# Machine Learning Autoencoder Applied to Communication Channels

## (Final report)

Eduardo Dadalto Camara Gomes*, Meryem Benammar†

*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
†Institut Supérieur de l'Aéronautique et de l'Espace (ISAE-SUPAERO), Université de Toulouse, 31055 Toulouse, FRANCE
Email: meryem.benammar@isae-supaero.fr

*Abstract*—Communication channel error correction is key to enable digital critical communication systems to work efficiently. The maximum a posteriori (MAP) decoder is proven to be the mathematically optimal solution for the problem. However, its implementation cripples the system applicability. This is because the MAP rule introduces a large delay for long code words. In this context, a deep neural network (DNN) is proposed to optimize the channel with an end-to-end autoencoder. The trained DNN autoencoder has a *one-shot* capability which outperforms a MAP decoder for a given encoder in terms of delay. ONE PHRASE FOR RESULTS

*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.
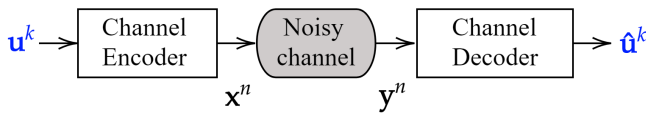


Fig. 1. Diagram of a simplified communication system.

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 $-10$dB to $15$dB - 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 $-10$dB to $10$dB. 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* $\mathbf{G}$.

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

The objective of the receiver is to estimate the original message. The empirical error probability, or a BER, in this process is 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.

The Hamming distance $d_H : \mathcal{X} \mapsto \mathbb{Z}$ is a metric in the word subset which in the context of linear coding is equal to equation 3. Noteworthy for the MAP rule, a non zero Hamming distance indicates that there is a block error, and ultimately at least a binary error in the received code word.

$$d_H(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} y_i \text{ XOR } x_i \tag{3}$$

### B. Binary Symmetric Channel

Briefly, a BSC is a noisy channel with a binary input $X \in \{0, 1\}$ on the left and a binary output $Y \in \{0, 1\}$ on the right. They are linked through a stochastic model characterized by a crossover probability of value $p$ as illustrated in Fig. 2. Typically, $p$ is smaller than $0.10$.
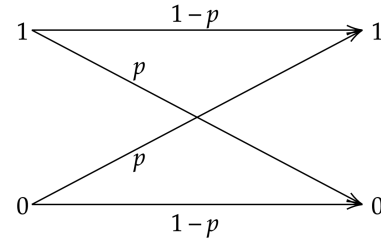


Fig. 2. Illustration of a binary symmetric channel with crossover probability $p$.

A BSC could be represented mathematically through conditional probabilities as follows.

$$Pr[Y = 0 | X = 0] = 1 - p$$
$$Pr[Y = 0 | X = 1] = p$$
$$Pr[Y = 1 | X = 0] = p$$
$$Pr[Y = 1 | X = 1] = 1 - p$$

### C. 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 **y**, which is written as

$$Pc := Pr[\mathbf{X} = f(Y)] \tag{4}$$

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

$$\Pr[X = f(Y)] = \sum_{f(y)y \in \mathcal{X} \times \mathcal{Y}} P_{XY}(f(y)y) \tag{5}$$

$$= \sum_{y \in \mathcal{Y}} P_Y(y) P_{X|Y}(f(y)|y) \tag{6}$$

The optimal decoder is finally given by

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

$$= \arg \max_{x \in \mathcal{X}} P_X(x) P_{Y|X}(y|x). \tag{8}$$

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.

### D. 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. 3 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 $\theta$ 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

$$f_l(\mathbf{r}_{l-1}; \theta_l) = \sigma(\mathbf{W}_l \mathbf{r}_{l-1} + \mathbf{b}_l) \tag{9}$$

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 $\sigma$ is the activation function [3].

Two common activation functions are the rectified linear unit (ReLU) and the sigmoid function. Both will be applied to the DNNs treated in this paper. In addition, the Softmax activation function is commonly used in multi classification problems and will be used to the One-hot decoding DNN architectures. Their mathematical expressions are shown below.

$$\text{ReLU: } \sigma_1(x) = max\{0, x\} \tag{10}$$

$$\text{Sigmoid: } \sigma_2(x) = \frac{1}{1 + e^{-\lambda x}} \tag{11}$$

$$\text{Softmax: } \sigma_3(\mathbf{z}) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{12}$$

where $x \in \mathbb{R}$, $\lambda > 0$, $i = 1, ..., K$ and $\mathbf{z} = (z_1, ..., z_K) \in \mathbb{R}^K$.

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 (13) is most common. It is derived from the
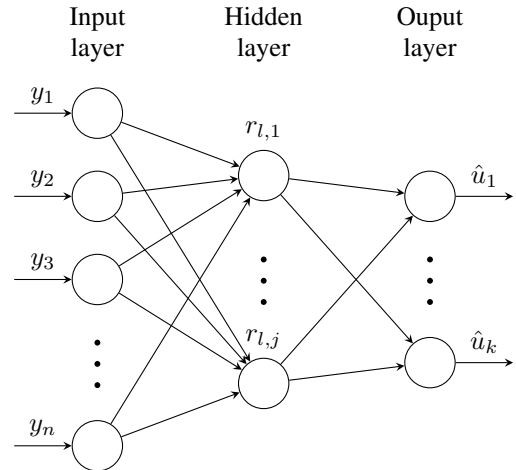


Fig. 3. MLNN representative diagram, where $\mathbf{y}^n$ is the input vector, $\mathbf{r}_l^j$ is a hidden layer vector and $\hat{\mathbf{u}}^k$ is the output vector.

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 $j = 1, ..., N_L$.

$$l(\mathbf{p}, \mathbf{q}) = -\sum_j p_j \log(q_j) \qquad (13)$$

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 (14) in terms of the parameter $\theta$ [3].

$$L(\theta) = \frac{1}{S} \sum_{i=1}^{S} l\left(\mathbf{u}_i^k, \mathbf{y}_i^n\right) \qquad (14)$$

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 measured. 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.

### E. 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.

$$x^n = \phi(u^k) \qquad (15)$$

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. 4 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.
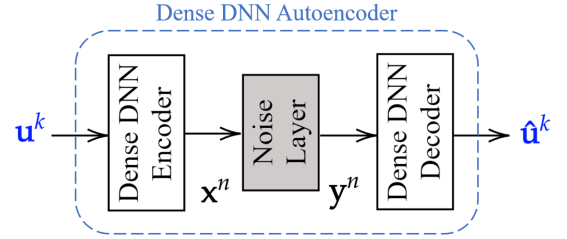


Fig. 4. Representation of a DNN autoencoder composed of dense layers.

## 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 four subsections. First, the predictions and error corrections statistical relevance are justified through the Monte Carlo simulations conducted. Second, the MAP algorithm for a linear code is shown. Then the experiments relating to the ML based decoders are assessed. Finally the implementation of the ML based autoencoder is explained.

### A. Monte Carlo Simulation

For calculating the BER versus the crossover probability $p$ with confidence over the results, a Monte Carlo simulation was elaborated. This stochastic computational algorithm uses pseudo-random sampling to solve different problems. Hence, we can work with a wide variety of transmission scenarios.

Applied to the context of this research, the messages are repeatedly randomly generated for a specific BSC of crossover probability $p$. They are coded by a linear block code or by a DNN encoder and then transmitted through the channel. Their information arrives corrupted by noise in the decoder and is subsequently partially corrected by a decoder. Finally the average BER is calculated and plotted. The interval of the horizontal axis is in between $0$ and $0.10$.

Every channel is tested with $100000$ messages. As a consequence of the *law of large numbers*, the empirical sampling of the BER represents sufficiently well the real distribution.

## B. MAP Rule

The MAP algorithm as cited previously, has a high time complexity ($\mathcal{O}(2^k)$), since it computes the Hamming distance between the received code word and every code word from the code book. For more details, the pseudo code of the algorithm is available below.

---

**Algorithm 1** MAP rule for BSC and linear block code.

---

**Input:** received block $\mathbf{y}^n \in \{0,1\}^n$, code word set $\mathcal{X}$ and generator matrix $G_{k \times n}$.

**Output:** message estimation $\hat{\mathbf{u}}^k \in \{0,1\}^k$.

**procedure** MAP DECODER($y, \mathcal{X}, G$)
    $p \leftarrow$ channel crossover probability
    **for** i in $range(2^k)$ **do**
        $distances[i] \leftarrow d_H(\mathbf{y}, word[i] \in \mathcal{X})$
    $\hat{\mathbf{x}} \leftarrow argmin(distances)$
    $\hat{\mathbf{u}} \leftarrow \hat{\mathbf{x}}G^{-1}$ **return** $\hat{\mathbf{u}}$

---

The linear encoding matrix $\mathbf{G}_{8 \times 16}$ was used for the encoding of the messages. Its dimension is equal to the linear block code name represented by the double $(n, k) = (16, 8)$. Thus, the *rate* of the code is equal to $0.5$. The linear block name and = rate is constant throughout the work.

$\mathbf{G} =$

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

To establish a reference model to use as comparison base to the DNN models and MAP decoder, the BER results for a scenario without codification was calculated. The ensemble of the no decoding curve and the MAP algorithm curve of Fig. 5 are replicated for comparison purposes with the methods of interest.

## C. Decoder

This section is divided in two for better identify the differences between the implementation of the array DNN decoder and the one-hot DNN decoder.

*1) Array Decoder:* The architecture of the DNN necessary to achieve a great performance in terms of BER is available in table I. What should be remarked is the first layer of the DNN. A *Lambda* layer is a layer in which the activation operations are user defined and the parameters are non-trainable. Thus, we were able to define a layer that act as a BSC to force our DNN to learn its effects in the training data. The fact that the parameters are non-trainable means that the noise layer is applied in forward propagation only.
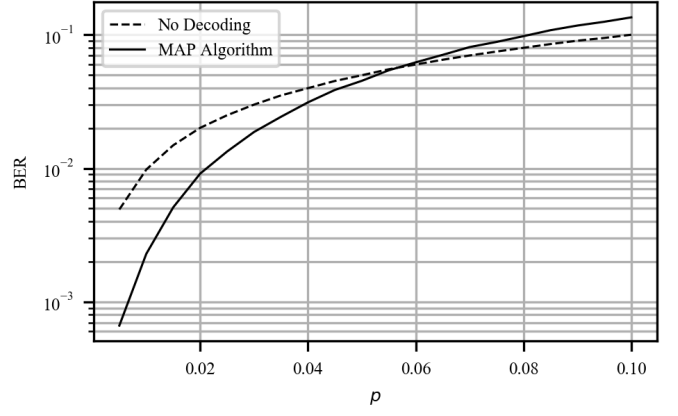


Fig. 5. No decoding and MAP algorithm performance in terms of BER versus channel crossover probability.

TABLE I
DNN ARRAY DECODER ARCHITECTURE.

| Channel | Lambda: $\mathbf{x} \oplus$ noise |
|---------|-----------------------------------|
| | Dense: 128, activation: ReLU, input size: 16 |
| Decoder | Dense: 64, activation: ReLU |
| | Dense: 32, activation: ReLU |
| | Dense: 8, activation: Sigmoid |
| **Total parameters:** 12776 | |

The choice of the crossover probability of the training channel, $p_t$, was based in the work of [17] which shows $p_t = 0.07$ to be the best in terms of decoding precision after the DNN is trained. In addition, a deeper NN showed best results in [3] and were adopted to solve the problem. The training parameters are displayed in table II.

Trainin set for array decoding.

The choices of training parameters is a mix between what other works in literature uses and a hyper parametric analytic approach, where several trials lead to the decision. First, the binary cross-entropy as loss function is widely used in literature and showed to converge to the desired result consistently during tests. For the optimizer, the *Adam* built-in Keras optimizer was chosen. Adam optimizer algorithm is similar to a classic stochastic gradient descent algorithm, but computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients [15]. Next, a batch size matching the size of the training set which is sufficiently small to still operate in the regime of small batch was chosen. In [18], the authors demonstrated better performance for a NN training with stochastic gradient descent and its variants operating in a small-batch regime, usually between 32-512 data points. Finally, the necessary number of epochs to training is sensitive to the problem, since it impacts directly the training computational cost. As a result, we tried several training epoch sizes and found $2^{16}$ to have precise results and be computationally viable.

*2) One-hot Decoder:* The architecture of the one-hot decoder is much simpler when compared to the array decoder.

| Loss func. | Optimizer | N. Epochs | Batch Size |
|---|---|---|---|
| Binary cross-entropy | Adam | $2^{16}$ | 256 |

TABLE III
DNN ONE-HOT DECODER ARCHITECTURE.

| Decoder | Dense: 256, activation: Softmax, input size: 16 |
|---|---|
| **Total parameters:** 4352 | |

TABLE V
DNN ARRAY AUTOENCODER ARCHITECTURE.

| Encoder | Dense: X, activation: ReLU, input size: 8 |
|---|---|
| | Dense: X, activation: ReLU |
| | Dense: 16, activation: Sigmoid |
| Channel | Lambda: $Round(\mathbf{x})$, input size: $n$ |
| | Lambda: $\mathbf{x} \oplus$ noise |
| Decoder | Dense: X, input size: 16 |
| | Dense: X, activation: ReLU |
| | Dense: 8, activation: Sigmoid |
| **Total parameters:** XX | |

TABLE VI
DNN ARRAY AUTODECODER TRAINING PARAMETERS.

| Loss func. | Optimizer | N. Epochs | Batch Size |
|---|---|---|---|
| Binary cross-entropy | Adam | $2^{14}$ | 256 |

It contains only one layer of 256 units which maps the input of size 16 to the $2^8 = 256$ possible messages. Hence, this characterizes a multi-class classification problem, meaning that the chosen activation function is the Softmax (Eq. 12). This training technique is known in the literature as *one-hot*.

The denomination one-hot comes from the output vector which is usually a long vector of zeros with one element equals to one. This *hot* vector element indicates which class the NN has classified the input. In this context, which message encoded in 256 different classes the code word refers to. Basically, a message is mapped into a class and a class only.

A straight forward consequence of this shallower NN is the number of training parameters almost three times less than the array decoder. Because of that, we could expect a lower training time. Also, we observed that the number of training epochs is substantially smaller for the one-hot decoder. It was necessary only $2^{14}$ epochs to achieve the results for the same batch size of 256 and Adam optimizer.

Despite the NN complexity benefits of the one-hot decoder, it is required to transform the predicted vector into a message. Consequently, an additional operation is needed. To transform from a one-hot vector to a message, a ordered collection of the messages was stored in memory and the correspondent message is determined by a $argmax$ operation in the one-hot vector, which identifies which message is the most likely to have been transmitted through the channel.

Trainin set for one-hot decoding, say how was pseudo-randomly generated, sa that the stastistic was checked to have 1/2 of 1s and 1/2 of 0s.

The one-hot decoder was trained both with and without a noise layer. The model without a integrated channel converged faster to the solution. Tables III and IV resume the architecture and training variables, respectively.

However, we still need a decoding matrix.... Introduces autoencoder that could train without G

### D. Autoencoder

Architecture, Training and simulation parameters, procedure to find a M epoch suitable, choices of p train, Talk about NN separation, no necessity for G, a slight different trainin approach (training data and labels are identical), same channel rate (16 units) for a message of 8 bits,

## IV. RESULTS AND ANALYSIS

Explain any different training procedure

### A. Decoders

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

We can infer from figures 6 and 7 that both the array decoder and the one-hot decoder are able to learn a decodification function. Moreover, its performances replicate the MAP algorithm's decoding capability. Hence, DNN decoders could substitute the mathematically optimal MAP decoder for a stochastic channel, without compromising the bit error probability.

TABLE IV
DNN ONE-HOT DECODER TRAINING PARAMETERS.

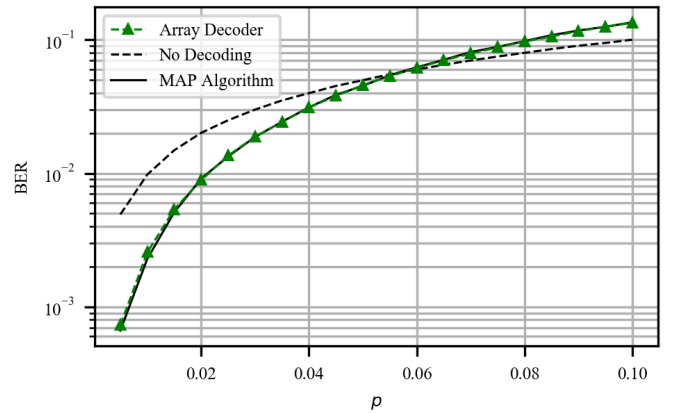| Loss func. | Optimizer | N. Epochs | Batch Size |
|---|---|---|---|
| Binary cross-entropy | Adam | $2^{14}$ | 256 |



Fig. 6. Array decoding BER performance. DNN array decoder trained with a channel crossover probability error of $p_t = 0.07$.
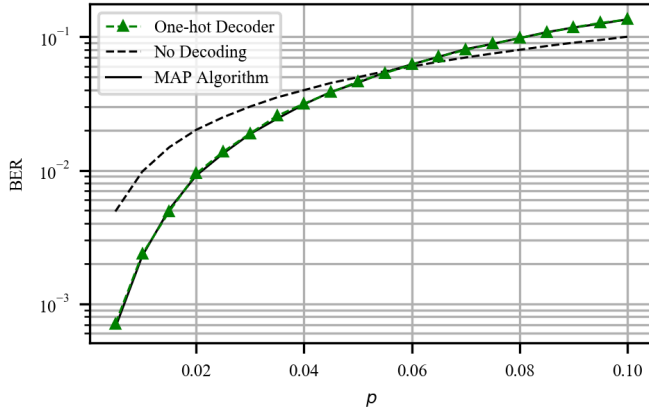
Fig. 7. One hot decoding BER performance. DNN one-hot decoder trained with a channel crossover probability error of $p_t = 0$.

## B. Autoencoder

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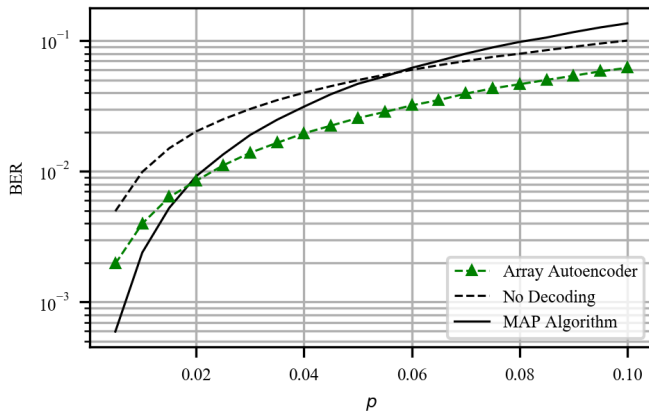


Fig. 8. Array autoencoder BER performance. DNN array autoencoder trained with a channel crossover probability error of $p_t = 0.03$.

## C. Decoding time analysis

In order to analyze the delay in decoding for each technique studied, a monte carlo simulation was conducted. There were selected 20 different BSC channels. For each channel, 100000 messages were transmitted and decodified for each decoder. This process was timed and repeated several times. In the end, an average time and standard deviation of the results were recorded. A normalization for the average MAP decoding time was done. The results are available in the table VII.

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

## V. CONCLUSIONS

Talk about feasibility of decoders, improved performance of the channel delay with DNN decoders, challenges in implementing this architecture in real life, challenges in training

the NNs, Future work, what else could be done with NN in error correction,

TABLE VII
DECODING TIME COMPARISON BETWEEN THE MAP ALGORITHM AND THE DNN DECODERS AND AUTOENCODERS. THE DATA IS NORMALIZED TO THE MEAN MAP ALGORITHM DECODING TIME.

| MAP | Array Decoder | One-hot Decoder |
|---|---|---|
| $1.00 \pm 0.02$ | $0.74 \pm 0.03$ | $0.76 \pm 0.02$ |
| Array. Autoencoder | | One-hot Autoencoder |
| $0.00 \pm 0.00$ | | $0.00 \pm 0.00$ |

## REFERENCES

[1] C. E. Shannon, "A mathematical theory of communication," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 5, pp. 3–55, Jan. 2001.
[2] F. D. Calabrese, L. Wang, E. Ghadimi, G. Peters, and P. Soldati, "Learning radio resource management in 5g networks: Framework, opportunities and challenges," *CoRR*, vol. abs/1611.10253, 2016.
[3] T. J. O'Shea and J. Hoydis, "An introduction to machine learning communications systems," *CoRR*, vol. abs/1702.00832, 2017.
[4] T. J. O'Shea, K. Karra, and T. C. Clancy, "Learning to Communicate: Channel Auto-encoders, Domain Specific Regularizers, and Attention," *arXiv e-prints*, Aug. 2016.
[5] D. Goldin and D. Burshtein, "Performance Bounds of Concatenated Polar Coding Schemes," *arXiv e-prints*, Oct. 2017.
[6] E. Worm, S. Member, P. Hoeher, S. Member, and N. Wehn, "Turbo-decoding without snr estimation," *IEEE Communications Letters*, pp. 193–195, 2000.
[7] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. IT-13, pp. 260–269, April 1967.
[8] P. Robertson, P. A. Hoeher, and E. Villebrun, "Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding.," *European Transactions on Telecommunications*, vol. 8, no. 2, pp. 119–125, 1997.
[9] H. J., R. P., and P. L., "iterative turbo decoding of systematic convolutional codes with the map and sova algorithms," pp. 21 – 29, 10 1994.
[10] M. Jordan and R. Nichols, "The effects of channel characteristics on turbo code performance," pp. 17 – 21 vol.1, 11 1996.
[11] M. Ibnkahla, "Applications of neural networks to digital communications-survey," *Signal Processing*, vol. 80, pp. 1185–1215, 07 2000.
[12] M. A. Nielsen, "Neural networks and deep learning," 2018.
[13] K. P. Murphy, *Machine learning : a probabilistic perspective*. Cambridge, Mass. [u.a.]: MIT Press, 2013.
[14] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *CoRR*, vol. abs/1603.04467, 2016.
[15] F. Chollet *et al.*, "Keras." https://keras.io, 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, 2006. PMID: 16764513.
[17] M. Benammar and P. Piantanida, "Optimal training channel statistics for neural-based decoders," in *52nd Asilomar Conference on Signals, Systems, and Computers, ACSSC 2018, Pacific Grove, CA, USA, October 28-31, 2018* (M. B. Matthews, ed.), pp. 2157–2161, IEEE, 2018.
[18] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *CoRR*, vol. abs/1609.04836, 2016.