Realizing Neural Decoder at the Edge with Ensembled Binary Neural Networks

A project thesis

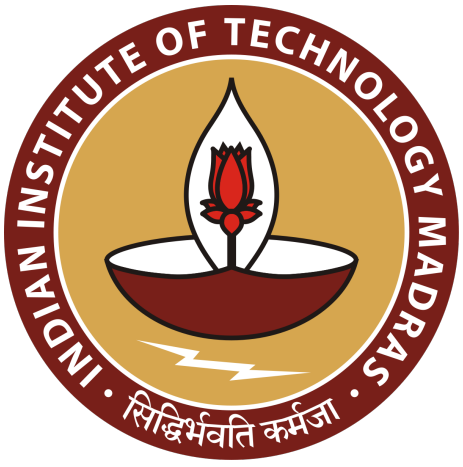
submitted by

Devannagari Vikas

in partial fulfillment of the requirements

for the award of the degree of

Master of Technology



Dept. of Electrical Engineering

IIT Madras

Chennai 600 036

Thesis Certificate

This is to certify that the thesis titled **Realizing Neural Decoder at the Edge with Ensembled Binary Neural Networks**, submitted by **Devannagari Vikas**, to the Indian Institute of Technology, Madras, for the award of the degree of Master of Technology, is a bona fide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Sheetal Kalyani
Project Guide,
Associate Professor,
Dept. of Electrical Engineering,
IIT Madras, 600 036

Place : Chennai
Date : June 2020

Acknowledgements

I am greatly indebted to Dr. Sheetal Kalyani for guiding me through the entire course of my M.Tech project. She always took the time and effort to discuss the problem and to suggest different methods to experiment. Her valuable remarks always gave new directions to my project.

I am thankful to all the professors whose courses helped me improve my knowledge in Wireless Communication and Signal Processing through the course of my two-year M.Tech program. Their classes always inspired me to think beyond classrooms into more practical scenarios.

A special thanks to Nancy and Thulasi who were my fellow associates in doing this project. This project would not have been possible without their contributions and insightful observations. A special word of thanks to all the wonderful people I met at IITM without whom life would not have been the same.

Abstract

Autoencoder is a powerful framework used for channel coding, which is proven to be efficient in many non-canonical channel settings. One such autoencoder is TurboAE, which has learnable interleaved encoder and iterative decoder inspired by the "Turbo Principle". But the TurboAE architecture has nearly $26e5$ parameters and consumes a memory of nearly 20.8 MB for reliable reconstruction of the data sent. Of these, the decoder alone has $25e5$ parameters. Because of this reason, it becomes challenging to deploy it in resource limited edge devices, which are expected to be ubiquitous in future wireless communications such as 6G. To address this issue, we explore the various network compactification techniques and make the neural decoder to be memory and computation efficient, with the following contributions: (a) usage of Binary and Ternary neural networks instead of their real counterpart can save the memory and computations by 64 times, at the cost of some performance degradation; (b) instead of relying on single weak learner such as Binary or Ternary neural networka, if we ensemble more than one such weak learners with the help of bagging, the ensembled network offers almost similar performance as the full precision real network by giving 16 times saving in memory and computation power.

Contents

Acknowledgements **i**

Abstract **ii**

List of Figures **iv**

List of Tables **v**

Abbreviations **vi**

1 Future Wireless Communications **1**

1.1 Introduction 1

1.2 Contribution 3

1.3 Organization of the thesis 4

2 Background **6**

2.1 Post training quantization 6

2.2 Binary neural network 7

2.3 Ternary neural network 8

2.4 Computational savings 8

3 Turbo Autoencoder **10**

3.1 Channel coding 10

3.2 TurboAE with binary and ternary iterative decoder 11

4 Proposed Approach **13**

4.1 Motivation for the proposed approach 13

4.2 Proposed Ensembled TurboAE-Bagging 13

5 Experiments and Results **15**

5.1 Experiments 15

5.2 Computation and memory savings at the edge devices 18

6 Conclusion **20**

List of Figures

3.1 TurboAE interleaved encoder (left), Channel (middle) and TurboAE iterative decoder (right) with block rate $\frac{1}{3}$. $g_\phi^v = g_\phi^b$ for BinTurboAE and $g_\phi^v = g_\phi^t$ for TernTurboAE. S in the decoder side is a sigmoid block 11

4.1 Architecture of the decoder of En-TurboAE-Bag: the final estimate $\hat{u}$ is the aggregate of $B = 4$ weak learners 14

5.1 Performance of Binary and Ternary networks compared to the quantized and real valued TurboAE 17

5.2 Performance of Ensembled versions compared to the binary and ternary TurboAE 17

List of Tables

5.1 Hyper-parameters of TurboAE 16
5.2 Savings vs performances at the edge device 18

Abbreviations

AI Artificial intelligence. 1

AWGN Additive White Gaussian Noise. 2

BCJR Bahl-Cocke-Jelinek-Raviv. 11

BER Bit Error Rate. 2

BLER Block Error Rate. 10

BNN Binary Neural Network. 7

CNN Convolutional neural network. 2

FLOP Floating-point operations. 8

IoT Internet-of-Things. 1

LDPC Low Density Parity Check. 2

MIMO Multiple-In Multiple-Out. 1

RNN Recurrent neural network. 2

RSC recursive systematic convolutional. 11

SIMD Single instruction, multiple data. 9

SWAR SIMD within a register. 9

TNN Ternary Neural Network. 8

TurboAE Turbo Autoencoder. 2

Chapter 1

Future Wireless Communications

1.1 Introduction

The future generations of wireless communication systems, or 6G, will provide a platform for a fully connected world. They are expected not only to be equipped with multi-band high speed transmission but also energy-efficient communication, low latency and high security. One of the technological breakthroughs to achieve the connectivity within 6G include the pervasive Artificial intelligence (AI) [1].

The field of AI has seen an enormous growth, which helps it find a way in many applications like object recognition, self driving cars and so on. In the context of wireless communications, AI can be applied to channel estimation and symbol detection which are extremely useful in the massive Multiple-In Multiple-Out (MIMO) communications. We can deploy artificial intelligence at each layer of the wireless network. For instance, we can use the machine learning algorithms to better adapt the network resources for various scenarios. At the physical layer, deep learning can be applied to modulation and coding schemes. They also assist with channel estimation.

With the advancement of research in artificial intelligence for 6G, the high speed data will allow an ubiquitous utilization of Internet-of-Things (IoT) containing billions of devices [2]. The interactions among themselves and with the access providers may result in excessive signal processing at the user end that gives rise to huge power consumption. Therefore economic energy usage to have elongated battery life in mobile devices has been a research direction with utmost importance [3, 4].

The frequent data exchange between the users and the access providers will

expose a risk to the communication. To prevent the data from being leaked or getting corrupted by channel noise, different physical layer encryption algorithms [5–7] are used as channel coding methods. In digital communication system, the aim of channel coding technique is to combat the effect of noise in the channel and to successfully reconstruct the message that was sent over the noisy channel. Finding an optimal code is of great interest from the point of view of both wireless communication [8] and information theory [9]. But finding the same for a non-canonical channel is challenging because the codes for canonical setting are adapted via heuristics to these channels. For the simple codes [10] designed by optimizing the minimum code distance, an optimal MAP decoder is computationally simple and it reduces the Bit Error Rate (BER). But the MAP decoders for the near-optimal codes like Low Density Parity Check (LDPC) [5], Turbo [11] and Polar [6] codes that achieve a capacity of Shannon's limit on Additive White Gaussian Noise (AWGN) channels, are computationally inefficient. The design of optimal decoders for canonical AWGN channel is completely handcrafted. When the channel deviates from the Gaussian setting in a practical scenario, time heuristic are used to design the decoder and it does not always exploit the power of encoders.

To fill up this gap, Neural Networks have been used to design the decoder while encoder is fixed as a near-optimal code [12]. Neural decoders have gained immense interest; for instance one that mimics the belief propagation based decoders for LDPC [13] and Polar codes [14] or Convolutional neural network (CNN) [11] and Recurrent neural network (RNN) [15] based decoders for optimal decoding of Convolutional and turbo codes. Deploying decoders for these codes takes up huge computation which is only possible because of recent advancement in signal processing. Though, the neural decoder alone have shown good performance in noisy channel, encoding has been challenging using this paradigm. Therefore designing neural code has been proposed where the encoder and decoder are jointly trained [16, 17]. Neural codes have shown results better than many state of the art codes but could not reach to a level of capacity achieving codes. Besides, these kind of joint optimization methods sometime leads to convergence at local minimum. To overcome this issue, the authors proposed, Turbo Autoencoder (TurboAE) [18] that uses CNN based over-complete auto-encoder model incorporating interleavers and de-interleavers to

achieve the performance of state of the art channel codes under AWGN scenario.

However, all the existing neural autoencoders have real valued network parameters and perform floating point operations during deployment. For instance, the TurboAE architecture has nearly $26e5$ parameters that takes up a memory of 20.84 MB considering a 64 bit floating point representation. Out of total $26e5$ parameters, the encoder has nearly $1.5e5$ parameters whereas the decoder has nearly $25e5$ parameters. Because of the huge number of parameters in the autoencoders, deploying it in a resource limited IoT setup is a challenging task. Furthermore, with the advent of edge computing in IoT scenario, the computation is decentralised to edge devices where the data is processed locally. Even though the encoder at the base station can take the load of heavy processing, the neural decoder is at the user end with limited memory and computing power and cannot withstand bulky processing of the edge devices.

Deep neural networks have been extensively used in real world machine learning applications like image/object recognition, self driving cars, mobile gadgets or smart home assistance and so on. If the networks are bulkier, they can not be easily deployed at the edge devices. Different model compression methods like pruning, quantization, knowledge distillation, efficient model design, tensor decomposition [19–22] etc have been proposed to reduce the memory requirement and the number of FLOP counts. These methods have encouraged the deployment of deep learning techniques for the real world applications. Binary Neural Networks [23] take this compression to the extreme level by taking the weights and activations to be 1-bit. Binary networks replace 64 bit floating points with 1 bit that gives a memory reduction of 64 times. Also the floating point multiply and addition operations are replaced with *xnor* and *popcount* operations. This saves the computation cost radically during the inference time.

1.2 Contribution

In the domain of wireless communication, the channel noise is real valued and till now, only the real valued Neural Decoders have been used for end-to-end training that had only floating point operations. In this work, we explore the possibilities of

using extreme compactification techniques in wireless applications where a simple coarser quantization on the trained network tremendously degrades the performance of the Neural Decoder. In this work, we also propose the techniques that allow the decoder to be memory and computation efficient but still have a performance as close as to the original neural decoder. The major contributions of our work are the following:

- The effect of quantizing the trained neural decoder to different levels after training are shown. The performances are compared to the original real valued TurboAE.

- We have proposed to use binary filters/weights/biases and binary activations in order to save in memory and computation at the edge.

- The performance is further improved by the use of ternary neural network where the weights take three levels $\{-1, 0, +1\}$ but the activation is still binary. The proposed architectures with binary and ternary weights are shown to be better than one where the trained network is quantized with 2 bit or 1 bit.

- An ensemble of multiple weak binary and ternary decoders is shown to perform close to the real valued TurboAE but achieve nearly 16 times saving in memory (if we consider four weak learners) and nearly 64 times speed up due to less computations that helps to achieve energy efficiency and low latency in the edge communication.

1.3 Organization of the thesis

This thesis is organized as follows:

*Chapter*2 reviews the topics involved in quantization techniques implemented on the neural networks in order to achieve the desired savings at the edge device end

*Chapter*3 discusses about the role of TurboAE in tackling the problem of channel

coding and then introduces the concept of binary and ternary iterative decoders

*Chapter*4 explains the proposed ensembled TurboAE using the bagging of weak learners such as binary and ternary networks

*Chapter*5 illustrates the simulation setup used for the experiments. Performance of different networks are analyzed in terms of BER graphs

*Chapter*6 summarizes the work done and provides some concluding remarks

Chapter 2

Background

In this chapter, we review the quantization techniques that are implemented on the neural networks so as to achieve compactification at the edge device. We denote a real valued neural network (NN) by $g_{\boldsymbol{\phi}}(.)$ where $\boldsymbol{\phi}$ represents the real valued network parameters. The output from the NN is given by $\mathbf{y} = g_{\boldsymbol{\phi}}(\mathbf{x})$ where $\mathbf{x}$ is the input features to the NN and can be real valued. The neural network $g_{\boldsymbol{\phi}}(.)$ can be of any type: a fully connected, a CNN or an RNN. In our work, as we mostly deal with CNN in the neural decoder, we discuss the following part with respect to the CNNs.

For a general CNN $g_{\phi}(.)$ of L layers, the parameters are nothing but the filters of the CNN and are given by $\phi = \{\mathbf{W}_1, \ldots, \mathbf{W}_L\}$ where $\mathbf{W}_l \in \mathbb{R}^{c_o \times c_i \times k}$ for l^{th} layer of one dimensional CNN. Here c_i and c_o represents number of input and output channels and k is the dimension of the filter. For a one dimensional CNN as used in TurboAE, if the input to l^{th} layer of CNN has spatial features of dimension h_{in}, then input to l^{th} layer is $\mathbf{a}_l \in \mathbb{R}^{c_i \times h_{in}}$. The output of l^{th} layer is $\mathbf{a}_{l+1} \in \mathbb{R}^{c_o \times h_{out}}$ where h_{out} is the dimension of the output.

2.1 Post training quantization

Post training quantization is an optimizing strategy which is used to make the model smaller and improve the CPU latency, which is important for low-power devices. But with this strategy, the models experience some degradation in the accuracy. There are several ways to choose from, to implement post training quantization. The simplest way is to quantize the full precision floating point number, such as weights and activations, to that of a number with 8-bit, 4-bit, 2-bit and 1-bit precision. That

means we are trying to represent the full precision number using 256, 16, 4 and 2 quantization levels respectively.

If a real valued network $g_{\phi}(.)$ is deployed in a 64 bit system, then its 8-bit, 4-bit, 2-bit and 1-bit quantized versions will occupy 8, 16, 32, 64 times lesser memory. However as we have discussed, the performance generally degrades to some extent.

2.2 Binary neural network

In a Binary Neural Network (BNN), the weights $\mathbf{W}$'s and activations $\mathbf{a}$'s are binarized using the *sign* function before taking convolution.

$$sign(r) = \begin{cases} +1, & \text{if } r \geq 0 \\ -1, & \text{otherwise.} \end{cases} \tag{2.1}$$

The binarized parameter $\mathbf{W}_l^b$ and $\mathbf{a}_l^b$ is given by:

$$\mathbf{W}_l^b = sign(\mathbf{W}_l) \tag{2.2}$$

$$\mathbf{a}_l^b = sign(\mathbf{a}_l) \tag{2.3}$$

So the real valued convolution is approximated with binary weights and activations as

$$\mathbf{W}_l * \mathbf{a}_l \approx \mathbf{W}_l^b \circledast \mathbf{a}_l^b \tag{2.4}$$

where $\circledast$ is convolution performed with bitwise operations. Even though the binarized weights are used for the forward pass, only the real valued latent weights are updated with the real valued gradients during backpropagation. However during inference, these latent weights can be dropped and binary network with the binary weights and activations can be deployed. The *sign* function is non-differentiable and has gradients as zero almost everywhere; thus not appropriate for the back propagation during the training. Therefore a straight through estimator [24] was proposed that binarizes in the forward pass but during backprop it just passes the gradients as it is to the previous layers. For instance, if $q = sign(r)$, then $g_r = g_q \mathbf{1}_{|r| \leq 1}$ where $g_r = \frac{\partial C}{\partial r}$, $g_q = \frac{\partial C}{\partial q}$ and C is the cost function of the NN. To have a stable update during the training, the updated real valued weights are clipped between $[-1, 1]$.

If a real valued network $g_{\boldsymbol{\phi}}(.)$ is deployed in a 64 bit system, then its binary version will occupy 64 times lesser memory and all the floating point operations can be converted to just xnor and popcount operations. However, because of this extreme compactification, the performance generally degrades very much.

2.3 Ternary neural network

Ternary Neural Network (TNN) [25] on the other hand, is proposed as an alternative to BNN, where 3 bits $\{-1, 0, 1\}$ are used unlike two bits $\{-1, 1\}$ in BNN. Therefore the ternarized parameter q is given by:

$$q = tern(r) = \begin{cases} +1, & \text{if } r > \Delta \\ 0, & \text{if } r < |\Delta| \\ -1, & \text{if } r < -\Delta \end{cases} \tag{2.5}$$

where $\Delta \approx 0.7 \cdot \mathrm{E}(|\mathbf{W}|)$, $\mathbf{W}$ indicate full precision weights.

The introduction of zero as another bit along with $\{+1, -1\}$ gives a better representation power and therefore better performance than BNN. But the zero weights need not to be saved during deployment. So the memory requirement of TNN is same as that of the BNN. Note that the activation is still binary and thus the computational complexity is also same as the BNN. Therefore with TNN, an improve in performance is possible without any additional requirement in memory or computation.

2.4 Computational savings

The convolution between real valued $\mathbf{W}_l \in \mathbb{R}^{c_o \times c_i \times k}$ and $\mathbf{a}_l \in \mathbb{R}^{c_i \times h_{in}}$ at l^{th} layer results in an output $\mathbf{a}_{l+1} \in \mathbb{R}^{c_o \times h_{out}}$. The total number of multiplication for l^{th} layer is $c_i \times k \times h_{out} \times c_o$ and the total number of addition for l^{th} layer is $(c_i - 1) \times (k - 1) \times h_{out} \times c_o$. The total Floating-point operations (FLOP) count for l^{th} layer of a real valued 1D-CNN is the summation of the number of multiplication and addition that is roughly twice of the number of multiplication, it is given by $2 \times c_i \times k \times h_{out} \times c_o$. For a binary counterpart, as the weights and activations are constrained to -1 or $+1$, the 64 bit floating point multiply-accumulation operations

are replaced by 1 bit XNOR-count opertions [23]. Note that the modern CPUs can perform a single multiplication and an addition in a single clock cycle, and thus the total number of operations in a binary network is $c_i \times k \times h_{out} \times c_o$. In recent 64-bit CPUs, 64 such binary operations can be performed in a single clock cycle that gives a speed up of nearly 64 times in a binary or ternary network [26]. Because the filters take only $+1$ or -1, only a limited number of filters are possible. So with BNN, the filter repetition can be exploited by using dedicated hardware/software. The implementation on GPU can be made faster by using Single instruction, multiple data (SIMD) within a register SIMD within a register (SWAR) where 64 binary variables are concatenated in a 64 bit register and a 64 times speedup on the bitwise operation like XNOR can be achieved.

Chapter 3

Turbo Autoencoder

3.1 Channel coding

The method of channel coding can be divided into three sub-problems: an encoder $f_\theta(.)$ at the transmitter, a channel $c(.)$ and a decoder $g_\phi(.)$ at the receiver. In a communication system, the encoder $x = f_\theta(u)$ encodes the binary bits $\mathbf{u} = (u_1, \ldots, u_K) \in \{+1, -1\}^K$ of block length K to get the codeword $\mathbf{x} = (x_1, \ldots, x_N)$ of length N such that the codeword satisfies the power constraints. The code rate is $R = \frac{K}{N}$, where $N > K$. The i.i.d. AWGN channel corrupts the encoded bits and generates $z_i = x_i + w_i$ such that $w_i \sim \mathcal{N}(0, \sigma^2)$ for $i = 1, \ldots, K$. The noise in the AWGN channel is represented by the signal to noise ratio $\text{SNR} = -10 \log_{10} \sigma^2$. After transmission through the channel, the decoder $g_\phi(z)$ receives the real valued noisy encoded bits z and map them to an estimate of the actual message sequence $\hat{\mathbf{u}} = (\hat{u}_1, \ldots, \hat{u}_K) \in \{+1, -1\}^K$ using either MAP principle or handcrafted decoding algorithm to have the least bit error rate (BER).

Channel coding aims to minimize the Bit Error Rate (BER) or the Block Error Rate (BLER) of the recovered message signal $\hat{u}$ given by $BER = \frac{1}{K} \sum_1^K Pr(\hat{u}_i \neq u_i)$ and $BLER = Pr(\hat{\mathbf{u}} \neq \mathbf{u})$.

Naively applying deep learning models by replacing encoder and decoder with general purpose neural networks does not perform well. So in [18], authors have proposed a TurboAE with interleaved encoding and iterative decoding using 1D convolutional neural networks. We propose to binarize the iterative decoder of TurboAE and inspect its performance in the following chapters. The basic architecture of TurboAE and the same of its binary and ternary versions are discussed here.

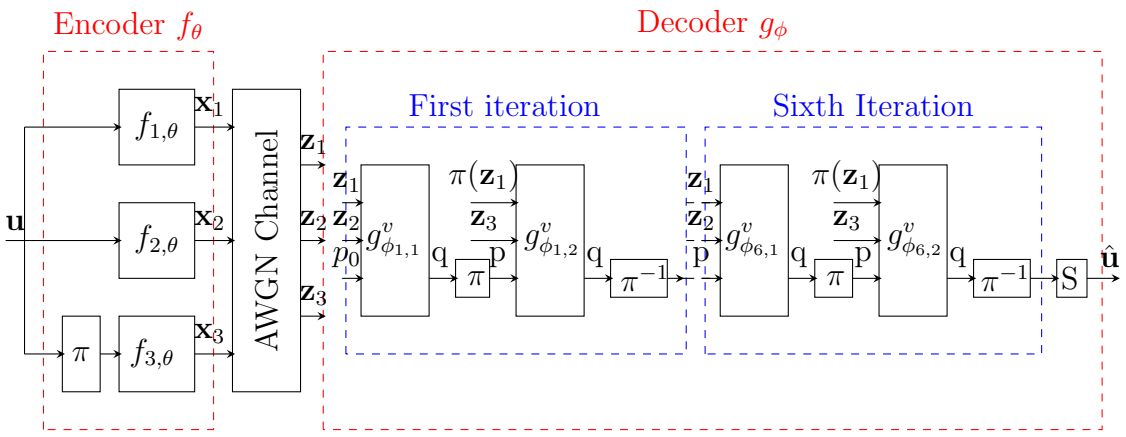


Figure 3.1: TurboAE interleaved encoder (left), Channel (middle) and TurboAE iterative decoder (right) with block rate $\frac{1}{3}$. $g^v_\phi = g^b_\phi$ for BinTurboAE and $g^v_\phi = g^t_\phi$ for TernTurboAE. S in the decoder side is a sigmoid block

3.2 TurboAE with binary and ternary iterative decoder

Turbo code is one of the first capacity approaching codes based on recursive systematic convolutional (RSC) code that has an optimal decoding algorithm namely the Bahl-Cocke-Jelinek-Raviv (BCJR). To add long range memory to the code, interleaving is used: out of two copies of input bits, the first one passes through the RSC code and the second goes through the interleaver before passing through the same RSC code as shown in Fig. 3.1(left). After the transmission through the channel, this code is then decoded by repeating alternatively between (i) and (ii): where (i) refers to the soft decoding based on the signal received from the first copy and (ii) indicates the de-interleaved version as a prior to decode the second copy as shown in Fig. 3.1(right). This iterative decoding method keeps re-estimating the posterior distribution on the transmitted bits.

Both the interleaved encoder and the iterative decoder are learnable as proposed in TurboAE [18]. The interleaver $\mathbf{x}^\pi = \pi(\mathbf{x})$ and the de-interleaver $\mathbf{x} = \pi^{-1}(\mathbf{x}^\pi)$ shuffles and shuffles back the input sequence with a random interleaving array known to both encoder and decoder respectively. A code rate of 1/3 is considered for the interleaved encoder f_θ that has three learnable blocks $f_{1,\theta}, f_{2,\theta}$ and $f_{3,\theta}$. The first

two takes the original message bit $\mathbf{u}$ to produce $\mathbf{x}_1$ and $\mathbf{x}_2$ whereas the third block takes the interleaved message $\pi(\mathbf{u})$ to return $\mathbf{x}_3$ as shown in Fig. 3.1. The encoded messages then pass through the channel and the received noisy messages are $\mathbf{z}_1$,$\mathbf{z}_2$ and $\mathbf{z}_3$. Our focus is mostly on the compression of the iterative decoder part so that it can be deployed at the edge devices. Thus we do not discuss much on the encoder part in this work. Interested readers may refer to [18] for more details on the encoder.

Considering M iterations of the iterative decoder, each iteration consisting of two decoders. First decoder in i^{th} iteration $g_{\phi_{i,1}}(.)$ takes the original noisy message $\mathbf{z}_1, \mathbf{z}_2$ and the prior distribution p on the transmitted bits and returns a posterior q that goes to the second decoder $g_{\phi_{i,2}}(.)$ via interleaving along with the interleaved noisy messages $\pi(\mathbf{z}_1)$ and $\mathbf{z}_3$. In the proposed binarized and ternarized TurboAE, named as *BinTurboAE* and *TernTurboAE*, the real valued decoders $\{g_{\phi_1}, \ldots, g_{\phi_M}\}$ are replaced with binary decoders $\{g^b_{\phi_1}, \ldots, g^b_{\phi_M}\}$ and ternary decoders $\{g^t_{\phi_1}, \ldots, g^t_{\phi_M}\}$ respectively. For ease of notation, we represent the complete binary decoder by g^b_ϕ and the ternary decoder by g^t_ϕ. But the main limitation of BinTurboAE and TernTurboAE is that they do not perform as good as the real valued TurboAE. In applications where a degradation in performance is acceptable at the cost of reduced computation and energy efficiency, BinTurboAE or TernTurboAE can be deployed at the Edge devices. As the performance of BinTurboAE is not as good as the real counterpart, each of these can be thought of as a single weak learner. Instead of relying on a single weak learner, we propose to ensemble a set of weak learners' outcomes to have a much reliable output which we will discuss in the next chapter.

Chapter 4

Proposed Approach

4.1 Motivation for the proposed approach

As discussed in the introduction chapter, the decoder section of TurboAE has huge number of network parameters and involves floating point operations. This makes it challenging to deploy this decoder in resource limited edge devices. To make this happen, we can replace the CNNs in the decoder with the BNNs, neural networks with binary weights and activations as discussed in chapter two. On the other hand, we can also replace CNNs in the decoder with TNNs. Usage of binary and ternary networks indeed reduces the memory requirement and the computations with the ternary network performing better than that of a binary network. But both of these networks suffer a great deal of performance degradation when compared with that of original network.

To reduce this gap, we propose to use the Ensemble Neural Network [27] instead of single BNN or single TNN. Basic idea behind Ensemble technique is to improve the accuracy of any network by combining many weak learners and to make it perform close to the full precision network, so that we can use it in the edge device. The two common ensemble strategies used are bagging and boosting. Here we use bagging to ensemble multiple binary or ternary networks.

4.2 Proposed Ensembled TurboAE-Bagging

Considering each decoder g_ϕ^v a weak learner, B such weak learners are trained separately with the complete dataset. The idea of "ensemble" is to get opinion from all

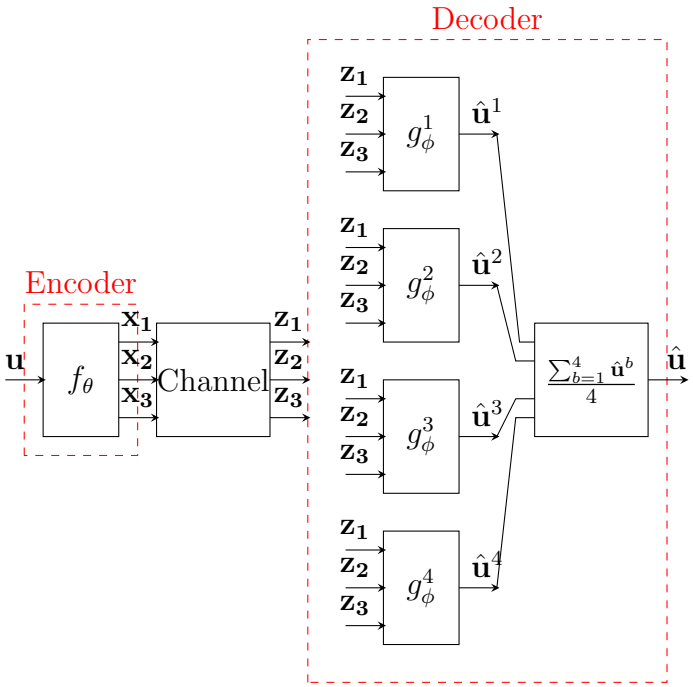


Figure 4.1: Architecture of the decoder of En-TurboAE-Bag: the final estimate $\hat{u}$ is the aggregate of $B = 4$ weak learners

these weak learners to arrive at better prediction. One of the many ways the weak learners can be ensembled is *Bagging*. The key idea of bagging is to average the weak classifiers. In this work, we have proposed to ensemble $four$ BinTurboAEs with the bagging method and it is called *BinTurboAE-Bag*. The same is done with TernTurboAE and it is named as *TernTurboAE-Bag*. Bagging is used in machine learning to improve stability and accuracy and to reduce the variance. Here in our work, the decisions from each one of these $four$ BinTurboAEs or $four$ TernTurboAEs ($\{\hat{u}^1, \ldots, \hat{u}^4\}$) are averaged to get the final prediction $\hat{\mathbf{u}} = \frac{1}{4}\sum_{b=1}^{4} \hat{\mathbf{u}}^b$ as shown in Fig. 4.1.

Chapter 5

Experiments and Results

5.1 Experiments

To validate the usefulness of the proposed compression techniques, we consider the setting used in [18] to train the encoder and decoder networks. A large batch size preferably greater than or equal to 500 is used to average the channel noise effects. We train encoder and decoder separately so as to avoid any possible local optima. BinTurboAE and TernTurboAE need lower learning rate than the real valued TurboAE. Hence we reduced the learning rate by 10 times whenever the validation loss gets saturated for higher training epochs. The hyper-parameters used in our experiment are shown in Table 5.1.

We provide results showing the performance in terms of BER vs SNR of the proposed BinTurboAE and TernTurboAE and compare them with QuantTurboAE, the quantized TurboAE to q levels after the training. For QuantTurboAE, the parameters of the trained TurboAE is quantized to different levels i.e. 8-bit, 4-bit, 2-bit and 1-bit. The saving in memory is 8, 16, 32 and 64 times respectively compared to the real valued TurboAE network as shown in Table. 5.2. However, this method does not save at all from the computation point of view. The 8-bit quantization after the training performs as good as the original TurboAE. But the 2-bit and 1-bit quantizations performs really bad as shown in Fig. 5.1. But instead of quantization after the training, if the network is trained with 1-bit quantization like the BinTurboAE, the network outperform 2-bit and 1-bit QuantTurboAEs. The Ternary network improves the BER performance even more by 0.5dB and performs similar to QuantTurboAE ($q = 4$). QuantTurboAE ($q = 4$) uses 4 bits to store

Table 5.1: Hyper-parameters of TurboAE

Loss	Binary Cross-Entropy (BCE)
Encoder	2 layers 1D-CNN, kernel size 5, 100 filters for each learnable encoding block
Decoder	5 layers 1D-CNN, kernel size 5, 100 filters for each learnable decoding block
Decoder Iterations	6
Info Feature Size F	5
Batch Size	500
Optimizer	Adam
Learning Rate	initially 0.0001 and reduced by 10 when test loss saturates for more number of epochs
Block Length K	100
Number of Epochs	800

each parameter whereas the TernTurboAE uses only 1 bit to store each parameter. And when compared to the real valued TurboAE, both the binary and the ternary networks save the memory requirement by about 64 times and the computations by converting all the floating point computations to xnor and pop-count operations at the decoder side. The performance gap between these proposed methods and the TurboAE still exists and needs ones attention. To close this gap, four such weak learners are ensembled and its performance is shown in Fig. 5.2.

The ensemble of just $four$ BinTurboAEs is implemented with the bagging method and it performs much better than that of a single BinTurboAE. The BinTurboAE-Bag even outperforms the real network in low SNR region by almost 1 dB as shown in Fig. 5.2. In the high SNR region, the BinTurboAE-Bag performs close to the real TurboAE. Now, the ensemble of four $four$ TernTurboAE is implemented with the same bagging method to see how this ensemble performs. The performance of TernTurboAE-Bag is slightly better than BinTurboAE-Bag as shown.

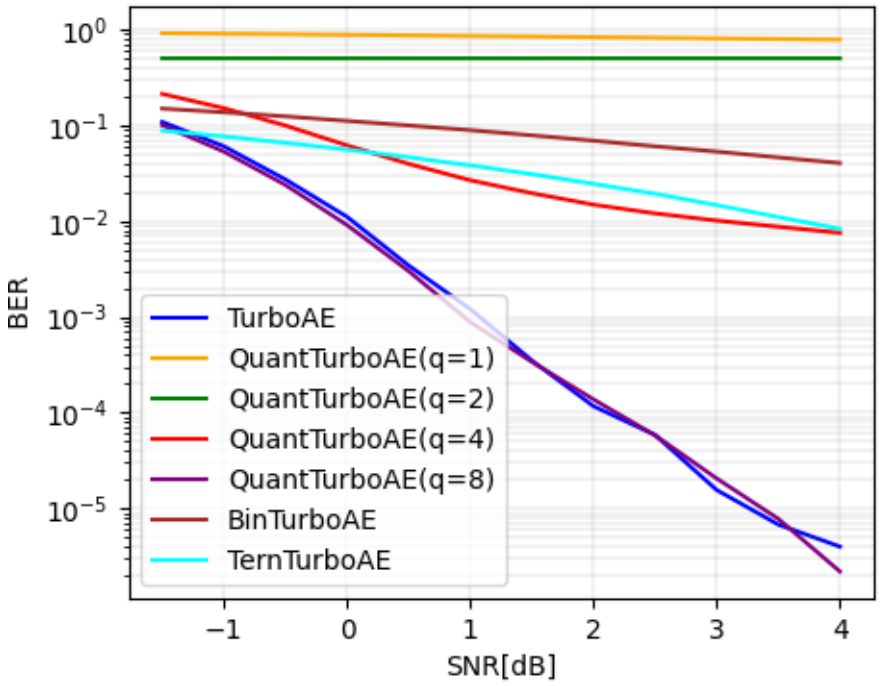


Figure 5.1: Performance of Binary and Ternary networks compared to the quantized and real valued TurboAE

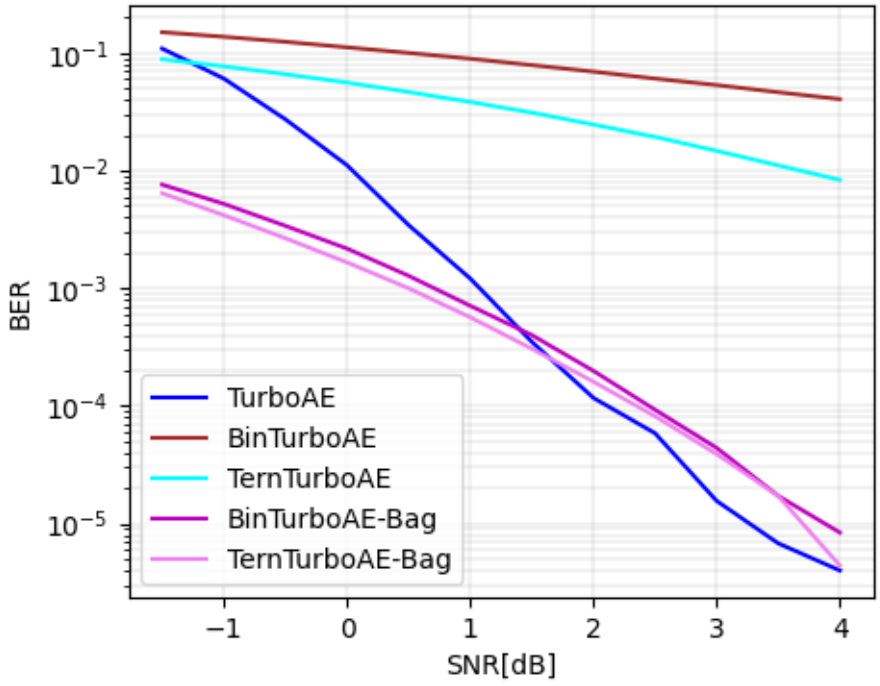


Figure 5.2: Performance of Ensembled versions compared to the binary and ternary TurboAE

This result is significant as the (Bin/Tern)TurboAE-Bag saves a lot in terms of the memory requirement (about $64/4 = 16$ times) and the number of computations (FLOPs are replaced with XNOR-count) at the edge device end but still not compromising much on the BER performance when compared to the TurboAE with real parameters.

5.2 Computation and memory savings at the edge devices

Decoding usually happens at the edge device. In real TurboAE, the iterative decoder has huge number of parameters that takes up a lot of memory. It also involves floating point operations thus making the computations slow at the edge devices. Our main goal is then to reduce the memory requirement and computations at the decoder side of the TurboAE so that the proposed decoders are suitable for deployment at the edge.

Table 5.2: Savings vs performances at the edge device

Model	**Memory savings**	**Computation**	**Speed up**	**BER at** 0 **dB SNR**
Full precision DNN	1x	$\simeq 4e8$ FLOPs	1x	$1e-02$
QuantTurboAE ($q = 4$)	$\simeq$ 16x	$\simeq 4e8$ FLOPs	1x	$6e-02$
BinTurboAE	$\simeq$ 64x	$\simeq 4e8$ xnor-count	64x	$1e-01$
TernTurboAE	$\simeq$ 64x	$\simeq 4e8$ xnor-count	64x	$6e-02$
Bin/Tern-TurboAE-bag ($B = 4$)	$\simeq$ 16x	$\simeq 16e8$ xnor-count	16x (64x if parallel processing is possible)	$1e-02$

The savings for each of the proposed techniques are shown in Table. 5.2. BinTurboAE and TernTurboAE takes up memory 64 times lesser than the real valued TurboAE. BinTurboAE-Bag and TernTurboAE-Bag takes a memory B times the memory taken by the BinTuboAE and TernTurboAE.

The number of FLOPs in the decoder of the real TurboAE at the edge devices is about $4e8$. Even though the memory savings in q bit Quantized network would be around $(64/q)$ times the real networks requirement, QuantTurboAE and TurboAE does not speed up the computations as the computations are still in 64 bit. As the Binary, Ternary and the Ensembled TurboAEs convert all the $4e8$ floating point operations to only bitwise operations, the computations are extremely fast with much lower power consumption. When 64 bitwise operations are performed in a single clock cycle, then the binary and ternary networks are 64 times faster thus gives very low latency than the real TurboAE network. Even though the computation in (Bin/Tern)TurboAE-Bag is B times more than the (Bin/Tern)TurboAE, the (Bin/Tern)TurboAE-Bag can be made equally fast like (Bin/Tern)TurboAE, if parallel processing is available at the edge as shown in Table. 5.2.

Chapter 6

Conclusion

In summary, we propose to use BinTurboAE and TernTurboAE with the aim of implementing the end-to-end learnt channel coding in the targeted low-power edge devices by reducing the memory requirement and the computations by nearly 64 times at the cost of performance degradation. We then propose BinTurboAE-Bag and TernTurboAE-Bag to improve the performance offered by a single BinTurboAE or single TernTurboAE respectively and achieve the performance close to the real network. The ensembled technique implemented with $four$ such weak learners is shown to consume 16 times less memory and the computation power, without suffering much loss in accuracy and perform close to the original real valued TurboAE.

Bibliography

[1] I. F. Akyildiz, A. Kak, and S. Nie, "6g and beyond: The future of wireless communications systems," *IEEE Access*, vol. 8, pp. 133 995–134 030, 2020.

[2] M. H. Alsharif, A. H. Kelechi, M. A. Albreem, S. A. Chaudhry, M. S. Zia, and S. Kim, "Sixth generation (6g) wireless networks: Vision, research activities, challenges and potential solutions," *Symmetry*, vol. 12, no. 4, p. 676, 2020.

[3] R. Mahapatra, Y. Nijsure, G. Kaddoum, N. U. Hassan, and C. Yuen, "Energy efficiency tradeoff mechanism towards wireless green communication: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 686–705, 2015.

[4] C. Huang, A. Zappone, G. C. Alexandropoulos, M. Debbah, and C. Yuen, "Reconfigurable intelligent surfaces for energy efficiency in wireless communication," *IEEE Transactions on Wireless Communications*, vol. 18, no. 8, pp. 4157–4170, 2019.

[5] D. J. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *Electronics letters*, vol. 32, no. 18, pp. 1645–1646, 1996.

[6] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[7] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Proceedings of ICC'93-IEEE International Conference on Communications*, vol. 2. IEEE, 1993, pp. 1064–1070.

[8] K. D. Rao, *Channel coding techniques for wireless communications*. Springer, 2015.

[9] R. W. Hamming, *Coding and information theory*. Prentice-Hall, Inc., 1986.

[10] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[11] Y. Jiang, S. Kannan, H. Kim, S. Oh, H. Asnani, and P. Viswanath, "Deepturbo: Deep turbo decoder," in *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, 2019, pp. 1–5.

[12] T. J. O'Shea, K. Karra, and T. C. Clancy, "Learning to communicate: Channel auto-encoders, domain specific regularizers, and attention," in *2016 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*. IEEE, 2016, pp. 223–228.

[13] E. Nachmani, Y. Be'ery, and D. Burshtein, "Learning to decode linear codes using deep learning," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2016, pp. 341–346.

[14] W. Xu, Z. Wu, Y.-L. Ueng, X. You, and C. Zhang, "Improved polar decoder based on deep learning," in *2017 IEEE International workshop on signal processing systems (SiPS)*. IEEE, 2017, pp. 1–6.

[15] H. Kim, Y. Jiang, R. Rana, S. Kannan, S. Oh, and P. Viswanath, "Communication algorithms via deep learning," *arXiv preprint arXiv:1805.09317*, 2018.

[16] V. Raj and S. Kalyani, "Backpropagating through the air: Deep learning at physical layer without channel models," *IEEE Communications Letters*, vol. 22, no. 11, pp. 2278–2281, 2018.

[17] T. J. O'Shea, T. Erpek, and T. C. Clancy, "Deep learning based mimo communications," *arXiv preprint arXiv:1707.07980*, 2017.

[18] 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 preprint arXiv:1911.03038*, 2019.

[19] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *4th International Conference on Learning Representations, ICLR*, 2016.

[20] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[21] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.

[22] A. Tjandra, S. Sakti, and S. Nakamura, "Tensor decomposition for compressing recurrent neural network," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–8.

[23] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[24] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[25] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," 2016.

[26] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[27] S. Zhu, X. Dong, and H. Su, "Binary ensemble neural network: More bits per network or more networks per bit?" in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4923–4932.