Improving the efficiency of deep learning methods for communication systems

A Project Report

submitted by

DEDDY JOBSON

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY (HONS.)

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

APRIL 2019

THESIS CERTIFICATE

This is to certify that the thesis titled **Improving the efficiency of deep learning methods for communication systems**, submitted by **Deddy Jobson**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology (Hons.)**, is a bona fide record of the research work done by him under our 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.

Prof. Srikrishna Bhashyam
Research Guide
Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 30th April 2019

ACKNOWLEDGEMENTS

I would like to thank my project guide Prof. Srikrishna Bhashyam for the insights he provided during our discussions, and for helping me out by directing me to relevant literature where required. I would also like to thank Dr. Nitin Chandrachoodan for providing assistance and helping us cross hurdles we faced while benchmarking our deep learning models. Last but not the least, I wish to thank the department of Electrical Engineering for partially funding the research project.

ABSTRACT

KEYWORDS: Deep Learning; Wireless Communication; Model Compression; Knowledge Distillation; Quantization; Weights Pruning

In this report we explore some of the applications of deep learning in communications systems. We bring out the disadvantages of deep learning when compared with preexisting models designed by experts in the field and provide theoretical reasoning behind it. The limitations we discuss here are primarily regarding computational efficiency of deep learning models. We then demonstrate some techniques like knowledge distillation, quantization of parameters and pruning of weights which allow one to compress the models and so improve the efficiency of deep learning based communication systems thereby laying the foundations for the implementation of data-driven deep learning models in embedded devices for practical applications. Lastly, we also provide directions for future work which would be of interest to the curious researcher.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

IITM	Indian Institute of Technology, Madras
ANN	Artificial Neural Network
WGN	White Gaussian Noise
RBF	Rayleigh Block Fading
CNN	Convolution Neural Network
SNR	Signal to Noise Ratio
LSB	Least Significant Bit

CHAPTER 1

INTRODUCTION

In this report, we discuss the the prevalence of deep learning in communication systems. More specifically, we look for ways to optimize and improve the efficiency of neural networks to improve inference speed, memory overheads, etc. The report is divided into the following sections.

In chapter 2, we give a brief introduction to deep learning and talk about it's working principles. We then present situations in communication systems design where deep learning provides solutions that perform as good as or better than the expert-designed counterparts. We provide our own replicated findings and conduct some additional experiments to better understand the role of deep learning in each task.

In chapter 3, we elaborate on the limitations of deep learning methods used in chapter 2. We emphasize primarily on the large computational complexity of neural networks and how it affects the latency and memory requirements of the communication network. We also talk about some ways used to measure the efficiency of neural networks in terms of computational and time requirements. The later chapters explore some solutions to alleviate the inefficiency bottlenecks.

Chapter 4 deals with knowledge distillation, the only compression method discussed here which does not directly compress the model. Instead the model—which will be referred to as the teacher—is used to assist in training a smaller model, the student. We explain in detail how the features learned by the teacher can be used to improve the performance of the student. We also look into some situations where knowledge distillation doesn't help as much as hoped and provide justification for its inefficacy in such cases.

In chapter 5, we explore the idea behind quantization of parameters and how it can help against the latency issues associated with neural networks. We also conduct experiments to analyze quantized neural networks and express the degradation in performance of the network in the form of an additional noise component. To reduce the accuracy

loss dealt by quantization, we propose a scheme which makes use of regularization of weights to impose helpful priors on the network.

In chapter 6, we explain the idea of pruning of weights in a neural network. It is the simplest of the discussed strategies to improve efficiency. We present the results obtained through experiments conducted to quantify the effect of pruning the weights of the fully connected layers both with and without retraining the weights that remain. Then, we repeat the above by pre-training the network with regularization to minimize the debilitating effect of pruning on the performance of the neural network.

We compare the above compression strategies in chapter 7 to highlight the points of strength and weakness in each of the methods discussed. This would give one a better idea of which method to use where. Since these methods are independent to each other (to a certain extent), we can use them in tandem to greatly reduce the model size without adverse effects on performance of the model for the communication task.

Finally, we conclude our report in chapter 8. Here, we summarize the work discussed in the preceding chapters and emphasize on its applications in the field of communication. We shall also briefly mention a couple of other compression techniques which have not been explored as thoroughly. They will serve as potential directions for future work. In order to help others use the work done here as a foundation for future research, we shall make the source code[3] of all our experiments available freely through Github.

CHAPTER 2

DEEP LEARNING IN COMMUNICATION SYSTEMS

Deep Learning is perhaps the most popular implementation of machine learning these days. While it first gained popularity in image classification, it soon found applications in handling unstructured data like audio, video, natural language, etc. for fields like healthcare[27], fraud detection[34], recommendation systems[42], advertising[41], self-driving cars[9], etc. In this report, we focus on some of the applications of deep learning in communication systems. We begin with a brief introduction to the working principles behind deep learning and then look into some problems in communication systems to which it presents suitable solutions.

2.1 Deep Learning

To understand how deep learning handles complex and unstructured environments, we first look at the human brain. In a simplistic sense, the human brain is a large network of neurons. Each neuron, on being given an input, passes it on only if the input signal exceeds a certain threshold. Therefore, a single neuron can be represented by the function compiled from data obtained from the work of Hodgkin and Huxley (1952) as depicted in figure 2.1. While a single neuron can't do much on it's own, the human ability to deal with large amounts of complex information emerges from the multitude of simple neurons, each playing it's small, almost insignificant role.

Artificial Neural Networks (ANNs) work in a similar fashion. Here however, since the way we tune the parameters of the networks relies on gradients, we don't make use of the same function such as used in the human brain (figure 2.1). We instead use "softer" or well-behaved functions. Some of the commonly used activation functions for ANNs like sigmoids and hyperbolic tan functions have been plotted in figure 2.2 obtained from Nwankpa et al. (2018). BircanoÄ§lu and ArÄśca (2018) provide a comprehensive overview of the many activation functions used for various tasks in deep

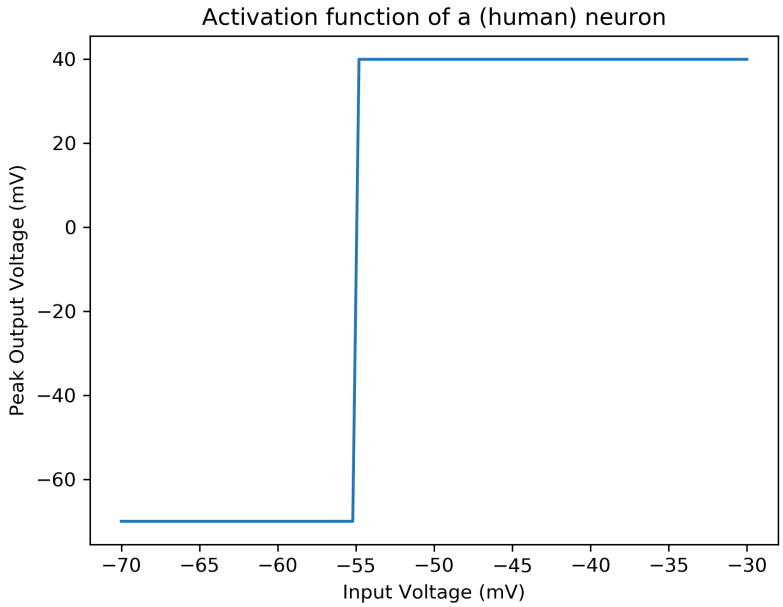


Figure 2.1: Activation potential function for human brains

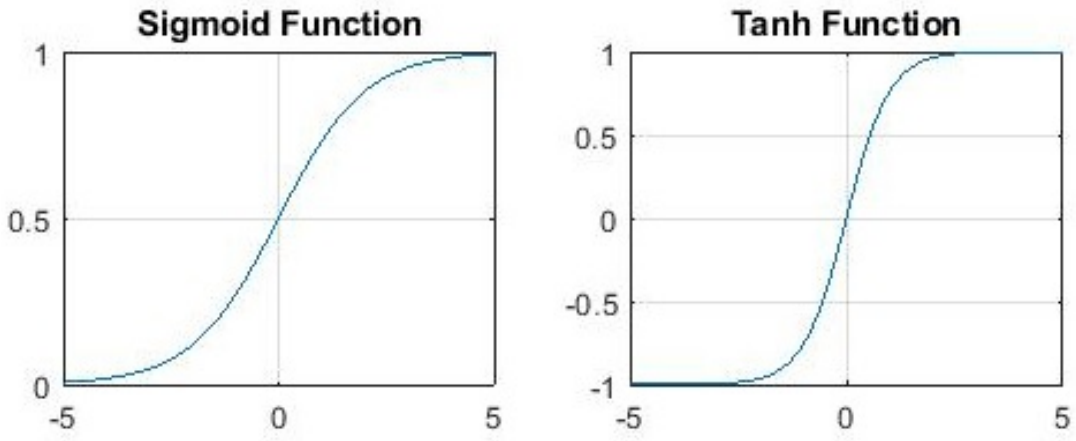


Figure 2.2: Activation functions used in ANNs. Source:Nwankpa et al. (2018)

learning. However, theoretically speaking at least, any non-linear activation function on top of an affine transformation of the input would work for a neuron.

The Universal Approximation Theorem[13] states that given enough neurons, a neural network can be used to approximate **any** continuous function. This means that as long as a function that serves as a desirable mapping between the input and output exists, it can be modelled using a neural network. For example, since a human can trivially classify objects through sight, a function that maps images captured in the retina (input) to the neural representation in the brain (output) exists. Therefore neural networks can be used for image classification, at times performing even better than humans[1].

In order to train deep neural networks such as those which have practical applications, a few more tricks are needed. The activation functions like sigmoids and tanhs were dropped in favor of the ReLU function. This function has a gradient of 1 which prevents gradients (g) from exploding or vanishing as they pass through the many layers of the networks.

$$\lim_{n\to\infty} g^n = \left\{ \begin{array}{ll} 0 & g < 1 \\ 1 & g = 1 \\ \infty & g > 1 \end{array} \right\}$$

Besides the RELU function, other innovations like batch normalization[21], Adam[22], etc., coupled with the advent of GPUs for parallelizing the training of large neural networks resulted in the rise of deep learning for practical applications.

2.2 Applications in Communication Systems

In communication systems we often encounter situations where we need to obtain a mapping of sorts. For example, in channel modelling, we wish to match the observed signal to the correct transmitted signal. So as long as some kind of estimation is possible, there exists a scope for deep learning models in communication. However, since communication is a well-established field, many experts have put in a lot of effort to push the performance of existing systems to near optimality. Therefore in most cases, even when deep learning can be used, the performance is equal to or only marginally better than current implementations designed using expert knowledge. Only when the environment is such that it cannot be modelled by experts, or other algorithmic deficiencies are present, can data-driven methodologies like deep learning provide substantial gains.

In some situations, the problem statement can only be partially modelled. This can happen if the situation possesses some known variables and some unknown variables. For example, in channel estimation, the noise distribution can be known to be both affine, but it may not be possible to model the exact distributions properly, or instead the exact distribution varies greatly with location. In that case, deep learning models can make use of priors set by experts to combine information learned through expert

modelling and data from the environment. This would result in a model that performs better than one that is either purely data driven or one that is solely modelled by experts.

2.2.1 Channel Estimation

As mentioned earlier, in channel estimation, we wish to match the observed signal to the transmitted signal. Typically, experts model the channel as

$$Y = X + N$$

where Y is the observed signal, X is the transmitted signal which we wish to estimate, and N is an additive noise. To simplify modelling, experts model the channel with White Gaussian Noise (WGN). While this works in most scenarios, there are situations when such models do not suffice. For example, in molecular MIMO communication, Lee et al. (2017) have shown the efficacy of using machine learning to model the channel. Deep learning methods can also be used to solve the same problem. This is done by modelling the entire communication system as a deep autoencoder, consisting of an encoder to best protect the transmission against noise and a decoder which deciphers the encoded embeddings into predictions for the input.

In Figure 2.4, we compare the performance of our neural network implementation of a one-bit-one-channel model against Binary Phase Shift Keying[35]. The implementation of BPSK was taken from [bps]. Each approach performs about as well as the other because the noise distribution is known in advance to the designer. To better visualize the decision rule learned by neural networks for channel estimation tasks in general, the decision boundaries learned by the neural network have been marked for a three-bit-two-channel scenario (figure 2.3). One can see that the decoder learns to choose the nearest (noiseless) encoding as the right mapping given a data point, and the encoder tries to keep the embeddings of different inputs as far apart as possible—given the energy constraints. This is similar to the objective of the traditional methods, the difference being the representation was learned using data rather than through mathematical modelling by experts.

The functioning of the encoder can be visualized with the help of constellation diagrams. These are obtained by taking the following steps:

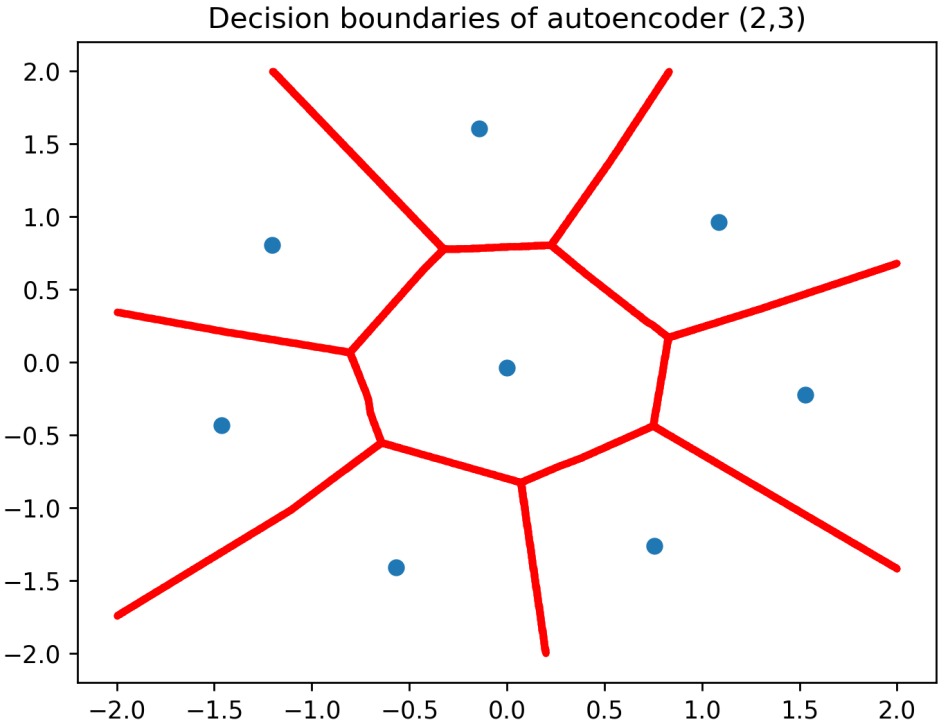


Figure 2.3: Decision boundaries obtained for 3-bit-2-channel estimator

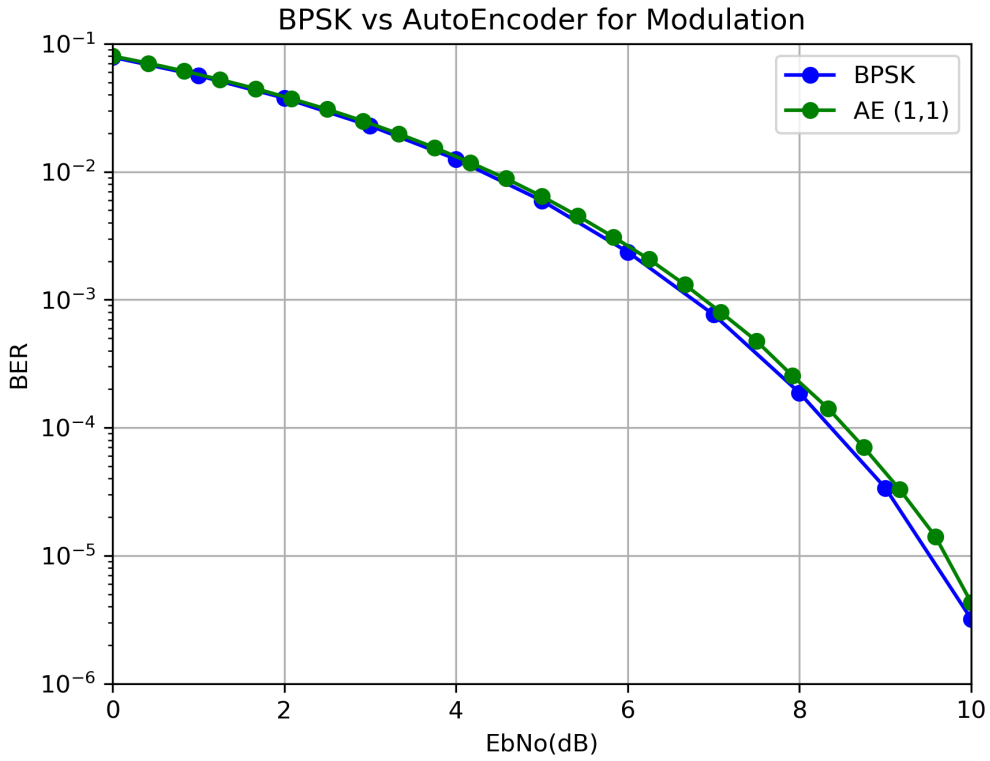


Figure 2.4: Comparison of BPSK and a 1-bit-1-channel AE model

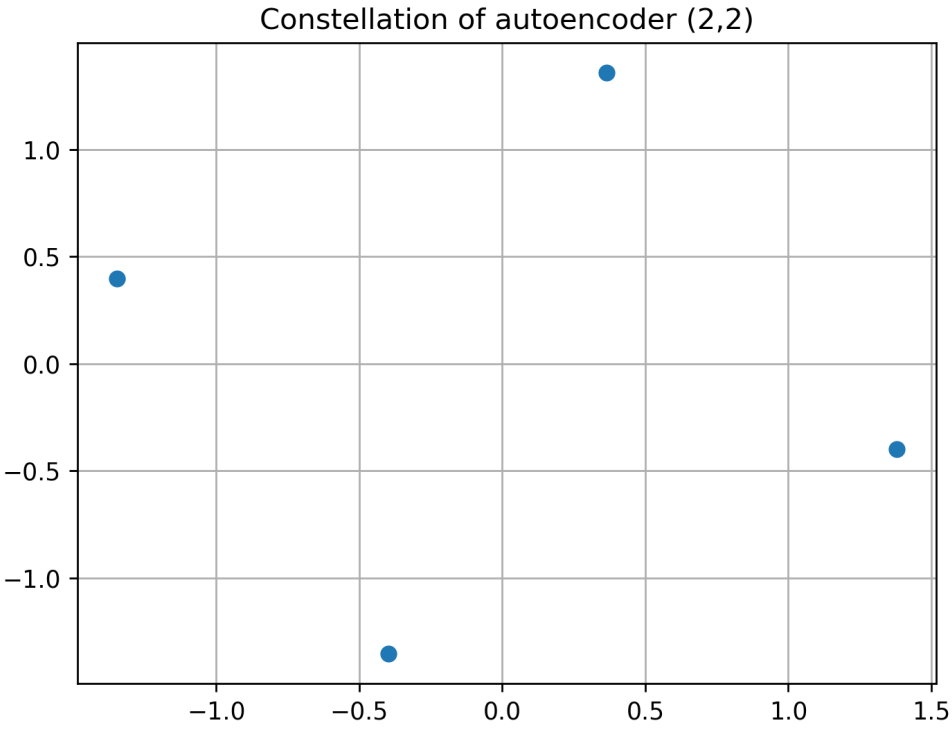


Figure 2.5: Constellation plot for a 2-bit-2-channel estimator

- Pass all possible input messages through the encoder to obtain high dimensional encodings.
- Convert the high dimensional encoded state to a two dimensional state using TSNE[39].
- Plot the two dimensional representations of the encodings.

For example, a 2-bit-2-channel estimator has a constellation plot as in figure 2.5. Here, the encodings are of the form

$$e^{j\left(\theta_0 + k\frac{\pi}{2}\right)} \qquad \forall k \in \{1, 2, 3, 4\}$$

which is the same as in Quadrature Phase Shift Keying.

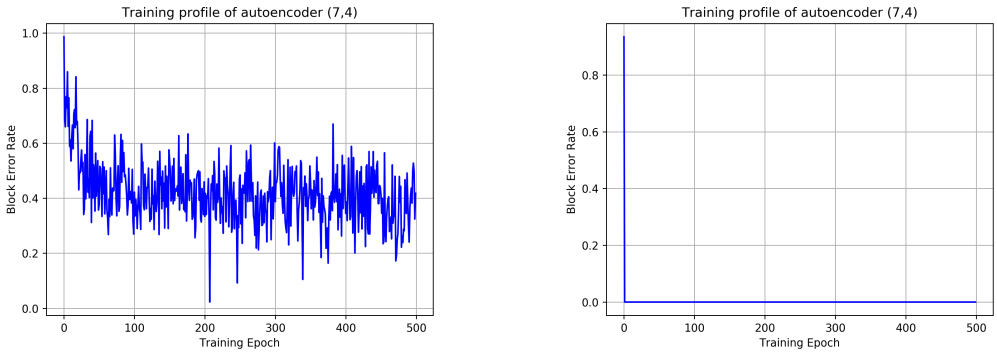
2.2.2 Channel Estimation without model

In the preceding subsection, even though no assumption was made regarding the distribution of the noise, the channel was modelled in the form of the input signal with an

additive independent noise

$$Y = X + N \qquad (X \perp N)$$

This allows one to compute gradients to train the network. However, this prior imposed on the model also limited the ability of the network to be used in new environments.



(a) Supervised model

(b) Policy Gradient model

Figure 2.6: Comparison of training profile for incorrect modelling of channel in the supervised case

Since the issue lies with having an explicit channel model, the logical solution is to remove it. But how then can the gradients be computed? To answer this, we make use of Reinforcement Learning. By treating the network as a stochastic policy maker, and training the policy by attempting to minimize the signal reconstruction error, one can make use of policy gradients to tune the parameters of the network. Since policy gradients are obtained empirically by sampling actions, they tend to be noisy and lead to (mildly) inaccurate optimization. This results in significantly larger train times and computational costs. But the key benefit achieved is a deep learning solution that makes use of no modelling by humans.

To make things more concrete, consider the following example. If the true channel model was $Y = X^2 + N$ and the supervised network modelled for a channel of the form $Y = X + N$, the neural network will fail to model the channel. This has been verified experimentally as can be seen in figure 2.6. The error rates of both trained models have been plotted for various SNR values in figure 2.7, thus proving the clear dominance of the reinforcement learning method when the channel cannot be modelled.

Following the work of Aoudia and Hoydis (2018), we implemented the policy gradient method to train neural networks for additive WGN and Rayleigh Block Fading

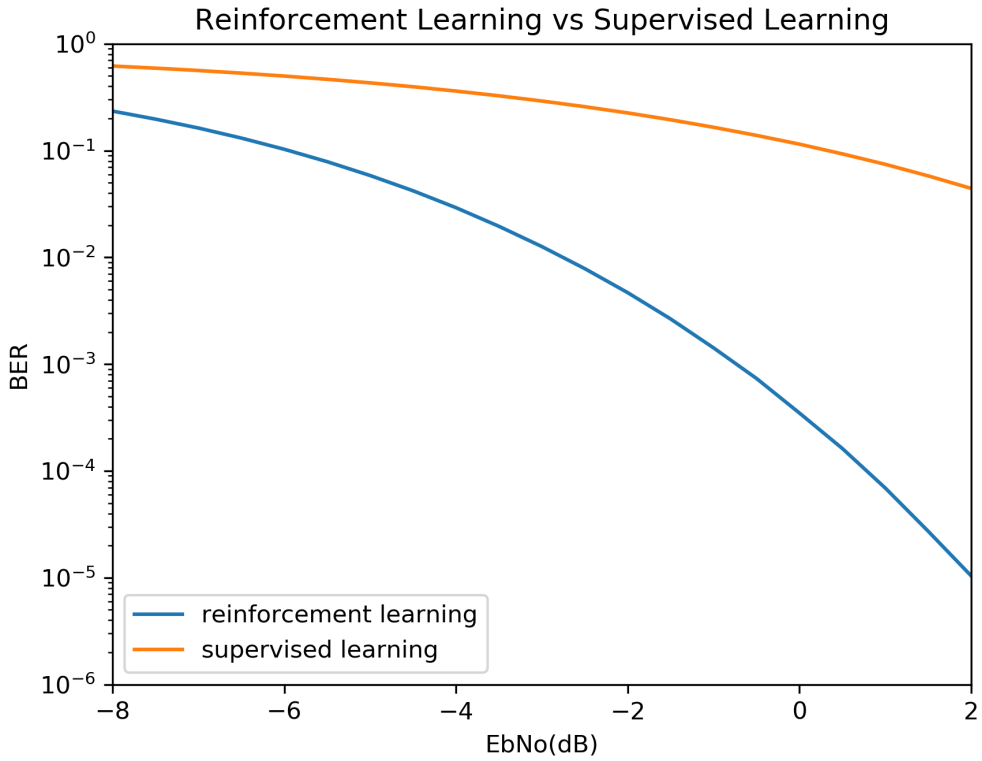


Figure 2.7: Comparison of Bit Error Rates (BER) for incorrect modelling of channel in the supervised case

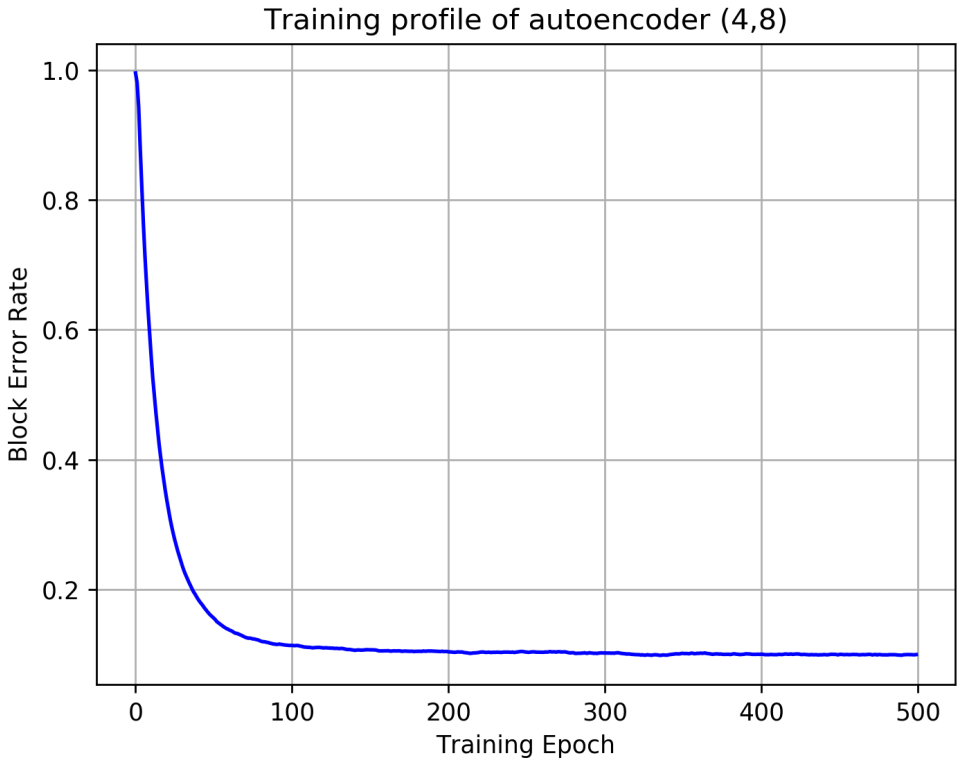


Figure 2.8: Loss Curve for channel without model

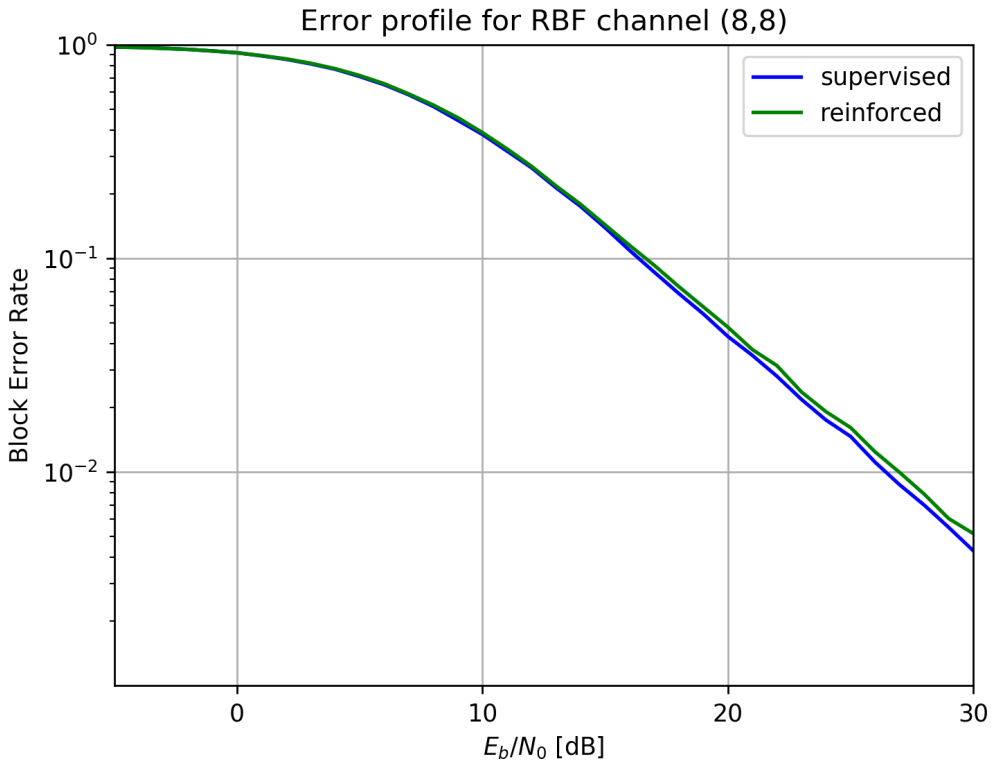


Figure 2.9: Loss Curve for an RBF channel which has been correctly modelled in the supervised case

(RBF) channels. However, the network converged much quicker than the original authors (see Fig. 2.8). Since the implementation details in the original paper were incomplete, the hyper parameters were tuned to get the best performance. While doing so, we took a large number of samples at each step when estimating the policy gradients. This resulted in policy gradients which were more accurate which may have led to the rapid convergence observed.

We also compared the performance of supervised versus policy gradient method against an RBF channel which was correctly modelled. This would give us a bound on the performance of the policy gradient model. Figure 2.9 shows the performances were nearly identical thus proving that the RL-based model performs at least as well as the supervised model in all types of channels. To better train the RBF network, we added an additional clamping layer that helps the network ignore the gradients of overly noisy samples. This protected the RBF network from the problem of explosive gradients, thus ensuring smooth convergence of training.

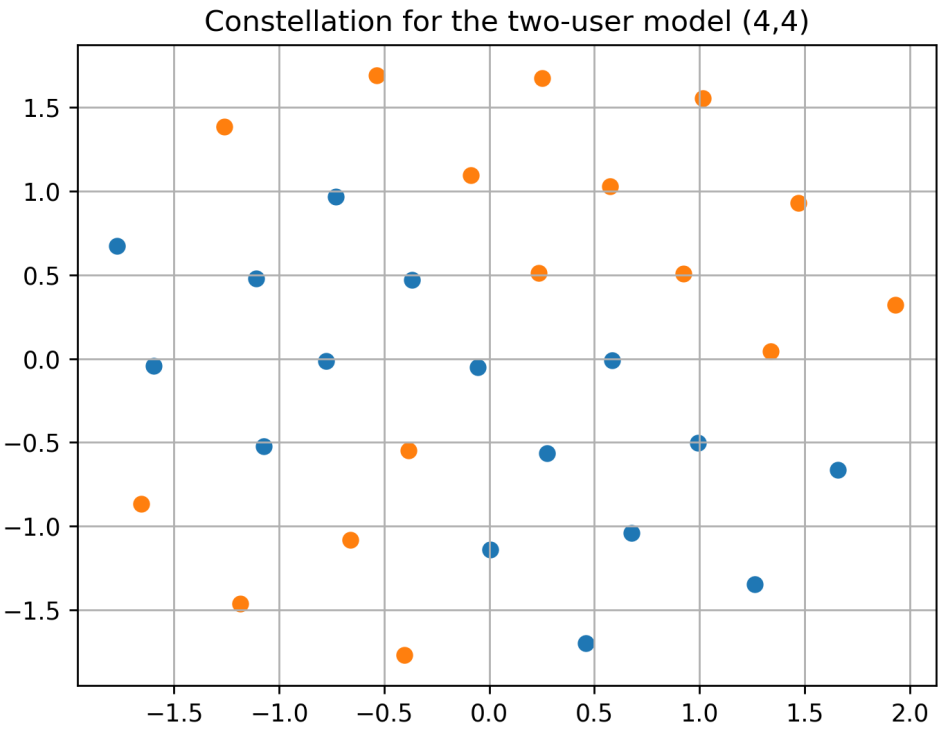


Figure 2.10: Constellation plot obtained for interference channel

2.2.3 Multiple Transmitters and Receivers

Deep Learning can also be used to find the best encoder-decoder configuration to protect the signal against interference from other systems using the same channel. This is especially important in the case of wireless communication where multiple users may be using the same channel at the same time. By following the work of O'Shea and Hoydis (2017), we implemented a two-transmitter-two-receiver model to understand how deep learning embeds input signals of one system to protect them from interference with transmitted signals from the other system. By plotting the input encoding in the form of a constellation plot (as can be seen in Figure 2.10), it becomes apparent that the orientation of the encodings of each system is, in a sense, perpendicular to the other. This orthogonality criterion effectively nullifies the effect of interference and so all contribution to the bit error rate is through noise in the channel.

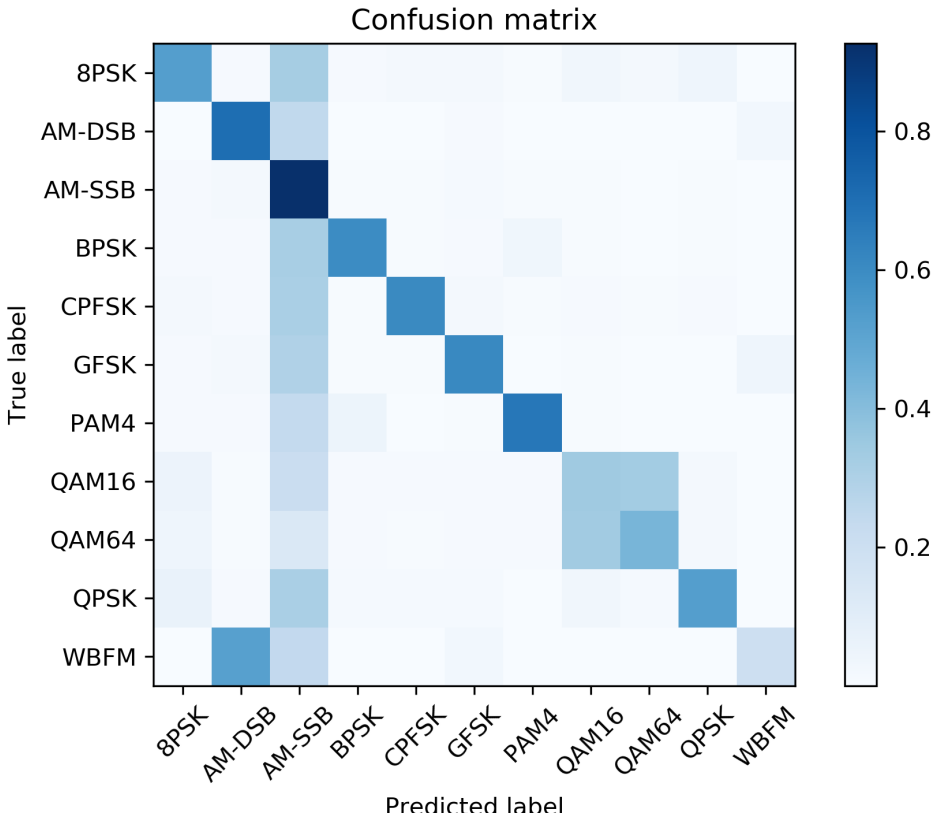


Figure 2.11: Confusion matrix for modulation recognition

2.2.4 Modulation Recognition.

Oftentimes in wireless communication, it becomes necessary to detect the type of modulation of the observed signal. This can be the case for several reasons. For example, if the device communicates with multiple other devices using different forms of modulation, the correct modulation type needs to be determined so that the appropriate decoding mechanism can be used. It can also be used to discard transmissions that do not need to be read. For example, if the system communicates solely using BPSK and detects a transmission that is of QPSK, then the message need not be decoded thus saving on computational costs.

O'Shea and Hoydis (2017) implemented a CNN architecture to solve the modulation recognition problem[5]. To visually analyse the predictive performance of neural networks for classification, we shall make use of a confusion matrix (Figure. 2.11). The confusion matrix does not only tell us how accurately the model predicted each class, but also which alternative does the classifier tend to get confused with. For example, in this case (2.11), we can see that the model gets especially confused between QAM16 and QAM64 modulation types. If this debilitates the system, a specialist binary classi-

fier can be designed to differentiate between the two specific modulation types. Besides that, we can also see that when the model is completely unsure between all classes (such as when the SNR is too low), the model defaults to Single Side-Band AM.

The above examples which have been elaborated on certainly do not form an exhaustive summary of all the applications of deep learning in communications systems. However, they provide enough use cases to allow us to bring out the limitations of applying deep learning models for commercial purposes, would therefore allow us to explore certain techniques to make them more feasible for practical usage. For a more thorough summary, one may look to the works of Simeone (2018) and O'Shea and Hoydis (2017) which survey data-driven approaches to communication systems.

CHAPTER 3

COMPRESSING DEEP LEARNING MODELS

In the preceding chapter, we explained how the universal approximation theorem guarantees the representative power of neural networks and gave a few example scenarios where neural networks perform as well as classical implementations designed through mathematical modelling by experts. However, one key limitation of the universal approximation theorem is it gives no upper bound on the number of neurons required to accurately approximate the function.

Since we don't have an upper bound to work with, practitioners of deep learning always end up using more neurons and parameters than necessary when designing neural networks for communication. So this is the price we pay for the enormous flexibility of neural networks: high computational costs. Implementing neural networks for communication purposes can be a bottleneck and slow down the overall system. Also, since most communication models are implemented on embedded devices, deep learning solutions involve significant memory overheads which hinder inference speed. For example, the ResNet-152 model16, which is a popular image classifier, takes 124.67ms to process a single image on an NVIDIA Jetson TX17 and takes around 244MB of storage space. This is beyond the ability of most microprocessors and embedded devices such as those used in the internet of things, where adaptive communicators can make use of deep learning[23] to handle a wide variety of environments.

The source of all the issues associated with deep learning is the same as the source of all the benefits, namely redundancy. The same redundant parameters that allow a neural network to approximate unknown functions without prior expert knowledge incur additional computational costs. Therefore the key to improving the efficiency of neural networks is to minimize the redundancies involved. But to detect the redundancies, one needs additional information. Therefore, most of the approaches we discuss here will be inherently tied to the training process of the neural network. More specifically, we shall discuss certain training strategies that be used intentionally in order to make it easier to eliminate such redundancies.

In the later chapters of the report, we shall explore various strategies employed to improve the efficiency of deep neural networks in general and analyze their performance in communication systems tasks.

CHAPTER 4

KNOWLEDGE DISTILLATION

Knowledge distillation is a method of compressing neural networks which was first proposed by Bucila et al. (2006). The version which is explored here is a generalization proposed by Hinton et al. (2015). This method is primarily used for classification tasks and is therefore applicable in all the aforementioned communication system tasks.

Knowledge distillation works on the assumption that while a small neural network can theoretically approximate the mapping as desired, it may fail to do so because of the optimization strategy. All current optimization strategies like SGD[43], Nestorov momentum, Adam[22], etc. mostly converge to local optima and fail to reach the best possible solution for the model architecture. One way of avoiding sub-optimal performance of neural networks due to local optima is to add additional layers. In doing so, the local optima get converted to saddle points due to the presence of additional parameters and the neural network can be much more easily optimized. The cost of doing so is the additional computational requirements for the resulting larger network.

Knowledge distillation solves this problem by training a large network which attains much better performance than the original network. This network shall be referred to as the teacher network. Then a new shallow network is designed with fewer parameters. This network is called the student network. Training the student network on its own results in sub-optimal performance as mentioned above. This is because the classification targets of the neural network for training are simple one hot vectors. But with the help of the teacher, one can generate soft (probability distribution) targets to train the student with more information. Since soft targets contain much denser information compared with one hot vectors, this results in the elimination of many local optima. Comparing the student network trained with and without knowledge distillation, we notice a significant improvement in overall performance with no change in model size. In context of this report, since the objective is to improve the efficiency of the neural network, we shall compare the performance of the teacher and student network.

It pays to better understand the role of the soft targets created through the teacher

network. The following interpretation was proposed by Hinton et al. (2015). When a neural network is trained for classification purposes, its objective is to predict the correct class. However, it also inadvertently captures the relative similarity between other classes in doing so. For example, let us consider the case where the network is trained to classify the type of modulation used. If the true modulation scheme used is QAM16, the network is likelier to misclassify the signal as QAM64 rather than say, BPSK modulation. This is captured in the probability it assigns to each of the classes. For example, it may assign QAM16 a probability of 0.8, QAM64 a probability of 0.19, and BPSK a probability of 0.01 thus highlighting the similarity between both QAM variants of modulation.

Therefore by training the student with the refined information learned by the much larger teacher model, the performance of the student model improves significantly as shall be shown in the succeeding experiments.

4.1 Experiments

We have conducted some experiments applying knowledge distillation on deep neural networks for communication purposes and shall elaborate on each of them in this section.

4.1.1 Channel Estimation

The network discussed in 2.2.1 contains one hidden layer. We shall consider the case where the number of channels is 8 and the number of bits transmitted in a message is 8. The encoder and decoder together contain 135944 parameters as can be seen in table 4.1. Here, we can see the majority of parameters lie in the first and last layers. The authors of O'Shea and Hoydis (2017) have added these layers to prevent the training process from being stuck at local optima.

We attempted to make use of knowledge distillation to circumvent the local optima using soft targets which were generated by passing the input through the teacher model. This would allow us to train a smaller model with just 4360 parameters (refer Table 4.2).

Layer type	Number of Parameters
Linear (encoder)	65792
Linear (encoder)	2056
Linear (decoder)	2304
Linear (decoder)	65792
Total	**135944**

Table 4.1: Teacher model for channel estimation

Layer type	Number of Parameters
Linear (encoder)	2056
Linear (decoder)	2304
Total	**4360**

Table 4.2: Student model for channel estimation

However, contrary to the the observations of O'Shea and Hoydis (2017), we observed that for the additive WGN channel, even the smaller student network by itself converged to the best optimum. This could be the case for the following reasons.

- Different optimizer for training. The reasoning made in [31] was for SGD and works for SGD while the neural network in our experiments was optimized using Adam. The authors too made use of Adam in their experiments.
- The batch size and stopping criterion were not mentioned in O'Shea and Hoydis (2017) so we optimized the hyper parameters of the network ourselves to maximize performance. This could have led to the better performance observed in the models we trained.

Figure 4.1 shows the performance of the teacher, student alone, and student with the help of the teacher. The difference in performance between the three versions is insignificant and perhaps even nil once statistical effects are taken into account.

Whatever the reason, since the student network on its own matched the teacher network in performance, using knowledge distillation for this task seemed to be unnecessary. The most probable reason for this is the size of the network used. The network, being merely two layers deep, was too simple to compress further through knowledge transfer. With that in mind, we looked for applications in communication systems which required the usage of much more complex neural networks.

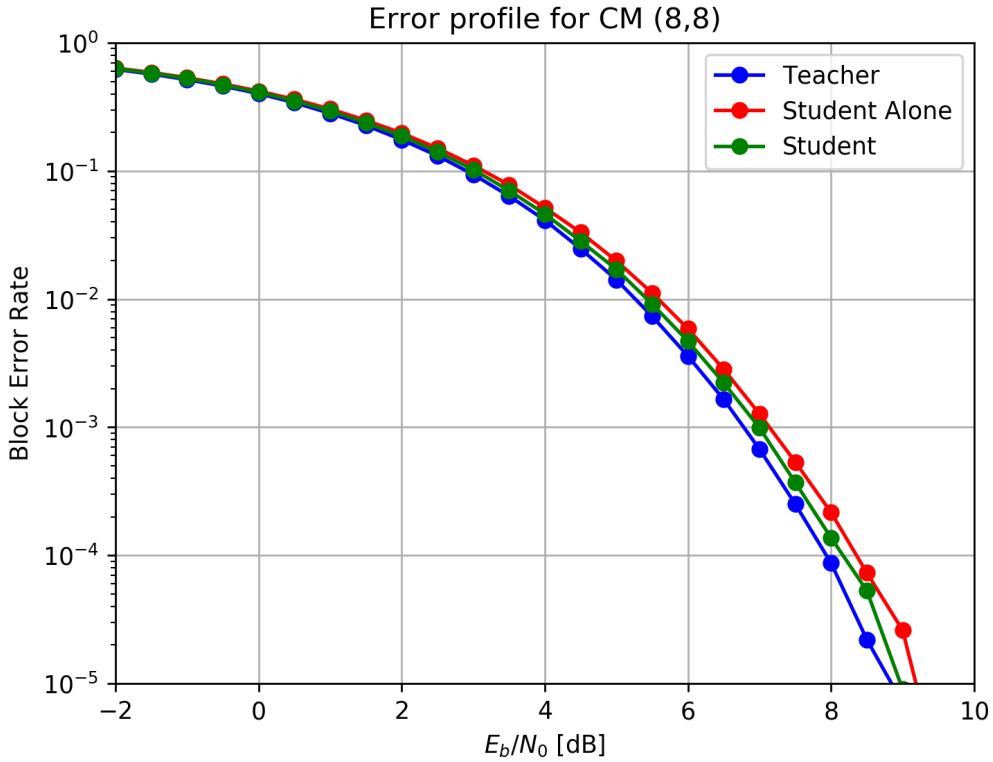


Figure 4.1: Knowledge distillation for channel modulation

4.1.2 Modulation Recognition

Convolution Neural Networks (CNNs) can be used for modulation recognition tasks. CNNs typically contain millions of parameters and are therefore serve as good candidates for demonstrating the power of knowledge distillation. O'Shea et al. (2016) have implemented a CNN model to distinguish between 11 different modulation schemes using signal data. Since the source code was made available, we made use of it for our experiments. To train the CNN, we made use of a dataset created by radioML[32]. Instead of using dataset (b) like the previous authors, we made use of the smaller dataset (a) for the experiments since the dataset (a) was sufficient to explore the qualitative aspects of knowledge distillation.

The student model was first trained on its own and it achieved a peak performance (at high SNR values) of about 70%. The width and depth of the network where then increased. The new model, designated as the teacher, was then trained (from scratch) to achieve a peak accuracy of about 80%. Finally the student model was trained with the help of soft targets obtained through the teacher model and this performed nearly as well as the teacher model.

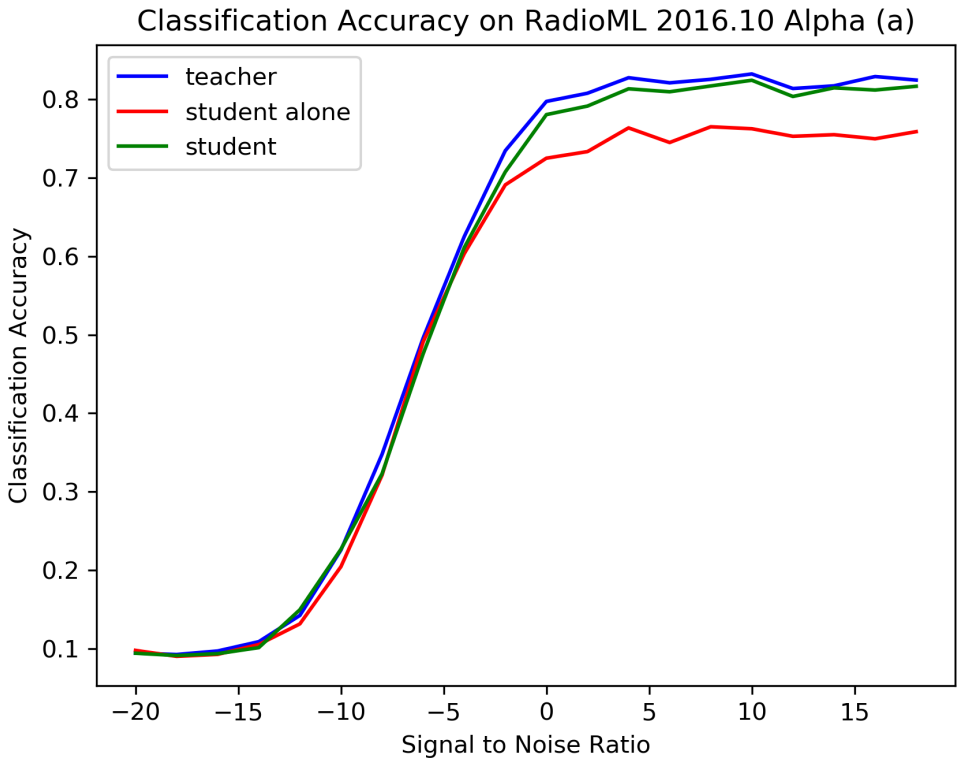


Figure 4.2: Knowledge distillation for modulation recognition

Note: In order to make the experiment rigorous, we have not adjusted any of the hyper parameters in the experiments. The only difference between the teacher and student models is the architecture. The training procedure, optimizer, learning rate, batch size, etc. are exactly the same in both cases. These steps were taken to ensure a fairer and more controlled evaluation of the models.

The architecture and size of the teacher model can be seen in table 4.3 while those of the student model can be seen in table 4.4. We measured the classification performance of the teacher and student models for various values of Signal to Noise Ratio (SNR)

Layer type	**Number of Parameters**
conv1 (Conv2D)	1024
conv1.5 (Conv2D)	196864
conv2 (Conv2D)	196736
conv3 (Conv2D)	98432
dense1 (Dense)	8651264
dense2 (Dense)	131328
dense3 (Dense)	2827
Total	**9,278,475**

Table 4.3: Teacher architecture for modulation recognition

Layer type	Number of Parameters
conv1 (Conv2D)	1024
conv2 (Conv2D)	122960
dense1 (Dense)	2703616
dense2 (Dense)	2827
Total	**2,830,427**

Table 4.4: Student architecture for modulation recognition

Deep Learning Model	Inference time per data point
Teacher	66.5 ms
Student	18.5 ms

Table 4.5: Benchmark results for Knowledge distillation for modulation recognition

and have plotted the results in figure 4.2. We can see that the student performance has significantly improved under the teacher's tutelage especially when the SNR is low. Comparing the teacher and student performance, we have managed to reduce the model size by over 3 times without significant compromise on accuracy.

Furthermore, all the deep learning models have been benchmarked on a Raspberry Pi 3, one of the higher-end embedded devices today. It has a quad core 64-bit 1.4GHz processor and 1GB of RAM. Benchmarking on this device on would give one an idea of how much latency to expect from average embedded devices in the future. To account for the effects of caching, we measured the execution time for 110 data points and discarded the first 10 time measurements. As can be seen in table 4.5, the student model inferred predictions over 3 times faster than the teacher model. The precise ratio between execution times (3.59) is roughly similar to the ratio between number of parameters (3.27). Therefore, if benchmark results are not available, the ratio of the number of parameters provides a suitable alternative to measure relative efficiency of the neural network.

4.2 Other variants

The type of knowledge distillation discussed here is the one proposed by Hinton et al. (2015). Over time, several variants of knowledge distillation have been proposed. Mirzadeh et al. (2019) have proposed a version where knowledge is transferred in stages through intermediate "teacher assistants". This increases the compressing power of

knowledge distillation which is especially useful for communication tasks. Huang and Wang (2017) formulated knowledge distillation as a distribution matching problem introducing new loss functions that help minimize the performance differences between teacher and student. Last but not the least, Mishra and Marr (2017) have shown how to use quantization, a method which shall be discussed in the next chapter in tandem with knowledge distillation to further improve the efficiency of deep neural networks.

CHAPTER 5

QUANTIZATION OF PARAMETERS

All current implementations of neural networks make use of 32 bit floating point numbers to represent the parameters of a neural network. To save on storage, one can make use of fewer bits to store and represent each parameter. While the process would introduce a significant amount of Least Significant Bit (LSB) errors, this could be treated as a form of noise, and neural networks have in general proven to be robust to noise.

The efficacy of quantization can also be tied to the redundancy inherent in neural networks. To more clearly see it, consider the following case: If a neural network is wider than minimally required, then the final output is going to be equivalent to the (weighted) mean of a set of random variables. Since the variance of the mean of a number of iid random variables is always less than the variance of one of the random variables, the effect of new (LSB) noise is also lessened.

The most common form of quantization is 8-bit quantization. In this case, the parameters are quantized layer-wise. For each fully connected layer, we took the minimum and maximum value from each weight tensor. We then allocate each parameter to one of the $2^8 = 256$ levels. Now for each weight matrix, we can store the minimum, maximum and an 8-bit number to indicate the level. To convert the 8-bit number to the original value, we make use of the following expression

$$true\ val = min + \frac{level}{256} * (max - min)$$

Using this method, the size of the model can be compressed by a factor of almost 4 (some extra space is required to store the lower and upper bound values). Furthermore, the smaller model would better fit in the typically small caches of embedded systems speeding up execution even more. This is true for all compression methods in general, but even more so for quantization. Last but not the least, some embedded devices make used of Single Input Multiple Data[38]. This speeds up computation by an additional factor of (almost) 4 thereby improving computational efficiency.

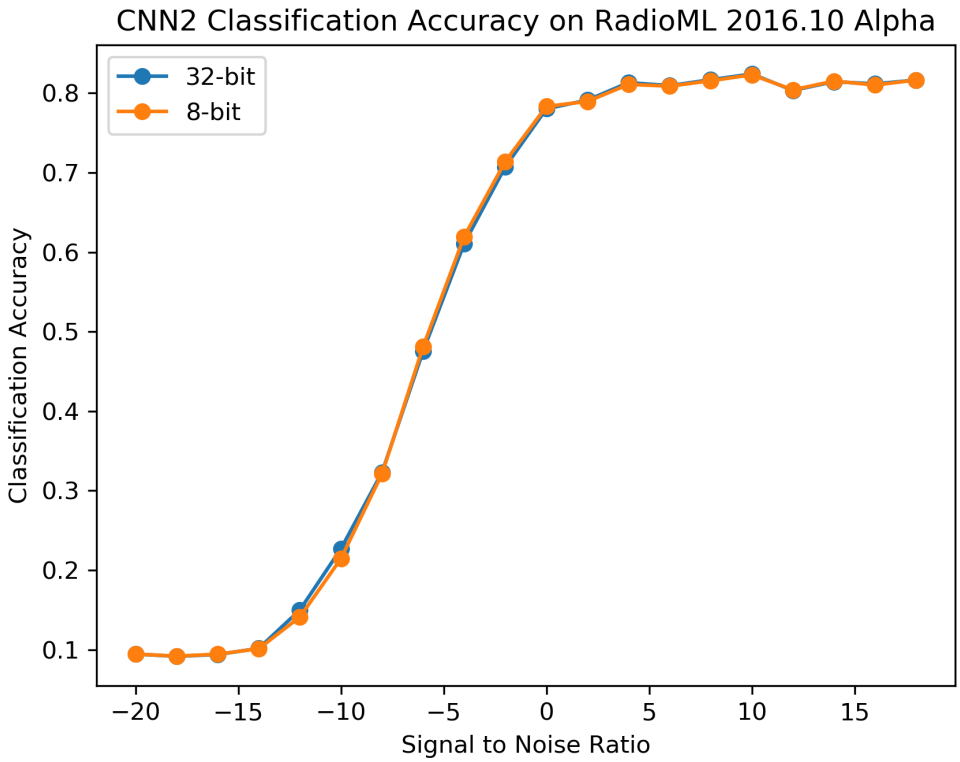


Figure 5.1: 8-bit quantization for modulation recognition

5.1 Modulation Recognition

We attempted to quantize the parameters of the CNN created by O'Shea et al. (2016) to analyze its performance in modulation recognition. On doing so, the model size reduced greatly while the inference time remained more or less the same. The accuracy of the models for various levels of SNR has been plotted as shown in figure 5.1. The performance of the quantized and unquantized models are nearly identical. This implies the quantized network is still larger than necessary. One could therefore attempt to quantize the model even further (to say 4-bits or 2-bits) and measure the performance to see if further compressing is feasible.

5.2 Channel Estimation

Here we analyze the effect of quantization of parameters on the neural network developed to solve the channel estimation problem. We first performed 8-bit quantization to understand the effect of simply quantizing a pretrained network. As can be seen in ta-

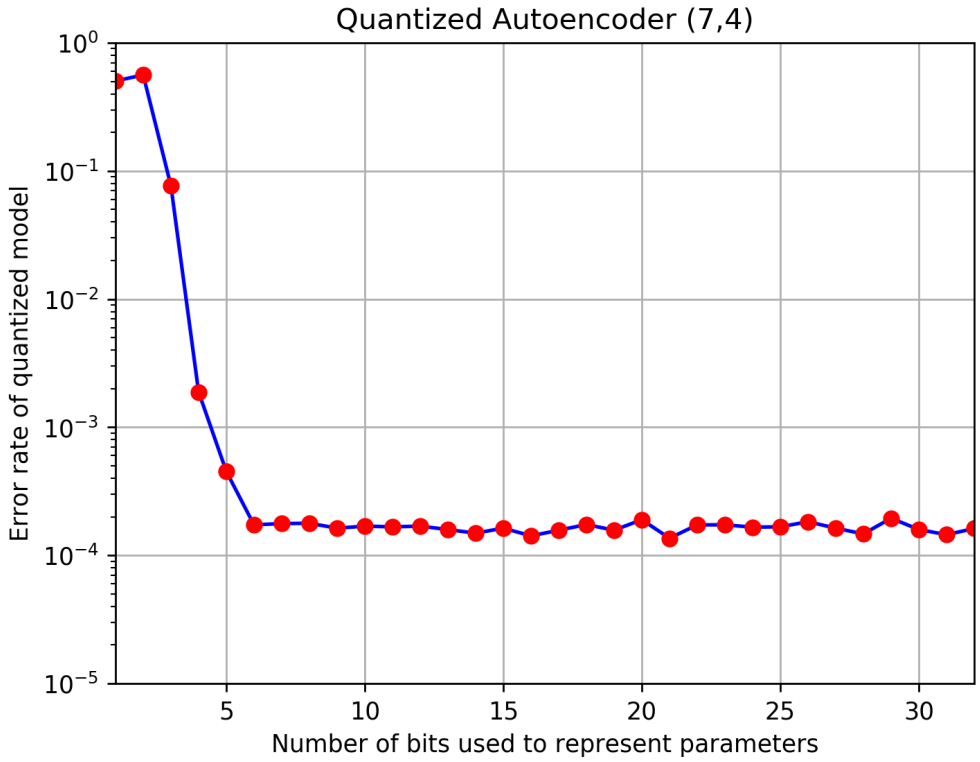


Figure 5.2: Error of classification for various quantization rates

Parameter type	**Accuracy**	**Error Rate**
Full Precision Floats	98.73%	1.29×10^{-2}
8-bit Integers	98.71%	1.29×10^{-2}

Table 5.1: 8-bit quantization for channel estimation

ble 5.1, the classification accuracy of 8-bit quantization is about the same as that of full precision floats. The model contained 7 channels with additive WGN (SNR=2.5119). The messages which were transmitted through all the channels contained 4-bits, giving us a total of 16 different possible messages. In figure 5.2, the parameters of the model have been quantized to various levels and the resulting quantized models have been evaluated. As can be seen, as long as the parameters are represented by at least 6 bits, the classification performance is not afflicted. This results in memory savings of approximately

$$\frac{32-6}{32}\times 100\% = 81.35\%$$

Based on the memory and performance constraints, one may choose the optimal quantization rate for the task at hand.

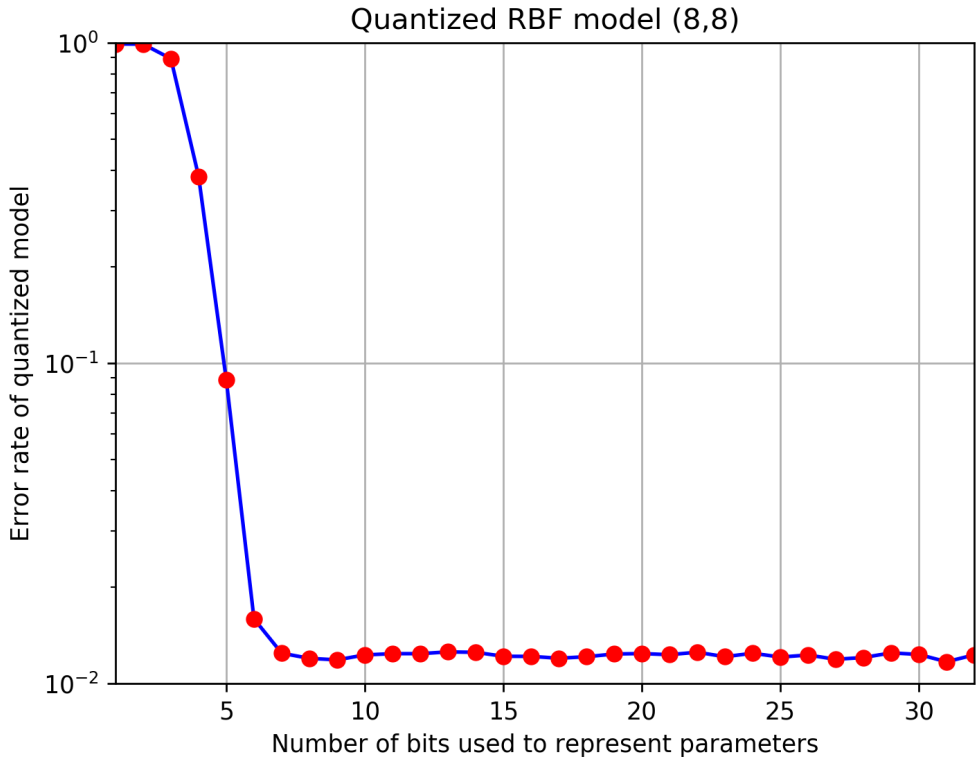


Figure 5.3: Error in RBF channel estimation for various quantization rates

5.3 RBF Channel

In the previous case, we have seen how the model fares against additive WGN. Now we analyze the performance of quantized networks in an RBF channel. This is necessary since the RBF neural network contains a division layer which tends to be very sensitive to noise, such as the LSB errors introduced by quantization. The plot of accuracy vs compression rate can be found in figure 5.3. Surprisingly, the effect of quantization is not more pronounced in the RBF channel as expected. As long as the network parameters are encoded with at least 7 bits, there would be no loss in error. This result stands testament to the robustness of deep learning models to noise.

5.4 l2 regularization for improved performance

The extent to which LSB errors afflict the network post-quantization depends on the largest and smallest parameter for each weight. The greater the range of weights within a layer, the greater the LSB errors and the greater the overall loss in accuracy due to

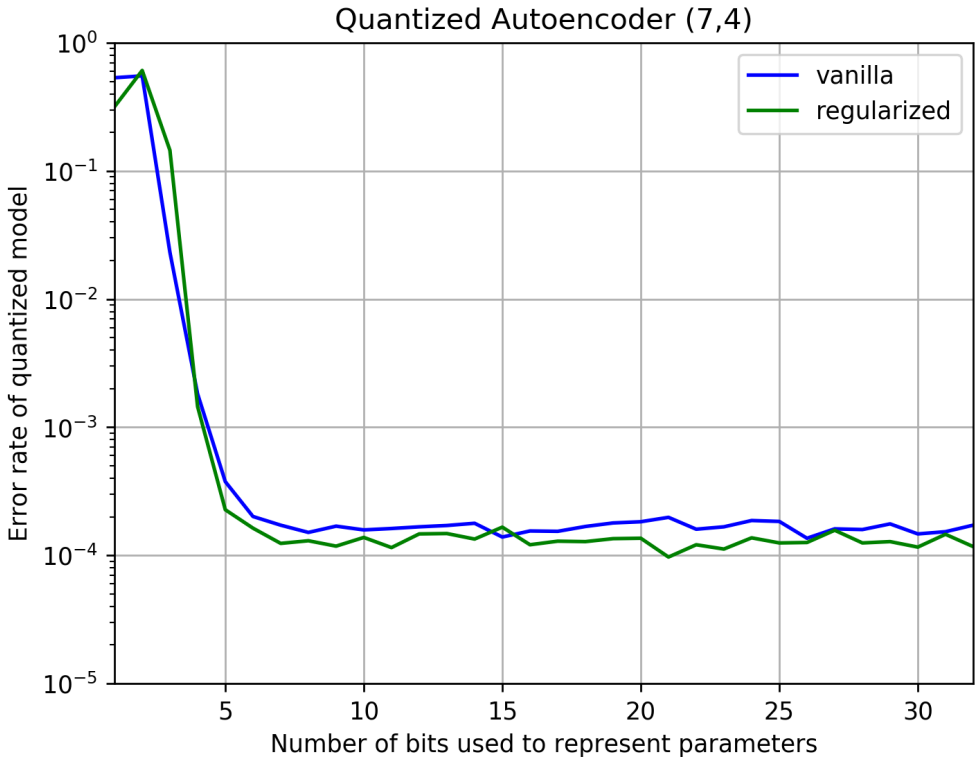


Figure 5.4: Channel estimation error for different quantization strategies

quantization. Therefore, in order to improve the robustness of trained models to quantization, we trained the networks with regularization to minimize the range of weights. Between l1 and l2 regularization, we decided to use l2 regularization since the error of quantization is most sensitive to singly large weights which increase the real-width of each bit representation thereby increasing LSB errors; the values of the intermediate weights do not matter. We compared the error rates of regularization+quantization with the vanilla version of quantization for an SNR of 2.51 and have tabulated it in table 5.2. As predicted, the regularized model was more robust to noise introduced by quantization. The performance of the proposed version of regularization+quantization has been compared against the standard vanilla version of quantization and has been plotted in figure 5.4. While performance has been improved, it is less significant than expected. This is primarily because the neural network was much more robust to quantization errors than expected. This left little room for improvement.

Parameter type	Accuracy	Error Rate
Full Precision Floats	98.70%	1.30×10^{-2}
4-bit quantization	97.23%	2.77×10^{-2}
4-bit quant+l2	97.42%	2.58×10^{-2}

Table 5.2: 4-bit quantization for channel estimation

5.5 Other variants

Quantization presents an effective way to compress neural networks for deploying on embedded devices. Due to its popularity, several variants of quantization have been proposed which can further improve the effectiveness of quantization in fields like communication. Lin et al. (2016) have shown how to optimize quantization by varying the widths of the bit representation to minimize quantization errors. This process would incur greater one-time costs but over time, the improvement in performance would make up for it, especially if the use-case is performance-critical. Hubara et al. (2016) have proposed a method for training and running inference on 1-bit weights and activations for image processing tasks. This work can also be extended to efficiently train communication system models directly on embedded devices. Since bit operations execute much quicker than float operations, this can increase the overall throughput of the communication system used.

CHAPTER 6

PRUNING OF PARAMETERS

Neural networks are typically trained with more neurons than required. This redundancy is crucial to enable the high representative power of neural networks for a wide variety of communication tasks. However, after training the network for the task at hand, the extra neurons serve no purpose and only make the network more expensive in terms of time and computational requirements. The simplest and most obvious solution would be to remove or prune the useless neurons. In this chapter, we discuss the idea of pruning the weights of a neural network.

In order to prune the neural network, one must remove the weights of the network that have the least impact on the final output. This would cause minimal affliction to the performance of the network. Therefore we remove the weights of the network that have the smallest absolute value. The weights are removed by setting them to zero. Then with the help of a sparse representation of the weight matrix, one can save on memory space. This process would add a certain amount of noise to the network. But as we have mentioned earlier in the case of quantization, neural networks are robust to small amounts of noise. Han et al. (2015b) have shown the efficacy of pruning in image recognition tasks as we shall do so in the context of communication.

The following are the steps involved in pruning

- Group all the weights of the network.
- Sort them according to magnitude.
- Set the bottom x percentile of weights to zero.

The hyperparameter x would decide the compression rate. The greater the value of x, the more the amount of compression but the greater the degradation in performance. Sometimes, in the case of classification, the final layer is spared from the pruning process since the following softmax function can exponentially amplify the errors, thus hurting performance.

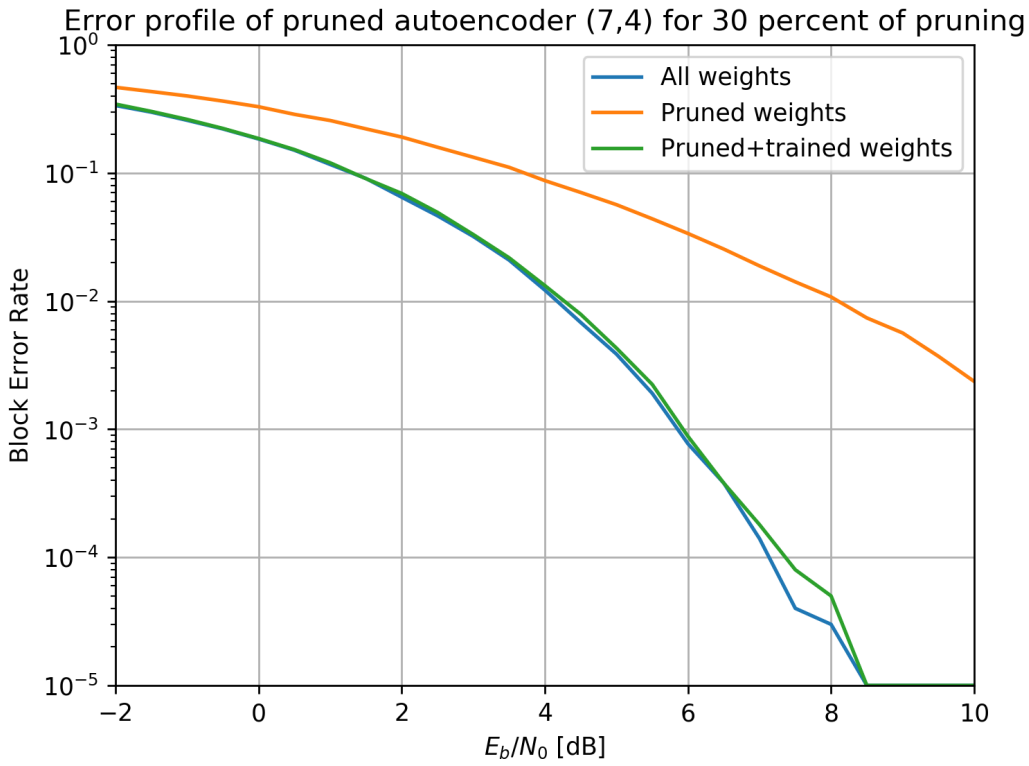


Figure 6.1: Pruning for channel estimation

The steps involved in pruning mentioned above is based on the work of See et al. (2016). For our experiments, we made use of an implementation of pruning in PyTorch which was taken from Github[4].

6.1 Channel Modulation

To solve an 8-bit-8-channel modulation problem, we designed a neural network which in total contained 135,944 parameters (for more details refer table 4.1). On pruning the network of half of the weights, the error rate rose by a factor of about 8. On retraining the remaining weights however, the performance of the network reaches the original performance levels. The results obtained have been tabulated in table 6.1.

We then plotted the performance of a 4-bit-7-channel model against noise after pruning, with and without retraining. In Figure 6.1, one can see the performance of the network for various levels of SNR. While the effect is not significant for low SNR, retraining becomes paramount to the efficacy of pruning. After retraining the pruned network however, it performs approximately as well as before for all SNR values.

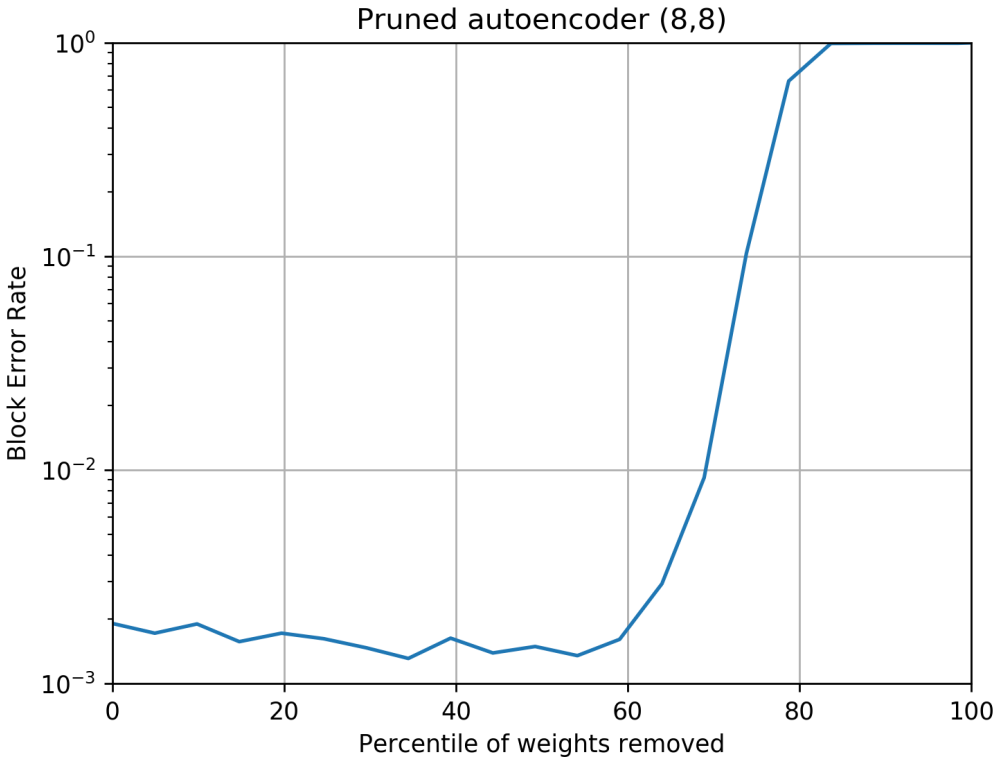
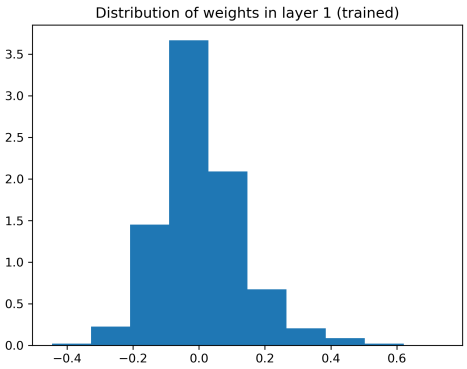


Figure 6.2: Effect of pruning rate on channel estimation error

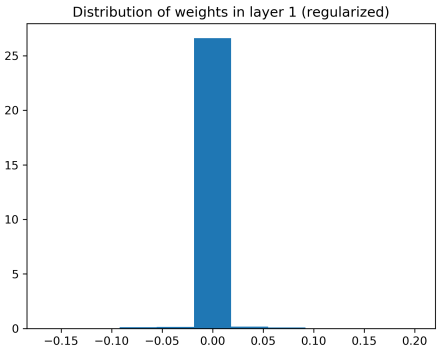
Network type	**Accuracy**	**Error Rate**
No pruning	99.47%	5.26×10^{-3}
Pruned network	95.63%	43.7×10^{-3}
Pruned then retrained	99.39%	6.08×10^{-3}

Table 6.1: 50 % pruning for channel estimation

To better understand the effect of pruning on the network, we have plotted the performance of the resulting network against the pruning rate (Figure 6.2). This would help practitioners of deep learning in communication systems decide the pruning rate based on the performance and efficiency constraints presented to them. As can be seen, for this case, upto half of all the weights can be removed from the fully connected layers without significantly affecting performance. In fact, in some cases, pruning has marginally improved the overall accuracy. Therefore one can greatly save on storage space without affecting performance.



(a) Without Regularization



(b) With l1-regularization

Figure 6.3: Comparison of weight distribution in the presence of regularization.

6.2 Regularized Training

Pruning works by setting small insignificant weights to zero in order to save on memory. Usually, after training the weights of a neural network, they belong to a normal distribution. For example in Figure 6.3a, a lot of the weights are close to zero. This property helps reduce the overall error introduced by pruning. However, as the earlier experiments have shown, pruning still introduced a substantial amount of performance degradation which can only be nullified through retraining. Here, we explore a strategy to make a network more resilient to noise introduced by pruning, namely regularization.

Regularization is a method used to set priors on the weights of a neural network. It is primarily used to avoid overfitting: good performance on training data but terrible on unseen data. Overfitting is usually characterized by large parameter values. Therefore, to penalize such parameters, an additional loss term is added. The most common type of penalization is called l2-regularization.

$$loss = CEloss + \alpha(weights)^2$$

The above modified loss function penalizes large weight values. So when the neural network is trained against the modified loss function, it attempts to use smaller weights while optimizing. This reduces overfitting thus improving the quality of the deep learning model.

In our case, the objective is to set priors such that the network prefers weights that are close to zero be exactly zero. This would minimize the overall effect of pruning

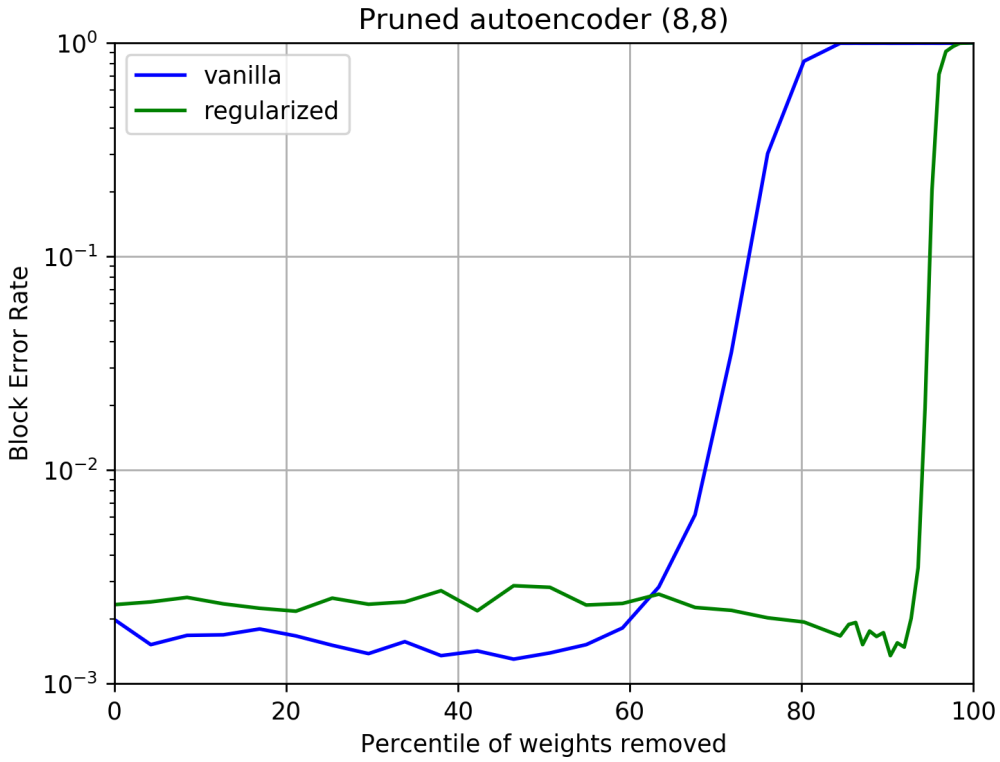


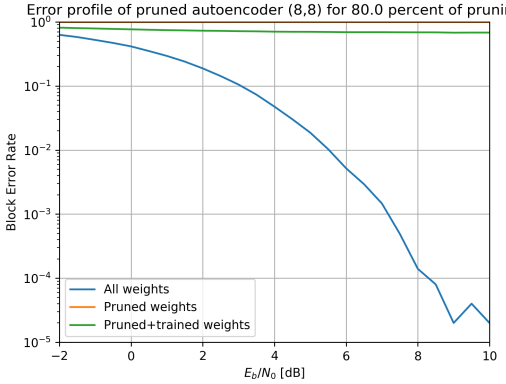
Figure 6.4: Estimation accuracy of pruning with and without regularization

on the network thus saving on memory with minimal performance drops. Instead of l2 regularization which we used with quantization we experiment with another regularizer: l1 regularization.

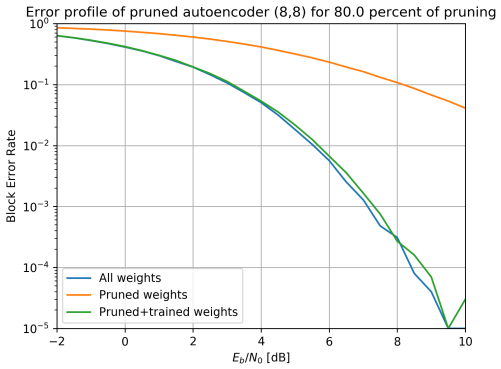
$$loss = CEloss + \alpha|weights|$$

By more strictly penalizing weights that are nearer to zero (having larger gradient update values), l1 regularization should perform better than l2 for this specific task. As can be seen in figure 6.3, training the network with l1 regularization pushed most of the weights to zero. This would minimize the overall changes made to the network while pruning, thus minimizing it's affect on the performance of the network.

The following is the procedure we follow: We first train the network with l1 regularization; then, we prune the network and measure its performance with and without retraining the network. As can be seen in figure 6.4, the regularized method does not perform as well as the vanilla version for low pruning rates. However, in order to speed up the network, one must make use of sparse representations of matrices and this is only possible for very high pruning rates ($> 90\%$). Given these constraints, pruning with regularization maintains its peak performance even when over 95% of the weights



(a) Without Regularization



(b) With l1-regularization

Figure 6.5: Comparison of pruning with and without regularization.

are pruned and is therefore the better option when compression is to be maximized. In figure 6.5, we compare the performance of pruning with and without regularization when 80% of the weights are pruned. The superiority of the regularized method is clearly visible, especially for high SNR values. This is in spite of retraining the same number of parameters in both cases. The reason for this difference in performance is after training the network with l1 regularization, the insignificant weights are easier to separate from the significant weights thus facilitating pruning.

6.3 Other variants of pruning

Besides the vanilla and regularized pruning methods which have been discussed here, several other variants have been proposed. We shall discuss a few of them briefly as pointers for future direction for research, as there currently exists a dearth of research on the efficacy of these methods in the context of communication systems. Han et al. (2015a) have shown how to compress neural networks using a strategy which combines pruning, trained quantization and Huffman coding to compress neural networks to increase the efficiency of the network by a factor of over 30 times without compromising on performance. This is one of the examples of using multiple compression strategies in unison. There are also other ways to set priors on neural networks to facilitate pruning. Rather than encouraging insignificant weights to be set to zero, as we did earlier, Liu et al. (2015) directly imposed a sparsity constraint on the weight matrices which allowed more weight connections to be dropped without degrading performance

in image classification. A promising next step would be to compare the performance of such sparsity constraints against simple l1 regularization for the purpose of pruning of weights in communication systems tasks.

CHAPTER 7

COMPARISON OF ABOVE METHODS

So far we have gone through the following methods of compressing neural networks in detail:

- Knowledge distillation
- Quantization of parameters
- Pruning of weights

While all of the above methods tackle the same root cause which is the redundancies inherent and necessary in neural networks, they are also different in many aspects. Since they are different, some of them may be preferable in some cases. In this chapter, we go through various situations where the methods reveal their differences therefore leading to the existence of a preference relation between them.

The objective of this report is to discuss the implementation of deep learning models on embedded devices for communications purposes. When compressing a model, some of the considerations that need to be taken into account are:

- Data requirements
- Execution time
- Memory overheads
- Universality

7.1 Data Requirements

One point of difference between the three different methods is the additional data requirements. For quantization—at least the version discussed in chapter 5—there are no additional data requirements. A trained model can be directly quantized to improved its efficiency. Though the performance has been improved when priors like l2 regularization are imposed, no data is explicitly used during the process. Pruning of weights can

theoretically be done without data. However, the performance of the method is greatly improved when the pruned model is trained again with data (Figure 6.1). Knowledge distillation on the other hand works by training a student model with the help of a teacher model. Therefore, without training data, this method simply cannot be used. One point to note is that the data required by knowledge distillation need not necessarily be labelled since soft targets are in any case generated through the teacher. Also, the amount of data required for knowledge distillation is far less than the amount used for training due to the inherent regularization feature when training with soft targets.

Therefore, if no data is available, quantization is the best way to go about compressing a neural network. Quantization is followed by weight pruning which should only be used if the space constraints are dire. Last and also the least, comes knowledge distillation. The method cannot be used without data and would therefore be unusable in this scenario.

7.2 Execution time

Another point of consideration when evaluating compression method is the attained speedup. Since the hardware used in communication systems are microprocessors which are much slower than devices used in data centers typically used to train large neural networks, one must also try to reduce the amount of time taken by the neural network to process one data point during inference. This would also give the increase in throuput rate or decrease in latency of the system.

Knowledge distillation compresses the model in the most literal sense and is therefore guaranteed to speed up the neural network.As can be seen in chapter 4, the speedup appears to be roughly equal to the ratio of model size.

Pruning of weights results in a sparser matrix. This can result in quicker matrix multiplications thus resulting in quicker inference times. However, in order to efficiently represent sparse matrices, the $(N \times N)$ matrices must have O(N) non-zero elements which would result in significant performance decrements. Even then, the algorithms that work on sparse matrices have to perform better than BLAS/LAPACK which have been intensely optimized for full-matrix multiplication. Therefore in most cases, pruning of weights does not result in a large speedup in inference.

In quantization, each parameter has an 8-bit integer representation rather than a 32-bit float representation. Since float operations are about as fast as integer operations in modern CPUs, there is not much speedup to be expected. However, some microprocessors contain SIMD[38] instructions which allow four 8-bit integers to fit in one 32-bit float register. In that case, quantization can result in a theoretical speedup of about four times. Using smaller integer representations coupled with dedicated hardware can further improve the efficiency of using quantization.

Comparing the above methods, knowledge distillation seems to be the best method as it's speedup is guaranteed and universal. Quantization of parameters can perform better once dedicated hardware is present. Pruning of weights currently results in the least amount of speedup due to lack of highly optimized algorithms for sparse matrix computations.

7.3 Memory Overheads

The primary goal of compressing neural networks is to minimize storage requirements. This is because communication systems are implemented on microprocessors which have low compute and memory capabilities. This becomes more important in the era of internet of things where multiple "smart" devices communicate with each other in environments which can't be modelled by experts and so require data-driven methods like the deep learning models discussed in chapter 2.

Comparing the above three methods, quantization and pruning with regularization result in the maximum amount of compression when it comes to memory overheads. Knowledge distillation takes the second place as it can't compress networks by a factor of about ten as in the other two methods.

7.4 Universality

Another factor to consider when comparing the various approaches of compressing neural networks is universality or compatibility. While there exist methods which work in very specific cases like for example; removal of redundant CNN filters; which only

	Distillation	Quantization	Pruning
Data requirements	unlabelled	none	supervised data or none
Speedup requirements	none	special hardware	fast sparse algorithms
Memory savings	~3x	~8x	~10x
Compatibility	universal	extra hardware	additional algorithms

Table 7.1: Comparison of various compression strategies

work when the deep learning solution consists primarily of convolution filters, we restricted our analysis to methods which work with a wider range of neural networks, especially those which contain fully-connected or dense layers.

Knowledge distillation can be considered the most universal of the three approaches: the student model is as compatible as the teacher model to hardware and so it can be implemented wherever needed.

While quantized networks theoretically work with the same hardware as the unquantized networks, its efficacy is maximized only when dedicated hardware or specialized instructions like SIMD[38] are present to leverage the reduced bit representation for operations like add and multiply.

Pruning of weights, like quantization and knowledge distillation, can work on all types of hardware. However, while the memory savings are significant, the speed can only be improved using algorithms[40] on sparse matrices that can outperform BLAS/LAPACK with full-matrices, which are hard to beat. While some algorithms[11] do exist, a standardized efficient implementation is yet to be made.

7.5 Mixed Methods

One must remember that the methods discussed above work in different ways and can so be used together to further push the limits of model efficiency. For example, one can first apply knowledge distillation to make the network as small as possible keeping performance the same and then quantize the weights to increase the throughput rate of the system[29]. Or one can quantize neural networks setting a significant number of them to be exactly zero. Then pruning those weights would lead to no further loss in performance while saving on memory[14]. Other combinations can also be experimented on depending on the type and application of the deep learning model.

The comparison between all the compression strategies discussed in the report has been briefly summarized in the form of a table and can be seen in table 7.1. The table also states the requirements for the abilities of method to be fully leveraged. This has been compiled using empirical evaluation done earlier in the report and therefore serve as guidelines rather than hard-and fast-rules.

CHAPTER 8

CONCLUSION

In this report, following a brief introduction to deep learning, we have explored various situations where deep learning can be used in communication systems. We also elaborated on a few specific cases where deep learning presents solutions which match expert-designed models. We then emphasized on the limitations of deep learning which are relevant to communication: time and space complexity.

Once the flaws were empirically shown and theoretically reasoned, we systematically explored various techniques to ameliorate the shortcomings of deep learning in communication systems like

- Training a smaller model using knowledge distillation
- Reduced bit representation of parameters as in quantization
- Removing redundant parameters through pruning of weights

Through these methods, the memory requirements of models could be reduced drastically without much compromise on performance. Furthermore, in some cases, the time complexity of the models could be significantly reduced, again, without significant degradation in performance. Overall, through the above methods, the computational efficiency of the deep learning models has been improved.

While we have discussed some of the more popular methodologies for compressing neural networks, there are many other ways to compress neural networks which may be of considerable interest to anyone in the field of communication engineering who wishes to implement deep learning models on embedded devices for practical usage. These include, but are not limited to

- SVD factorization of weight matrices for dense layers
- Removal of redundant filters in CNNs
- Dropping of dead RELU neurons

The above methods are not as universal as the methods which were elaborated on earlier since they work on specific layers or activation functions. But they still are different from the discussed methods and can therefore be used in conjunction with the earlier methods to further improve the efficiency of the network.

Cheng et al. (2017) have provided a summary of various forms of model compression strategies and have measured their efficacy in classification tasks. However, to the best of our knowledge, there has been no survey of compression techniques which evaluates their usefulness in communication systems. We submit this report, therefore, as a first step to the survey and evaluation of methods developed to compress deep learning models used in the field of communication.

REFERENCES

1. What i learned from competing against a convnet on imagenet. http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/. Accessed: 2019-03-30.

2. Bpsk bit error rate calculation using python. http://www.raymaps.com/index.php/bpsk-bit-error-rate-calculation-using-python/. Accessed: 2019-03-30.

3. Source code of all implementations. https://github.com/deddyjobson/DL-comm-sys. Accessed: 2019-04-10.

4. Neural network pruning pytorch implementation. https://github.com/wanglouis49/pytorch-weights_pruning/. Accessed: 2019-03-30.

5. Rf datasets for machine learning. https://www.deepsig.io/datasets. Accessed: 2019-04-21.

6. Fayçal Ait Aoudia and Jakob Hoydis. End-to-end learning of communications systems without a channel model. *CoRR*, abs/1804.02276, 2018. URL http://arxiv.org/abs/1804.02276.

7. Simone Bianco, Rémi Cadène, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. *CoRR*, abs/1810.00736, 2018. URL http://arxiv.org/abs/1810.00736.

8. C. BircanoÄ§lu and N. ArÄśca. A comparison of activation functions in artificial neural networks. In *2018 26th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4, May 2018. doi: 10.1109/SIU.2018.8404724.

9. Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. URL http://arxiv.org/abs/1604.07316.

10. Cristian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven, and Dimitrios Gunopulos, editors, *KDD*, pages 535–541. ACM, 2006. ISBN 1-59593-339-5. URL http://dblp.uni-trier.de/db/conf/kdd/kdd2006.html#BucilaCN06.

11. Aydin Buluç, Jeremy T. Fineman, Matteo Frigo, John R. Gilbert, and Charles E. Leiserson. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proceedings of the Twenty-first Annual Symposium on Parallelism in Algorithms and Architectures*, SPAA '09, pages 233–244, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-606-9. doi: 10.1145/1583991.1584053. URL http://doi.acm.org/10.1145/1583991.1584053.

12. Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *CoRR*, abs/1710.09282, 2017. URL http://arxiv.org/abs/1710.09282.

13. George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

14. Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

15. Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1135–1143. Curran Associates, Inc., 2015b. URL http://papers.nips.cc/paper/5784-learning-both-weights-and-connections-for-efficient-neural-netw pdf.

16. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

17. Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015. URL http://arxiv.org/abs/1503.02531.

18. A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544, 1952. doi: 10.1113/jphysiol.1952.sp004764. URL https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1952.sp004764.

19. Zehao Huang and Naiyan Wang. Like what you like: Knowledge distill via neuron selectivity transfer. *CoRR*, abs/1707.01219, 2017. URL http://arxiv.org/abs/1707.01219.

20. Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *CoRR*, abs/1609.07061, 2016. URL http://arxiv.org/abs/1609.07061.

21. Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

22. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL http://arxiv.org/abs/1412.6980. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

23. Nicholas D. Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. An early resource characterization of deep learning on wearables, smartphones

and internet-of-things devices. In *Proceedings of the 2015 International Workshop on Internet of Things Towards Applications*, IoT-App '15, pages 7–12, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3838-7. doi: 10.1145/2820975.2820980. URL http://doi.acm.org/10.1145/2820975.2820980.

24. C. Lee, H. B. Yilmaz, C. Chae, N. Farsad, and A. Goldsmith. Machine learning based channel modeling for molecular mimo communications. In *2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5, July 2017. doi: 10.1109/SPAWC.2017.8227765.

25. Darryl D. Lin, Sachin S. Talathi, and V. Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 2849–2858. JMLR.org, 2016. URL http://dl.acm.org/citation.cfm?id=3045390.3045690.

26. Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

27. Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. Deep learning for healthcare: review, opportunities and challenges. *Briefings in Bioinformatics*, 19(6):1236–1246, 05 2017. ISSN 1477-4054. doi: 10.1093/bib/bbx044. URL https://doi.org/10.1093/bib/bbx044.

28. Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant: Bridging the gap between student and teacher. *CoRR*, abs/1902.03393, 2019. URL http://arxiv.org/abs/1902.03393.

29. Asit K. Mishra and Debbie Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *CoRR*, abs/1711.05852, 2017. URL http://arxiv.org/abs/1711.05852.

30. Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *CoRR*, abs/1811.03378, 2018. URL http://arxiv.org/abs/1811.03378.

31. T. O'Shea and J. Hoydis. An introduction to deep learning for the physical layer. *IEEE Transactions on Cognitive Communications and Networking*, 3(4):563–575, Dec 2017. ISSN 2332-7731. doi: 10.1109/TCCN.2017.2758370.

32. Timothy J O'Shea and Nathan West. Radio machine learning dataset generation with gnu radio. *Proceedings of the 6th GNU Radio Conference*, 2016.

33. Timothy J O'Shea, Johnathan Corgan, and T. Charles Clancy. Convolutional radio modulation recognition networks. *arXiv preprint arXiv:1602.04105*, 2016.

34. E. L. Paula, M. Ladeira, R. N. Carvalho, and T. MarzagÃčo. Deep learning anomaly detection as support fraud investigation in brazilian exports and anti-money laundering. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 954–960, Dec 2016. doi: 10.1109/ICMLA.2016.0172.

35. Stanimir Sadinov, P Daneva, Panagiotis Kogias, J Kanev, and K Ovaliadis. Binary phase shift keying (bpsk) simulation using matlab. *ARPN Journal of Engineering and Applied Sciences*, 14:222–226, 01 2019.

36. Abigail See, Minh-Thang Luong, and Christopher D. Manning. Compression of neural machine translation models via pruning. *CoRR*, abs/1606.09274, 2016. URL http://arxiv.org/abs/1606.09274.

37. Osvaldo Simeone. A very brief introduction to machine learning with applications to communication systems. *CoRR*, abs/1808.02342, 2018. URL http://arxiv.org/abs/1808.02342.

38. Gürkan Solmaz, Rouhollah Rahmatizadeh, and Mohammad Ahmadian. A study on simd architecture.

39. Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL http://www.jmlr.org/papers/v9/vandermaaten08a.html.

40. Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, July 2005. ISSN 1549-6325. doi: 10.1145/1077464.1077466. URL http://doi.acm.org/10.1145/1077464.1077466.

41. Shuangfei Zhai, Keng-hao Chang, Ruofei Zhang, and Zhongfei Mark Zhang. Deepintent: Learning attentions for online advertising with recurrent neural networks. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1295–1304, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939759. URL http://doi.acm.org/10.1145/2939672.2939759.

42. Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Comput. Surv.*, 52(1):5:1–5:38, February 2019. ISSN 0360-0300. doi: 10.1145/3285029. URL http://doi.acm.org/10.1145/3285029.

43. Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 116–, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5. doi: 10.1145/1015330.1015332. URL http://doi.acm.org/10.1145/1015330.1015332.