COMPACT AUTOENCODER FOR END TO END COMMUNICATION USING DIFFERENT PRUNING TECHNIQUES

A project thesis

submitted by

Vudurupati Srikanth

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 **compact autoencoder for end to end communication using different pruning techniques** , submitted by **Vuduru-pati Srikanth (EE19M033)**, 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.

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

Place : Chennai
Date : June 2021

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 the two years of my 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

KEYWORDS: Autoencoder,Neural Network,weights and biases,Pruning, Fully connected network, Sparse network,Block Error Ratio.

An autoencoder is an artificial neural network which can learn to code the data efficiently. An autoencoder tries to learn a representation for a set of data, typically for dimensionality reduction, by training the network to ignore internal noise. Along with the encoding, a reconstructing side is learned, where the Autoencoder tries to generate from the reduced encoding a representation as close as possible to its original input. The performance of the Autoencoder can be maintained the same even if we remove some of the weights and biases from the network also know as pruning. With the pruning, the network becomes sparse. This is addressed in the thesis where we try to apply different pruning techniques on the Autoencoder network and compared it with a fully connected network. The Block Error Ratio(BLER) for the output generated by each of the networks is analyzed and the results are plotted.

Contents

List of Figures

List of Tables

Chapter 1

DEEP LEARNING FOR THE PHYSICAL LAYER

1.1 Introduction

To date, the wireless network evolution was primarily aimed for higher rates, which mandated an increase in the network capacity. As the demand for wireless capacity will continue to grow, the development of the Internet of Everything (IoE) systems, connecting millions of people and billions of machines is yielding a shift from the rate centric enhanced mobile broadband (eMBB) services of yesteryears toward ultra-reliable, low latency communications (URLLC) [1]. The recent upsurge of diversified mobile applications, especially those supported by Artificial Intelligence(AI), is opening the ways for the future evolution of wireless communications [2].
The strict spectral efficiency, reliability, and latency requirements associated with next-generation communication systems imply that manual configuration of the network will no longer be possible. Rather, network intelligence and automation will occupy centre stage, helping to build an increasingly intelligent network [3]. Along with the exponential growth in the number of wireless devices, services, and applications, a corresponding demand for higher rate wireless communications has burgeoned in recent times .In the past few years, the field of artificial intelligence (AI) has witnessed rapid growth, leading to its application in a broad variety of areas More particularly, in the the domain of wireless communications[3].

1.2 AI for the physical layer

Traditionally, physical layer modelling has been model-oriented—a manner in which mathematical models following a specific structure are proposed and optimized under constraints to meet a list of pre-determined performance specifications. For example, for the channel estimation, a channel model is considered along with additional parametric configurations. These model-based solutions normally perform well if the derivation of mathematical models is relatively straightforward or there exists a closed-form solution. The models can then be validated by field measurements or numerical simulations. However, in real-world situations, the applicability of such model-based solutions falls short in complex environments, due to factors such as non-linearity inside systems and uncontrollable interference among others. On the other hand, another approach which is based on statistics or data sets, build the model through learning from the data. This method is particularly useful when the theoretical analysis is intractable or when a closed-form solution is difficult to obtain.

To date, artificial intelligence has proved its usefulness in multiple physical layer procedures. For example, in channel estimation and symbol detection, deep learning approaches reported in [4], [5] has explained that the proposed deep learning-based symbol detection algorithms can provide robust and accurate results with reduced complexity. Furthermore, a deep learning method based on the deep neural network architecture also shows an improved channel estimation accuracy under the consequences of non-linearities of power amplifiers, and quantization errors induced by hardware impairments [6].

An autoencoder-based communication system is designed to reconstruct the transmitted signals from channel impairment based on trained deep neural networks in an end-to-end manner [7]. Furthermore,self-supervised learning is a growing trend for user localization since it has been demonstrated that proper methods can significantly decrease the size of the labelled dataset for efficient processing [8]. Neural networks can give solutions to complex problems in communications and signal processing due to their non-linear processing, the capacity of learning and generalization and parallel distributed architecture.

1.3 Organisation of thesis

This thesis is organised as follows:

chapter 2 explains in detail the Neural Networks and an overview of the Autoencoder. The definition of pruning and its necessity is explained in brief.

chapter 3 introduces the supervised learning for the Autoencoder Network. It gives an idea about the network that we used and how it is being trained.

chapter 4 explains various pruning methods that we apply on the Autoencoder Network defined in the previous chapter.

In *chapter 5* the simulation set-up used for experiments is explained. each pruning method is applied on the network with different threshold values and the results are plotted.

chapter 6 summarizes the work done and provides some concluding remarks and observations.

Chapter 2

AUTOENCODER NEURAL NETWORK

2.1 Introduction

A neural network is a loop of algorithms that attempts to identify underlying relations in a set of data within a process that simulates the way the human brain works. In this reason, neural networks refer to systems of neurons. Neural networks can adjust to varying input so the network produces the best feasible result without needing to redesign the output models. The theory of neural networks has its origins in artificial intelligence. A neural network works likewise to the human brain's neural network. A "neuron" in a neural network is a mathematical function that receives and classifies data according to a particular architecture. The network shows a strong similarity to statistical methods such as regression analysis and curve fitting. A neural network comprises layers of interconnected nodes. Each node is similar to multiple linear regression. The neuron feeds the signal generated by various linear regression into an activation function that can be nonlinear.
The input layer receives input data of patterns. The output layer has output signals or classifications to which input patterns may be mapped. Deep layers fine-tune the input weightings till the neural network's performance of error is smallest. It is hypothesized that deep layers extrapolate important characteristics in the input set of data that have predictive capability regarding classifying the outputs. This represents feature extraction, which performs a service comparable to statistical techniques such as principal component analysis(PCA).

2.2 Overview of Autoencoder

An Autoencoder is an artificial neural network that learns how to efficiently compress and encode the set of data and learns how to build the data back from compressed encoded representation that is closest to the original input. An autoencoder aims to learn a representation for a set of data, generally for reducing the dimension of the data, by training the network to neglect internal Snoise. Along with the compression, reconstructing is learned, where the autoencoder attempts to produce from the reduced encoding an output representation that is as close to the original input as possible. Reducing the dimension of input is one of the features of an autoencoder. Principle Component Analysis (PCA) uses a linear transformation to project data into low dimensional space, unlike PCA an autoencoder can model complex non-linear functions also.[9]

An autoencoder is used to learn suitable data codings in a supervised way. In the figure2.1 there is an internal (deep) layer that represents a code used to represent the input, and it is constituted by two parts: an encoder that maps the input data into the code word, and a decoder that maps the code word to a reconstruction of the input.

Implementing the replicating task perfectly would replicate just the signal, and this is why autoencoders normally are limited in ways that force them to reconstruct the input approximately, preserving only the most important features of the data in the copy.

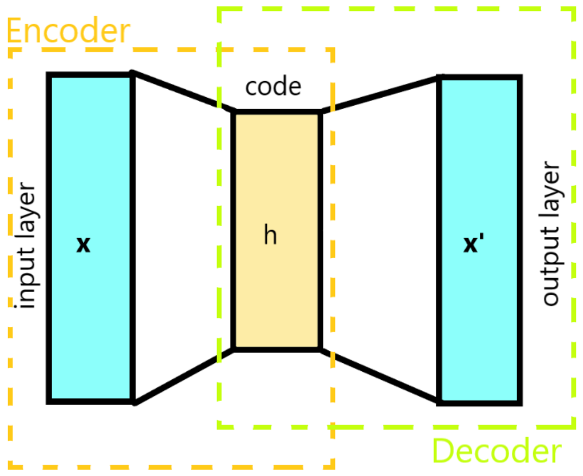

Figure 2.1: Block diagram of autoencoder

In a simple form, a communications system consists of a transmitter, a channel, and a receiver, it is shown in Fig 2.2. The transmitter needs to transmit one out of M

available messages $M = 1, 2, .., m$ to the receiver with n discrete uses of channel. To this step, it uses the transformation of the information s to produce the transmitted signal $x = f(s)$.

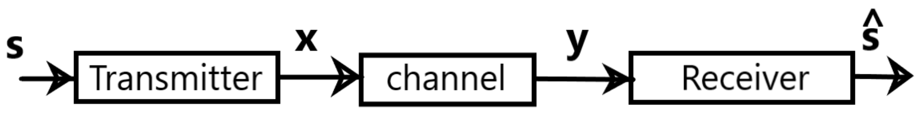

Figure 2.2: A simple communications system

Generally, the goal of an autoencoder is to represent its input in low-dimensional code at some of the hidden layer which allows regeneration at the output with minimum error. In this way, the autoencoder learns to compress the data non-linearly and reconstruct the input. It tries to learn representations x of the possible messages s that are robust to the channel conditions mapping x to y (i.e., fading, distortion, noise etc.) so that the transmitted information can be recovered with a minimum probability of error.

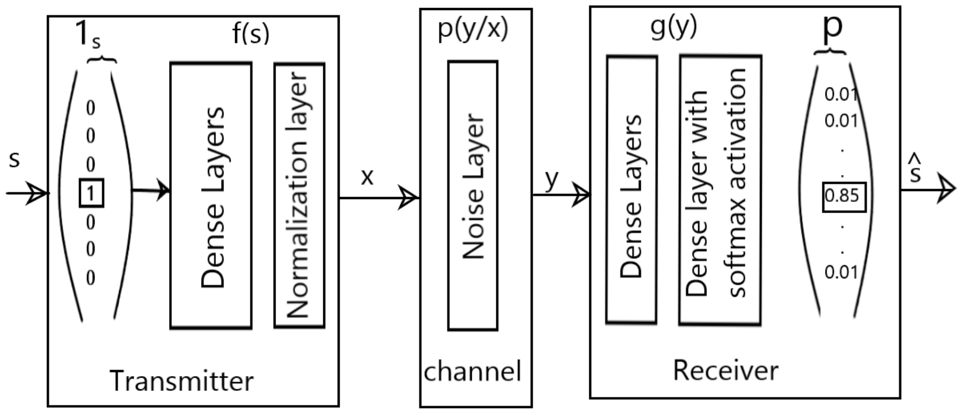

Figure 2.3: A communications system over an AWGN channel

An example of such an autoencoder is shown in Fig.2.3 Here, the transmitter has a feedforward Neural Network with many dense layers followed by a normalization layer. The input s to the transmitter is encoded as a one-hot vector i.e., an M-dimensional vector, the s^{th} position of which is equal to one and zero otherwise. The decoded vector $\widehat{s}$ corresponds then to the index of the element of p with the greatest probability. The autoencoder can be trained end-to-end using Stochastic

Gradient Descent on the set of all messages $s \in M$ using the categorical cross-entropy loss function. Finally,by comparing the s with $\widehat{s}$ the Block Error Rate (BLER) of the autoencoder for various SNR can be obtained.

2.3 Motivation for Pruining

A neural network can be considered a solution finder for any mathematical problem. Every consecutive layer of a network is connected by weights and biases. These weights are the degrees of freedom to approximate the relation between the input and the output.

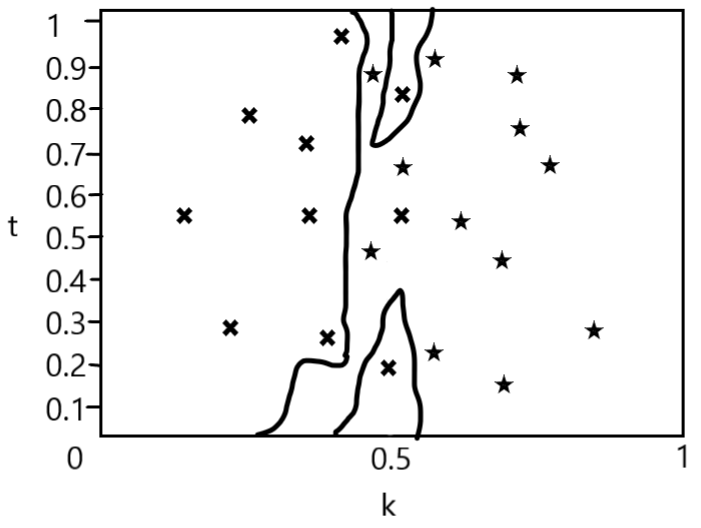

Figure 2.4: overfitting of the training data points

In larger networks, it often leads to a problem, since it means training a lot of parameters that too with a scarce data set, which can easily lead to overfitting and poor generalization.

Figure 2.4, is an illustration of classification problem where we aim to classify "$*$" and "$\times$" data points. it is an example of setting very tight boundaries for the classification.[10]

By reducing the degrees of freedom to approximate the required function, i.e., by removing some of the weights in the network, overfitting of the data points can be reduced while maintaining the minimum effect on the error.This is called as ***pruning*** [10]. Pruning is a data compression technique in machine learning and search algorithms that reduces the size of Neural Network by removing sections that

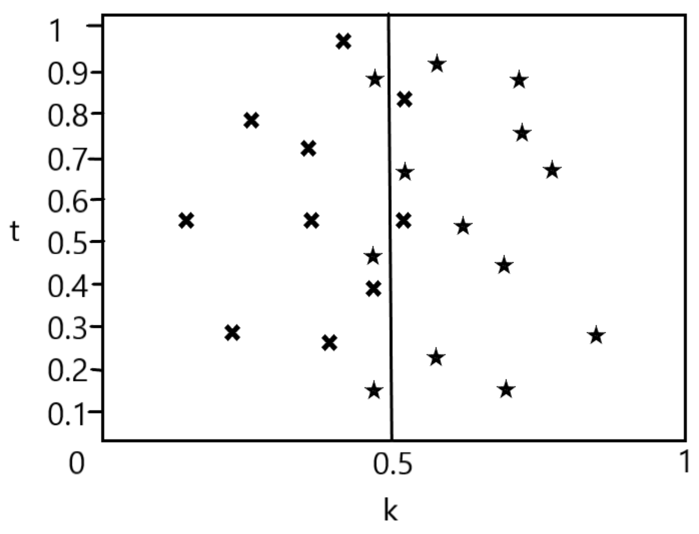

Figure 2.5: proper fitting the training data points

are non-critical and redundant to classify instances. Pruning reduces the complexity of the final classifier, and hence improves predictive accuracy by the reduction of overfitting.

In figure 2.5, by reducing the number of parameters, classification is possible with an almost similar error[10]. Hence, the removal of some of the weights from the neural network is advantageous in two ways. Storage space required for the network reduces and the time taken for the execution reduces.

can we blindly remove some of the weights and call it pruning? The answer is a no. Which weights are to be removed from the network is decided by knowing the importance of the weights in the network.

Chapter 3

SUPERVISED LEARNING OF THE AUTOENCODER

3.1 Introduction

Supervised learning is a type of machine learning in which networks are trained using well-labelled training data, and on basis of that information, machines predict the output. The labelled data indicates some input data is already assigned with the exact output.

In supervised learning, the training data given to the networks work as the supervisor that teaches the networks to predict the output correctly. Supervised learning is a method of giving input data as well as correct output data to the machine learning model. A supervised learning algorithm tries to find a mapping function to map the input variable x with the output variable y.In the real world, supervised learning can be used for Image classification, Risk Assessment, spam filtering, Fraud Detection, etc. In supervised learning, models are trained using a labelled data, where the model learns about each type of data. Once the learning process is completed, the model is examined based on the test dataset (a subset of the training set), and then it predicts the output.

3.1.1 Steps involved in supervised learning

- Determine the type of training dataset.
- Collect the labelled training data.
- Divided the training dataset into a training dataset, test dataset, and validation dataset.

- Determine the input features of the training dataset, which should have sufficient information so that the model can precisely predict the output.
- Determine the proper algorithm for the model, such as a Neural Network, decision tree, etc.
- Perform the algorithm on the training dataset.
- Evaluate the efficiency of the model by giving the test set. If the model predicts the correct output, which indicates our model is accurate.

3.1.2 Types of supervised learning Algorithms

Regression

Regression algorithms are applied if there is a correlation between the input variable and the output variable. It is utilised for the prediction of continuous variables, such as curve fitting, Weather forecasting, Market Trends, etc.

Classification

Classification algorithms are applied when the output variable is categorical, which means there are two categories such as True-false, Yes-No etc.

3.2 Deep Learning basics

3.2.1 Forward Propagation

A Neural Network with L number of layers represents a mapping $f(r_0;\theta) : R^{N_0} \mapsto R^{N_L}$ of an input vector $r_0 \in R^{N_0}$ to an output vector $r_L \in R^{N_L}$ through L repetitive steps:

$$r_l = f_l(r_{l-1};\theta_l), l = 1, ..., L \tag{3.1}$$

where the mapping carried out by the lth layer is $f_l(r_{l-1};\theta_l) : R^{N_{l-1}} \mapsto R^{N_l}$.This mapping depends on the output of the vector r_{l-1} from the previous layer and also on the set of parameters in the network θ_l.The parameters are weights and biases in each of the layers.The mapping can be statistical ,i.e., f_l is a function of random variable.We denote $\theta = \theta_1, ..., \theta_L$ the set of all parameters present in the network. The lth layer which is intermediate is called *fully-connected* or *dense* if $f_l(r_{l-1};\theta_l)$ is of the form

$$f_l(r_{l-1};\theta_l) = \sigma(W_l r_{l-1} + b_l) \tag{3.2}$$

where $\sigma(.)$ is an activation function $W_l \in R^{N_l \times N_{l-1}}$, and $b_l \in R^{N_l}$ [11].The set of parameters for lth layer is $\theta_l = W_l, b_l$.This is the *forward propagation* where output of lth layer is the input for the $l + 1$th layer [12]. [13] gives several other types of layers along with their mapping functions and the parameters that are used. All layers with random mappings generate a new stochastic mapping during each iteration. For example, the noise layer simply adds a vector of elements with Gaussian distribution of zero mean to the input. Thus, it generates a random output for the same input each time it is called. The non-linearity is introduced by the activation function $\sigma(.)$ in 3.2 which is important for the so-called expressive power of the Neural Network. There would be not much of an advantage of stacking multiple layers on top of each other if there is no non-linearity [14].Some of the commonly used activation functions are listed in [11].

3.2.2 Backward Propagation

Neural Networks are generally trained using training data which is labelled, i.e., a set of input-output vector pairs $(r_o, r^*_{L,i}), i = 1, ..., S$, where r$^*_{L,i}$ is the expected output of the neural network when $r_{0,i}$ is given as an input. The aim of the training is to penalize the loss function

$$L(\theta) = \frac{1}{s} \Sigma l(r^*_{L,i}, r_{L,i}) \tag{3.3}$$

with respect to the internal parameters in θ, where $l(u, v) : R^{N_L} \times R^{N_L} \mapsto R$ is the loss function which results a real number indicating the loss for each iteration and $r_{L,i}$ is the output of the Neural Network when $r_{0,i}$ is given as an input.Several commonly used loss functions are provided in [15].The popular algorithm to find good sets of parameters θ is the stochastic gradient descent(SGD) which starts with some random initial values of $\theta = \theta_0$ and then updates θ iteratively as

$$\theta_{t+1} = \theta_t - \eta \nabla \tilde{L}(\theta_t) \tag{3.4}$$

This is the *Backward propagation* where $\eta > 0$ is the learning rate and $\tilde{L}(\theta_t)$ is the loss function over a random mini batch examples $S_t \subset 1, 2, ..., S$ with S_t as size of mini-batch.

$$\tilde{L}(\theta) = \frac{1}{S_t} \sum_{i \in S_t} l(r^*_{L,i}, r_{L,i}) \tag{3.5}$$

3.3 Input dataset

we need to transmit 16-QAM symbols through an AWGN channel.Each symbol requires $log_2(16)$ number of bits to transmit. we vectorize the information into one-hot encoded form where each symbol $s \in s_1, s_2, ..., s_{16}$ such that it is a 16 dimensional vector,the s_i position of which is equal to one and zero otherwise. The input dataset is limited to 16 in number, making it a very small dataset. The same data is shuffled and used as mini-batches. However, the AWGN noise generated in the noise layer is random.It adds a random noise component each time it is activated. Thus, it mimics the practical environment.

Chapter 4

VARIOUS PRUNING TECHNIQUES

4.1 Introduction

In a Neural Network each pair of layers is connected by weights and biases. Our aim is to prune some of these parameters and make the Network sparse.As all the parameters are not equally important, decision is to be taken on which parameters are to be pruned.

4.2 magnitude based pruning

Depending on the magnitude of each of the connection, we decide to keep them or remove them. The reason for considering the magnitude as a parameter is that while calculating the output of any node the equation that represents forward propagation of a neural network is given by $Y = \sigma(Wx) + b$. where $\sigma(.)$ is an activation function, and W is the weight matrix. So, if there are some weights in the network whose values are small and very close to zero. If we make those weights as zeros, the effect it has on the Y values will be very minimum. Hence, the magnitude of weights is a parameter for the pruning.

4.2.1 Threshold for pruning

From the distribution of magnitudes of parameters ,i.e, the weights and biases whose values are close to zero are removed. The threshold here in our case is how much percentage of weights we want to remove from the fully connected network and make it sparse. The results for pruning by removing 5%, 10%, 15%, 20%, 25% of total weights are removed and compared with the fully connected network.

4.3 Gradient based pruning

This method of pruning collects some information at the time of training itself. During the training process, both the forward propagation and the backward propagation are taken care of by the optimizer that we use. In our case, we used Adam Optimizer with a learning rate of 1e-3.
During the forward propagation of the training, the optimizer finds the output corresponds to present weights and biases. But, the actual output we obtained will be different from what is the correct output. During the backpropagation, to make the actual output as correct output, the optimizer updates the weights accordingly to penalize the cost function or the error by calculating the gradients for every parameter in the network

4.3.1 Gradient calculation

The gradient is nothing but the derivative of the cost function with respect to each parameter in the network. The gradient indicates how the cost function will change with respect to a change in the internal parameters (weights and biases) [16]. We aim to capture the gradients corresponding to every connection in the network throughout the training process and find the exponential average of gradients over all the training epochs as:

$$g_i(t+1) = 0.8 g_i(t) + 0.2 \frac{\partial L(t)}{\partial w_i} \tag{4.1}$$

4.3.2 Sensitivity calculation

The sensitivity of a weight w is the difference in error occured with and without the weight[10].

For every w_{ij} in the network, the sensitivity S_{ij} with respect to w_{ij}, will be defined here as[17]

$$S_{ij} = L(w_{ij} = 0) - L(w_{ij} = w_{ij}^f) \tag{4.2}$$

upon the completion of the training the final value of the connection is w_{ij}^f.
the sensitivity S_{ij} defined in 4.2 can be written as

$$S = \frac{L(0) - L(w^f)}{0 - w^f}(0 - w^f) \tag{4.3}$$

where w = w_{ij} and loss L is expressed as a function of w, considered that all other parameters except w_{ij} are fixed.

Generally learning process does not start with $w = 0$, but randomly chosen initial value w^i. since L(0) is unknown, we will approximately calculate the slope of L(w) while moving from 0 to w^f by the slope measured between w^i and w^f, namely

$$S = \frac{L(w^i) - L(w^f)}{w^i - w^f}(0 - w^f) \tag{4.4}$$

The initial and final weights,w^i and w^f, respectively, are readily available during the training phase. But, for the numerator of eq(4.4), it was assumed that only one weight, namely w, had been changed, keeping other weights as fixed.But, This is not the case during normal learning.
To elaborate, consider an example of a network having only two weights, denoted u and w (the extension to more weights will become obvious). For this case the numerator in eq(4.4) is

$$L(u^f, w^i) - L(u^f, w^f) \tag{4.5}$$

i.e., only the influence due to the changes in w is considered into account, figure 4.1 clarifies the situation.

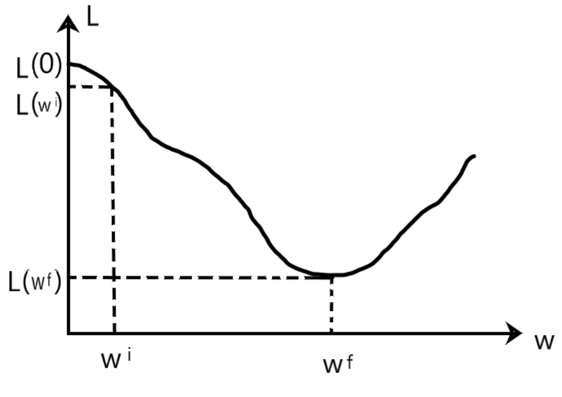

Figure 4.1: The Loss as a function of individual weight

The error L(u,w) is illustrated by constant value contours.The initial value of weight is designated by ***I*** in fig **??** and the learning path is the dashed line from ***I*** to ***F***,the final point.For a precise evaluation of S, the numerator of eq(4.4) can be evaluated as L(w=w^f) $- L(w = 0) = \int_{\boldsymbol{A}}^{\boldsymbol{F}} \frac{\partial L(u^f,w)}{\partial w}\, dw$ The integral is calculated along the line from point ***A*** to ***F*** as an approximation, which corresponds to w=0

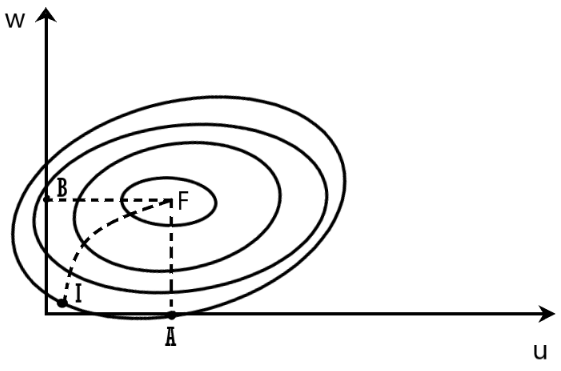

Figure 4.2: Learning as motion on an error surface

to the final weight value $\boldsymbol{F}$. However, the training phase starts at point $\boldsymbol{I}$ rather than at $\boldsymbol{A}$.

We approximate the integral by summation,over the discrete iteration steps during training.The sensitivity of the connection is

$$L(w = w^f) - L(w = 0) \cong \int_{\boldsymbol{I}}^{\boldsymbol{F}} \frac{\partial L(u^f, w)}{\partial w} \, dw \tag{4.6}$$

This expression will be approximated by replacing the integral by summation. Thus the estimated sensitivity to the removal of connection w_{ij} will be evaluated as

$$\widehat{S_{ij}} = \sum_{0}^{N-1} \frac{\partial L}{\partial w_{ij}}(n) \, \Delta w_{ij}(n) \, \frac{w_{ij}^f}{w_{ij}^i - w_{ij}^f} \tag{4.7}$$

where N are the number of training epochs.
The terms that the above estimate of the sensitivity uses are available during the course of training. Also, every optimizer uses gradients to know the direction of change, so the gradient, are available. Therefore, the only extra computational demand for implementing our procedure is the summation in eq(4.7). This overhead keeps track of the accumulated terms that build up to S_{ij} in eq(4.7).
For our case of back-propagation, weights are updated by the Adam optimizer according to eq(4.8)

$$\Delta w_{ij} = -\eta \, \frac{\partial L}{\partial w_{ij}} \tag{4.8}$$

after completion of training, we are provided with a list of sensitivity numbers, one per each connection. At this point, a decision can be taken on pruning those weights which are having the smallest sensitivity numbers.[17]

If a particular weight is having very low sensitivity throughout the training process, It means that the gradient of this weight is very small. so even if there is any change in the value of this weight, i.e., making it to zero, the cost function is not much affected. Hence, those weights which are having the least sensitivity can be removed from the network.

4.3.3 Threshold for pruning

From the distribution of sensitivities of parameters ,i.e, the weights and biases whose values are close to zero are removed. The threshold here in our case is how much percentage of weights we want to remove from the fully connected network and make it sparse. The results for pruning by removing 55%, 60%, 65%,70%,75% of total weights are compared with the fully connected network.

Chapter 5

SIMULATION RESULTS

5.1 Results of magnitude based pruning

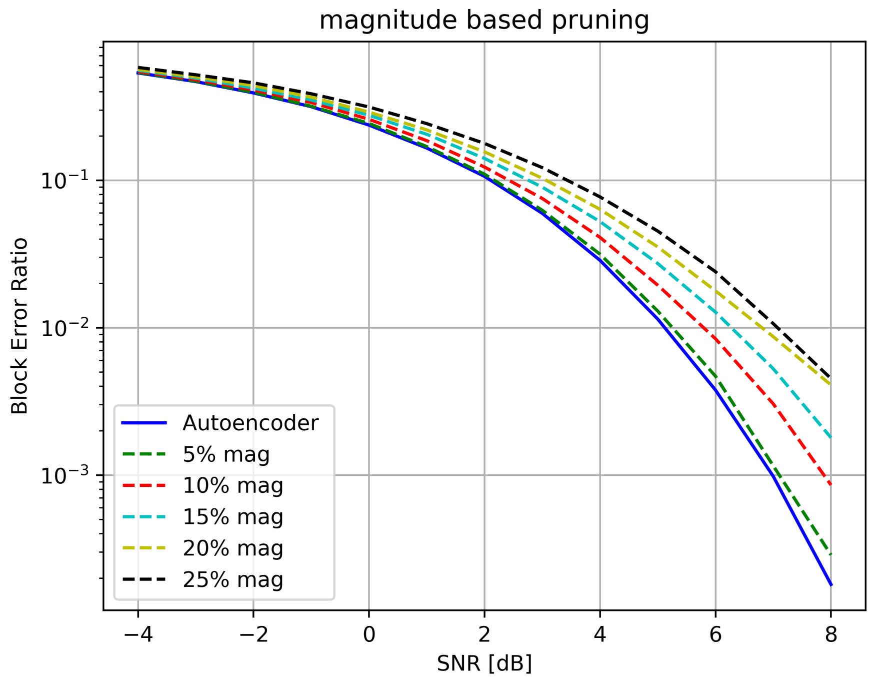

Figure 5.1: BLER performance of magnitude based pruning

Observations:

As per the section 4.2,depending on the magnitude of the connections in the network, we decide to remove those conncections which have small magnitude and close to zero. By removing respective percentage of connections,the BLER performance of the fully connected Autoencoder is compared with the sparse Network.It is observed that by removing upto 10 percentage of weights, the performance is nearly the same.The time saved for the network to run on the test data set is also shown in table 5.1

% of weights pruned	% time saved for computation
5%	6%
10%	7%
15%	9%
20%	12%
25%	15%

Table 5.1: magnitude based pruning

5.2 Results of Gradient based pruning

pruning with respect to gradient

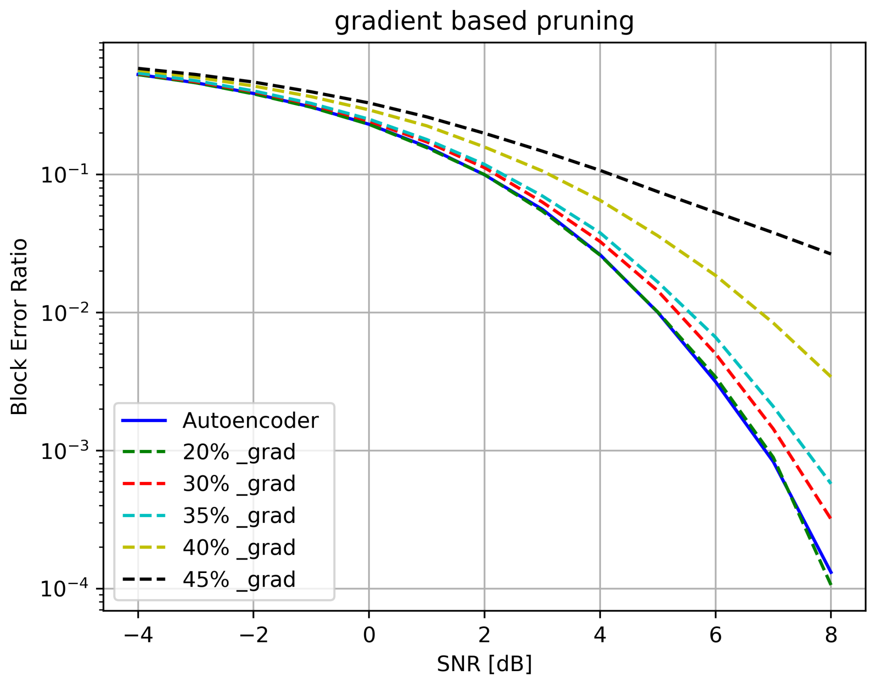

Figure 5.2: BLER performance of gradient based pruning

Observations:

As per the section 4.3.1,depending on the gradinet of the connections in the network, we decide to remove those conncections which have small gradient. By removing

% of weights pruned	% time saved for computation
20%	4%
30%	8%
35%	11%
40%	14%
45%	16%

Table 5.2: gradient based pruning

respective percentage of connections,the BLER performance of the fully Autoencoder is compared with the sparse Network.It is observed that by removing upto 30 percentage of weights, the performance is nearly the same.The time saved for the network to run on the test data set is also shown in table 5.2

Pruning with respect to sensitivity

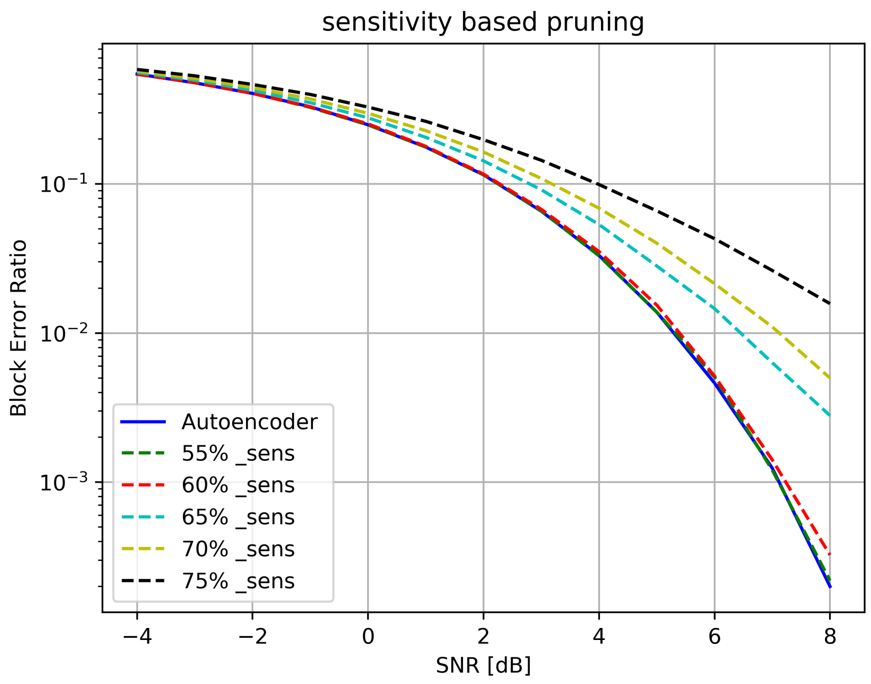

Figure 5.3: BLER performance of sensitivity based pruning

Observations:

As per the section 4.3.2,depending on the Sensitivity of the connections in the network, we decide to remove those conncections which have small sensitivity. By

% of weights pruned	% time saved for computation
55%	6%
60%	10%
65%	13%
70%	17%
75%	19%

Table 5.3: sensitivity based pruning

removing respective percentage of connections,the BLER performance of the fully Autoencoder is compared with the sparse Network.It is observed that by removing upto 60 percentage of weights, the performance is nearly the same.The time saved for the network to run on the test data set is also shown in table 5.3

Chapter 6

CONCLUSIONS

The sensitivity-based pruning performed much better than magnitude-based pruning. Because, when we use the sensitivity of weights as a criterion for the pruning, those weights which are highly sensitive to calculate the final Loss function are considered for pruning. so, even if a connection is having a very small magnitude, we can not remove it. Because, magnitude wise it may be small, but the Loss function might be very sensitive to that connection. The storage space for the Network is reduced and also the computational time required for the network to evaluate on the test set is also reduced.

Bibliography

[1] W. Saad, M. Bennis, and M. Chen, “A vision of 6g wireless systems: Applications, trends, technologies, and open research problems,” *IEEE Network*, vol. 34, no. 3, pp. 134–142, 2020.

[2] K. B. Letaief, W. Chen, Y. Shi, J. Zhang, and Y.-J. A. Zhang, “The roadmap to 6g: Ai empowered wireless networks,” *IEEE Communications Magazine*, vol. 57, no. 8, pp. 84–90, 2019.

[3] 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.

[4] H. Ye, G. Y. Li, and B.-H. Juang, “Power of deep learning for channel estimation and signal detection in ofdm systems,” *IEEE Wireless Communications Letters*, vol. 7, no. 1, pp. 114–117, 2018.

[5] N. Samuel, T. Diskin, and A. Wiesel, “Deep mimo detection,” 2017.

[6] T. Demir and E. Björnson, “Channel estimation in massive mimo under hardware non-linearities: Bayesian methods versus deep learning,” *IEEE Open Journal of the Communications Society*, vol. 1, pp. 109–124, 2020.

[7] T. J. O’Shea and J. Hoydis, “An introduction to deep learning for the physical layer,” 2017.

[8] M. Arnold, J. Hoydis, and S. ten Brink, “Novel massive mimo channel sounding data applied to deep learning-based indoor positioning,” 2019.

[9] Y. Dai, J. Guan, W. Quan, C. Xu, and H. Zhang, “Pca-based dimensionality reduction method for user information in universal network,” in *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*, vol. 01, 2012, pp. 70–74.

[10] R. Reed, “Pruning algorithms-a survey,” *IEEE transactions on neural networks*, vol. 4 5, pp. 740–7, 1993.

[11] B. Ding, H. Qian, and J. Zhou, “Activation functions and their characteristics in deep neural networks,” in *2018 Chinese Control And Decision Conference (CCDC)*, 2018, pp. 1836–1841.

[12] L. Buturovic and L. Citkusev, “Back propagation and forward propagation,” in *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, vol. 4, 1992, pp. 486–491 vol.4.

[13] S. Albawi, T. A. Mohammed, and S. Al-Zawi, “Understanding of a convolutional neural network,” in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6.

[14] Z. Dlugosz and R. Dlugosz, “Nonlinear activation functions for artificial neural networks realized in hardware,” in *2018 25th International Conference ”Mixed Design of Integrated Circuits and System” (MIXDES)*, 2018, pp. 381–384.

[15] A. Demirkaya, J. Chen, and S. Oymak, “Exploring the role of loss functions in multiclass classification,” in *2020 54th Annual Conference on Information Sciences and Systems (CISS)*, 2020, pp. 1–5.

[16] M. C. Mozer and P. Smolensky, “Skeletonization: A technique for trimming the fat from a network via relevance assessment,” in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 1. Morgan-Kaufmann, 1989. [Online]. Available: https://proceedings.neurips.cc/paper/1988/file/07e1cd7dca89a1678042477183b7ac3f-Paper.pdf

[17] E. Karnin, “A simple procedure for pruning back-propagation trained neural networks,” *IEEE Transactions on Neural Networks*, vol. 1, no. 2, pp. 239–242, 1990.