Lossy and Lossless compression for matrices presented at Neural Networks

Pesaresi Seminar

Gabriel Carmona Tabja

gabriel.carmona@phd.unipi.it
Università di Pisa

February 16, 2024

Part I: Motivation

Part II: Lossy Compression

Part III: Lossless Compression

Part I

MOTIVATION

ResNet-50

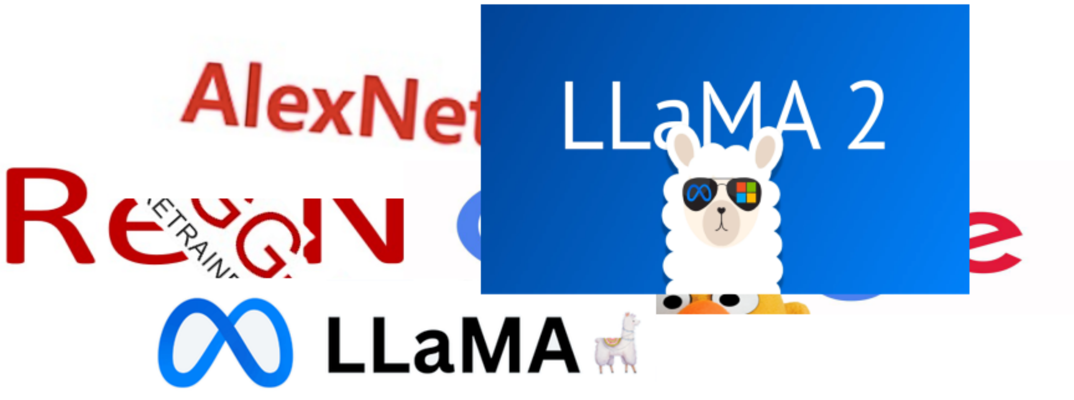
PRETRAINED MODEL

NEURAL NETWORKS TODAY

Neural Networks Today

Neural Networks Today

NEURAL NETWORKS TODAY


AlexNet
LLaMA 2
LLaMA


AlexNet
LLaMA 2
Stanford
Alpaca

Neural Networks Today

They have similar problems:

- Over-Paramtrized
- High Complexity in operations
- HUGE use of space
- HUGE consume of energy
- Difficult to use locally

NEURAL NETWORKS TODAY

- Some amount of parameters
  - Llama 2: three versions with 7, 13 and 75 billion of parameters
  - Alpaca: 7 billion of parameters
  - BERT 109 million of parameters
- Some details in Llama 2 7B:
  - 32 layers
  - Each layer has 7 matrices
    - 4 are size 4096×4096
    - 3 are size 11008×4096

Why is interesting to do matrix compression?

The models are a combination of layers, where each layer has a Matrix/Tensor.
The Matrices/Tensors are a great porcentage of the model.
Then we will be very interesting on compressing those matrices, with these two objective:

- Reduce the usage of space
- Maintain the performance of the model

Categories of Compression Techniques

The techniques to compress matrices can be divided in two:

- Lossy Compression
- Lossless Compression

Now lets study about them!

Part II

Lossy Compression

Definition

Definition 1.1

Lossy Compression or irreversible compression is the class of data compression methods that uses inexact approximations and partial data discarding to represent the content.

Some techniques known to compress Matrices or Models are:

- Pruning
- Quantization
- Knowledge Distillation
- Low-Rank Factorization

Pruning

Pruning can be found in two ways:

- Structured Pruning
- Unstructured Pruning

PRUNING

STRUCTURED PRUNING (ANWAR ET AL., 2017)

Elimination of entire structural components: neurons, channels, or layers.

- Objective: target a set of weights at once.
- Pros:
  1. Reduce model complexity.
  2. Reduce memory usage.

  Maintaining overall structures.
- Cons: leaves redundancies!!

Pruning

Unstructured Pruning (Zhang et al., 2018)

Elimination of connections considered irrelevant for the overall network behavior.

- Simple pruning:
  - Let α be a constant and $W \in \mathbb{R}^{n \times m}$ a matrix.
    1. If $w_{ij} \leq \alpha$, then $w_{ij} = 0$.
    2. Otherwise, $w_{ij} = w_{ij}$.
  - α can be layer-specific or set globally.
- More Complex Pruning:
  - Use of regularization terms (L_1 or L_2).
  - Using optimization strategies.

Pros: It is simple and generates a sparse matrix (something that we will like later).
Cons: Ignores model structure!!

Quantization

Quantization Definition

Quantization's objective is to fix the amount of bits that you can use to represent the weights.
The most simple quantization is to reduce the amount of bits in the numeric representations.

- 64 bits using double precision floating point
- 32 bits using single precision floating point
- 16 bits using half precision floating point
- 16 bits using integer
- $8, 4, 2$ bits using even shorter integers
- 1 bit!

Quantization's objective is to fix the amount of bits that you can use to represent the weights.
The most simple quantization is to reduce the amount of bits in the numeric representations.

- 64 bits using double precision floating point
- 32 bits using single precision floating point
- 16 bits using half precision floating point
- 16 bits using integer
- $8, 4, 2$ bits using even shorter integers
- 1 bit!

WARNING! Clearly reducing the representation this harshly can produce severe decay in performance.

Quantization

Quantization - Via Share Weights (SW)

Quantization via Share Weights has three phases:

1. Partition the weights into k categories and transform all into a unique representative value c_i, for the i-th category.
2. Cumulative retraining of the weights.
3. Storage of the shared weights.

In this part of the presentation, we will focus on the first point!

Quantization

Scalar Quantization via Clustering-based SW (Gong et al., 2014)

- Given a matrix $W \in R^{n \times m}$, can flatten in the vector $w \in R^{1 \times nm}$.
- Apply k-means over w to divide the weights in k clusters, obtaining $cinR^{1 \times k}$ (cluster centers).
- Now, $W_{ij} = z$, where $zin[1, k]$ and c_z is the representative value for w_{ij}.
- With this, we can encode each centers in $\log_2(k)$ bits and save the vector c!
- Another version is called: Entroy Constrained Scalar Quantization
  - Same idea, but minimizing the distortion while you don't exceed a threshold based on the entropy

Quantization

Product Quantization via Clustering-based SW (Gong et al., 2014)

- Divide the matrix $W \in \mathbb{R}^{n \times m}$ into s groups:

$$W = [W^1, W^2, \ldots, W^s],$$

where $W^i \in \mathbb{R}^{n \times (m/s)}$.

- We applied k-means on each submatrix W^i, obtaining $c^i \in \mathbb{R}^{k \times (m/s)}$, where c^i_j is the representative vector for the j-th row in the submatrix i.

With this, we can encode each vector c^i_j using $\log_2(k)$ bits and store each vector.

QUANTIZATION

QUANTIZATION VIA UNIFORM SW (CHOI ET AL., 2020)

Given the matrix $W \in \mathbb{R}^{n \times m}$:

- Select representative weights uniformly in the weight domain.
- Transform weight w_{ij} to:

$$w'_{ij} = \delta \cdot \text{round}\left(\frac{w_{ij} + d}{\delta}\right) - d$$

  where $\delta > 0$ is the interval size and $d \in \left[-\frac{\delta}{2}, \frac{\delta}{2}\right]$.

- Given the matrix $W \in \mathbb{R}^{n \times m}$, we get:
  - $w_{\text{min}} = \min W$
  - $w_{\text{max}} = \max W$
- Thanks to this, we get the following:
  - $P(w = w_{\text{min}}) = \frac{w_{\text{max}} - w}{w_{\text{max}} - w_{\text{min}}}$
  - $P(w = w_{\text{max}}) = \frac{w - w_{\text{min}}}{w_{\text{max}} - w_{\text{min}}}$
  - $E(w \| W = w') = w'$
- Because of pseudorandomly, the quantized matrix is highly compressible!
- This case was $k = 2$, but $k > 2$!

Knowledge Distillation (KD) (Ba and Caruana, 2014)

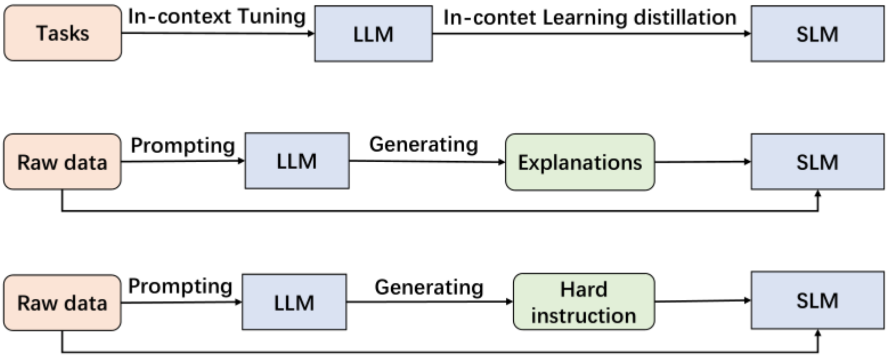
Definition

- The learning of a thinner model (**student**) is guide by a larger model (**teacher**)
- The output of the teacher act as a soft targets for the training process
- Objective: exploit the logits of the outputs of the teacher to distill the information to the student
- The student is trained minimizing the cross entropy between the logits of the teacher and student
- There are two types of KD: White-Box and Black-Box

Knowledge Distillation (KD) (Ba and Caruana, 2014)

White-Box KD and Black-Box KD

- White-Box KD:
  - Student has access to the predictions AND parameters of the teacher.
  - Benefits: deeper understanding of teachers structures and representations.
- Black-Box KD:
  - Student only has access to the predictions of the teacher.
  - Emergent Abilities of this type:
    - In-Context Learning
    - Chain-of-Thought
    - Instruction Following



Low-Rank Factorization

- Given a matrix $W \in \mathbb{R}^{n \times m}$ of full rank r, it can be decomposed as $W \approx AH$, where $A \in \mathbb{R}^{n \times r}$ and $H \in \mathbb{R}^{r \times m}$.
- Other approaches:
  - SVD

Some previous results

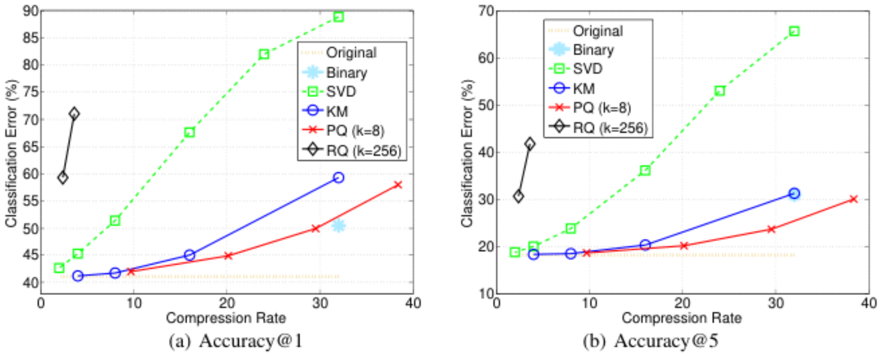


Figure. Comparison of different compression methods on ILSVRC dataset.[1]

[1] plots taken from this paper: Gong et al., 2014

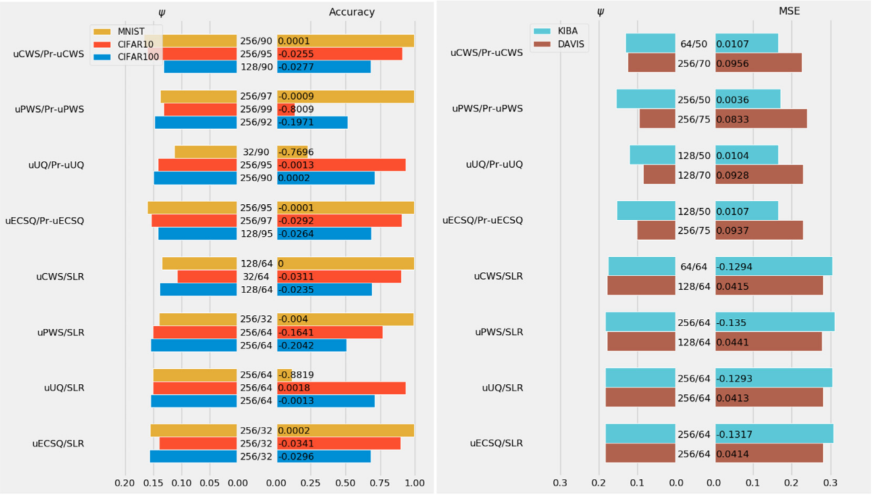


Figure. Best performance when quantizing convolutional layers and applying SLR or pruning followed by quantization to FC layers of VGG19 (a) and DeepDTA (b)[2]

[2] plots taken from this paper: Marinó et al., 2023

Part III

Lossless Compression

Definition

- Lossless compression or reversible compression is a class of data compression that allows the original data to be perfectly reconstructed from the compressed data with no loss of information.
- But also there is a characteristic that is important to maintain...

  Apply operations DIRECTLY in the COMPRESSED information!
- In this context, matrix/vector multiplication!

COMPRESSED SPARSE COLUMN (SAAD, 2003)

$$w = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 3 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 \end{pmatrix}$$

$$nz = [1, 1, 1, 3, 1, 5, 5]$$

$$ri = [0, 2, 1, 2, 0, 2, 4]$$

$$cb = [2, 2, 1, 0, 2]$$

COMPRESSED SPARSE COLUMN (SAAD, 2003)

$$w = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 3 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 \end{pmatrix}$$

$$nz = [1, 1, 1, 3, 1, 5, 5]$$

$$ri = [0, 2, 1, 2, 0, 2, 4]$$

$$cb = [2, 2, 1, 0, 2]$$

We define:

- $w \in \mathbb{R}^{n \times m}$
- $s \in [0, 1]$: ratio on non-zero elements
- b: is the amount of bits to encode the elements in nz

Space in bits: $snm(b + \log n) + m \log n$

Huffman Address Map Compression (Marinó et al., 2023)

But first!

Huffman Address Map Compression (Marinó et al., 2023)

But first!

Definition 3.1 (Shanon's Entropy)

Given a set of symbols $Z = \{z_1, \ldots, z_n\}$ *and the probability distrbution* Pr*.*

$$H_Z = -\sum_{z \in Z} Pr(Z) \cdot \log Pr(Z)$$

Huffman Address Map Compression (Marinó et al., 2023)

But first!

Definition 3.1 (Shanon's Entropy)

Given a set of symbols $Z = \{z_1, \ldots, z_n\}$ *and the probability distrbution* Pr*.*

$$H_Z = -\sum_{z \in Z} Pr(Z) \cdot \log Pr(Z)$$

Definition 3.2 (Huffman Codes)

Given a set of symbols $Z = \{z_1, \ldots, z_n\}$ *and the corresponding counting* $\{c_1, \ldots c_n\}$*.*
Huffman encoding will encode the elements minimizing:

$$\sum_{i=1}^{n} \frac{c_i}{n} \cdot l_i$$

where l_i *is the length of the code* i*.*

Huffman Address Map Compression (Marinó et al., 2023)

$$w = \begin{pmatrix} 0 & 5 & 2 & 4 \\ 4 & 1 & 3 & 1 \\ 6 & 0 & 5 & 3 \\ 0 & 5 & 0 & 2 \end{pmatrix}$$

Huffman Address Map Compression (Marinó et al., 2023)

$$w = \begin{pmatrix} 0 & 5 & 2 & 4 \\ 4 & 1 & 3 & 1 \\ 6 & 0 & 5 & 3 \\ 0 & 5 & 0 & 2 \end{pmatrix}$$

1. Apply Huffman Encoding to each element of the matrix w

Huffman Address Map Compression (Marinó et al., 2023)

$$w = \begin{pmatrix} 0 & 100 & 101 & 110 \\ 110 & 1110 & 11110 & 1110 \\ 11111 & 0 & 100 & 11110 \\ 0 & 100 & 0 & 101 \end{pmatrix}$$

2. Now, use the canonical variant of Huffman Codes (CHC)

symbol	**code**
0	0
5	100
2	101
4	110
1	1110
3	11110
6	11111

l	**first_symbol**	**first_code_l**
0	0	0
1	0	0
2	1	16
3	1	16
4	4	28
5	5	30
6	-	32

Huffman Address Map Compression (Marinó et al., 2023)

$$HAM(w) = 0\ 110\ 11111\ 0\ 100\ 1110\ 0\ 100\ 101\ 11110\ 100\ 0\ 110\ 1110\ 11110\ 101$$

3. We join the binary string column-based order

symbol	code
0	0
5	100
2	101
4	110
1	1110
3	11110
6	11111

l	first_symbol	first_code_l
0	0	0
1	0	0
2	1	16
3	1	16
4	4	28
5	5	30
6	-	32

Huffman Address Map Compression (Marinó et al., 2023)

$$HAM(w) = 0\ 110 \| \ 1111 \| 1\ 0\ 10 \| 0\ 111 \| 0\ 0\ 10 \| 0\ 101 \| \ 1111 \| 0\ 100 \| \ 0\ 110 \| \ 1110 \| \ 1111 \| 0\ 101$$

$$C_{\text{HAM}}(w) = \{6, 15, 10, 7, 2, 5, 15, 4, 6, 14, 15, 5\}$$

4. Divide the bitstream in integers of b bits (e.g. $b = 4$)

symbol	**code**
0	0
5	100
2	101
4	110
1	1110
3	11110
6	11111

l	**first_symbol**	**first_code_l**
0	0	0
1	0	0
2	1	16
3	1	16
4	4	28
5	5	30
6	-	32

If w doesn't have repeated elements: bits$(HAM) \leq 3nm \log nm + (nm)^2 + b - 2 \log nm$
If w has $k < nm$ distinct elements: bits$(HAM) \leq nm + nm \log k + B_k$

Sparse Huffman Address Map Compression (Marinó et al., 2023)

- If the matrix is sparse and very large, HAM is in trouble.
- sHAM does:
  - Use CSC over the matrix.
  - Use HAM for vector *nz*.
  - The other vectors stay normal.
- Space:
  - If the matrix contains *snm* non-zero distinct elements (excluding 0):

$$\text{bits}(sHAM(w)) \leq snm\,(3\log(snm) + snm + b + \log n) - \log(snm) + m\log n$$

  - If the matrix contains *snm* non-zero elements and $k < snm$ distinct elements (excluding 0):

$$\text{bits}(sHAM(w)) \leq snm(1 + \log k \log n) + m\log n + B_k$$

Grammar-Compressed (Paolo et al., 2022)

$$w = \begin{pmatrix} 5 & 0 & 2 & 3 \\ 4 & 1 & 3 & 1 \\ 5 & 0 & 2 & 3 \\ 5 & 0 & 2 & 0 \end{pmatrix}$$

$$V = [5, 2, 4, 3, 1]$$

$$S = \begin{array}{l} <1,1><2,3><4,4>\$ \\ <3,1><5,2><4,3><5,4>\$ \\ <1,1><2,3><4,4>\$ \\ <1,1><2,3>\$ \end{array}$$

Grammar-Compressed (Paolo et al., 2022)

$$w = \begin{pmatrix} 5 & 0 & 2 & 3 \\ 4 & 1 & 3 & 1 \\ 5 & 0 & 2 & 3 \\ 5 & 0 & 2 & 0 \end{pmatrix}$$

$$V = [5, 2, 4, 3, 1]$$

$$S = \begin{array}{l} R_1 < 4, 4 > \$ \\ < 3, 1 >< 5, 2 >< 4, 3 >< 5, 4 > \$ \\ R_1 < 4, 4 > \$ \\ R_1 \$ \end{array}$$

GRAMMAR-COMPRESSED (PAOLO ET AL., 2022)

$$w = \begin{pmatrix} 5 & 0 & 2 & 3 \\ 4 & 1 & 3 & 1 \\ 5 & 0 & 2 & 3 \\ 5 & 0 & 2 & 0 \end{pmatrix}$$

$$V = [5, 2, 4, 3, 1]$$

$$S = \begin{array}{l} R_2\$ \\ < 3, 1 >< 5, 2 >< 4, 3 >< 5, 4 > \$ \\ R_2\$ \\ R_1\$ \end{array}$$

Grammar-Compressed (Paolo et al., 2022)

$$w = \begin{pmatrix} 5 & 0 & 2 & 3 \\ 4 & 1 & 3 & 1 \\ 5 & 0 & 2 & 3 \\ 5 & 0 & 2 & 0 \end{pmatrix}$$

$$V = [5, 2, 4, 3, 1]$$

$$S = \begin{array}{l} R_2\$ \\ R_3 R_4\$ \\ R_2\$ \\ R_1\$ \end{array}$$

Grammar-Compressed (Paolo et al., 2022)

$$w = \begin{pmatrix} 5 & 0 & 2 & 3 \\ 4 & 1 & 3 & 1 \\ 5 & 0 & 2 & 3 \\ 5 & 0 & 2 & 0 \end{pmatrix}$$

$$V = [5, 2, 4, 3, 1]$$

$$S = R_2\$R_5\$R_2\$R_1\$$$

$$R = \{R_1 \rightarrow < 1, 1 >< 2, 3 >, R_2 \rightarrow R_1 < 4, 4 > R_3 \rightarrow < 3, 1 >< 5, 2 >, R_4 \rightarrow < 4, 3 >< 5, 4 >, R_5 \rightarrow R_3 R_5$$

Time Complexity of Matrix/Vector multiplication

- HAM: $O(nm \log k)$
- sHAM: $O(snm \log k)$
- Grammar-Compressed: $O(|R| + |C|)$
- HAM and sHAM inscrease linearly based on the amount of the elements in the matrix
- Grammer-Compressed increased based the grammar rules

Part IV

Is it solved?
There are some open problems...

REFERENCES I

Anwar, S., Hwang, K., & Sung, W. (2017). Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *13*(3), 1–18.

Ba, J., & Caruana, R. (2014). Do deep nets really need to be deep? *Advances in neural information processing systems*, *27*.

Choi, Y., El-Khamy, M., & Lee, J. (2020). Universal deep neural network compression. *IEEE Journal of Selected Topics in Signal Processing*, *14*(4), 715–726.

Gong, Y., Liu, L., Yang, M., & Bourdev, L. (2014). Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*.

Marinó, G. C., Ghidoli, G., Frasca, M., & Malchiodi, D. (2021). Compression strategies and space-conscious representations for deep neural networks. *2020 25th International Conference on Pattern Recognition (ICPR)*, 9835–9842. https://doi.org/10.1109/ICPR48806.2021.9412209

Marinó, G. C., Petrini, A., Malchiodi, D., & Frasca, M. (2023). Deep neural networks compression: A comparative survey and choice recommendations. *Neurocomputing*, *520*, 152–170.

Paolo, F., Giovanni, M., Travis, G., Dominik, K., Gonzalo, N., STRIANI, M., & Francesco, T. (2022). Improving matrix-vector multiplication via lossless grammar compressed matrices. *48th International Conference on Very Large Databases*.

Saad, Y. (2003). *Iterative methods for sparse linear systems*. SIAM.

References II

Zhang, T., Ye, S., Zhang, K., Tang, J., Wen, W., Fardad, M., & Wang, Y. (2018). A systematic dnn weight pruning framework using alternating direction method of multipliers. *Proceedings of the European conference on computer vision (ECCV)*, 184–199.