Autoencoder Compression by Symmetry Aware Pruning and Quantization

Chinmay Talegaonkar
ECE
UCLA

Rajas Mhaskar
Computer Science
UCLA

Abstract

In this project we study compression approaches on autoencoders, focusing on pruning and quantization of weight matrices. We evaluated an approach based on the idea of using the network structure symmetry of autoencoders for low rank matrix factorization. We first evaluated DeepCompression (Han et al., 2015) on an autoencoder trained on MNIST dataset and observed a symmetry in the pruning thresholds of the encoder and decoder layers. We then analyzed the idea of using joint low rank approximation on concatenated encoder and decoder weight matrices to make use of this symmetry. Our analysis highlights various facets of the proposed idea, and also gives insights into hardware applicability of this approach. For quantization, we evaluate the post training quantization scheme (Krishnamoorthi, 2018) on the trained autoencoder model and observed the loss in accuracy for weight quantization to half precision FP16 and int8 data types. We then analyzed the impact of the size of representative data used for quantizing activations on the accuracy of the model. Lastly we looked at the ranges and distributions of the weight and bias matrices for all the layers of the autoencoder network and present some findings that can be used to improve the quantization schemes.

Keywords— AutoEncoders, Compression, Pruning, Quantization, Low Rank Approximation, Network Symmetry

1 Introduction

The past decade has seen an exponential rise in the use of deep learning systems in vision, speech and text applications (Liu et al., 2017). CNNs and LSTMs form the basis of most deep learning systems currently used in practise. However, recent emphasis on unsupervised learning and representation learning has to led to a surge in the use of autoencoders and its variants (Baldi, 2012).

Autoencoders consist of the same units as that of LSTMs or CNNs, i.e. fully connected layers or convolution layers. However, the architecture is designed primarily to extract compressed latent embeddings from input data in an unsupervised manner which can then be used for other downstream tasks. For example, high dimensional images/ textual inputs can be compressed into low dimensional vector embeddings. These embeddings characterize the input data efficiently and find applications in a variety of applications like Machine Translation, object detection, etc. Autoencoders and it's variants generally consist of an encoder and a decoder module. The encoder takes an input, and converts it to a latent low dimensional embedding. The decoder is then tasked with constructing the input again with this embedding as the input.

With the rapidly scaling model sizes, deploying neural networks on edge devices requires compressing the network, for it to be accommodated on chip memory. (Chin et al., 2018). Pruning and quantization approaches have been explored throughly for CNNs, LSTMs and fully connected networks. However, the recent focus on unsupervised learning in deep learning research warrants an analysis of prevailing network compression techniques. Pruning approaches can also result performance loss due to unstructured sparsity (Yu et al., 2017). However, autoencoders and other unsupervised approaches have an architectural symmetry. Thus, a compression approach that leverages this structural symmetry, while retaining the dense nature of weight matrices (or provides structured sparsity) quite valuable. Quantization of weights into 8 or 16 bits is essential for successful deployment on edge devices. While this results in better performance and energy efficiency on SIMD machines used on edge, they can have adverse impact on the accuracy of the network. Further recent advances in quantization have quantized weights to less than 8 bits. Most efforts in quantization have focused entirely on convolutional or recurrent networks. An analysis of quantization techniques on autoencoders thus becomes valuable in understanding their impact on model performance and size. In this regard we make the following contributions through this work.

1.1 Contributions:

First, we present an analysis of deep compression (Han et al., 2015) on autoencoders, and show the existence of symmetry in the pruning thresholds of the encoder and decoder layers. This observation is confirmed through an ablation study over different layer sizes. Second, we propose a joint low rank approximation strategy to compress the encoder and decoder layers. We present an evaluation of this strategy highlighting its potential strengths and weaknesses from a hardware perspective. We analyze the effect of this scheme on FLOPS and memory accesses for autoencoder layers.
For the weights quantization, we quantize the trained autoencoder model to 8 bit signed integers and 16 bit half precision floating point. We look at the degradation of the autoencoder performance post quantization. For quantizing the activations, we pass a representative MNIST data to the network to determine the range of the activations. We present an analysis of the performance loss versus the size of the representative data. Finally we analyze the distribution of weights across all the layers of the autoencoder to get an insight into the layer-wise behavior of the weights.

2 Related Work

Pruning and Quantization have been extensively studied recently to reduce the size of the networks for efficient inference on edge devices. Deep compression (Han et al., 2015) was an early effort in this direction that followed the three pronged strategy of pruning , weight sharing and huffman coding. It is worthwhile to mention here that the quantization strategy used in deep compression is unlike the typical quantization we know of today. Deep compression clusters weights based on their proximity, approximates the cluster by the centroids and uses a shared floating point weight map. For the weights of the network we only store the index bits for the shared map and these are referred to as the quantization bits. Unlike Deep Compression,quantization schemes today convert the floating point weights into either 8 or 16 bit integer or half precision floating point format. (Krishnamoorthi, 2018) details the post training quantization support in TensorFlow and describes the different types of quantizers used today. It also presents details about the post training quantization of weights and activations for a fully quantized 8 bit inference on fixed point hardware.
(Patrick H. Chen, June 2018) proposed a low rank matrix approximation technique for compressing neural language models. Their method relied on using an SVD based low rank approximation for the word embeddings matrix. They propose using a low rank approximation of a weighted embedding matrix which takes into account the frequency of occurence of words. The paper uses a low rank approximation to compress RNN/LSTM based neural networks. Similarly, (Chong Li, 2018) introduces COBLA (Constrained Optimization Based Lowrank Approximation) for finding an optimal low rank approximation subject to constraints in the number of multiply and accumulate (MAC) operations and memory footprint. They propose to add these hardware bounds as constraints to their solvers to produce an approximation that results in significant reductions in computation and memory footprint. This method focusses mostly on convolution neural networks such as VGG-16 used in vision applications.
Quantization is a topic of active research in the deep learning community. (Guo, 2018) gives a brief survey on the existing methods for quantization and discusses the current challenges and trends in quantization schemes. There is also a brief discussion on binarized neural networks that could be a game changer for efficient implementation on parallel hardware. Similarly (Markus Nagel, 2019) introduces an approach that relies on equalizing weight ranges across consecutive layers while simultaneously correcting for any bias that may arise due to quantization.

3 Method

3.1 Symmetrical Pruning Thresholds in Autoencoders

The autoencoder is generally implemented as a symmetric architecture in practice. We consider a symmetric autoencoder (see figure 1 for the layout of the network) for this project, trained on re-

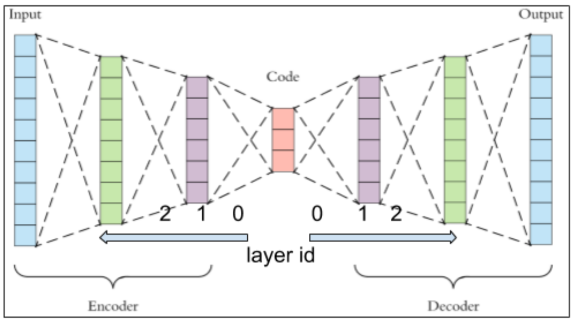


Figure 1: **Autoencoder layout and layer indexing scheme for pruning experiments.** The layer indexing starts from the centre and increases outwards for both encoder and decoder. An autoencoder made of fully connected layers was used for pruning experiments.

constructing MNIST digits. On pruning the autoencoder layers using the pruning strategy used in (Han et al., 2015), we observe a symmetry in the pruning thresholds of the encoder and decoder layers. We confirmed this hypothesis by evaluating it over different layer sizes. Formalizing this notion of symmetry, the layers are indexed with the center of the architecture as origin (for both encoder and decoder), since the autoencoder has a symmetric structure about the center. Let the pruning threshold for the i^{th} encoder matrix be P_i^e, and the i^{th} decoder matrix be P_i^d. We observe that $\forall i\ P_i^e \approx P_i^d$. Section 4 shows numerical results confirming this observation.

An intuitive explanation to this observation arises from the functioning of the autoencoder. An ideal autoencoder would have the same input and output, since it would be capable of perfectly reconstructing the input. Coupled with the symmetric nature of the autoencoders, this served as the key motivation to evaluate the idea of using a symmetry based compression strategy.

3.2 Joint Low Rank Approximation of Encoder-Decoder Weights

One of the ways to compress Neural Networks is to use low rank approximation on weight matrices. For convolutional layers/filters, this translates to using separable filters. For an autoencoder, we denote a weight matrix between layer m and n as W_{mn}. Using Singular value decomposition on W_{mn}, we get $W_{mn} = USV^T$, where $U \in \mathcal{R}^{m\times m}, V \in \mathcal{R}^{n\times n}$ and S is a diagonal matrix $\in \mathcal{R}^{m\times n}$. Assuming m is less than n, rank of W_{mn} can at most be m. A rank r approximation of W_{mn} can be taken as follows - $W_{mn}^r = U_r S_r V_r^T$ where U_r denotes a submatrix of U containing first r columns of U, S_r is an $r \times r$ matrix with first r rows and columns of S (since S is a diagonal matrix. V_r denotes a sub-matrix of V containing the first r columns of V. For low rank matrices, r is much smaller than m, hence the total number of parameters required to store W_{mn}^r are $(m+n+1)r$, while number of parameters in the original weight matrix are mn. One can observer that when $r < \frac{mn}{m+n+1}$, the low rank approximation would result in a compression. Thus, low rank approximation can be applied to every weight matrix of the network to obtain compressed weight matrices. Any vector multiplication to W_{mn}^r can be expressed using U_r, S_r, V_r. Thus, one needs to store these matrices instead of the entire W_{mn}, which enables compressing the matrix representation

Using the structural symmetry of autoencoders, and the observed symmetry in pruning thresholds, we propose an approach that leverages this symmetry to make the low rank approximation more efficient for autoencoders. Let W_{mn} be a weight matrix for the encoder matrix. Let the corresponding decoder weight matrix be W_{nm}. Instead of applying low rank approximation independently on these 2 matrices, we concatenate W_{mn} and W_{nm}^T, and apply the low rank matrix approximation on this concatenated matrix. The idea of concatenating the matrix comes from the observation that the pruning thresholds are symmetric for corresponding encoder and decoder layers. Further, we plot the elements of the S matrices of the SVD of W_{mn}, W_{nm} and the concatenated matrix (W_{cat}). The plots show that concatenating the matrix results in a sharper falling curve for the singular values, thus indicating higher compressibility. To improve the applicability of this approach, the matrices W need to be low rank. To enforce this, we use L2 regularization on the weights of the autoencoder while training it. The loss function used is

$$\mathcal{L}(x,\hat{x}) = ||x-\hat{x}||_2 + \lambda \sum_i ||W_i||_F$$

Here x is the true input, $\hat{x}$ is the output reconstructed by the autoencoder, λ is the regularization parameter for L2 regularization (weight decay). W_i is the weight matrix of the i^{th} layer and F denotes the Frobenius norm of the matrix. Increasing λ enforces smoothness in weights, thus promoting the likelihood of W_i to be low rank. Section 4 shows an ablation study that shows the effect of λ on the singular value decay rate of the matrices. The rank r for each of the concatenated weight matrices is selected using an error threshold ϵ.

$$r = \text{argmin}_r \frac{||W_i^r - W_i||}{||W_i||} \leq \epsilon_i$$

Here ϵ_i denotes the value of ϵ for the i^{th} encoder-decoder concatenation. The norm used here is the Frobenius norm. Using a higher ϵ results in a lower estimate of r. We compare the compression capability of this approach with individually compressing the encoder and decoder matrices, and present hardware insights gained from this analysis in section 4.

3.3 Post Training Quantization in Autoencoders

In most cases it is desirable to quantize the weights and activations of the network for faster inference without having to retrain the entire model. An advantage of these techniques is that they are simple to use and allow quantization with limited data. A typical uniform quantizer to a n-bit fixed point integer looks at the range of the floating point inputs (x_{min},x_{max}) and transforms an input x as follows -
x_{int} = round(x/ Δ) + z
x_Q = clamp(0,$N_{levels} - 1$,x_{int})
$x_{de-quant}$ = (x_Q - z) Δ

Here N_{levels} = 2^n is the number of discrete levels that we can represent in n bits, x_Q is the quantized output of the quantizer. Δ is the scale factor while z is the zero point. The zero point is an integer that has a special significance because it ensures that zero is quantized with no error. This ensures that routine operations such as zero padding does not result in quantization error. It is easy to see that quantization is bound to create some losses at the network output due to the precision loss while converting from single precison floating point to lower bit precision. While this has been studied extensively for convolutional neural networks, we experiment with a trained autoencoder network and look at the output loss for quantization to half precision floating point (FP16) and 8 bit integer.

There are essentially two ways we can do the post training qunatization - Quantizing only the weights or quantizing both the activation and weights. Quantizing the activation adds the overhead of calibrating the activations for the given input data. This typically involves passing some sample input data thorugh the network and storing the activation ranges at each layer. After a sufficiently large number of samples are run on the network, the range of the activations stabilize and we can use this range to quantize the activation for the test data. The hope is that the sample data passed is sufficiently large to calibrate the test data. In our work we try to quantify the size of the representative data needed to achieve a stable activation range. We vary the size of the representative data and observe the output loss. The results for both these experiments are shown in section 4.

3.4 Distribution of Weights and Bias Across Layers

The autoencoder has a symmetric structure consisting of an encoder and decoder. Further each layer has a weight matrix for multiplying with the input neurons and a bias vector to add to the output. During training both these matrices are updated. In our first experiment in this category we took at a trained network and plotted a histogram of the weight matrix and the bias vector for all the layers. The goal of this exercise was to look at the range of the weights and bias vector across all layers to look for a pattern that can aid quantization.

Secondly note that in the quantization method discussed in the last subsection, zero has a special signifcance. All quantizers today ensure that there is no quantization loss for zeros so that operations such as padding can be done safely. If the inputs to the quantizer are all positive or all negative then we need to extend the range to include zero in the input quantization range. This usually results in higher precision losses in the actual inputs as they are represented by fewer number in the integer range. Further if the quantized inputs are distribute symmetrically around 0 (there are almost same positive and negative numbers) then we can use a signed symmetric quantizer that quantizes to (-N_{level} /2, N_{level} /2 -1). A symmetric quantizer restricts the zero point to 0. The quantization equations then reduce to -
x_{int} = round(x/ Δ)
x_Q = clamp(-$N_{levels}/2$, $N_{levels}/2$ -1,x_{int})
$x_{de-quant}$ = x_Q * Δ

The histogram plots reveal interesting results on suitability of autoencoders for such quantizers.

4 Evaluation and Results: Pruning

We present here our findings and ablation studies performed for pruning of AutoEncoders. We use a layered autoencoder architecture with ReLU as the activation function. Some of the ablation studies are performed on different layer sizes, but the standard autoencoder architecture adopted for presenting results is as follows-
encoder_l_0 Linear(in=784, out=256)
encoder_l_1 Linear(in=256, out=128)
encoder_l_2 Linear(in=128, out=64)
decoder_l_2 Linear(in=64, out=128)
decoder_l_1 Linear(in=128, out=256)
decoder_l_0 Linear(in=256, out=784)
Here Linear represents linear layers with bias enabled. The training set has 60000 samples and the test set has 10000 samples. A batch size of 1000 was used.

4.1 Deep Compression on Autoencoders

We performed 2 sets of experiments for pruning. First we evaluate deep compression (Han et al., 2015) on autoencoders. The next set of experiments evaluates our proposed approach of jointly approximating encoder and decoder matrices and presents analysis on the same.

1. *Deep compression Evaluation on Autoencoders.* For the pruning experiments, we consider an autoencoder with FC layers. The input image is vectorized and fed as an input to the autoencoder. The autoencoder was trained for 100 epochs with a learning rate of 10^{-5}. Deep compression consists of 3 stages, a) weight pruning, b) weight quantization c) Huffman encoding. Using deep compression on 3 stage autoencoder with the standard configuration, we observed the following compression rates for each of the stages -:
   - Weight Pruning - 17.93x compression
   - Weight Sharing - 9x compression
   - Huffman Coding - 4.11x compression
2. *Symmetry in Pruning Thresholds* We performed an ablation over different layer configurations, and observed that the pruning thresholds for corresponding encoder and decoder layers are very close. Table 1 elicits this observation for one of the configurations.

Layer id	Encoder Thresholds	Decoder Threshold
0	0.036	0.042
1	0.089	0.092
2	0.1509	0.15507

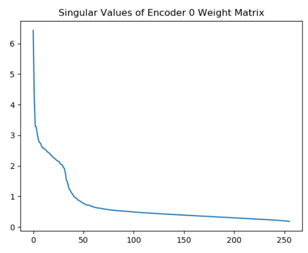
Table 1: **Symmetry in Pruning Thresholds for corresponding encoder and decoder Layers.** The layer id corresponds to the layer number measured from the centre of the autoencoder for both encoder and decoder as shown in figure 1

3. *Minor performance degradation due to deep compression* The test mean squared loss (MSE) per sample for the uncompressed autoencoder was 2.4×10^{-4}, and the test MSE per sample for the compressed autoencoder was 3.5×10^{-4}. For classification tasks using CNNs, network compression won't affect classification accuracy a lot, since the classification uses a softmax operation to select the final label, however for regression tasks, like image reconstruction for which the autoencoder is trained, the loss difference is visible.
4. *Inferring sparsity from Huffman encoding results* Furthermore, we observe that compression using Huffman encoding for Autoencoders is much lesser than that for Lenet shown in (Han et al., 2015). We suspect the reason being that Deep compression creates sparser weights for LeNet, which enables a higher compression using Huffman coding. This suggests that using sparsity for compression might not be the best solution for autoencoders, but this claim warrants further analysis.

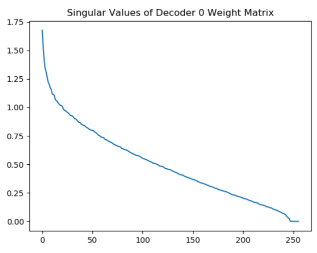
The key takeaway from this set of experiments is that the symmetrical structure of the autoencoders, which is also reflected surprisingly in the pruning thresholds can be used to devise a compression scheme that leverages this symmetry. We used this insight to devise the joint low rank approximation of encoder and decoder matrices as described in section 3.

4.2 Joint encoder-decoder Compression

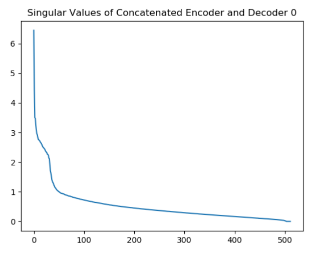
We perform an ablation study to evaluate the proposed joint compression idea. We first show that



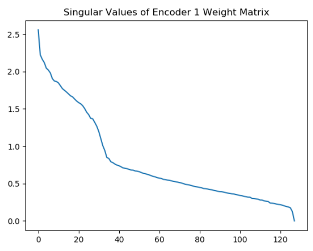
(a) Enc 0



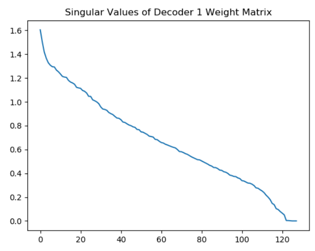
(b) Dec 0



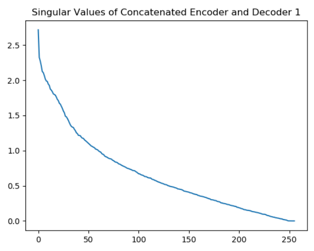
(c) Conc 0



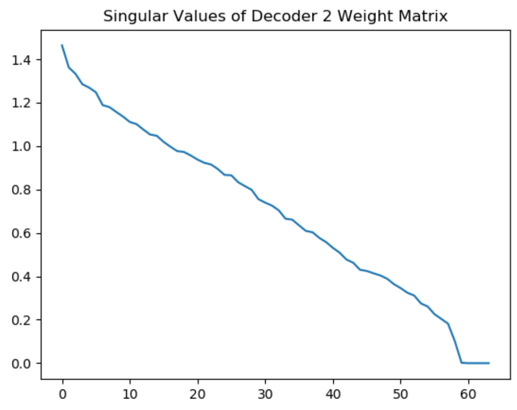
(d) Enc 1



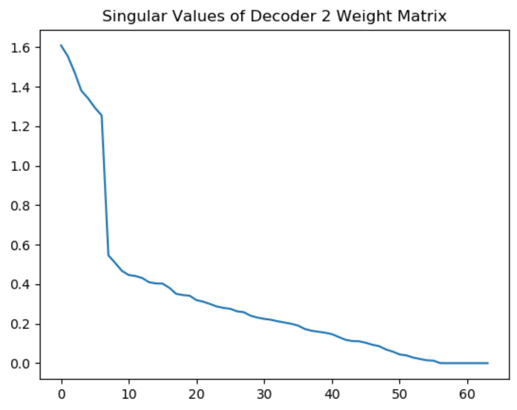
(e) Dec 1



(f) Conc 1



(g) Singular Value Fall off for decoder 2 with $\lambda = 10^{-5}$



(h) Singular Value Fall off for decoder 2 with $\lambda = 10^{-3}$

Figure 2: (a),(d) - singular value curve for encoder matrices. (b),(e) - singular value curves for decoder matrices. (c),(f) - singular value curves for concatenated encoder and decoder matrices. (g),(h) - effect of weight decay. Higher weight decay leads to sharper fall off in decoder matrix

joint low rank approximation of the encoder and decoder matrices results in more efficient compression than individually compressing them. The ϵ thresholds for individual compression are taken to be half of that for the joint compression case, since the concatenated matrix has twice the number of elements than the encoder or decoder matrix considered individually. (We also tried dividing ϵ by $\sqrt{2}$ for a conservative estimate and get results of similar nature) To reason out why joint approximation works better, we plot the singular values of the Encoder, Decoder and the concatenated encoder and decoder in figure 2 (figures 2a - 2f) for 2 of the layers. Table 2 shows the compression rates using the 2 strategies and the corresponding model accuracies for different ϵ values. The figure shows that encoder 1 has a sharp fall off curve for singular values as compared to the decoder, thus making the encoder more compressible. However, when the encoder and decoder matrices are concatenated we observe that the the concatenated matrix has a sharp fall off curve, thus supporting the idea that compressing a concatenated matrix is more favorable to compressing the encoder and decoder separately. To evaluate our hypothesis of the role of weight decay (section 3) we plot the fall off curve for for the third decoder layer for architectures trained with 2 different weight decay coefficients in Figures 2g and 2h to show that for a higher λ the fall off curve becomes steeper, thus making the matrix more compressible, with not much loss in accuracy (table 2). We observe that the joint approach gives better compression rates than individually compressing the layers, but with reduction in ϵ, the relative advantage of the joint approach also reduces. Also, the model accuracy for the individual compression seems to be better than joint compression. However, at $\epsilon = 0.2$, we observe that the accuracies for both joint and individual compression are reasonably close to the true accuracy, but the joint compression approach is 26% better than individual compression. What works to our advantage here is the fact that slight model inaccuracy of the autoencoder is still acceptable for generating suitable representations for downstream classification tasks, while providing a massive savings of 26%. We discuss this approach from a hardware perspective in section 6.

5 Evaluation and Results: Quantization

For the quantization experiments we have a simple 6 layer network as follows -
Encoder L1 - FC layer 128 units, i/p-784, o/p-128
Encoder L2 - FC layer 64 units, i/p-128, o/p-64
Encoder L3 - FC layer 32 units, i/p-64, o/p-32
Decoder L1 - FC layer 64 units, i/p-32, o/p-64
Encoder L2 - FC layer 128 units, i/p-64, o/p-128
Encoder L3 - FC layer 784 units, i/p-128, o/p-784

The autoencoder simply reconstructs the original image back from the test image given as input. The loss function takes the sum of squared differences between the input image and the reconstructed image for all 784 pixels across all test images. The reported loss figures are average loss for a test image obtained by dividing the total loss across all the 10,000 test images of the MNIST data set by 10,000, number of test images.

5.1 Post Training Quantization

Loss in FP32	Loss in FP16	Loss in INT8
13.178498	13.178310	13.74225

The figure indicates the value of the loss function as mentioned above for three models - the original tensorflow model with FP32 weights, quantized model with Fp16 weights and quantized model with INT8 weights. The model sizes reduced by a factor of 2 for FP16 and 4 for INT8 as expected. We observe that the model performance surprisingly improved in FP16 by a marginal amount but that can be safely ignored as noise. Further note that when we quantize the model to int8 the loss increases by around 4.2% as expected. We note however that while the quality of the reconstructed image decreases, it is still within acceptable limit and we cannot perceive any change in the reconstructed image before and after quantization.

Next we vary the size of the representative data set for quantizing the activations and observe the loss function.

D=500	D=1000	D=2000	D=5000
13.74285	13.74225	13.74407	13.74591

The test data has 10,000 images and we see that choosing a representative data that is 5-10% the size of the test data produces results within acceptable range. It is interesting to observe that varying the representative data size has little impact on the loss function. This indicates that activations for the autoencoder network are in general well behaved and a small sample of data gives a reasonable estimate of the activation range. It is does futile to have large representative data sets for the autoencoder network.

5.2 Distribution of Weights and Biases across Different layers

In this experiment we look at the range of weights and biases in each of the 6 layers of the network as shown in table 3.
One key observation from the table is that both the weights and sparsity across all layers have both positive and negative values. As a consequence 0 is in the range for every single layer and hence need not be handled separately. Thus an autoencoder seems to be an ideal candidate for using a symmetric quantizer.
Another interesting thing to note is that the range of the bias vectors are considerably lower than the range of the weight vectors. This behavior holds true in all the layers of the autoencoder. This presents a possibility of having a different quantization scheme with fewer bits for the bias term. Also note that there are in general fewer bias terms compared to the multiplication weights. Hence the fewer bias terms with a small range can be ideal candidates for quantizing in fewer bits compared to the weight matrix.

$\epsilon_1, \epsilon_2, \epsilon_3$	Joint Compression	Individual Compression	Loss after Individual LR	Loss after Joint LR	True Loss
0.8,0.8,0.8	38.14x	4.420x	5.3e-4	9.8e-4	4.21e-4
0.5,0.5,0.5	8.602x	2.86x	4.58e-4	7.049e-4	4.21e-4
0.2,0.2,0.2	2.308x	1.83x	4.323e-4	4.520e-4	4.21e-4
0.1,0.1,0.1	1.617x	1.511x	4.23e-4	4.344e-4	4.21e-4
0.1,0.05,0.1	1.571x	1.448x	4.229e-4	4.334e-4	4.21e-4
0.05,0.05,0.05	1.323x	1.292x	4.22e-4	4.267e-4	4.21e-4

Table 2: **Comparing individual and joint low rank compression.** ϵ_i denotes the Frobenius norm threshold for layer i for the concatenated matrix. For individual compression, the threshold is halved since the number of elements in the encoder/decoder is half that of the concatenated matrix.

Layer	1	2	3	4	5	6
Weight min	-0.3328	-0.4246	-0.4371	-0.4519	-0.3311	-0.6357
Weight max	0.2944	0.5988	0.6968	0.536	0.415	0.291
Bias Min	-0.227	-0.219	-0.176	-0.221	-0.159	-0.166
Bias Max	0.327	0.2848	0.2603	0.267	0.3191	0.0680

Table 3: **Range of the weights and biases across all layers**

6 Hardware Insights

6.1 Joint encoder-decoder pruning

We highlight here the impact of our devised jointly pruning strategy on hardware. The pruning experiments show the potential of coming up with more efficient compression schemes. Our compression scheme is suitable for GPUs (or massively parallel hardware) since we don't encounter the overheads due to unstructured sparsity that is generally seen in pruning approaches commonly. Our experiments show that jointly approximating the encoder and decoder layers is superior to approximating them individually. A caveat to this approach is the resulting increase in compute operations. However, for GPUs, most applications are memory bound hence this extra computation won't be a bottle-neck for most cases. On using a low rank approximation, the number of memory access requests to GPU are reduced (as $r < \min(m, n)$). S_r is a sparse matrix, but it is a structurally sparse (diagonal) matrix, hence can be dealt with efficiently by expanding the ISA to account for accessing a diagonal matrix. A deeper analysis of this aspect is shown below

6.2 Effect on Joint Compression on Compute

Here we formally analyse the effect of the increased compute operations resulting from the low rank compression scheme (joint case). Assuming the input has a batch size B. For a given layer id i - let the encoder and decoder matrices be of size $W_i^e = m \times n$ and $W_i^d = n \times m$. Hence the input matrix to the encoder is of the size $n \times B$.

- If the weight matrix is stored without any compression, then for the encoder, number of computations are $(2n-1)mB$ and the number of memory accesses are mn + nB + mB. (Assuming addition and multiplication to be separate)
- In case of joint compression (assuming compression till rank r), encoder computations are $m(2r-1) + mn(2r-1) + (2n-1)mB$ where $(2r-1)(m+mn)$ computations are used to reconstruct weight matrix from compressed representations. Memory operations are $mr + r + nr + nB + mB$.
- Since the encoder and decoder are transpose of each other, similar analysis holds for decoder, by replacing m with n in all expressions.

Consider the standard autoencoder configuration defined above, and the case in table 2 where $\epsilon = 0.2$, for layer id 0. $B = 1000$ for our experiments. Table 4 below shows the number of compute operations for each of the encoder layers with and without joint compression. From table 4 we can see that after joint compression, the decrease in memory operations is greater than the increase in compute operations. Although we could not execute the matrix multiplication kernels on the GPU for roofline analysis due to lack of time, based on the data for the kernels in miniproject 1, we stipulate that even after joint compression the workload is still memory bound, hence the joint low rank compression scheme could potentially be beneficial for GPU performance.

Layer id	Rank (r)	Mem access decrease	FLOPS increase
0	129	33.8 %	12.87 %
1	65	23.63 %	6.46 %
2	34	19.89 %	3.38 %

Table 4: **Change in memory access and compute operations after Joint Compression** Column 3 and 4 show the percentage decrease in memory accesses and increase in FLOPS due to joint compression respectively. This relative change in percentage is calculated w.r.t original FLOPS and memory accesses required to load the weight matrix

6.3 Quantization

Quantizing a network to 8 bits or 16 bits without loss of network accuracy can lead to drastic improvements in performance and energy efficieny of networks. Due to the lower memory footprint of the quantized network, the memory bandwidth requirements of the systems reduces. Further fixed point hardware is more energy efficient than floating point hardware and hence quantizing improves the energy performance as well. In this work we demonstrate that quantizing an autoencoder has negligible effect on its performance. We obtained upto 4X reduction in model size by quantizing to 8 bits with small loss in accuracy. Further due to the well behaved nature of the activations, we need a relatively small representative data to quantize the activations of an autoencoder. As a smaller representative data set reduces the quantization time significantly, we can quantize an autoencoder in a short period of time. Finally the study of the weights distrubution indicated that autoencoders are ideal candidates for symmetric quantization. Processors with support for signed 8 bit operations would thus be great for running autoencoder networks. The study also indicated a possibility of quantizing the bias vectors in fewer bits for further reduction in model size.

7 Conclusions

We evaluated pruning and quantization methods on autoencoders and demonstrated that some of the prevalent methods such as deep compression, low rank factorization and post training quantization can be applied to autoencoders. Further our analysis gave us interesting insights for tailoring these techniques for the autoencoder networks. Our experiments indicated that the encoder and decoder layers have similar pruning thresholds. Leveraging this observed symmetry, we proposed and evaluated a low rank approximation based approach to compress the encoder and decoder jointly. Our experiments show that for acceptable error thresholds, joint compression can be more effective than individual compression, despite the relative increase in compute operations. Our calculations show that the relative increase in compute operations is less than the memory savings resulting from joint compression, thus motivating the effectiveness of this approach. For quantization, we evaluated the post training quantization strategy on autoencoders and showed that we can successully quantize to int8 and fp16 with little loss in performance. A key insight from this exercise was that the activations in autoencoders are well behaved and hence a small representative data is sufficient for quantizing activations. Quantization experiments reveal that the bias vectors have a significantly lower range and hence can possibly be quantized in fewer bits. Lastly the symmetric nature of weights in all the layers hinted towards a possible use of a symmetric quantizer. This can lead to lower precision loss due to quantization as well as faster inference on SIMD machines suporting signed 8 bit operations.

Statement of Contribution *Chinmay:* Pruning experiments and ideas, Report writing. *Rajas:* Quantization experiments and ideas, Report writing We worked together as a team on this project and discussed ideas and experiments with each other.

References

Pierre Baldi. 2012. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49.

Ting-Wu Chin, Cha Zhang, and Diana Marculescu. 2018. Layer-compensated pruning for resource-constrained convolutional neural networks. *arXiv preprint arXiv:1810.00518.*

C.J. Richard Shi Chong Li. 2018. Constrained optimization based low-rank approximation of deep neural networks. *The European Conference on Computer Vision (ECCV).*

Yunhui Guo. 2018. A survey on methods and theories of quantized neural networks. *arXiv:1808.04752*.

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

Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv:1806.08342*.

Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E Alsaadi. 2017. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26.

Tijmen Blankevoort Max Welling Markus Nagel, Mart van Baalen. 2019. Data-free quantization through weight equalization and bias correction. *The IEEE International Conference on Computer Vision (ICCV), 2019*.

Yang Li Ciprian Chelba Cho-jui Hsieh Patrick H. Chen, Si Si. June 2018. Groupreduce: Block-wise low-rank approximation for neural language model shrinking. *arXiv:1806.06950*.

Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. 2017. Scalpel: Customizing dnn pruning to the underlying hardware parallelism. *ACM SIGARCH Computer Architecture News*, 45(2):548–560.