an active area of research and investigation in the ML world. There have been a number of important works in this field over the past several years including investigations into hyper-gradients and making hyper-parameters differentiable during architecture search [33] as well as investigations into how genetic algorithm or particle swarm style optimization can be used to guide this search [34]. These are important questions required to optimize neural network architectures and training processes on existing tasks, and for adapting them to new datasets and tasks. We have begun investigating this area based on distributed concurrent architecture search processes, an approach that has been discussed widely in the context of asynchronous parameter sharing systems for the training of very large networks [35].

C. Complex-valued neural networks

Owing to the complex baseband representation, we naturally deal with complex numbers in communications and most related signal processing algorithms rely on phase rotations, complex conjugates, absolute values, etc. For this reason, it would be desirable to have neural networks operate on complex rather than real numbers [36]. However, none of the previously described ML libraries (see Section I-F) currently support this, and there are several reasons for it. First, it is possible to represent all mathematical operations in the complex domain with a purely real-valued neural network of twice the size, i.e., each complex number is simply represented by two real values. For example, a neural network 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 neural network $\mathbf{y} = \mathbf{W}\mathbf{x}$, 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 network has four parameters while the complex-valued network has only two. Second, a complication arises in complex-valued neural networks since traditional loss and activation functions are generally not holomorphic so that their gradient is not defined. A solution to this problem is Wirtinger calculus [37]. Although complex-valued neural networks 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, but keep them as an interesting topic for future research.

D. ML-augmented signal processing

Rather than trying to immidiately replace all expert knowledge developed during the last century with ML, an alter-

[7] http://yann.lecun.com/exdb/mnist/
[8] http://www.image-net.org/

nate approach might be to augment existing algorithms for certain applications with the help of ML on sub-tasks. This is especially important as it provides the ML algorithm with side information which it would need to infer otherwise from the training data. This problem arises for example in channel coding. A naive deep neural decoder taking noisy observations of transmitted symbols as input and providing the corresponding information bits as output would need to see all possible codewords during the training phase. Since this number grows exponentially with the codelength, such a learning system suffers from the "curse of dimensionality" which prevents scaling to larger code length. A nice idea of how this problem can be circumvented is [12] which reinterprets the Tanner graph of a high density parity check (HDPC) code as a neural network with artificially introduced, trainable weights. Since the structure of the code is provided by the Tanner graph, it is sufficient to train this deep learning decoder with a single codeword. This approach is shown to improve on the standard belief propagation algorithm. However, as recently shown in [20], there is some hope that neural networks can learn to decode the full codebook of structured codes after having been trained on only a subset of it, although it is still unclear which type of architecture would achieve the best generalization performance. Other relevant examples of ML-augmented signal processing are [13] and [14] in the context of compressed sensing. The concept of RTNs as presented in Section II-B is another way of incorporating side information into a large variety of ML algorithms.

E. System identification for end-to-end learning

In Section II-A, we have tacitly assumed that the transfer function of the channel is known so that the back-propagation 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 [38], which is widely used in control theory. Transfer learning [39] 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).

F. Reinforcement learning for radio control tasks

Radio control planes are often faced with complex resource management challenges. These consist of optimizing the use of power, spectrum, and network resources while optimizing the user experience for latency, bandwidth, application function, and user satisfaction. In the context of cellular handset operation, for both base station resource allocation of spatial and spectral resources, as well as spectrum sensing for interference awareness, spectrum coordination and regulatory compliance, we are faced with a wide range of distinctly control-system problems that hold significant potential for optimization using neural network driven reinforcement learning approaches. Recent work in this field has provided drastic progress in the capacity for learning complex reward seeking control systems in increasingly difficult environments [40], [41], [42], [43]. This class of approaches could be applied in the radio resource management layer to optimize and control complex communications systems with many operating parameters efficiently. Some applications of these techniques in the context of radio resource managment for 5G systems are discussed in [44].

G. Algorithm learning

The more layers a neural network has, the more sequential processing steps it can perform on the input data. For this reason, deep neural networks are often more successful than shallow ones for a large number of problems in computer vision. Recurrent neural networks can perform in theory an infinitely large number of processing steps and are, therefore, Turing complete [4]; but they are notoriously difficult to train and harder to paralelize. Similar, to Turing's idea of enriching a finite-state machine by an infinite memory tape, memory-augmented neural networks, such as the Neural Turing Machine [45], add an addressable memory to recurrent neural networks with the goal of simplifying the learning process [46]. Such constructs could potentially learn arbitrary algorithms which could generalize from a few short training examples to very long sequences of input data. For example, rather than treating decoding as a hopeless classification problem, memory-augmented neural networks could try to learn a decoding (and even encoding) algorithm which does not suffer from the curse of dimensionality. However, work on these types of ML models is still early and has a long way to mature for such complex algorithmic tasks.

H. 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—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) [47] as well as gesture and emotion recognition [48], could be achieved using ML on radio signals.

IV. Conclusion

We have discussed a number of promising new applications of ML to radio communications system design, which offer

attractive properties with regards to their performance, complexity, and generalization capabilities. We have introduced a new way of thinking about communications as an end-to-end reconstruction optimization task using an autoencoder to jointly learn transmit and receive functions without prior knowledge. We believe that this is the beginning of a wide range of studies into ML based communications systems 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 a collection of key areas of future investigation and look forward to the continued growth and adoption of ML approaches in communications as these problems are better characterized and solved.

REFERENCES

[1] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, Jul. 1948.

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

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

[4] 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.

[5] 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*, vol. 1. Citeseer, 2011, p. 4.

[6] T. O'Shea and J. Hoydis, "Source code," https://github.com/-available-after-review, 2016.

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

[8] J. Mitola III, "Software radio–cognitive radio," *IEEE Commun. Mag.*, vol. 33, no. 5, pp. 26–38, 1995.

[9] T. W. Rondeau, "Application of artificial intelligence to wireless communications," Ph.D. dissertation, Virginia Polytechnic Institute and State University, 2007.

[10] H. Newquist, *Brain Makers*. Editors & Engineers, Limited, 1994.

[11] 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.

[12] E. Nachmani, Y. Beery, and D. Burshtein, "Learning to decode linear codes using deep learning," *arXiv preprint arXiv:1607.04793*, 2016.

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

[14] A. Mousavi and R. G. Baraniuk, "Learning to invert: Signal recovery via deep convolutional networks," *arXiv preprint arXiv:1701.03891*, 2017.

[15] 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.

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

[17] T. J. O'Shea, K. Karra, and T. C. Clancy, "Learning to communicate: Channel auto-encoders, domain specific regularizers, and attention," *arXiv preprint arXiv:1608.06409*, 2016.

[18] T. J. O'Shea and J. Corgan, "Convolutional radio modulation recognition networks," *arXiv preprint arXiv:1602.04105*, 2016.

[19] T. J. O'Shea, J. Corgan, and T. C. Clancy, "Unsupervised representation learning of structured radio communication signals," *arXiv preprint arXiv:1604.07078*, 2016.

[20] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning-based channel decoding," in *arXiv preprint arXiv:1701.07738*, 2017.

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

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

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

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

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

[26] 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.

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

[28] J. Estaran, R. Rios-Müller, M. Mestre, F. Jorge, H. Mardoyan, A. Konczykowska, J.-Y. Dupuy, and S. Bigo, "Artificial neural networks for linear and non-linear impairment mitigation in high-baudrate IM/DD systems," in *Proc. 42nd European Conf. Optical Communication (ECOC)*. VDE, 2016, pp. 1–3.

[29] 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.

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

[31] 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.

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

[33] D. Maclaurin, D. Duvenaud, and R. P. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *Proc. 32nd Int. Conf. Machine Learning*, 2015.

[34] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, Feb. 2012.

[35] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1223–1231.

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

[37] 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.

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

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

[40] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[41] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[42] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *arXiv preprint arXiv:1602.01783*, 2016.

[43] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *arXiv preprint arXiv:1611.05397*, 2016.

[44] F. D. Calabrese, L. Wang, E. Ghadimi, G. Peters, and P. Soldati, "Learning radio resource management in 5G networks: Framework, opportunities and challenges," *arXiv preprint arXiv:1611.10253*, 2016.

[45] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, 2016.

[46] W. Zaremba and I. Sutskever, "Reinforcement learning neural Turing machines," *arXiv preprint arXiv:1505.00521*, 2015.

[47] 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.

[48] M. Zhao, F. Adib, and D. Katabi, "Emotion recognition using wireless signals," in *Proc. 22nd Annu. Int. Conf. Mobile Computing and Networking*. ACM, 2016, pp. 95–108.