Jestr

Journal of Engineering Science and Technology Review 18 (2) (2025) 8-20

Research Article

JOURNAL OF
Engineering Science and Technology Review

www.jestr.org

Artificial Intelligence-based Decoding Algorithms for Binary and Duo-Binary Turbo Codes

Dharamveer Russeawon[1] and Yogesh Beeharry[2,*]

[1]*Huawei Technologies (Mauritius) Co. Ltd*
[2]*Department of Electrical and Electronic Engineering, Faculty of Engineering, University of Mauritius, Réduit, Mauritius*

Received 20 July 2024; Accepted 22 March 2025

Abstract

This work proposes the enhancement of Long-Term Evolution (LTE) binary and Digital Video Broadcasting – Return Channel Satellite (DVB-RCS) duo-binary turbo codes by replacing the conventional Max-Log MAP decoder with a Deep Learning Neural Network (NN). The binary turbo decoder is replaced with a Recurrent Neural Network (RNN) composed of two bi-LSTM (bi-Long-Short Term Memory) layers containing 200 hidden units each. The duo-binary turbo decoder is replaced with DTNET which is an RNN with 2 bi-LSTM layers and comprising 250 hidden units each. All NNs designed in this work outperform the conventional turbo decoder at $\frac{E_b}{N_0}$ values lower than 0 dB. The trade-off for achieving this performance is the increased computational complexity during the training phase of the model. The algorithms used with Quadrature Phase Shift Keying (QPSK) modulation generalize for code rates above the code rate trained on. The results suggest that a "rate-less" and block size-adapting turbo decoder can be achieved by training an RNN with 2 bi-LSTM layers comprising at least 200 hidden units on the lowest code rate with data from a QPSK modulated communication system.

Keywords: LTE Turbo codes, Duo-Binary Turbo codes, Recurrent Neural Network, bi-Long Short Term Memory, Max-Log-MAP

1. Introduction

Paramount importance has been accorded to reliable digital communication in the modern information age. A key issue in the communication field is the design of codes to allow robust and efficient decoding in a noisy channel. Channel coding was first introduced in 1948 by Claude Shannon in his landmark paper [1]. The basic idea behind channel coding is the addition of redundant bits that convey information about the message bits in the decoding process. According to Shannon, it is possible to achieve arbitrarily small transmission errors at a maximum bit rate for any channel. While it is not possible to operate beyond the Shannon limit [1], it is stated that creatively and intelligently designed codes can perform very close to the Shannon limit albeit under certain restrictions such as block size and the type of channel. Some landmark codes that approach the theoretical Shannon limit are Turbo codes, Low-Density Parity Check (LDPC) codes, and Polar codes.

With codes operating very close to the Shannon limit, it is wise to push towards their robustness. In the case of Turbo codes, the known optimal decoding algorithm is Maximum a-posteriori (MAP). Due to the computational inefficiency involving a huge number of multiplication operations, the MAP algorithm is often not implemented. Instead, LOG-MAP or MAX-LOG-MAP algorithms are implemented which are sub-optimal algorithms having cheaper computational costs. To achieve a better bit error rate (BER) and higher computational efficiency, there have been attempts to create a Neural Network (NN) based decoder for Turbo codes. For a channel decoding task, the input is the noise-corrupted sequence and the output is the corresponding encoded noiseless sequence.

Conventional iterative decoders using the algorithms such as MAX-Log-MAP rely on approximations and handcrafted metrics while RNN-based approaches have the property of learning decoding techniques directly from data. Thus RNN-based Turbo decoding advances the error correction coding field significantly enabling improved performances under varying noise conditions and channel impairments. The data-driven approach is particularly impactful in low signal-to-noise ratio (SNR) regimes where the decoding adaptability and robustness are enhanced. Given that the requirement for modern communication systems is low latency without significant degradation in error performance as trade-off, systems with reduced computational complexity are desired. RNN-based decoders have been found to fit these requirements, thereby making them highly suitable for modern communication systems [2, 3]. Decoding challenges exist in non-standard channel models for application in IoT and 5G communications. RNN-based decoding have demonstrated their ability to address these challenges making them a suitable candidate [4]. Thus, this work contributes to the state-of-the-art in turbo decoding through the strong learning abilities of RNN and providing an efficient and scalable potential solution for next generation communication systems.

This work aims to investigate whether a NN can enhance the BER performance of conventional binary and duo-binary Turbo codes. Concerning the works conducted in [5], a Recurrent Neural Network (RNN) with directionality is deemed necessary for good performance. The data needed for the training process is generated using the DVB-RCS duo-binary turbo encoder and the Long-Term Evolution (LTE) binary turbo encoder with a modification to the interleaver. The conventional turbo decoder is replaced with an RNN comprising 2 bi-LSTM (bi-Long-Short Term Memory) layers

*E-mail address: y.beeharry@uom.ac.mu
ISSN: 1791-2377
doi:10.25103/jestr.182.02

having at least 200 hidden units each. Owing to no previous works on duo-binary turbo decoding by NN architectures, we propose an RNN with 2 bi-LSTM layers and 250 hidden units in each layer. One of the main objectives of this work is to come up with a code-length and code-rate independent decoder. The generalization capacity of the RNN is examined through the use of different block sizes and code-rates input to the RNN for decoding. The computational complexity lies in the training of the neural network. However, once the model is obtained, there is no need for re-training at different code lengths and code rates.

The paper is organized as follows. Section 2 contains an overview and theoretical framework surrounding turbo codes and NNs. Previous works relating to the use of NNs in turbo decoding are also explored. Section 3 illustrates the methodology followed for the software implementation of the NNs for turbo decoding. Section 4 presents the analysis and discussion of the results obtained with the methodology followed. Section 5 sheds light on the major findings of this work and suggests some further works.

2. Overview

The following sub-sections provide a review of the NN-based decoding of turbo codes and the related parameters used in previous research. A theoretical framework on the essential concepts in turbo codes and NNs is also provided.

2.1 Previous Research on NN-based Turbo Decoding

To approach the Shannon limit, several channel codes were invented. However, these channel codes were the product of human ingenuity and only came periodically across the century. As such, there was a need to enhance existing state-of-art error-correction codes which led to several attempts to implement channel codes using NNs on various occasions. Channel codes have been implemented using feedforward NNs as well as RNNs. The authors in [6] implemented a feedforward NN using a multilayer perceptron structure. The BER generated by the neural decoder was close to the optimal MAP decoder with the latter having a E_b/N_0 gain of 0.8 dB over the neural decoder at a BER of 0.01 for a 37/21 turbo code. In [7], the BCJR algorithm was reformulated using matrix manipulations to implement a feedforward NN that was tested for performance in computer simulations.

In [8] and [9], an auto-encoder model was used for turbo encoding and decoding. The authors of [8] introduced the interleaving process into the neural decoding process. The results obtained at low Signal-to-Noise ratio (SNR) ranges, show that the introduction of the interleaving process makes a significant difference. The performance matches and even beats the conventional turbo codes whereas the NN trained, without the interleaving included, performs worse than the turbo code. However, large block sizes for auto-encoder training require very large memory which is not feasible without sufficient computing resources. It is also judged that the Turbo auto-encoder encounters a lot of difficulty during the training at high SNRs thereby hindering its learning process. The loss is the main driving force of the NN training which pushes the system towards its minimum. Training at high SNRs causes fewer bits to be in error resulting in a tiny loss that ultimately makes training very difficult.

Authors in [5] and [10] use bi-directional gated RNNs to perform the decoding process. The NN in [10] shows poor generalization over high SNRs after training a block length of 64 bits with only 3000 training examples and a range of SNR from -2 to 2 dB. The authors of [5] train a NN for a block size of 100 using 1.2 million examples but using only one training SNR which performs much better over the remaining SNRs. The system also demonstrates generalization over other longer block lengths.

With the above-mentioned networks performing the 'learning to decode' task, the authors in [11] approach the problem differently to obtain a network that has fewer parameters to train by replacing only the iterative decoders with suitable subnets. Three iterations by the TurboNet in [11] outperform the LOG-MAP and the MAX-LOG-MAP algorithms and perform the decoding task in less time and with fewer parameters than the BCJR RNN used in [5]. A complexity analysis by the author in [11] shows that the RNN in [8] has 3.85M trainable parameters as compared to 17.8k parameters being trained in [11].

2.2 Training Hyper-parameters and parameters used in other researches

Some of the training parameters and hyper-parameters used in similar research for the use of NNs in decoding tasks are given in the following sub-sections.

2.2.1 Training SNR

In [10], an RNN with 2 layers of bi-directional gated recurrent units (bi-GRU) is shown to outperform the conventional turbo codes (packet size 64) at low SNR although it lags at higher SNR. It proposes training 2 separate turbo decoders for high and low SNRs respectively to counter the problem. The authors in [8] propose an RNN with a similar structure as [10] except with batch normalization that is trained for code-rate ½ and a block size of 100. For experiments conducted in [8], a relationship is derived between the ideal training SNR and the code rate unlike [10] which uses a set of SNRs ranging from -2 to 2 dB for training. The NN in [5] matches the performance of the MAP optimal decoding algorithm. Strong generalization is shown in [5] as the NN performs equally well on block sizes of 1,000 and 10,000 when trained on a block of 100. According to the empirical results derived in [5], with a turbo code of code rate r, the training SNR is given in equation (1).

$$SNR_{Train}(dB) = \min\{SNR_{test}(dB), 10\log_{10}(2^{2r}-1)\} \quad (1)$$

2.2.2 Amount of data required, Mini-batch size and optimizer used

According to authors in [5], the number of training examples that are required to approach the performance of turbo codes is dependent on the block size used. Additionally, the batch size must be optimal: too small of a mini-batch size leads to very slow convergence due to the update at each step whereas a very large mini-batch size may result in poor generalization and memory problems. In [5], the two layers of bi-GRU with 200 hidden units are trained with 1.2 million examples to replicate the turbo code performance with a batch size of 200 (using ADAM optimizer). The authors in [11] find a mini-batch size of 500 suitable for the training of a block size 40 turbo code with ADAM optimizer.

2.2.3 Learning Rate and Number of Epochs

The learning rate determines the time taken in the training process as well as its convergence. In [5], the learning rate chosen is of the order of 10^{-3} and converges well as its performance approaches that of turbo codes. The RNN in [11] converges with a learning rate of 10^{-5} in a 2-layered bi-LSTM structure. The epoch signifies the number of times the whole

training dataset is used in the training process. Similar research for RNN-based decoding in [5, 10, 11] uses epoch numbers of 50, 30, and 10 respectively.

2.2.4 Number of Layers and Hidden Units

It is indicated in [5] that one-layer structured RNNs and single-direction RNNs perform worse than 2-layer RNN structures. As such, it is deemed important to have a bi-directional RNN with at least 2 layers for good performance. Two bi-LSTM layers of 800 units were sufficient to have a good BER in [11]. Additionally, the author in [10] stipulates that less than 200 GRU units in the 2-layered NN leads to divergence during training.

2.3 The Turbo Encoding & Decoding Principles

Turbo codes were invented in 1993 by Claude Berrou [12] and were the first practical codes to approach the Shannon limit. Turbo codes find applications in mobile communication, Universal Mobile Telecommunications Systems (UMTS), LTE as well as deep-space satellite communications.

2.3.1 Binary Turbo Encoding & Decoding in the LTE Standard

Turbo codes are described as a parallel concatenation of Recursive Systematic Convolutional (RSC) codes. The details of the Turbo encoder can be obtained from [12][13] and [14]. Details of the corresponding decoder can be obtained from [12] and [15].

2.3.2 Max-Log MAP Algorithm

The Log-Likelihood Ratio in this algorithm is computed as follows:

$$L(u_k|\mathbf{y}) = max^*{}_{R1}\left[A_{k-1}(s) + \Gamma_k(s', s)\right) + B_k(s)] - max^*{}_{R0}\left[A_{k-1}(s) + \Gamma_k(s', s)\right) + B_k(s)] \quad (2)$$

where,
$L(u_k|\mathbf{y})$ represents the log-likelihood ratio for transmitted symbol u_k at time instant k, given the received symbol $\boldsymbol{y}$
A represents the forward metric
B represents the backward metric
Γ represents the state transition metric
s', s represent the previous and current states respectively
$R1$ and $R0$ denotes the transitions with output bits 1 and 0 respectively

Details on the computations of the different metrics can be obtained from [16].

2.3.3 Duo-Binary Turbo Encoding / Decoding

Duo-binary turbo encoder consists of two duo-binary Circular Recursive Systematic Codes (CRSC) in parallel concatenation. Details of the duo-binary CRSC encoder used in Digital Video Broadcasting – Return Channel Satellite (DVB-RCS) can be obtained from [17].

Duo-binary turbo codes can also be iteratively decoded using the usual MAP algorithm. However, there are 4 LLRs to be computed for Duo-Binary turbo code since the pair (A, B) can take on 4 values. Modifying the equation for binary codes, the following is obtained for duo-binary turbo codes:

$$LLR = LLR = \ln \frac{P(u_k = i|\mathbf{y})}{P(u_k = i|\mathbf{y})} \quad (3)$$

where,
$\mathbf{y}$ is the received noisy vector,
u_k is the message symbol that can take on values i equal to 00, 01, 10 and 11.

For duo-binary turbo codes, a state has 4 transitions in the trellis because of 4 possible values of (A, B). Although the BER of the duo-binary turbo code can be superior to that of the classical turbo code, the computational complexities and decoding times increase even more with the number of iterations.

2.4 Neural Networks

The concept of NNs is derived from the human brain which consists of synapses and neurons. To recognize an object, specific neurons are fired in the human brain which has millions of neurons with billions of connections between them. Artificial neurons make up an Artificial Neural Network (ANN) which is commonly called Neural Network for simplicity. The perceptron is one such artificial neuron that was conceived in the 1950s by F. Rosenblatt. As shown in Figure 1, the basic mathematical model around the perceptron was that for a given number of inputs, X, the perceptron assigned weights, W to the inputs, and all of them were added. The output was decided upon a threshold that would typically be learned in training by going through a great number of examples and epochs whereby an epoch represents one pass of the whole dataset into the NN [18].

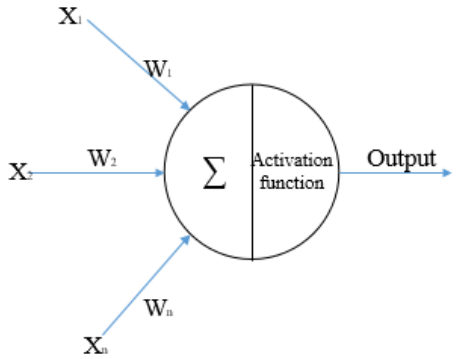

Fig. 1. Perceptron model

As a single unit, the perceptron would not be able to make good decisions about complex problems. However, a network composed of several layers and a handful number of neurons in each layer would be able to make a good decision about a complex problem when given enough training. In a multi-layered structure, the common intuition is that each layer performs a feature extraction of increasing abstraction level. The multilayer perceptron is commonly referred to as Vanilla NN when there is only one hidden layer. Generally, increasing the number of hidden layers for a NN helps to better model at the cost of more training time [19].

Long Short-Term Memory (LSTM)

LSTM networks are commonly used for sequence-to-sequence learning of long-term dependencies in a sequence. LSTM networks have gates that are used to selectively remember relevant information. As a lot of gradients are multiplied during the backpropagation process, successive multiplications of gradients much less than 1 tend to zero which gives rise to a vanishing gradient. With very small gradients, the error becomes negligible making the NN incapable of learning whereas exploding gradients cause a very large change in parameters which can potentially lead to divergence. The gates incorporated in the LSTM network are

forget, input, and output gates which help in selective read, write and forget. The forget gate, in particular, helps in deterring the vanishing gradient problem [20]. The structure of the LSTM block is shown in Figure 2 [21].

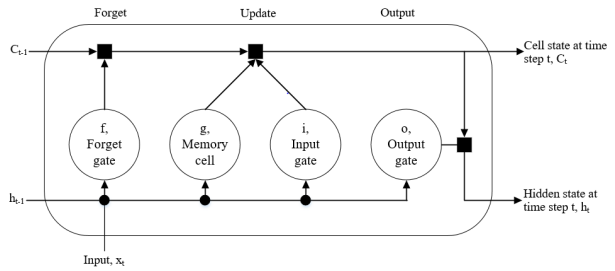

Fig. 2. LSTM block

The core elements of the LSTM block are explained in [22]. The cell and hidden states contain information about the sequence whereas the gates selectively allow information to flow in the block. The difference between the cell state and the hidden state is that the cell state (long-term memory) stores information over longer time steps than the hidden state (working memory). The functions of each gate found in the LSTM block are outlined in Table 1.

Table 1. Function of each gate in the LSTM block

Gate	Function
Input gate	The input gate passes the previous hidden state and the current input into a sigmoid function to update the cell state to retain relevant information.
Output gate	The output gate determines the hidden state to be fed to the next block by combining the previous hidden state, the present input and the new cell state.
Forget gate	The forget gate is used to forget information; its value ranges from 0 to 1 with the value close to 0 meaning forgetting the given information and 1, retaining the information.

The Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) designed to overcome the vanishing gradient problem by introducing gating mechanisms. A step-by-step description of the equations governing the LSTM structure is as follows:

Step 1: Forget Gate
The forget gate determines which information from the previous cell state C_{t-1} should be discarded. It uses a sigmoid activation function to generate a value between 0 (forget) and 1 (retrain).

$$f_t = \sigma\left(W_f . [h_{t-1}, x_t] + b_f\right) \quad (4)$$

f_t: Forget gate vector (shape: n-dimensional)
x_t: Current input vector
h_{t-1}: Previous hidden state
W_f: Weight matrix for the forget gate
b_f: Bias term for the forget gate
σ: Sigmoid activation function

Step 2: Input Gate
The input gate decides what new information to add to the cel state. It consists of two parts:

1. A sigmoid layer (i_t) determines which parts of the input are relevant.
2. A *tanh* layer $(\widetilde{C_t})$ creates candidate cell state update

$$i_t = \sigma(W_i . [h_{t-1}, x_t] + b_i) \quad (5)$$

$$\widetilde{C_t} = tanh(W_C . [h_{t-1}, x_t] + b_C) \quad (6)$$

i_t: Input gate vector
$\widetilde{C_t}$: Candidate cell state update
W_i, W_C: Weight matrices for input gate and candidate state
b_i, b_C: Bias terms for input gate and candidate state
$tanh$: Hyperbolic tangent activation function

Step 3: Update Cell State
The new cell state C_t is computed by combining the old cell state C_{t-1} modulated by the forget gate f_t, and the candidate state $\widetilde{C_t}$, modulated by the input gate i_t.

$$C_t = f_t \odot C_{t-1} + i_t \odot \widetilde{C_t} \quad (7)$$

C_t: Updated cell state
$\odot$: Element-wise multiplication

Step 4: Output Gate
The output gate determines the parts of the cell state C_t that will be passed to the hidden state h_t. This involves applying a sigmoid function (o_t) and modulating the cell state with tanh (C_t).

$$o_t = \sigma(W_o . [h_{t-1}, x_t] + b_o) \quad (8)$$

$$h_t = o_t \odot tanh(C_t) \quad (9)$$

o_t: Output gate vector
h_t: Current hidden state
W_o: Weight matrix for the output gate
b_o: Bias term for the output gate

3. System Model

The system for the binary Turbo code implemented with Binary Phase Shift Keying (BPSK) and Quadrature Phase Shift Keying (QPSK), named TNET and TNETQPSK respectively, is shown in Figure 3. The packet size of 40 bits and code rate of ½ are used.

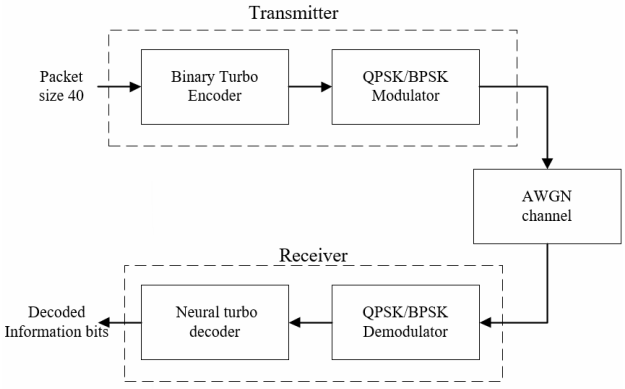

Fig. 3. Overview of the system used

The different stages involved in the neural turbo decoding are as follows:

i. Data processing
ii. Deep NN design
iii. Training of the NN
iv. Testing of the models

In the data processing stage, 1.2 million packets were deemed sufficient for training an RNN with 200 bi-GRU units on a block size of 40 ($\text{code} - \text{rate} = \frac{1}{2}$). The $\frac{E_b}{N_0}(dB)$ values used for training the NN with BPSK and QPSK modulation

Table 2 shows the activations and learnables in TNET and TNETQPSK. A summary of the main parameters and hyper-parameters considered for the training of TNET and TNETQPSK are given in Table 3.

schemes are 3.010 and 0.0 respectively. These values have been selected after performing intensive individual simulations and analyzing the performance with each $\frac{E_b}{N_0}(dB)$ value. The testing of the models is performed by generating new sets of random messages repeating the whole process and using the NN decoder.

3.1 Deep NN Design for Binary Turbo Codes

The layers used for TNET and TNETQPSK are shown in Fig. 4.

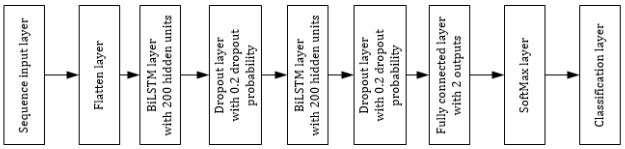

Fig. 4. Layers used for TNET and TNETQPSK.

Table 2. Activations and learnables in the TNET and TNETQPSK.

Layer	Activations	Input weights	Recurrent Weights	Bias	Learnable
Sequence input	1×80×1	-	-	-	0
Flatten	80	-	-	-	0
Bi-LSTM	400	1600×80	1600×200	1600×1	449600
Dropout	400	-	-	-	0
Bi-LSTM	400	1600×400	1600×200	1600×1	961600
Dropout	400	-	-	-	00
Fully connected	2	2×80	-	2×1	802
SoftMax	2	-	-	-	0
Classification	-	-	-	-	0
Total learnable					1412002

Table 3. Summary of training parameters for TNET and TNETQPSK

Parameter	Value/Setting
Block size trained	40
Code-rate used	$\frac{1}{2}$
Number of training examples	1.08 million
Optimiser	ADAM
Mini-batch size	500
Learning rate	3×10^{-3}
Epoch number	100
L2 regularisation constant	10^{-4}

The performance of the models was tested using three criteria: the validation accuracy during training, the BER performance against MAX-LOG-MAP and the generalization capability. 10% of the 1.2 million packets generated are used for validation purposes. As all data were generated randomly with no specific correlations between successive sequences, it was deemed appropriate to select the last 10% of the 1.2 million training examples for validation data. The validation frequency is set at 500 to monitor the training progress and any divergence. The BER performance is computed after decoding and de-multiplexing the noisy sequence. This predicted sequence is then compared against the expected sequence.

The generalization capability of the model can be tested by passing different block sizes into NN and examining the BER performance.

a. Different code rates other than the rate trained on can also be tested to see if the models can adapt to them. The code rates at which the generalization is tested are $\frac{8}{22},\frac{8}{20},\frac{8}{18},\frac{8}{16},\frac{8}{17},\frac{8}{14},\frac{8}{13},\frac{8}{12}$. These code rates are divided into smaller code rates: $\frac{8}{22},\frac{8}{20},\frac{8}{18},\frac{8}{16}$ and bigger code rates $\frac{8}{16},\frac{8}{17},\frac{8}{14},\frac{8}{13},\frac{8}{12}$. Code-rate $half\ is$ kept into both BER graphs as a point of reference for comparison among all code-rates.
b. Similar to the analysis done for code rate, the models are also used to decode different block sizes other than the block sizes used for training. Small and large block sizes used in the LTE standard are employed in the analysis for generalization evaluation for smaller blocks. The smaller block sizes tested are 40, 48, 56, and 64 bits whereas the bigger block sizes tested are 400, 200 and 96. The block size of 40 is also included in the BER graph for a bigger block size as a point of reference for ease of comparison.

3.2 Deep NN Design for Duo-Binary Turbo Codes

The NN trained for duo-binary turbo code is named DTNET. Due to the computational constraints imposed by the lengthy sequence length of the duo-binary turbo code, it becomes increasingly difficult to train the network with voluminous data. Since the duo-binary turbo code is composed of 2-bit streams with 48-bit size each, 1.2 million packets (inclusive of validation data) were thus generated for the training of DTNET. The training $\left(\frac{E_b}{N_0}\right)$ was set to 0 dB.

As no previous work has been done for duo-binary turbo codes in the context of NN decoding, the layers shown in Fig. 5 were proposed for the training process. The proposed network is based on intuition following the one for binary Turbo codes. To minimize the complexity of the network and shorten the training time, only 2 bi-LSTM layers were used

for DTNET. However, the number of hidden units was increased to 250 to better model the more complex decoding Table 4 shows the activations and learnables for DTNET. As the demultiplexed received vector is fed to DTNET, the sequence length is always 288, regardless of the rate although the punctured elements are replaced with zeros.

process. The dropout layers and L2 regularisation were used to avoid overfitting.

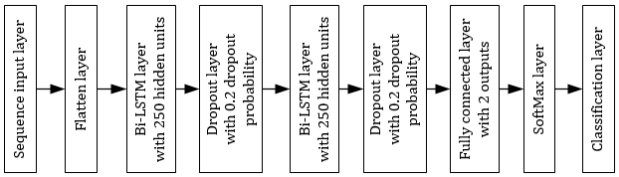

Fig. 5. Layers used for DTNET

Table 4. Activations and learnable for DTNET

Layer	Activations	Input weights	Recurrent Weights	Bias	Learnable
Sequence input	1×288×1	-	-	-	0
Flatten	288	-	-	-	0
Bi-LSTM	500	2000×288	2000×250	2000×1	1078000
Dropout	500	-	-	-	0
Bi-LSTM	500	2000×500	2000×250	2000×1	1502000
Dropout	500	-	-	-	0
Fully connected	2	2×500	-	2×1	1002
SoftMax	2	-	-	-	0
Classification	-	-	-	-	0
Total Learnable					2581002

A summary of the main parameters and hyper-parameters considered for the training of DTNET is given in Table 5. Due to the huge computational load imposed by training long sequence length (288 bits for 48-bit frame size for duo-binary turbo code), the epoch number was set to 5.

Table 5. Summary of training parameters

Parameter	Value/Setting
Frame Size	48
Code-rate	$\frac{1}{2}$
Number of training examples	1.08 million
Optimiser	ADAM
Mini-batch size	200
Learning rate	3×10^{-3}
Epoch number	5
L2 regularisation	10^{-4}

The same criteria as with binary Turbo codes are used to test the performance of DTNET. The training and validation accuracy are first examined. A BER comparison is then done with the conventional duo-binary turbo decoding. Its adaptability with other code rates and block sizes is evaluated through the generation of different BER graphs on different code rates and frame sizes.

The frame sizes for which DTNET is tested are 48, 64, 212 and 440 (used in DVB-RCS standards [17]). 48 and 64 bits are the shortest frame sizes according to the DVB-RCS standards whereas the other frame sizes are all above 200. The code rates for which DTNET is tested are $\frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{3}{4}$ and $\frac{6}{7}$. Code-rates $\frac{1}{3}$ and $\frac{2}{5}$ are the low code-rates whereas code-rates $\frac{3}{4}$ and $\frac{6}{7}$ are the high code-rates.

3.3 Design Choices

The RNN used for binary Turbo decoding comprises of 2 bi-LSTM layers with 200 hidden units each while 250 hiddent units are used with the 2 bi-LSTM layers for duo-binary Turbo decoding. Binary Tirbo decoding is performed on short sequences with less complex interleaving patterns. This can be effectively captured with 200 hidden units per layer. In contrast, duo-binary Turbo decoding involves the processing of longer sequences and higher code-rates, thereby requiring an increase in the number of hidden units per layer. The 50 additional hidden units enhance the network's ability to model intricate dependencies and iterative decoding processes for duo-binary Turbo codes. This increase in computational complexity is balanced with the decoding performance [23, 24, 25]. The parameter selection aligns with prior works demonstrating the efficacy of bi-LSTM architectures in sequential data tasks while ensuring computational feasibility in resource-constrained environments [25].

The choice of the number of training epochs and learning rate for RNN-based turbo decoding is critical to balancing convergence speed, generalization, and computational efficiency. In this work, the number of training epochs was determined through empirical evaluation to ensure that the model adequately learns the decoding task without overfitting, while the learning rate was selected using grid search to achieve stable and consistent optimization. Specifically, a learning rate of 10^{-3} provided a good trade-off between convergence speed and avoiding gradient explosion or vanishing issues commonly encountered in training RNNs [26]. Training for 100 epochs was sufficient to achieve convergence, as indicated by the stabilization of the loss function and minimal improvement in bit error rate (BER) performance beyond this point. This configuration ensures that the model can effectively decode in low signal-to-noise ratio (SNR) conditions without significant degradation in higher SNR regimes. Moreover, the ability of the trained model to adapt to dynamic channel conditions, such as fading or interference, was validated through extensive simulations. These settings are particularly advantageous in real-world communication systems, such as 5G networks and IoT applications, where channel conditions can vary rapidly, and computational resources are constrained [4, 27]. By optimizing the training parameters, the proposed RNN-based decoder demonstrates robustness and efficiency, ensuring practical utility in dynamic environments.

4. Results and Discussions

The performance of the three networks, TNET, TNETQPSK and DTNET are evaluated and presented. For each NN, the

training and validation accuracies are first depicted. The model is then tested against the MAX-LOG-MAP algorithm to compare its BER performance. Finally, the generalization capability is shown through the simulation for different block sizes and code rates.

4.1 Training and validation accuracy of TNET

The training and validation accuracy indicates how well TNET has been able to model the dataset provided to it. For a block size of 40, there exist 2^{40} unique training examples. As such, the data provided to TNET during training comprises only 0.0000982% of the whole dataset. Fig. 6 shows the training and validation accuracy for TNET.

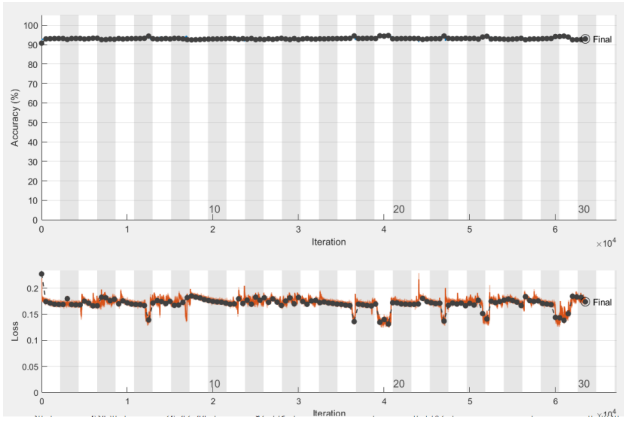

Fig. 6. Training and validation accuracy for TNET

The black dots represent the validation accuracy and the iterations are in units of 10^4. It can be observed from Fig. 6 that the training and validation accuracies are very close and little to no overfitting has occurred. It can be deduced that TNET has modelled the training dataset very well. As a training error occurred, the checkpoint network at 10 epochs was trained for 30 more epochs. After the 30 epochs, the training and validation accuracies were 93.04% and 92.85% respectively. As the accuracy and loss of TNET fluctuate a lot, it is possible to obtain a network with higher accuracy. Through the use of checkpoints in the training process, a network having a training and a validation accuracy of 94.11% and 93.92% respectively was obtained.

4.2 BER Performance of TNET versus MAX-LOG-MAP algorithm

Fig. 7 illustrates that TNET has a worse BER for $\frac{E_b}{N_o}$ values greater than 0 dB. As compared to the turbo decoder, the BER decreases slowly for TNET with increasing $\frac{E_b}{N_o}$. It is possible to improve the performance of TNET through longer training with a larger training dataset.

From Fig. 7, it may be observed that TNET only performs well for other high code rates in the region of $\frac{E_b}{N_0}$ less than 2 dB. The code rate ½ is used as a benchmark for analysis. TNET performs well only for code-rate $\frac{8}{17}$ as the maximum $\frac{E_b}{N_0}$ gain for code-rate $\frac{1}{2}$ is 0.34 dB over the former code-rate. This may be because code-rate $\frac{8}{17}$ is numerically very close to the code-rate trained on $\left(\frac{1}{2}\right)$. Fig. 7 also shows the same pattern of worsening BER performance as the rate decreases or is further from code-rate $\frac{1}{2}$. As such, TNET only adapts to code-rate $\frac{8}{17}$.

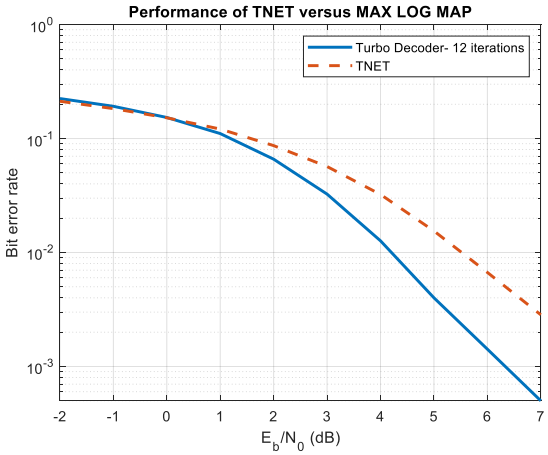

Fig. 7. BER performance of TNET versus Turbo decoder using MAX-LOG-MAP algorithm (12 iterations)

4.3 Code-Rate Adaptability for TNET

The code-rate adaptability is tested by decoding turbo codes of different code rates. The code rates are divided into high and low code rates. Fig. 8 and Fig. 9 show the BER performances of high and low code rates respectively.

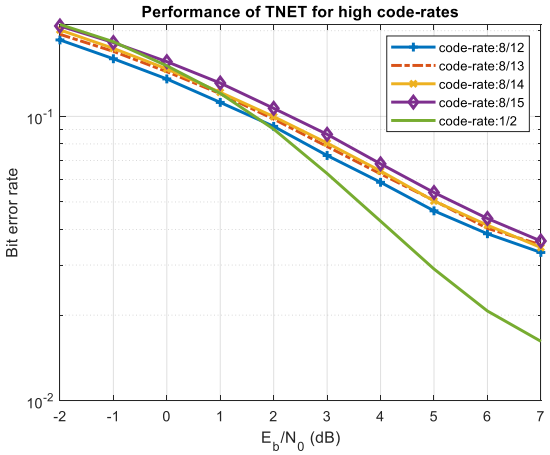

Fig. 8. Performance of TNET for high code-rates

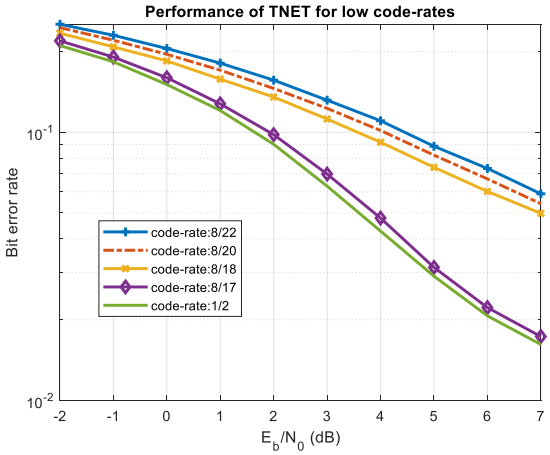

Fig. 9. Performance of TNET for low code-rates

4.4 Generalization to different block sizes for TNET

The generalisation is broken down into two categories; short block sizes close to the block size trained on and long block sizes further away from the block size trained on. The BER performance of TNET on each type of block size is shown in

Fig. 10 and Fig. 11. The code rate is kept at $\frac{1}{2}$ for the performance on the different block sizes.

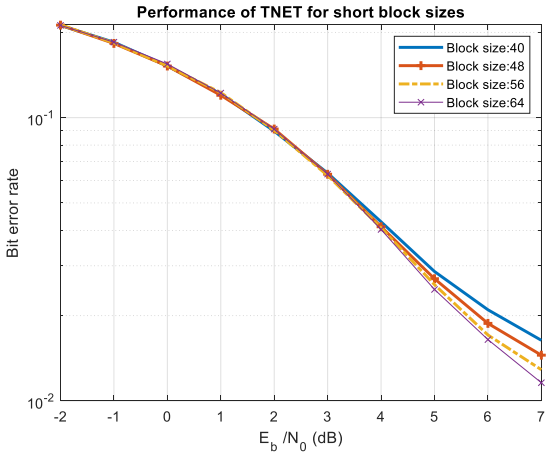

Fig. 10. Performance of TNET on short block sizes

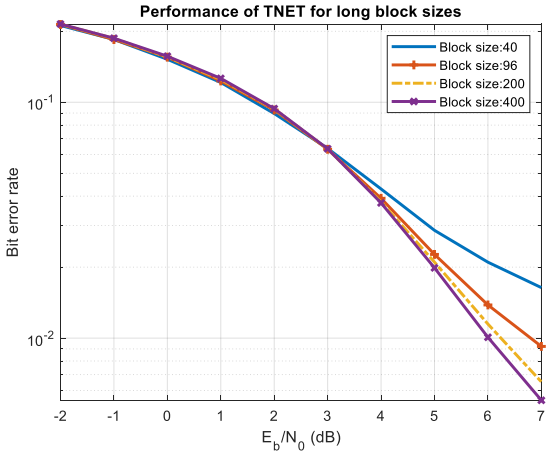

Fig. 11. Performance of TNET on long block sizes

Figures 10 and 11 show very strong generalizations for every block size tested. For $\frac{E_b}{N_0}$ values from -2 to 3 dB, the BER performances for all block sizes are the same. For $\frac{E_b}{N_0}$ value greater than 3 dB, the BER performance improves for other tested block sizes. TNET's $\frac{E_b}{N_0}$ gain observed at a BER of 2×10^{-2} for each simulated block size over the performance of TNET with block size 40 used is given in Table 6. It may be deduced that the BER performance improves with increasing block size.

Table 6. Gain of tested block sizes concerning block size 40 performance

Block size	48	56	64	96	200	400
Gain at BER 2×10^{-2} over block size 40 performance (dB)	0.372	0.583	0.677	0.95	1.12	1.21

4.5 Training and validation accuracy of TNETQPSK

Fig. 12 shows the training and validation accuracy of TNETQPSK. After training for 40 epochs, the training and validation accuracy reached 84.89% and 84.63% respectively. Due to little variation and the network having good accuracy at the final point, the network was used for further analysis. Both the training and validation accuracy of TNETQPSK are lower than those of TNET suggesting that the modulation scheme also has an impact on the model's accuracy.

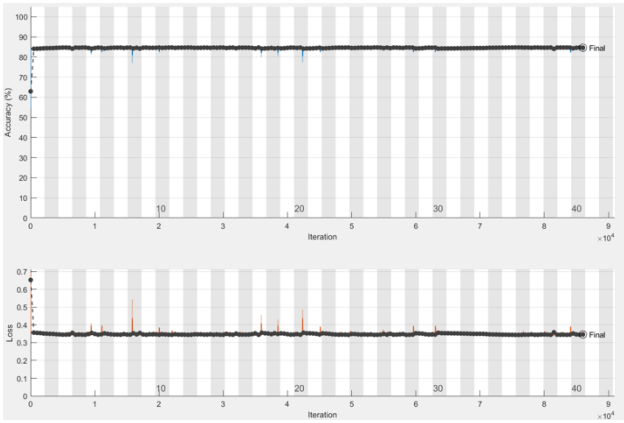

Fig. 12. Training and validation accuracy of TNETQPSK

4.6 BER Performance of TNETQPSK versus MAX-LOG-MAP algorithm

Fig. 13 shows the BER performance of TNETQPSK against the turbo decoder using MAX-LOG-MAP algorithms with 12 iterations. TNETQPSK is found to have similar characteristics as TNET. It outperforms the turbo decoder at $\frac{E_b}{N_0}$ values lower than 0 dB but the BER does not decrease as rapidly as observed with the turbo decoder. The lower accuracy of the training process is reflected in the BER performance of TNETQPSK. It can also be observed that TNET provides a 1 dB gain in $\frac{E_b}{N_0}$ over TNETQPSK at a BER of 10^{-2}.

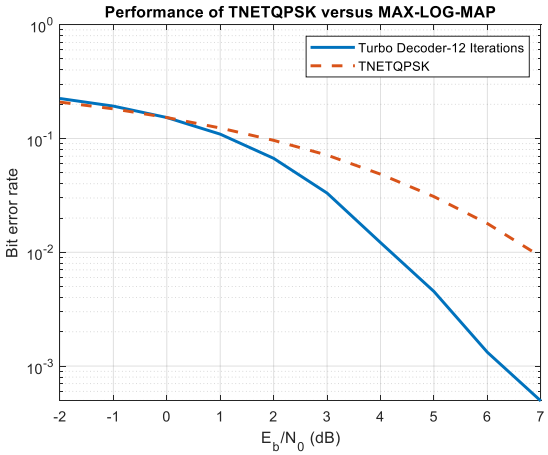

Fig. 13. Performance of TNETQPSK versus Turbo decoder using MAX-LOG-MAP (12 iterations)

4.7 Code-Rate Adaptability for TNETQPSK

Fig. 14 and 15 show the performances of TNETQPSK on low and high code-rates respectively. From Fig. 14, it can be observed that TNETQPSK's BER performance improves with increasing code rate. TNETQPSK's $\frac{E_b}{N_0}$ gain for other code-rates at a BER of 10^{-1} over code-rate $\frac{1}{2}$ is tabulated in **Table** 7, which indicates that the $\frac{E_b}{N_0}$ gain increases for code-rates higher than the one trained on. TNETQPSK also displays a good performance for code-rate 8/17 with an $\frac{E_b}{N_0}$ degradation of only 0.28 dB when compared to code-rate ½. TNETQPSK can thus generalize code rates above $8/17$.

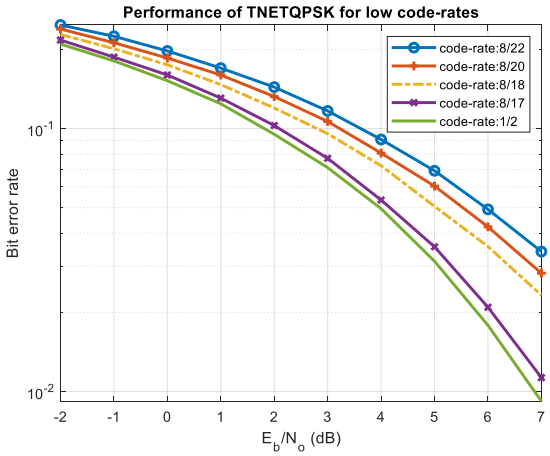

Fig. 14. Performance of TNETQPSK for low code-rates

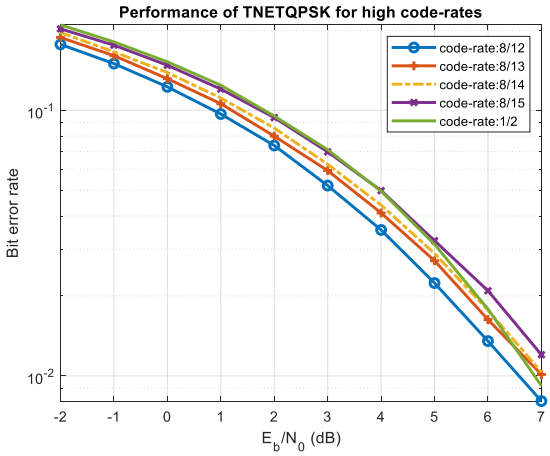

Fig. 15. Performance of TNETQPSK for high code-rates

Table 7. TNETQPSK gain of other code-rates over code-rate ½ performance

Code-rate	8/17	8/15	8/14	8/13	8/12
Gain at BER 10^{-1} concerning code-rate 1/2 performance (dB)	-0.28	0.07	0.405	0.615	0.955

4.8 Generalization to different block sizes for TNETQPSK

Fig. 16 and 17 show the performance of TNETQPSK on short and long block sizes respectively.

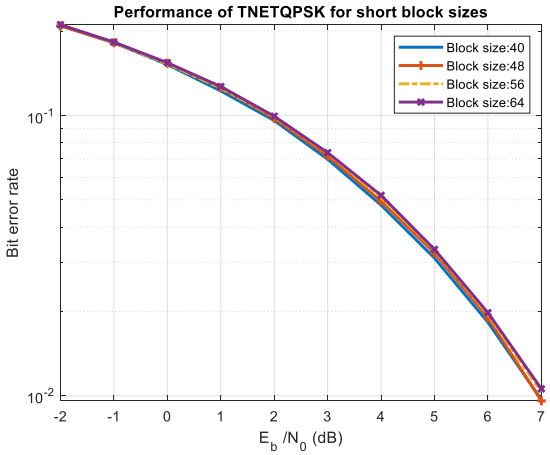

Fig. 16. Performance of TNETQPSK on short block sizes

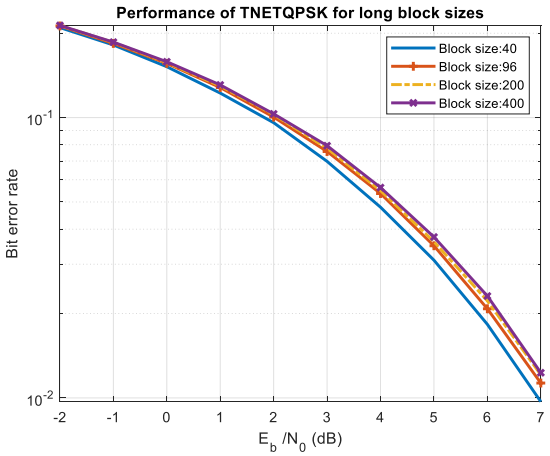

Fig. 17. Performance of TNETQPSK for long block sizes

Fig. 16 indicates little deviation from the BER performance from a block size of 40 for other block sizes. However, the BER performance does not improve with increasing block size as depicted by Fig. for TNET. It is more evident from Fig. 17 that the BER performance degrades slightly with increasing block size. It is observed that the maximum $\frac{E_b}{N_0}$ gain for block size 40 concerning block size 400 is 0.4 dB. As such, it is possible to say that TNETQPSK can generalize fairly to all block sizes tested owing to very slight degradation.

4.9 Training and validation accuracy of DTNET

As the training and validation accuracy fluctuation is very little, the network was taken for further analysis after 4 epochs. Fig. 18 shows the training and validation accuracy for DTNET. DTNET achieves a training accuracy and validation accuracy of 72.46% and 72.76% respectively. This can be explained by the greater sample space of duo-binary turbo codes composed of 2^{96} unique code words for the frame size of 48 bits. As such, using only 1.08 million training examples covers a tiny portion of the sample space.

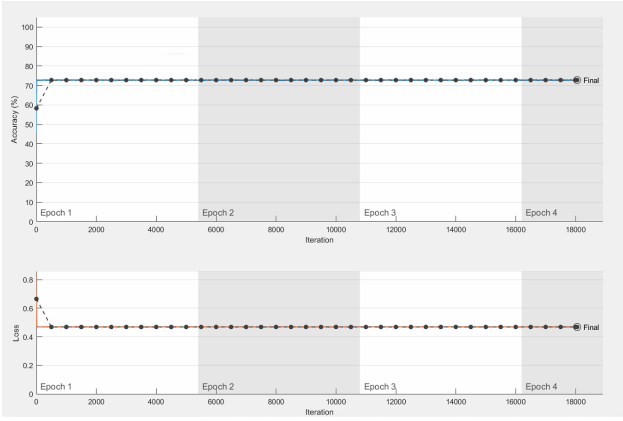

Fig. 18. Training and validation accuracy of DTNET.

4.10 BER Performance of DTNET versus MAX-LOG-MAP algorithm

Fig. 19 shows the BER performance of DTNET against the conventional duo-binary decoder running 12 iterations of the MAX-LOG-MAP algorithm. As was observed for TNET and TNETQPSK, DTNET also shows enhanced BER performance at $\frac{E_b}{N_0}$ values lower than -0.5 dB but the BER does not drop as fast as the duo-binary turbo decoder. This is due to the low accuracy reached during training which suggests

that more data and/or layers of abstraction might be needed for overall BER performance improvement.

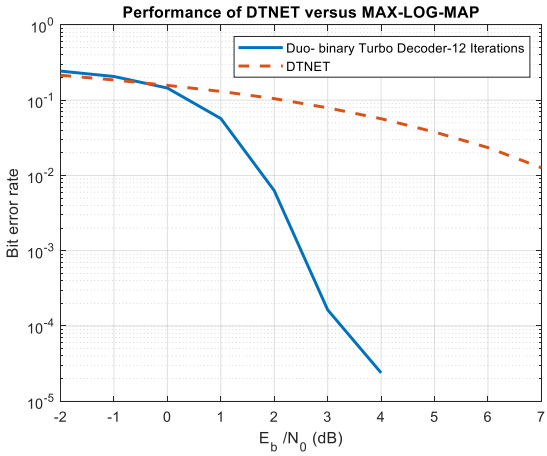

Fig. 19. Performance of DTNET versus MAX-LOG-MAP (12 iterations)

4.11 Code-Rate Adaptability for TNETQPSK

Fig. 20 shows the BER performance of DTNET for varying code rates. Table 8 shows the gain of other code-rates over code-rate ½ performance.

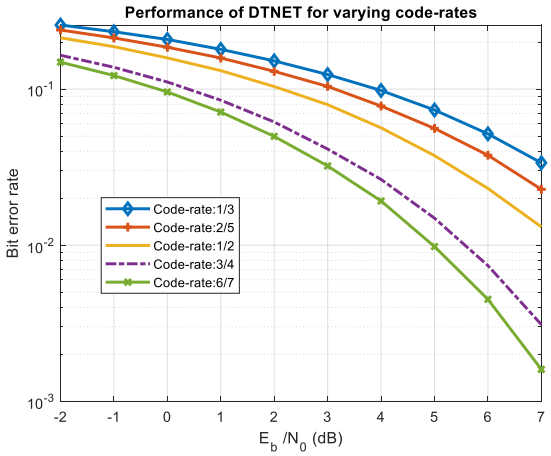

Fig. 20. Performance of DTNET for varying code-rates

Table 8. DTNET gain of other code-rates over code-rate ½ performance

Code-rate	1/3	2/5	3/4	6/7
Gain at BER 10^{-1} concerning code-rate 1/2 performance (dB)	-1.75	-1.00	1.75	2.33

As observed from Fig. 8, DTNET follows the same pattern as TNETQPSK whereby increasing the code rate above the code rate trained on ameliorates the BER performance. As TNET is the only NN that does not generalize well with different code rates, it may be deduced that the modulation scheme used impacts the rate adaptability, as it is common for both DTNET and TNETQPSK simulations.

4.12 Generalization to different block sizes for DTNET

Fig. 21 shows the BER performance of DTNET for varying frame sizes.

As seen in Fig. 21 the BER curves for different frame sizes are closely packed suggesting strong generalization to varying and long frame sizes. As generalization is present, the BER performance of DTNET for various frame sizes can be improved through more data and/or layers. TNET, TNETQPSK and DTNET show good generalization to the block sizes (and frame sizes) tested. This indicates that 2 bi-LSTM layers with at least 200 hidden units each are enough for block size (or frame size) generalization.

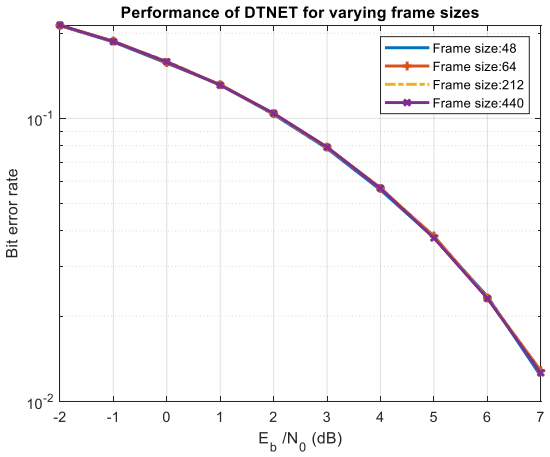

Fig. 21. Performance of DTNET for varying frame sizes

4.13 Comparative Analysis

As an alternative to traditional Turbo decoding techniques, several other systems, apart from RNNS, have been devised in this area. Some of the alternative systems are autoencoders and advanced machine learning-based methods such as graph neural networks (GNNs) and transformer models. Autoencoders perform well when it comes to learning end-to-end representations of communication systems. In comparison RNNs have the ability to provide finer-grained sequence-level modelling which make them a better fit for iterative decoding tasks. Advanced machine learning-based methods, such as graph neural networks (GNNs) and transformer models offer high capacity for parallelisation and complex dependency modelling. However, RNNs have lower computational overhead which make them more suitable in resource-constrained environments. Several studies have demonstrated that optimised RNN-based decoders can achieve comparable or superior decoding performance to these techniques. Moreover, the bi-directional capabilities of bi-LSTM layers in RNN-based decoders enable improved decoding accuracy.

4.14 Trade-offs

The training of RNNs is a resource intensive procedure when considering communication environments with constraints on hardware capabilities. The main cause for the computational cost is due to Backpropagation Through Time (BPTT) involving the computation of gradients for each layer and time step for the whole network. This impacts the demand for processing power and memory when handling long sequences. In addition, RNNs require the storage of intermediate states, gradients and model parameters impacting the memory usage, thereby overloading systems with restricted RAM or storage. Techniques like optimised RNN architectures (e.g. LSTMs and GRUs), quantization and checkpointing can be employed to balance performance and resource consumption. Using specialised hardware accelerators such as GPUs and TPUs have demonstrated to significantly reduce training times and computational costs [28]. Despite these advancements, there is still a strong need for research in the area of real-time communications in resource constrained-environments.

4.15 Practical Implications

RNN-based decoding methods are promising for modern communication systems such as 5G and satellite communications. The main purpose of RNNs is to enhance error correction in ultra reliable low-latency communications (URLLC) and massive machine type communications (mMTC) for efficient and accurate decoding. Turbo codes are required in high speed and high-reliability channels. RNNs fit this purpose together with the ability of modelling complex sequential patterns. Similarly, the robustness of RNN-based decoders can counter the challenges in noisy and high latency environment leading to reliable communications over long distances. The deployment of these techniques has to face a number of challenges in real-world scenarios. The limited resources of edge devices and embedded systems (FPGA or ASIC implementations) can be overwhelmed by the high memory and computational requirements of RNNs. Additionally, meticulous optimisations are required with existing hardware accelerators and the integration of RNNs for ensuring real-time performance under varying channel conditions. Further research is required in hardware-software co-design, model compression techniques and advancements in lightweight RNN architectures for addressing these challenges and paving the way for scalable and efficient AI-driven decoding solutions [29, 30]. Additionally, hybrid solutions combining RNNs with traditional algorithms could offer a pathway to balance performance and efficiency, paving the way for real-time deployment of RNN-based turbo decoders in next-generation communication systems.

4.16 Limitations

Even though RNN-based turbo decoding mechanisms have shown significant advancements, they still have certain limitations. One important challenge is their performance under the high SNR conditions. In these cases, traditional decoding algorithms are observed to perform best due to their deterministic optimisation strategies [31, 32]. The data-driven characteristic of RNN decoders make them struggle to generalise in noisy or biased training data case scenarios leading to suboptimal performance. Moreover, the training of RNN models have high computational complexities when considering long code lengths or high-dimensional inputs. This can be restrictive due to the requirement of extensive computational resources as compared to advanced deterministic decoders like LDPC-specific decoding algorithms [33]. The inherent sequential characteristic of RNNs aids in the decoding of temporal dependencies. However, it can result in higher latency compared to paralellised decoding approaches such as Polar code decoders leveraging successive cancellation list (SCL) algorithms [34]. Thus it can be inferred from these limitations that there is a need for hybrid approaches combining the strengths of RNNs with traditional decoders or even devise novel neural network architectures for robustness and improved efficiency.

4.17 Error Analysis

RNNs have the tendency to deviate from optimal performance when atypical conditions of noise patterns or channels which deviate significantly from the training distribution are encountered. Burst errors which are low probability events but common in practical scenarios of fading channels or interference-dominated environments can cause RNNs to struggle. These situations can be gracefully handled by traditional decoders under certain conditions. At high SNRs, neural networks fail to resolve subtle decoding ambiguities that traditional decoders can handle effectively. By identifying such error patterns, several techniques can be explored as future work. Mechanisms like adversarial training, data augmentation or hybrid decoder designs can be considered for mitigating these error limitations and enhance the robustness.

4.18 Discussions

The experimental results are organized into distinct segments to ensure a clear presentation of findings related to RNN-based turbo decoding. First, the training performance of the proposed decoder is analyzed by examining the accuracy and loss metrics over successive epochs, highlighting the convergence behavior and stability of the training process. Second, the bit error rate (BER) performance is evaluated under various signal-to-noise ratio (SNR) conditions and compared against conventional decoding methods, such as the Max-Log-MAP algorithm, to demonstrate the superiority of the RNN-based approach. This segment provides insights into decoding efficiency and robustness across standard additive white Gaussian noise (AWGN) channels. Finally, generalization capabilities are assessed by testing the trained RNN decoder on non-standard channel models, including fading and burst-error scenarios, as well as unseen code rates and block lengths. This comprehensive evaluation underscores the versatility and adaptability of the proposed decoder in diverse communication environments, ensuring its applicability in real-world systems.

v. Conclusion & Future Works

In this work, the aim was to investigate turbo decoding through RNNs and produce a learned model that can be used for further predictions irrespective of new code rates and block lengths. The binary turbo decoder was replaced with an RNN with 2 bi-LSTM layers having 200 hidden units in each layer. The RNNs trained for BSPK and QPSK were called TNET and TNETQPSK respectively. The NN replacing the duo-binary turbo code was named DTNET and is an RNN with 2 bi-LSTM having 250 hidden units in each layer. For the binary turbo code, the simulations were carried out in MATLAB® using the LTE turbo code. The simulations for duo-binary turbo code were carried out according to the DVB-RCS standards and the AWGN channel was considered for all simulations.

As per the results obtained, TNET attained an accuracy of 94.11% with very little overfitting as the validation accuracy was at 93.85%. TNET showed poor performance when compared to the MAX-LOG-MAP algorithm with 12 iterations. This is most probably due to insufficient training as computation resources were restricted. However, it could adapt to code-rate $\frac{8}{17}$ while being trained on code-rate $\frac{1}{2}$. TNET showed very strong generalisation for longer and similar block sizes as it was performing well on a block size of 400 while being trained on a block size of 40.

After training with the same parameters as TNET, TNETQPSK only reached an accuracy of 84.89% although there were no signs of overfitting with the validation accuracy at 84.63%. The lower accuracy of TNETQPSK led to a poor performance against the MAX-LOG-MAP algorithm with 12 iterations. However, TNETQPSK was able to generalize to all the block sizes tested although it did not continually improve the BER performance with increasing block size. Moreover,

TNETQPSK is generalised to all code rates above $\frac{8}{17}$ whereas TNET is generalised only to code-rate $\frac{8}{17}$.

Using the same amount of data as TNET and TNETQPSK and 250 hidden units, DTNET has a training and validation accuracy of 72.46% and 72.76% respectively. Although its BER performance is poor in comparison to the MAX-LOG-MAP algorithm, DTNET adapts seamlessly to all frame sizes tested. Additionally, DTNET shows improvement in BER for code-rates greater than the one trained on.

An important observation from all the simulations carried out is the fact that the generalization for all block sizes is possible when using 2 bi-LSTM layers with at least 200 units in each layer. It can also be deduced that the modulation scheme used has an impact on the final accuracy of the model. Moreover, adaptation for code rates above the one with which training was performed can be achieved by using QPSK modulation instead of BPSK modulation. Therefore, it is possible to achieve a "rate-less" turbo decoder that can perform decoding on any code rate by training TNETQPSK or DTNET on the lowest code rate available for the particular communication system.

Future works could expand on the training of the RNNs proposed in this work by having more hidden units and/or layers. In addition to the block size and code-rate adaptability of the proposed model, training of the RNN architecture can be performed with more data to obtain a model with higher accuracy and comparable BER performance to the MAX-LOG-MAP algorithm. Another interesting direction of research would be to replace only a portion of the turbo decoder to perform the LLR computations by using bi-LSTM layers.

Transformers and Graph Neural Networks (GNNs) have also emerged from advances in deep learning. Current research has shown that these techniques can become potential candidates for enhancing turbo decoding while overcoming the limitations of RNN-based approaches. Transformers have the ability to capture long-range dependencies and can process entire input sequences in parallel. Thus, the latency is significantly reduced in the decoding tasks as compared to the sequential operations with RNN [35]. The additional benefit of transformers is the ability to handle the decoding of long codewords or Turbo codes with high memory order. GNNs are characterised by their ability to naturally operate on graph-based data. They can thus model the graphical structure or the bipartite representation of parity-check relationships in turbo decoding more efficiently. This leads to GNN-based decoding systems which can learn dependencies between codeword bit and parity checks, thereby improving the bit error rate (BER) performance and the robustness in various channels [36, 37]. The ability to capture complex interactions in the data by these methods allows for better generalisation to fading or correlated noise channel models. Moreover, hybrid architectures of traditional decoding algorithms with transformers or GNNs could help with significant advancements in the Turbo decoding field with new levels of error performances and adaptability for next-generation communication systems.

Acknowledgments

The authors would like to thank the University of Mauritius for the necessary facilities provided for the conduction of this research work.

References

[1] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 623–656, Jul. 1948.

[2] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein and Y. Be'ery, "Deep learning methods for improved decoding of linear codes," *IEEE J. Sel. Topics Signal Process*, vol. 12, no. 1, pp. 119-131, Jan. 2018.

[3] J. Kim, W. Lee and T. Moon, "Communication algorithm design for neural network-based decoding of codes," in IEEE Int. Conf. Commun. (ICC), Vancouver Convention Center, Vancouver, BC, Canada, Feb. 2018, pp. 1-8.

[4] T. Gruber, S. Cammerer, J. Hoydis, and S. T. Brink, "On deep learning-based channel decoding," in *2017 51st Annual Conf. Inform. Sci. Sys.s (CISS)*, Baltimore, MD, USA: IEEE, Mar. 2017, pp. 1–6. doi: 10.1109/CISS.2017.7926071.

[5] Y. Kim, Y. Jiang, R. B. Rana, S. Kannan, S. Oh and P. Viswanath, "Communication algorithms via deep learning," *arXiv*. [Online]. Available: https://arxiv.org/abs/1805.09317. [Accessed: July 2019].

[6] R. Annauth and H. C. S. Rughooputh, "Neural network decoding of turbo codes," in. *Int. Joint Conf. Neural Netw.. Proceed. (Cat. No.99CH36339)*, Washington, DC, USA, 1999, pp. 3336-3341

[7] M. Sazli and M. Husnu, "Neural Network Applications to Turbo Decoding," *ProQuest*. [Online]. Available: https://surface.syr.edu/eecs_etd/105. [Accessed: July 2019].

[8] Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh and P. Viswanath, "Turbo Autoencoder: Deep learning based channel codes for point-to-point communication channels," *arXiv*. [Online]. Available: https://arxiv.org/abs/1911.03038. [Accessed: July 2019].

[9] A. Felix, S. Cammerer, S. Dorner, J. Hoydis and S. T. Brink, "OFDM-Autoencoder for End-to-End Learning of Communications Systems," *arXiv*. [Online]. Available: https://arxiv.org/abs/1803.05815. [Accessed: July 2019].

[10] R. Sattiraju, A. Weinand and H. D. Schotten, "Performance Analysis of Deep Learning based on Recurrent Neural Networks for Channel Coding," in *IEEE Int. Conf. Adv. Netw. Telecommunic. Sys. (ANTS)*, Indore, India, Sep. 2018, pp. 1-6.

[11] He, Yufeng, Zhang, Jing, Wen, Chao-Kai, Jin and Shi, "TurboNet: A Model-driven DNN Decoder Based on Max-Log-MAP Algorithm for Turbo Code," *arXiv*. [Online]. Available: https://arxiv.org/abs/1905.10502. [Accessed: July 2019].

[12] C. Berrou, A. Glavieux and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc. IEEE Int. Conf. Communic.*, Geneva, Sep. 1993, pp. 1064-1070.

[13] 3GPP, "3GPP specification series," *Specifications by Series*. [Online]. Available: https://www.3gpp.org/DynaReport/36-series.htm. [Accessed: 24 April 2020].

[14] M. Synthia and M. S. Ali, "Performance study of turbo code with inter-leaver design," *Int. J. Sci. Eng. Res.*, vol. 2, no. 7, pp. 1-5, Jul. 2011.

[15] L. Bahl, J. Cocke, F. Jelinek and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Transact. Inform. The.*, vol. 20, no. 2, pp. 284-287, Mar. 1974.

[16] S. A. Abrantes, "From BCJR to turbo decoding: MAP algorithms made easier," *Repositório Aberto da Universidade do Porto*. [Online]. Available: https://paginas.fe.up.pt/~sam/textos/From%20BCJR%20to%20turbo.pdf. [Accessed: 23 April 2020].

[17] European Telecommunications Standards Institute (ETSI), "Digital Video Broadcasting (DVB); Interaction channel for satellite distribution systems," *European Standard (Telecommunications Series)*. [Online]. Available: Interaction channel for satellite distribution systems. [Accessed: July 2020].

[18] M. Nielsen, "Neural Networks and Deep Learning," *Neural Networks and Deep Learning*. [Online]. Available: http://neuralnetworksanddeeplearning.com/. [Accessed: April 2020].

[19] D. Rumelhart, G. Hinton and R. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.

[20] R. Grosse, "Lecture 15: Exploding and Vanishing Gradients," *University of Toronto*. [Online]. Available: http://www.cs.toronto.edu/~rgrosse/courses/csc321_2017/readings/L15%20Exploding%20and%20Vanishing%20Gradients.pdf. [Accessed: July 2020].

[21] The MathWorks, Inc., "Long Short-Term Memory Networks," *MATLAB Help Center*. [Online]. Available: https://www.mathworks.com/help/deeplearning/ug/long-short-term-memory-networks.html. [Accessed: 24 April 2020].

[22] M. Phi, "Illustrated Guide to LSTM's and GRU's: A step-by-step explanation," *Towards Data Science.* [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21. [Accessed: 27 April 2020].

[23] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computat.*, vol. 9, no. 8, pp. 1735-1780, Nov. 1997.

[24] D. Bahdanau, K. Cho and Y. Bengio, "Neural Machine Translation by Jointly Learning To Align and Translate," *arXiv*. [Online]. Available: https://arxiv.org/abs/1409.0473.

[25] F. A. Gers, J. Schmidhuber and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," *Neural Computat.*, vol. 12, no. 10, pp. 2451-2471, Oct. 2000.

[26] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceed. Int. Conf. Artif. Intell. Stat. (AISTATS)*, 2010, pp. 249-256.

[27] A. Goldsmith, *Wireless communications*. Cambridge; New York: Cambridge University Press, 2005.

[28] Y. Bengio, P. Simard and P. Frasconi, "Learning Long-Term Dependencies with Gradient Descent is Difficult," *IEEE Transact. Neural Netw.*, vol. 5, no. 2, pp. 157-166, Mar. 1994.

[29] M. Shlezinger, Y. C. Eldar and S. Ten Brink, "Deep Learning for Orthogonal Time Frequency Space (OTFS) Modulation," *IEEE Transact. Wirel. Communic.*, vol. 19, no. 12, pp. 8040-8054, Aug. 2020.

[30] K. N. Rathi, P. S. Rathi and S. W. Kim, "Neural Network-based Turbo Decoding for 5G NR: An Overview," *IEEE Access*, vol. 9, pp. 11267-11279, May 2021.

[31] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier and Xiao-Yu Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Transact. Communic.*, vol. 53, no. 8, pp. 1288-1299, Aug. 2005.

[32] B. Hassibi and H. Vikalo, "On the sphere-decoding algorithm I. expected complexity," *IEEE Trans. Sign. Proc.*., vol. 53, no. 8, pp. 2806-2818, Aug. 2005.

[33] T. Richardson and R. Urbanke, *Modern Coding Theory*, 1st ed. Cambridge University Press, 2008. doi: 10.1017/CBO9780511791338.

[34] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Trans. Inf. Theory*, vol. 61, no. 5, pp. 2213-2226, May 2015.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, "Attention Is All You Need," in Proceedings *Adv. Neural Inf. Process. Syst. (NeurIPS)*, Long Beach, CA, USA, Jun. 2017, pp. 1-15.

[36] Z. Ye, Y. J. Kumar, G. O. Sing, F. Song and J. Wang, "A Comprehensive Survey of Graph Neural Networks for Knowledge Graphs," *IEEE Access*, vol. 10, pp. 75729-75741, Jul. 2022.

[37] V. Ninkovic, O. Kundacina, D. Vukobratovic, C. Häger, and A. G. I. Amat, "Decoding Quantum LDPC Codes Using Graph Neural Networks," in *GLOBECOM 2024 - 2024 IEEE Global Communic. Conf.*, Cape Town, South Africa: IEEE, Dec. 2024, pp. 3479–3484. doi: 10.1109/GLOBECOM52923.2024.10901425.