Vanilla Recurrent Neural Networks for Convolutional Decoding

Henri Bellamy
CentraleSupélec
henribellamy@icloud.com

***Abstract*—In this paper, we propose a simple neural network for convolutional decoding. The proposed model achieves performance close to the theoretical limit in terms of decoding, i.e., bit error rate (BER), while maintaining a relatively low decoding time. The enhanced decoding speed is due to the use of a vanilla recurrent neural network (Vanilla RNN) instead of more complex architectures, such as gated recurrent units (GRUs) or long short-term memory (LSTMs), which are typically used in deep convolutional decoding. To improve decoding performance, several hyperparameters are analyzed. In addition, different training techniques are explored, such as Gaussian annealing. Once trained, the model exhibits strong generalization capabilities, effectively adapting to a wide range of SNR values and varying block lengths.**

Deep learning, recurrent neural network, convolutional code

I. Introduction

Error correction codes are essential in modern communication systems, ensuring reliable data transmission over noisy channels. Over the past few decades, significant advancements have been made, with convolutional codes playing a pivotal role in enhancing performance. Today, these codes have brought us close to the Shannon limit, enabling efficient data transfer even under challenging conditions, such as high-speed and mobile communications. Their impact can be seen in fields like wireless communication, satellite systems, and 5G networks, where minimizing bit errors is crucial for optimal performance.

In order to decode a convolutional code, the distance between the noisy received code sequence and the estimated code sequence has to be minimized. This is usually performed by the Viterbi algorithm which determines the shortest path in a weighted graph, given by the trellis representation of the convolutional encoder [1]. The Viterbi algorithm is optimal in terms of maximum likelihood decoding, as it selects the most probable sequence of transmitted symbols based on the received noisy data.

However, neural networks (NNs) can also be employed for convolutional decoding. According to the universal approximation theorem [2], a sufficiently large NN can approximate any continuous function with arbitrary precision. Since an NN can be designed to process noisy input data and output probability distributions over each bit, and given that an explicit optimal decoding algorithm exists, as provided by the Viterbi algorithm, it theoretically has the potential to achieve near-optimal decoding performance.

Using an NN for convolutional decoding presents several advantages. Firstly, it has been shown that deep convolutional decoders are offering a notable advantage in terms of robustness compared to the Viterbi algorithm [3]. Secondly, once trained, NNs perform decoding through multiplication and addition operations. These operations are not only highly parallelizable but also remain independent of K, the memory length of the code. In contrast, the computational complexity of the Viterbi decoding algorithm grows exponentially with K [1] making NNs a more scalable alternative for large memory lengths

As explained in [3], [4], the NN needs to be exposed to at least 90% of the codeword during training phase. However, this condition appears impractical: using only 100 bits per block with a 1/2 rate code would generate 2^{100} codewords. Nevertheless, convolutional codes encode blocks sequentially, processing bits one by one through a memory-based mechanism. This allows the neural network to learn a local structure by exploiting temporal dependencies, enabling it to generalize effectively even when exposed to only a small subset of the possible codewords. Thus, since convolutional codes process bits sequentially, the encoded data can be interpreted as times series where each series represents a block. Given the temporal dependencies between the bits, the use of recurrent neural networks (RNNs) is a natural choice.

This approach has already been explored in previous works where GRU [2], [3], [4] and LSTM [4] were utilized for convolutional decoding. These studies have shown promising results in approximating the Viterbi algorithm, though they often come with relatively high computational decoding costs.

In this work, we aim to develop a simplified network architecture, which is characterized by reduced computational decoding complexity while maintaining performance levels on par with more advanced models.

Figure 1 illustrates the system model. A k-bit dataword $\mathbf{u} \in \mathcal{A}$ where $\mathcal{A} = \{0,1\}^k$ is encoded into an n-bit codeword $\mathbf{m} \in \{0,1\}^n$. The encoded bits are then modulated using Binary Phase Shift Keying (BPSK) to produce the transmitted signal $\mathbf{x}$. During transmission, Additive White Gaussian Noise (AWGN) is introduced, resulting in the received signal

$$\mathbf{y} = \mathbf{x} + \mathbf{n},$$

where $\mathbf{n} \sim \mathcal{N}(0, \sigma^2)$ represents gaussian noise with zero mean and variance σ^2. AWGN is widely used as a reference model for evaluating the performance of communication systems under standard noise conditions [4].

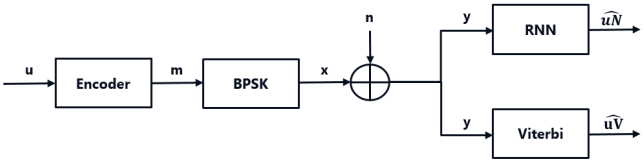


Figure 1: System model

The Viterbi algorithm aims to recover the original data sequence by maximizing the posterior probability. Specifically, it seeks the optimal function $f^\star : \mathbb{R}^n \to \mathcal{A}$ defined as:

$$f^\star(\mathbf{y}) = \arg\max_{\mathbf{u}\in\mathcal{A}} \mathbb{P}(\mathbf{u}|\mathbf{y}),$$

where $\mathbf{y}$ represents the received signal. In contrast, the RNN-based decoder is trained to learn the optimal decoding function $f^\star$ directly from the training data [3]. Specifically, the training dataset comprises pairs $(\mathcal{R}, \mathcal{S})$, where $\mathcal{R}$ contains the noisy encoded blocks, and $\mathcal{S}$ represents the corresponding input codewords. The objective is for the RNN to approximate the optimal decoding function $f^\star$.

In this work, we focus on a systematic convolutional code with generator polynomials $(7, 5)$ and a coding rate of $1/2$.

The paper is organized as follows: Section II describes the system and explains the neural network architecture. Section III outlines the training method and justify the choices made regarding the hyperparameter set, including a discussion of the gaussian annealing technique. Section IV compares different models in terms of decoding performance and computational complexity. Finally, Section V provides the concluding remarks.

II. Network architecture

Table I presents the overall architecture of the proposed neural network. It consists of two main components: recurrent layers and dense layers, also known as fully connected layers. The recurrent layers process the noisy received sequence, capturing temporal dependencies by mapping each sequence into a higher-dimensional representation. The dense layers then leverage these extracted dependencies to estimate probability distributions over each bit. It is worth mentioning that the network outputs soft predictions in the form of probabilities, and a hard decision is applied afterward to obtain the final decoded bit sequence.

A. Recurrent layers

Several studies have demonstrated that bidirectional state propagation is essential for achieving near-optimal decoding performance [2], [3], [4]. This finding has been further corroborated by our own empirical experiments. As mentioned in the introduction, we employ Vanilla RNN layers to attain near-optimal decoding performance while maintaining a relatively low computational cost.

The objective is to design the Vanilla RNN to process an input sequence consisting of pairs of noisy encoded bits and output a sequence of d-dimensional tuples, where d depends on the number of units per layer.

TABLE I: Network architecture of the proposed decoder

Layer	Activation	Parameters	Output Shape
Dropout	-	0.2	$(\boldsymbol{B}, \mathbf{k}, 2)$
Bi-Recurrent 1	tanh	64	$(\boldsymbol{B}, \mathbf{k}, 64)$
Batch Normalisation	-	-	$(\boldsymbol{B}, \mathbf{k}, 64)$
Dropout	-	0.2	$(\boldsymbol{B}, \mathbf{k}, 64)$
Bi-Recurrent 2	tanh	64	$(\boldsymbol{B}, \mathbf{k}, 64)$
Batch Normalisation	-	-	$(\boldsymbol{B}, \mathbf{k}, 64)$
Dense 1	sigmoid	128	$(\boldsymbol{B}, \mathbf{k}, 128)$
Dense 2	sigmoid	128	$(\boldsymbol{B}, \mathbf{k}, 128)$
Dense 3	sigmoid	1	$(\boldsymbol{B}, \mathbf{k}, 1)$

This formulation aligns with a sequence-to-sequence model, where each encoded bit originally represented as an element of $\mathbb{R}^2$ is mapped into a higher-dimensional space. The purpose of this transformation is to provide a richer representation of the bits, thereby enhancing the model's ability to capture relevant features and extract meaningful patterns from the encoded sequence.

To design an effective RNN architecture, several hyperparameters must be carefully examined. Our experiments primarily focused on the number of layers, the number of units per layer and the activation function used in each layer.

We found that using only a single recurrent layer was insufficient to achieve optimal decoding performance, while increasing the number of layers beyond two did not yield significant improvements, an observation consistent with previous findings in [3]. Consequently, we limited the network to two bidirectional recurrent layers.

We also explored different layer configurations with unit counts increasing exponentially from 1 to 256. The results indicated that using fewer than 16 units led to suboptimal decoding performance, while increasing beyond 128 units provided no further gains and significantly increased computational complexity, leading us to select 64 units per layer.

Among the various activation functions tested, $\tanh$ demonstrated the best decoding performance, aligning with observations reported in [1], [2].

B. Dense layers

The dense neural network (DNN) processes an input sequence consisting of k vectors, i.e., k time steps, where each vector belongs to $\mathbb{R}^{64}$. In other words, each encoded bit is now represented in a 64-dimensional space. In the first dense layer, each of the 128 units receives a 64-component input and outputs a scalar. As a result, the sequence is transformed into k vectors in $\mathbb{R}^{128}$, which serve as the input to the second dense layer. This second layer maintains the same output dimensionality as the first.

Finally, each 128-dimensional vector is fed into a single unit in the last layer, which produces a scalar output for each time step, corresponding to each bit. Since the sigmoid activation function is applied, the resulting scalar values can be interpreted as probabilities. Consequently, the network outputs a probability distribution over the decoded bits.

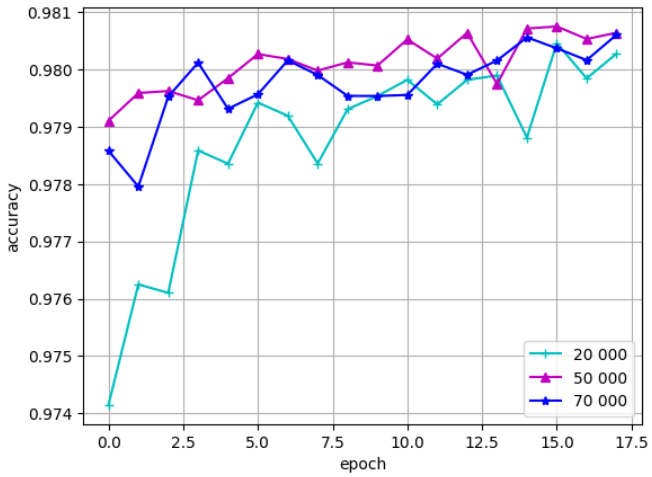


Figure 2: Impact of training dataset size on model accuracy

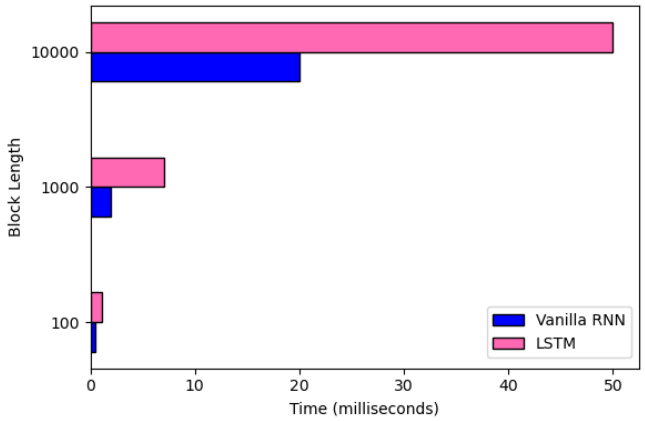


Figure 3: Average decoding time for Vanilla RNN and LSTM

C. Batch Normalisation and Dropout

It has been demonstrated that deep learning models achieve better performance when trained with normalized input data [5]. In our case, the input data is already normalized to have zero mean and a fixed variance. However, as the data propagates through the network, this normalization may be lost due to successive transformations. To address this issue, batch normalization layers are incorporated after each layer to maintain a consistent data distribution and improve training stability.

Additionally, dropout is employed as a regularization technique to enhance decoding performance. By randomly deactivating a fraction of neurons at each iteration, dropout mitigates overfitting and encourages the model to learn more robust and generalizable features.

III. Training Method

Once the network architecture has been defined, a significant part of the work focuses on determining the most effective training strategy.

A. Deep learning basics

1) Loss function: The decoding problem can be interpreted as both a binary classification task or a regression task.

From a binary classification perspective, the neural network aims to predict whether each bit of the original message is 0 or 1. In this case, a hard decision is applied according to the rule:

$$\hat{u}_i = \begin{cases} 1 & \text{if } \mathbb{P}(u_i = 1 \mid y_i) > 0.5, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

where each prediction is discretized into 0 or 1. The loss function used in this context is the binary cross-entropy (BCE), defined as:

$$\mathcal{L}_{\text{BCE}} = -\sum_{i=1}^{k} u_i \log \hat{p}_i + (1 - u_i) \cdot \log(1 - \hat{p}_i) \tag{2}$$

Alternatively, from a regression perspective, the network seeks to approximate the continuous function $f^\star$. The output of the network remains a continuous value, which must then undergo a hard decision step to determine the predicted bit. In this case, the commonly used loss function is the mean squared error (MSE), defined as:

$$\mathcal{L}_{\text{MSE}} = \sum_{i=1}^{k} (u_i - \hat{p}_i)^2 \tag{3}$$

Both approaches yield similar decoding performance. Ultimately, we opted for the classification-based approach.

2) Optimizer and learning rate: Despite the fact that [2] suggests that *RMSprop* optimizer provides slightly better results with a classification approach, our experiments showed that the *Adam* optimizer achieved the best performance for both the MSE and BCE loss functions.

The learning rate was determined using the Hyperas optimization framework, specifically tailored to our network architecture. The optimal learning rate obtained through this process was 0.0013.

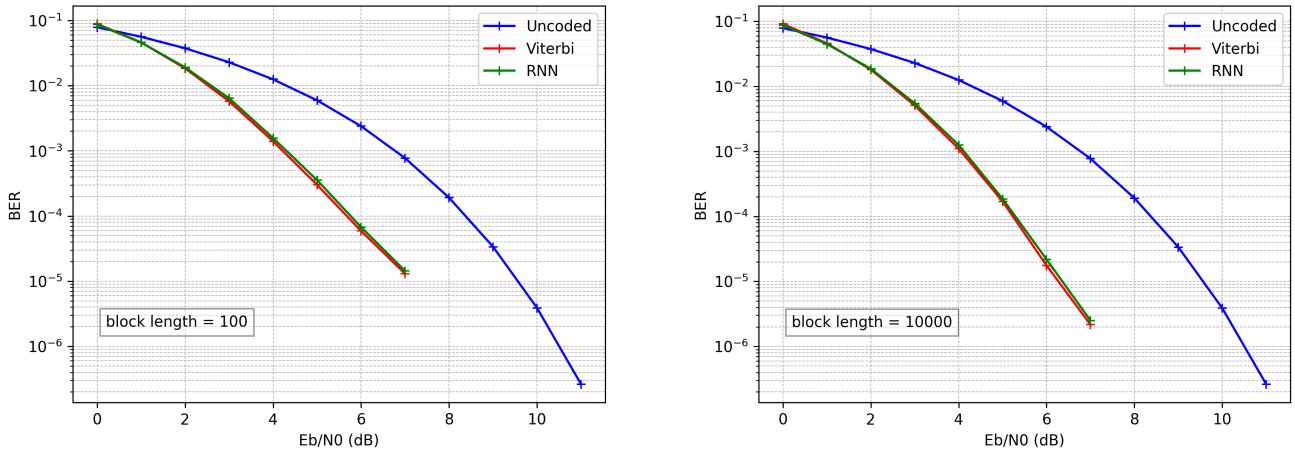
3) Batch size and epochs: Our experiments confirmed the findings of [2], which suggest that smaller batch sizes lead to better network performance. Based on this observation, we selected a batch size $\boldsymbol{B}$ of 16 sequences.

Figure 2 presents the training accuracy curve, where it can be observed that after a few epochs, the model reaches its peak accuracy and ceases to improve. The rapid convergence is attributed to the simplicity of the architecture. Throughout this study, we opted for 15 training iterations.

4) Metric: Among the various evaluation metrics, the most natural choice is the BER, as it directly quantifies the decoding performance by measuring the fraction of incorrectly predicted bits. The BER is defined as:

$$\text{BER} = \frac{1}{k} \sum_{i=1}^{k} \mathbb{1}_{\{u_i \neq \hat{u}_i\}} \tag{4}$$

Other metrics were explored but did not lead to any performance improvements.



(a) (b)

Figure 4: BER performance of our Vanilla RNN decoder for block lengths of 100 and $10,000$.

B. Important Hyperparameters

1) Training SNR: It is standard practice to sample both training and test data from the same distribution. However, empirical studies [2], [3], [4], as well as our own experimental results, have demonstrated that the neural network achieves its best performance when trained on encoded blocks corrupted by noise at very low SNR values.

Furthermore, an analytical study [2] established that, for a given network architecture and coding scheme, an optimal training SNR exists. In our experiments, this SNR was found to be 2 dB.

2) Training block length: Once trained, the neural network can decode blocks of varying lengths. However, during training, a specific block length must be chosen. Our experiments, as well as findings reported in [4], indicate that training with blocks of length 100 significantly enhances the network's generalization ability. Notably, a model trained on 100-bit blocks can effectively decode much longer sequences, including 10,000-bit blocks, without requiring additional retraining.

This result highlights the strong generalization capacity of RNNs. Furthermore, it demonstrates the ability of the network to decode previously unseen codewords. This suggests that temporal dependencies are efficiently captured, allowing the model to exploit local structural patterns in the received sequences.

3) Training dataset size: The size of the training dataset is a critical factor influencing the performance of the neural decoder. A sufficiently large dataset is required to ensure proper generalization while avoiding overfitting. Figure 2 illustrates the impact of dataset size on the model accuracy. Based on our experiments, a training set consisting of $50,000$ encoded and noisy blocks was found to provide the best decoding performance.

C. Gaussian Annealing

During our experiments, we evaluated the Gaussian Annealing technique, commonly used in optimization to enhance model robustness and convergence. This method involves introducing a Gaussian noise component during training, which gradually decreases. The objective is to prevent the model from getting trapped in suboptimal local minima by encouraging a more diverse exploration of the solution space in the early stages of training. However, as noted in [1], due to the simplicity of our model, this method did not yield any noticeable improvement in decoding performance. Consequently, it was not incorporated into the final model.

IV. Results

We present the BER curves of our RNN decoder over AWGN in Figure 4. Additionally, in Figure 3, we compare the average decoding time of a dataword using an RNN and an LSTM decoder for block lengths of 100, 1000 and $10,000$.

V. Conclusion

The results demonstrate that Vanilla RNNs can achieve near-optimal decoding performance while maintaining a relatively low decoding time.

References

[1] K. Hueske, J. Götze, and E. Coersmeier, "Improving the Performance of a Recurrent Neural Network Convolutional Decoder," in *Proc. IEEE ISSPIT*, 2007, pp. 445–450.

[2] D. Tandler, S. Dörner, S. Cammerer, and S. ten Brink, "On Recurrent Neural Networks for Sequence-based Processing in Communications," *arXiv preprint arXiv:1905.09983*, Nov. 2019.

[3] K. Yashashwi, D. Anand, S. R. B. Pillai, P. Chaporkar, and K. Ganesh, "MIST: A Novel Training Strategy for Low-latency Scalable Neural Net Decoders," *arXiv preprint arXiv:1905.08990*, May 2019.

[4] H. Kim, Y. Jiang, R. Rana, S. Kannan, S. Oh, and P. Viswanath, "Communication Algorithms via Deep Learning," *arXiv preprint arXiv:1805.09317*, May 2018.

[5] F. Chollet, *L'apprentissage profond*. Paris, France: Eyrolles, 2018, ISBN: 978-2491674007.