Machine Learning Autoencoder Applied to Communication Channels

Eduardo Dadalto Camara Gomes*
*Institut Supérieur de l'Aéronautique et de l'Espace (ISAE-SUPAERO), Université de Toulouse, 31055 Toulouse, FRANCE
Email: eduardo.dadalto-camara-gomes@student.isae-supaero.fr

***Abstract*—Communication channel error correction is key to enable digital critical communication systems to work efficiently. Although the optimal decoder for noisy channels, the maximum a posteriori (MAP) decoder, has already a mathematically proven optimal error probability minimization; its implementation cripples system applicability, since it introduces a large delay for large code words. In this context, a deep neural network (DNN) is proposed to optimize the channel with an end-to-end autoencoder. The DNN autoencoder outperforms a MAP decoder form a given encoder in terms of delay, because of the first's *one-shot* capability once trained.**

***Index Terms*—communication system, machine learning, autoencoder, channel decoding, maximum a posteriori (MAP) decoder**

I. Introduction

A. Motivation

A point to point communication channel is a system in which two terminals exchange information through a noisy channel as Fig. 1 illustrates. As a result of channel imperfections, Shannon theorized in [1] the ultimate reliable data rate that can be transmitted through a communication system with an arbitrarily small probability error.

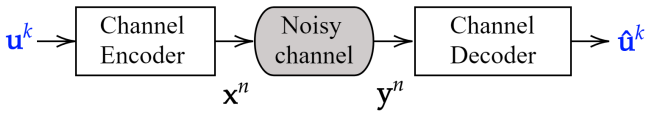

Fig. 1. Diagram of a simplified communication system.

Since then, the research community in digital communication developed a series of algorithms to minimize error probability over a channel. In practice, the bit error rate (BER), an approximation of the probability of error, is targeted. However, the challenge of finding an *efficient* solution i.e., with both low latency and low error probability, for low signal to noise ratio (SNR) channels, remains. Low latency and high bandwidth wireless communication are key to critical systems, such as airplanes, satellites, cellular communication and 5G operations. The latter was studied by F. D. Calabrese et al. in [2] which demonstrated that individual based radio resource management (RRM) algorithms were outperformed by a general learning framework, resulting in significant expense reductions, while increasing performance of the network.

Based on this vision, Tim O'Shea and Jakob Hoydis [3] pertinently noted that traditional algorithms in the field have foundations in probability theory e.g. maximum a posteriori (MAP) rule, maximum likelihood decoder (MLD) and turbo codes. Hence, they are usually built on top of mathematically convenient models. Even though they are theoretically optimal error corrector codes, these models often do not account for all the real system's imperfections, which leads to errors when implemented in practice.

As opposite to structured algorithms, machine learning (ML) algorithms do not require rigidly designed models and can take non-linearities effortlessly into account. These characteristics make these algorithms candidates for being used as channel decoder. Moreover, with ML based channel encoder and channel decoder, the design of communication systems as independently working blocks becomes obsolete, as a deep neural network (DNN) is able to actuate end-to-end in the system. As a result, an optimized autoencoder with a stochastic layer that models channel's imperfections can substitute the block based representation. Therefore, ML based communication systems could be a better representation of realistic systems and could optimize information transmission of different blocklengths and with low decoding latency, resulting ultimately in gain of bandwidth over standard methods. Thereby, ML algorithms are cardinal for state of the art communication applications.

B. Related work

Recently, a significant amount of work in radio communication theory has emerged, introducing ML elements to the communication system. O'Shea et al. in [4] developed a channel autoencoder with optimized impairment and regularization layers to emulate channel impairments. They studied this architecture over an additive white Gaussian noise (AWGN) channel, founding "some promising initial capacity" for this scheme. In their research, results in terms of BER over SNR for a DNN based autoencoder and for a convolutional neural network (CNN) based autoencoder were treated. They used a range of SNR - from -10dB to 15dB - with QPSK and QAM16 modulation as benchmarks. This analysis was conducted for a binary input message.

T. Gruber et. al. in [5] proved that a deep learning-based channel decoder could actually learn a decoding algorithm rather than just being a simple classifier. They introduced code words that were not been used in the training set, and the trained NN was able to correctly decode it. They also observed that structured codes are easier to learn than unstructured ones.

NN for structured codes are able to generalize to the full codebook even if they have not seen all the training examples. They trained the NN for very short blocklengths ($N \leq 64$) in order to compare with MAP decoding performance.

C. Problem statement

In this work, we will implement a ML autoencoder for a BSC that performs similar to the MAP decoder in real applications for a range of SNR from -10dB to 10dB. In its most simple form, a channel autoencoder includes an encoder, a noisy channel and a decoder. Using state-of-the art DNN algorithms to find the best solution of the problem, this work aims to contribute to set up a higher standard in terms of performance in bit-error correction and reduced delay for digital communication applications. In a near future, this disruptive methodology for error correction using ML could replace mathematically optimal decoders which are the current guideline.

D. Notation

Throughout the work, vectors will be written in bold font weight and may have a subscript which indicates their layer in a NN and may have a superscript which indicates their size length. In order to homogenize notation, for the encoding phase of the communication system, $\mathbf{u}^k$ represents an input *source message* and $\mathbf{x}^n$ its correspondent code word. For the decoding portion, $\mathbf{y}^n$ represents the output of the noisy channel and $\hat{\mathbf{u}}^k$ the estimated source message. These definitions are illustrated in Fig. 1.

In addition, every indispensable concept is introduced either in italics or is defined by a commonly used abbreviation present in others scientific works.

II. Theoretical Background

A. Channel coding

Consider the communication system illustrated in Fig. 1. The left portion of the chain, the transmitter, wants to communicate a message $\mathbf{u}$ through a noisy channel. The right portion of the chain, the receiver, may interpret this possible corrupted stream of bits into the original message. Usually this prediction $\hat{\mathbf{u}}$ carry errors which we aim to minimize its probability of occurring. In this scenario, the theory of channel coding discuss possible solutions.

In particular, linear block codes represented by a pair (n, k) called *code name* will be implemented. Where k is the length of a source message and n is the length of a code word. Briefly, a linear encoder will index a message $\mathbf{u}^k \in \mathcal{U} \subseteq \{0,1\}^k$ to a code word $\mathbf{x}^n \in \mathcal{X} \subseteq \{0,1\}^n$. This mapping is done by the *generator matrix* $\mathbf{G}$.

$$\mathbb{E} : \mathbf{u} \mapsto \mathbf{uG} \tag{1}$$

Then, the objective of the receiver is to estimate the original message with an empirical error probability, or a BER, defined as

$$P_{eb} = \frac{1}{M}\sum_{u} Pr(\hat{u} \neq u) \tag{2}$$

where M is the total amount of bits transmitted, u represents one bit transmitted and $\hat{u}$ its estimation.

B. Maximum a posteriori decoder

The concept of a MAP decoder algorithm for sequences, is choosing a message which maximizes the a posterior probability of the corrupted code word y received by the decoder [6]. This algorithm is known for its optimal error correction capability for white noise interference. This algorithm is also referred as the Viterbi decoder [7].

Mathematically, we want to maximize the average probability of making a correct decision when analysing the channel output $\mathbf{y}$, which is written as

$$Pc := Pr[\mathbf{X} = f(Y)] \tag{3}$$

where $x \sim X$ and $y \sim Y$ are multivariate random variables. We derive below the decoder that maximizes P_c.

$$\Pr[X = f(Y)] = \sum_{f(y)y \in \mathcal{X}\times\mathcal{Y}} P_{XY}(f(y)y) \tag{4}$$

$$= \sum_{y\in\mathcal{Y}} P_Y(y)\, P_{X|Y}(f(y)|y) \tag{5}$$

The optimal decoder is finally given by

$$f(y) = \arg\max_{x\in\mathcal{X}} P_{X|Y}(x|y) \tag{6}$$

$$= \arg\max_{x\in\mathcal{X}} P_X(x) P_{Y|X}(y|x). \tag{7}$$

However, the MAP decoder is practically nonviable to implement [8]. Thus, its variants cited in this paragraph are better suited for practical cases. The Log-MAP is an optimal decoder, having equivalent performance to the MAP decoder. While the Max-Log-MAP is a suboptimal decoder and will not be treated in this paper. these variants are widely used as decoders for turbo codes [9]. Refer to [8] for specific details in the derivation of the Log-MAP algorithm. Furthermore, the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm is a bit-wise MAP rule with lower complexity widely used in practice, even for long data packets.

Turbo decoding using Log-MAP decoder for an additive white Gaussian noise (AWGN) channel and for a BSC was studied in [10]. They analyzed the BER for small SNR and concluded that the results are sensible to SNR estimation of the real channel, since the estimation of this parameter is required for the metric calculation of the algorithm [8]. It means that if the real SNR is smaller, the decoder will not perform well. Hence, for channel characteristics that change over time, the application of a MAP algorithm is debatable. This conclusion serve as motivation for ML autoencoders, which could adapt to different SNR conditions, outperforming the MAP decoder for realistic applications.

C. Neural network basics

Neural networks (NN) are a set of connected *neurons* able to approximate any kind of function through an architecture composed of several single processing units (neurons) connected together forming a network. They are defined by [11] as parallel distributed, learning- and self-organizing information processing systems.

In the context of learning algorithms implemented by a NN, the key issue is to find a suitable architecture that delivers the best results. To define and build this infrastructure, the activation function, the loss function and the structure must be decided. Nielson [12] in his book explores a vast amount of NN architectures and how they work.

Multi-layer feed forward neural network (MLNN) as shown in Fig. 2 will be applied in this work. They are known for the universal approximation property [11] and for the versatility of increasing the number of layers, creating a DNN able to undertake complex classification problems with low classification error and low dimensionality. In more general terms, a MLNN with L layers, also called *depth*, and parameter θ is a mapping of an input $\mathbf{y}^n \in \mathbb{R}^n$ to an output $\hat{\mathbf{u}}^k \in \mathbb{R}^k$ through L iterative steps. For a fully-connected network, each layer vector is calculated iteratively in the forward propagation. Each neuron is composed of a linear combiner and an activation function defined as

$$f_l\left(\mathbf{r}_{l-1};\theta_l\right) = \sigma\left(\mathbf{W}_l\mathbf{r}_{l-1} + \mathbf{b}_l\right) \tag{8}$$

where $\mathbf{W}_l \in \mathbb{R}^{N_l \times N_{l-1}}$ is the weight matrix between layers $l-1$ and l, $\mathbf{r}_{l-1} \in \mathbb{R}^{N_{l-1}}$ is a vector containing the hidden layer $l-1$ values, $\mathbf{b}_l \in \mathbb{R}^{N_l}$ is the bias vector and σ is the activation function [3].

Two common activation functions are the rectified linear unit (ReLU) and the sigmoid function. Both will be applied in the NN treated in this paper. Their mathematical expressions are shown below.

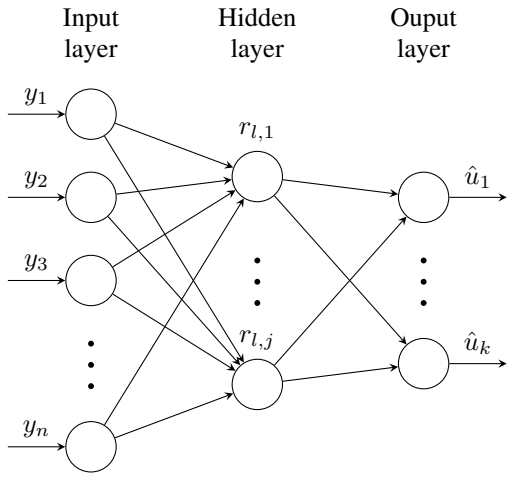

Fig. 2. MLNN representative diagram, where $\mathbf{y}^n$ is the input vector, $\mathbf{r}_l^j$ is a hidden layer vector and $\hat{\mathbf{u}}^k$ is the output vector.

$$\text{ReLU: } \sigma_1(x) = max\{0, x\} \tag{9}$$

$$\text{Sigmoid: } \sigma_2(x) = \frac{1}{1+e^{-\lambda x}} \tag{10}$$

where $x \in \mathbb{R}$ and $\lambda > 0$.

For a classification problem in communication, the *categorical cross-entropy* loss function $l(\mathbf{p},\mathbf{q}) : \mathbb{R}^{N_L} \times \mathbb{R}^{N_L} \mapsto \mathbb{R}$ defined in (11) is most common. It is derived from the log-likelihood of a training set where q_j is the estimated probability of the outcome j and p_j is the true probability. Basically, the function measures the dissimilarity between p_j (what you expected) and q_j (what you obtained) [13] where $j = 1, ..., N_L$.

$$l(\mathbf{p},\mathbf{q}) = -\sum_j p_j \log\left(q_j\right) \tag{11}$$

With all these elements set, training the neural network to calculate an accurate weight matrix $\mathbf{W}$ for forward propagation requires a labeled *training data set*. This set is composed of pairs $(\mathbf{y}_i^n, \mathbf{u}_i^k)$, where $i = 1, ..., S$ and S is the number of training sets. It matches the input and its respective desired output. The objective is to minimize the overall loss function defined in (12) in terms of the parameter θ [3].

$$L\left(\theta\right) = \frac{1}{S}\sum_{i=1}^{S} l\left(\mathbf{u}_i^k, \mathbf{y}_i^n\right) \tag{12}$$

This problem is an optimization problem and can be solved through different algorithms. For NN, an efficient way of computing this minimization is implementing the back-propagation (BP) algorithm. BP is classified as a supervised learning algorithm that is able to optimize a function based on some parameter and is highly parallelizable in computational terms [11] [12].

One of the greatest advantage of NN is its ability to generalize the training set, delivering the right results even for input data not contained in the training set. With the weight matrix trained, the NN finds the correct output values for unseen inputs. In order to validate this principle, a *validation data set* is used and the loss is mesured. Based on this value, we can evaluate the NN overall accuracy.

For implementing all the algorithms, Keras, a high-level python API for ML, will be used together with TensorFlow 1.13.1, a ML python friendly open-source library [14] [15]. Both are available on a free license terms.

D. Autoencoders

Autoencoders based on DNN can learn an appropriate correspondence between input source messages and estimated source message output through different noisy channels. The statistics of the channel's noise is shown in [5] to be irrelevant, since the NN extract it during training phase. The autoencoder applicability is compelling because the weights computed during training can be separated in two groups, one correspondent to the nodes before the channel and one posterior to the channel. As a consequence, we are able to

design a non linear encoder and its correspondent non linear decoder with ease. This new representation has ultimately far more possibilities than conventional encoding patterns.

Mathematically, the non linear autoencoder's encoding function $\phi : \{0,1\}^k \mapsto \{0,1\}^n$ maps the message input to its code word.

$$x^n = \phi(u^k) \tag{13}$$

Whilst autoencoders are more complex dimensional wise compared to NN decoders, the readily access to graphical processing units (GPUs) contributed to the feasibility of this technique, tackling the *curse of dimensionality* challenge in ML. This type of hardware made possible the approach of the layer-by-layer supervised training by stochastic gradient descent optimization applied in this work. [16].

A common architecture of DNN autoencoders is shown in Fig. 3 where the encoder and the decoder are composed of dense layers of different widths. In addition, a noise layer and a rounding layer acts on the forward propagation but not on the BP, since their activation functions are not differentiable. More details on the autoencoder's architecture is discussed is section III.

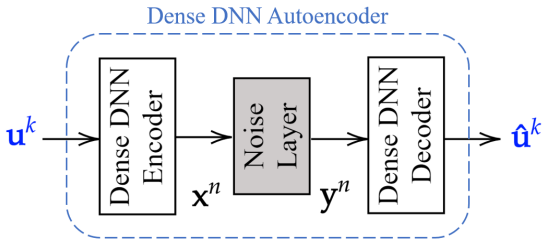

Fig. 3. Representation of a DNN autoencoder composed of dense layers.

III. Implementation and Methodology

This section explains the main implementation procedures in order to replicate the results and understand in depth the methodology behind the work. For organizational purposes, the section is divided into two subsections, where the first concerns the experiments relating to the ML based decoders and the second treats the implementation of the ML based autoencoders.

A. Decoder

The linear encoding matrix $\mathbf{G}_{8\times16}$ is shown below.

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Introduce encoding equations

1) Array Decoder: Architecture, Training parameters, cite choice of p train [17], number of epochs, Talk about noise layer and its removal in prediction,

2) One-hot Decoder: Architecture, Training parameters, Explain what is a One hot decoding, talk about the matrix transformation from one hot to messages, p train equals 0, number of epochs

B. Autoencoder

Architecture, Training and simulation parameters, procedure to find a M epoch suitable, choices of p train, Talk about NN separation,

IV. Results and Analysis

A. Decoders

BER Curves from both decoders, analyse why the number of epochs is so different.

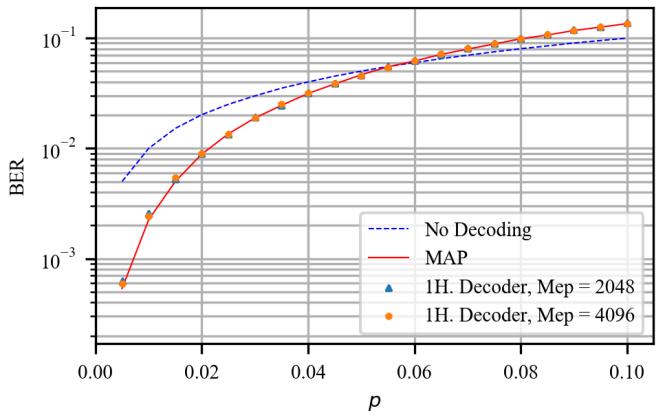

Fig. 4. Test caption to test image.

B. Autoencoder

BER Curves of simulation varying p, best choice of p, analyse autoencoder with best choice of p

C. Decoding time analysis

compare time in decoding between map, autoencoder and a decoder.

V. Conclusions

References

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

[2] 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.

[3] O'Shea, Tim & Hoydis, Jakob, "An Introduction to Machine Learning Communications Systems", 2017.

[4] T. O'Shea, K. arra, T. C. Clancy, "Learning to Communicate: Channel Auto-encoders, Domain Specific Regularizers, and Attention", 2016.

[5] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning based channel decoding," accepted for CISS 2017, arXiv preprint arXiv:1701.07738, 2017.

[6] W. Alexander, H. Peter & W. Norbert, "Turbo-Decoding Without SNR Estimation",IEEE Communications Letter, vol. 4, no. 6, pp. 193-195, 2000.

[7] A. J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm", IEEE Transactions on Information Theory, 13 (2): 260–269, 1967.

[8] Robertson, P. , Hoeher, P. and Villebrun, E. (1997), Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding. Eur. Trans. Telecomm., 8: 119-125. doi:10.1002/ett.4460080202.

[9] J. Hagenauer,P. Robertson, L. Papke: iterative ("Turbo") decoding of systematic convolutional codes with the MAP and SOVA algorithms. In: ITG-Fachbericht 130. October 1994, p. 21-29.

[10] M. Jordan and R. Nichols, "The effects of channel characteristics on turbo code performance," in Proc. Milcom'96, McLean, VA, Oct. 1996, pp. 17–21.

[11] I. Mohamed, "Applications of neural networks to digital communications - a survey", 1997.

[12] M. Nielsen, *Neural Networks and Deep Learning*.

[13] M. Kevin, "Machine Learning: A Probabilistic Perspective", MIT, ISBN 978-0262018029, 2012.

[14] M. Abadi, A. Agarwal, P. Barham, et al., "TensorFlow: Large-scale machine learning on heterogeneous systems", 2015. Software available from tensorflow.org.

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

[16] 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, July 2006.

[17] M. Benammar and P. Piantanida, "Optimal training channel statistics for neural-based decoders," in Asilomar, Oct 2018, pp. 2157–2161.