UNIVERSITY OF PISA
Department of Computer Science
Master Programme in Data Science and Business Informatics

MASTER THESIS

Minimizing Entropy for Training and Quantization (METaQ): A novel algorithm for neural network training and compression

SUPERVISORS
Prof. Antonio FRANGIONI
Prof. Paolo FERRAGINA

CANDIDATE
Andrea CARDIA

ACCADEMIC YEAR 2023-24

Abstract

A neural network compression strategy is developed during the training phase by adding a regularization term $\phi(w)$ to the loss function, aiming to minimize the entropy of the network's weights. $\phi(w)$ results from a non-differentiable optimization problem that is computationally very complex to solve. However, fortunately, to train the network, it is not necessary to explicitly find $\phi(w)$; instead, it is sufficient to provide its (sub)gradient to standard machine learning tools (such as PyTorch) to guide the training towards a low-entropy weight configuration (in addition to achieving good accuracy). This subgradient can be computed using the optimal Lagrange multipliers β^* associated with the set of constraints involving the weights w in the problem that defines $\phi(w)$.

In this work, we will develop a procedure to determine β^* by applying Lagrangian relaxation and optimization techniques, also developing ad hoc methods for certain subproblems when necessary for efficiency reasons. Once a trained network with low entropy is obtained, the compression strategy culminates with the quantization of the weights to the reference values of the buckets where training has directed them. The task of encoding the weights is entrusted to well-known compression algorithms that achieve entropy.

The tests were conducted on the LeNet-5 network using MNIST, although the strategy is also applicable to larger networks.

Indice

1 INTRODUCTION

Artificial Neural Networks (ANN) are nonlinear statistical models [Hastie et al., 2017] inspired by the functioning of the human brain, designed to process information in a way similar to how biological neurons process signals. These systems consist of a set of nodes, called artificial neurons, organized into layers and interconnected by edges, each associated with a weight.

Formally, denoting by a and b two neurons in the l-th layer, we can indicate the weight associated with their connection as $w_{a,b}^{(l)}$.[1]

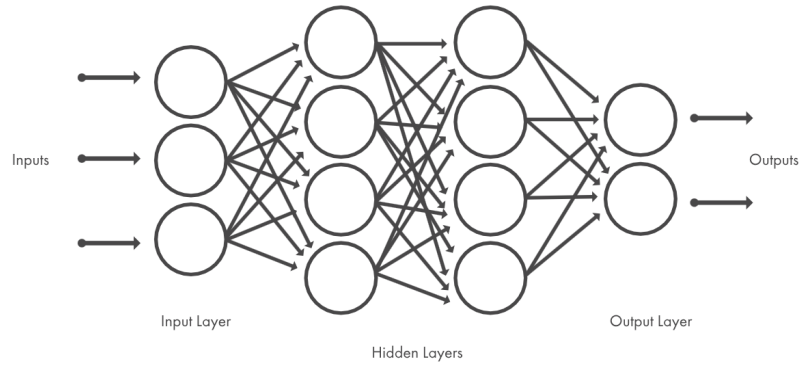

Figura 1: Example of a neural network architecture with 3 inputs and 2 target classes. The set of connections between the input layer and the first hidden layer is represented by the matrix $w^{(1)}$. Source: mathworks.com

The set of these objects forms a three-dimensional tensor corresponding to the collection of weight matrices for each layer. Often, the indices a, b, and l are omitted unless strictly necessary, and the entire set of network weights is simply denoted as w to keep the notation concise. In the remainder of this work, we will follow this convention, referring to n as the total number of weights in the network.

The network receives input data in the first layer (called the input layer), where the data is appropriately organized into N records whose structure is to be captured. It then performs a dot product with the weights of the current layer, followed by a mathematical transformation—typically through an activation function, which is nonlinear to detect potential nonlinearity in the problem. The result is then transmitted to the neurons of the next layer. At the final layer, the output is the value $\hat{y}_i(w)$ corresponding to record i, based on the given weight configuration w.

The goal of training techniques is to minimize the error made by the network in its predictions compared to the true value of the function being modeled. This error is measured through a mathematical function, known as the loss function, which evaluates the discrepancy between the network's output and the expected result. This function, denoted as $L(w)$, depends on the weight configuration w and returns a real number.

[1] Without loss of generality, throughout the rest of the discussion, we will use the term `weight` to also refer to the concept of `bias`.

For example, in a regression problem, the objective is to predict a continuous value, and a common loss function is the mean squared error (MSE), which measures the average of the squared differences between the actual values y_i associated with record i and the corresponding predictions $\hat{y}_i(w)$, as follows:

$$L(w) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i(w))^2 \tag{1}$$

For a classification problem, instead, the cross-entropy loss is often used. For a multiclass problem with M classes, it is expressed as:

$$L(w) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log(\hat{y}_{i,j}(w)) \tag{2}$$

There are numerous other choices for $L(w)$, such as MAE [Mitchell, 1997], Huber Loss [Tong, 2023], KL divergence [Van Erven and Harremos, 2014], or Dice Loss [Li et al., 2019] (to name just a few). However, this document does not aim to provide a detailed discussion on the choice of the loss function; for a comprehensive review, refer to [Wang et al., 2020].

What is important to emphasize here is that training consists of adjusting the weights w in such a way as to minimize the error $L(w)$. In other words, the goal is to find the network's weight configuration w^* such that:

$$w^* = \arg\min_{w} L(w) \tag{3}$$

ANNs excel in tasks such as image recognition [Quraishi et al., 2012], natural language understanding [Fanni et al., 2023], financial data analysis [Kurani et al., 2023], and even the creation of original content [Elbadawi et al., 2024], among many others. Their success lies in the fact that they are universal function approximators [Schäfer and Zimmermann, 2006]. Thus, whenever there is a process in which a certain input corresponds to a certain output, they aim to mimic and identify that process.

The fundamental concept behind neural networks is their ability to model complex relationships between data, making them a powerful tool for solving problems that are difficult to address with traditional algorithms.

Thanks to training techniques, they can learn autonomously from data, improving their performance as they receive more information [Haykin, 2009].

1.1 Problem presentation

What is typically done in equation (3) is adding a regularization term to the objective function that directs the network's training towards a weight configuration that satisfies not only the minimum of the function $L(w)$ but also some other property.

In this case, (3) becomes:

$$\min\{L(w) + \lambda\Omega(w) : w \in \mathbb{R}^n\} \tag{4}$$

Where λ is a hyperparameter, that is, a parameter that must be set before the model training and is not directly learned from the data. It is used to balance the terms $L(w)$ and $\Omega(w)$.

A typical example is ensuring that the weights do not overfit the instances seen by the network during training (training set) but rather generalize better in their predictions on unseen instances (test set). If we allowed the network weights to take any value, we might be able to correctly classify[2] the instances from the training set even in 100% of the cases, without actually learning the underlying structure of the data.

For this reason, during the training of a neural network, it is undesirable for the weights to grow too much (in absolute value). Therefore, in the problem (3), a term is added to minimize the norm of the weights.

The most common choices to limit this growth of the weights are either

$$\Omega(w) = ||w||_1 = \sum_{i=1}^{n} |w_i| \tag{5}$$

or

$$\Omega(w) = ||w||_2 = \sum_{i=1}^{n} w_i^2 \tag{6}$$

We do not discuss the difference here but refer to [Melkumova and Shatskikh, 2017] for a detailed discussion.

Another example is structured pruning regularization [Cacciola et al., 2024], a regularization technique that promotes sparsity at the structural level (e.g., channels, filters, layers) in deep learning models, encouraging models that are more efficient in terms of memory and computation. There are several similar techniques that apply regularization to achieve analogous effects.

In this work, we choose to integrate a term into the network training problem that takes into

[2]This argument is not limited to classification problems.

account, in addition to the standard regularization to avoid overfitting, also the minimization of the entropy of the network's weights.

The weight tensor can indeed be seen as a sequence of values to which a metric can be associated, considering how well the space it occupies in memory lends itself to compression.

By training the network to minimize this metric, it is possible to obtain, once training is completed, a network that not only possesses excellent predictive capability for the problem at hand but is also more amenable to compression in terms of memory usage.

Achieving this result is crucial for integrating artificial intelligence algorithms into applications on devices with limited resources, such as smartphones, IoT devices, or embedded systems [Han et al., 2016].

Moreover, reducing the entropy of the weights can act as a form of regularization that limits the model's complexity, reducing the risk of overfitting. In other words, the network is "forced" to focus on the most relevant information rather than learning noise or non-general details.

1.1.1 Buckets Organization

To develop a training strategy while simultaneously compressing the neural network, we construct $\Omega(w)$ in such a way as to guide the training towards a quantization of the weights, i.e., towards a limited number of possible values for the weights.

Let C be the number of buckets into which we want to direct the weights, and let c_b be a variable that counts the number of weights that fall into bucket b, where $b = 1, \ldots, C$. Then, we can write the Shannon entropy as:

$$H = \sum_{b=1}^{C} c_b \log_2 c_b \tag{7}$$

The minus sign that is usually found in front of the entropy formula is actually conventional and serves to ensure that entropy is always a positive quantity. In fact, what is typically defined is the entropy over probabilities; these are values between 0 and 1, and thus their logarithm is always a negative quantity. Therefore, it is necessary to artificially insert a minus sign in front of the entropy summation to guarantee its positivity. However, in our case, c_b is a count, not a fraction, so we can define the entropy with the positive sign.

The C buckets we have defined are equidistant from each other, with a width of $\frac{1}{C}$. In Figure 2, there is a pictorial representation of this arrangement.

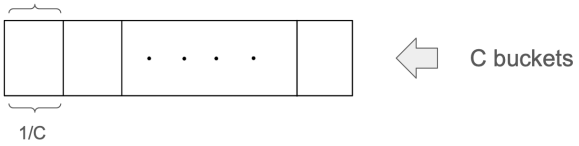

Figura 2: Buckets organization

The goal is to train the network so that the entropy is minimized. This will make the network more compressible once the weights are quantized.

Such quantization consists in assigning to all the weights within a specific bucket the central value of the bucket. If we assume that the weights are within a certain range $[w_0 - r, w_0 + r]$[3], the weights within the interval $[w_0 - r, w_0 - r + \frac{2r}{C})$ (i.e., the first bucket) will be assigned the central value $w_0 - r + \frac{r}{C}$, those in the interval $[w_0 - r + \frac{2r}{C}, w_0 - r + 2\frac{2r}{C})$ (i.e., the second bucket) will be assigned the central value $w_0 - r + 2\frac{r}{C}$, and so on.

1.1.2 METaQ Problem

In light of what has been presented in the previous sections, the problem of training the network with an entropy compression term (METaQ Problem) can be written as follows:

$$\min\{L(w) + \lambda(\alpha R(w) + (1 - \alpha)\phi(w)) : \mathbb{R}^n\} \tag{8}$$

where λ is the hyperparameter introduced earlier that balances how much weight to give to $L(w)$ and how much weight to give to the total regularization $\Omega(w)$ during training; $\Omega(w)$ is composed of two parts, $R(w)$ and $\phi(w)$, whose relative importance is balanced by a second hyperparameter α, which expresses as a percentage how much $R(w)$ contributes to the total regularization $\Omega(w)$. The function $R(w)$ represents the standard regularization term, while the function $\phi(w)$ represents the entropy regularization term, and is itself given by the following optimization problem:

[3]One could also think of $w_0 = \frac{1}{2}$ and $r = \frac{1}{2}$ to model the case where the weights are in $[0, 1]$, but for greater generality, we leave the interval in its general form. It is not necessarily the case that the most performant configuration of the network has weights in $[0, 1]$.

$$\phi(w) = \min \sum_{b \in \mathfrak{B}} c_b \log_2(c_b) \tag{9}$$

$$w_i = \sum_{b \in \mathfrak{B}} v_b x_{i,b} \qquad i \in I \tag{10}$$

$$\sum_{b \in \mathfrak{B}} x_{i,b} = 1 \qquad i \in I \tag{11}$$

$$c_b = \sum_{i \in I} x_{i,b} \qquad b \in \mathfrak{B} \tag{12}$$

$$x_{i,b} \in \{0, 1\} \qquad i \in I, b \in \mathfrak{B} \tag{13}$$

In the problem (9)-(13), $\mathfrak{B}$ is the set of buckets into which we want to direct the weights, I is the set of weights in the network with the total number $|I| = n$, v_b is the quantization value of bucket b, which, according to the construction made in Section 1.1.1, can be written as $v_b = w_0 - r + b\frac{r}{C}$, where $b = 1, \ldots, C$[4], $w_i \in [w_0 - r, w_0 + r]$ is the i-th weight of the network, and finally, the variables $x_{i,b}$ are binary variables defined as follows:

$$x_{i,b} = \begin{cases} 1, & \text{if weight } i \text{ belongs to bucket } b \\ 0, & \text{otherwise} \end{cases} \tag{14}$$

In optimization theory, the variables $x_{i,b}$ are called decision variables, and they play a crucial role since they are the true variables of optimization problems, including in the definition of $\phi(w)$.

From the definition of $x_{i,b}$, it follows that in the problem (9)-(13), the constraint (10), combined with the constraint (11), which imposes that only one of the $x_{i,b}$ is 1 in a given bucket, tells us that each weight value w_i in the network must equal exactly one of the representative values v_b of the buckets. The constraint (10) is important because it is the only point where the dependence of ϕ on w is explicitly stated.

The constraint (12), which defines c_b, simply imposes that c_b is the total number of weights associated with bucket b.

From the integrality of the $x_{i,b}$ expressed in (13), the integrality of c_b follows.

Thus, the combination of (10) and (11) defines the quantized representation of the weights, the constraint (12) prepares the data to compute the entropy, and the constraint (13) ensures the discrete nature of the quantization.

However, the problem (9)-(13) as defined involves a large number of binary decision variables

[4]In the special case where $[w_0 - r, w_0 + r] \to [0, 1]$, i.e., if $w_0 = \frac{1}{2}$, $r = \frac{1}{2}$, the first bucket goes from 0 to $\frac{1}{C}$ with a central value of $\frac{1}{2C}$, as also obtained from the expression $w_0 - r + b\frac{r}{C}$ for $b = 1$.

$x_{i,b}$ (if we imagine an 8-bit quantization corresponding to $C = 256$ levels, we have up to $256 \times n$ decision variables) and constraints. Exactly solving a problem of this scale would require enormous computational power and would be impractical for real-world applications, not only for large neural networks but also for smaller networks.

Fortunately, this problem can be easily circumvented. In fact, to solve the training problem, what is necessary to provide to machine learning tools like `pytorch` is a gradient of the function we want to minimize. Typically, for standard loss functions and regularization terms, these gradients (i.e., $\nabla L(w)$ and $\nabla R(w)$) are well-known functions for the same tools. What we need to find is $\nabla \phi(w)$.

An important result from Lagrangian relaxation theory comes to our rescue: the optimal Lagrange multipliers β^* associated with the constraint (10) are precisely a (sub)gradient for $\phi(w)$ to feed into tools like `pytorch` to solve the METaQ problem.

The goal of this work is to illustrate how to find, through optimization techniques and algorithms, the vector β or at least an approximation $\overline{\beta}$ to solve the METaQ Problem in equation (8).

1.2 Literature review

We have seen the many advantages that neural network compression brings with it. Applications of these methods find ample space in so-called Large Language Models [Zhu et al., 2024], neural networks designed to understand and generate natural language, where the dimensionality of the models, as can be easily inferred from the name, is very large. The most famous ones, such as GPT4 [Achiam et al., 2023], LLaMA 3 [Dubey et al., 2024], PaLM [Chowdhery et al., 2023], and Claude [Enis and Hopkins, 2024], to name just a few, have hundreds of billions of parameters, which translates into a considerable dimensionality that is unlikely to have a direct application on client devices. This is why the largest companies in the sector are moving towards the implementation of compression strategies for these models [Knight, 2025], to ensure lower resource usage and energy consumption, not to mention that enabling execution on local devices increases user control over their data, improving privacy.

In the landscape of neural network compression algorithms, pruning algorithms certainly find ample space [Cheng et al., 2024]. Pruning removes non-essential weights or neurons in a neural network, thus reducing the overall number of parameters. It is based on the observation that many connections in over-sized models have a negligible impact on performance. The most common techniques are structured pruning (See for example [Santacroce et al., 2023] or [Ma et al., 2023]), which removes entire filters, channels, or layers, and unstructured pruning ([Frantar and Alistarh, 2023], [Syed et al., 2023], and [Zhang et al., 2023]), which removes individual weights, leaving the network more irregular.

Another family of compression algorithms consists of the so-called quantization algorithms. Quantization reduces the numerical precision of the network's weights and/or activations, for example,

by switching from 32-bit floating-point numbers (FP32) to lighter representations such as INT8 or even binary. It is further divided into two main types: post-training quantization (PTQ), where quantization is applied after training (as in [Liu et al., 2021]), and quantization-aware training (QAT), which integrates quantization during training, allowing the network to adapt ([Liu et al., 2023], [Liu et al., 2022], [Ding et al., 2023], [Fu et al., 2023]).

Still referring to the families of compression algorithms, we also briefly mention knowledge distillation techniques and low-rank factorization methods.

Naturally, all these types of algorithms can sometimes also be applied in pipelines, as in [Han et al., 2016].

The strategy presented in this work can be placed somewhere between QAT and PTQ algorithms; indeed, during training, there is no true quantization. The introduction of the term (9) in the METaQ problem (the training problem of the network with the addition of the entropy regularization term) of equation (8) allows the network weights to *be directed* towards quantization, which will then be actually performed after training by replacing the weights with the reference value of the bucket assigned during training (and not by modifying the memory space occupied by the weights).

For the solution of the problem (9)-(13), [Pappalardo and Passacantando, 2012] was an indispensable guide for the foundations of optimization theory and operations research, while [Frangioni, 2005] provided the basis for the theory of Lagrangian relaxation.

The historical cornerstone [Kelley, 1960], together with the instructive application found in [Farruggia et al., 2019], allowed tackling the resolution of the Lagrangian relaxation subproblem of (9)-(13), providing the intuition for a specialized algorithm. Since it was possible to characterize the optimal solution of the relaxation in a simple way, it became feasible to efficiently determine the value of the Lagrangian solution as done in [Frangioni and Gorgone, 2013].

[D'Antonio and Frangioni, 2009], [Beck and Teboulle, 2009a], and [Frangioni, 2020] were of fundamental help in solving the dual problem; the first two with subgradient strategies and the latter as an invaluable guide in the sophisticated world of bundle methods.

Finally, for the network compression part, it is important to note that in this work we limited ourselves to considering, for simplicity's sake, zero-order entropy, meaning that each symbol of the source (the set of weights) was considered independent of the others, and no relationship or context between the symbols was taken into account. This is clearly a limitation, since in the neural network the weights are interconnected. For this reason, in addition to algorithms like Huffman coding and arithmetic coding, something like the FM-index [Ferragina et al., 2004] and its variants could have been used to exploit higher-order entropy k.

1.3 Thesis content

The organization of this document is as follows.

In Section 2, we will discuss the approximations made when handling the problem (9)-(13). We will then define a Lagrangian relaxation of the problem, which will allow us to decompose it into two solvable subproblems. We will discuss the analytical solution of the first and develop a custom algorithm to efficiently solve the second. Much of the discussion will focus on the latter point. At this stage, after stating and proving optimization theorems, we will use these results to solve the previously formulated Lagrangian relaxation through some appropriately discussed non-differentiable optimization algorithms. We will see that the solutions found in this way will also yield the sought approximation $\overline{\beta}^*$ for $\nabla\phi$.

In Section 3, we will discuss how, once a low-entropy trained network is obtained, it can be compressed using well-known compression algorithms.

In Section 4, after introducing the well-known MNIST dataset and the famous LeNet-5 architecture, specifically designed for its classification, we will present the results obtained by applying the strategy described and analyzed throughout this work.

Finally, in Section 5, we will discuss the implementation issues of METaQ, the precautions to take if one wishes to replicate the results on other networks, and potential future developments.

Section 6 contains acknowledgements, while Appendix 1 will summarize all the hyperparameters present in METaQ with their descriptions.

2 LAGRANGIAN RELAXATION

The first thing to do in order to handle (9)-(13) is to ensure that $\phi(w)$ is at least well-defined everywhere. For now, in fact, $\phi(w)$ is not even continuous, since the right-hand side of (10) spans a discrete set of values, while the left-hand side spans a continuous one. The problem is that $x_{i,b}$ is a binary variable, which makes the problem discontinuous. Therefore, the first thing to do is to relax (13) to:

$$x_{i,b} \in \{0,1\} \qquad i \in I, b \in \mathfrak{B} \tag{15}$$

In the problem (9)-(13), the constraints (12) tie together the x corresponding to different weights. If it were possible to decouple the c from the x, the source of non-linearity (i.e., the entropy) could be eliminated.

This is possible through Lagrangian relaxation techniques. What is done, in fact, is to introduce

the Lagrange multipliers $\xi_{bb\in\mathfrak{B}}$, each of which is associated with one of the constraints in (12). The goal at this point is to solve a problem in which these constraints do not need to be satisfied exactly (hence the term "relaxed"), but rather they should be satisfied "as much as possible." Once the LHS and RHS are moved to the same side, the constraints are moved into the objective function, and we minimize, along with what was previously to be minimized, the deviation of this difference from 0. The solution that will be found will therefore be one that drives this term toward 0, i.e., towards satisfying the relaxed constraints. The problem thus becomes one of finding the function (concave in the variables ξ)

$$\phi_w(\xi) = \min\left\{\sum_{b\in\mathfrak{B}} c_b \log_2(c_b) + \sum_{b\in\mathfrak{B}} \xi_b(\sum_{i\in I} x_{i,b} - c_b) : (10), (11), (15)\right\} \tag{16}$$

to maximize it

$$\max\{\phi_w(\xi) : \xi \in \mathbb{R}^C\} \tag{17}$$

In correspondence with the solution ξ of (17), the corresponding optimal multipliers β associated with the constraints (10) (or more precisely, their approximation $\overline{\beta}^*$) will have been found, and thus a (sub)gradient $\nabla\phi(w)$.

In fact, holds the following:

Th 1. *Let $\phi(w) = \min f(x) : g(x, w) = 0, x \in X$, where $f(x)$ is convex and $g(x, w) = 0$ represents n linear constraints in w, each of which has a derivative with respect to the n-th component of w equal to 1 (as in Equation* (10)*).*

Then, the optimal Lagrange multipliers λ^* associated with the constraints $g(x, w) = 0$ are a super-gradient for the function $\phi(w)$.

Dimostrazione. Let $w' = w + \Delta w$ a perturbation of w.

In this point we have

$$\phi(w + \Delta w) = \min\{f(x) : g(x, w + \Delta w) = 0, x \in X\} \tag{18}$$

The Lagrangian function is:

$$L(x, \lambda, w) = f(x) + \lambda^t \cdot g(x, w) \tag{19}$$

And therefore

$$\phi(w) = \min(L(x, \lambda, w)) = \min(f(x) + \lambda^t \cdot g(x, w) \tag{20}$$

Let $x^*(w)$ the optimal value, that is

$$\phi(w) = L(x^*(w), \lambda, w) = f(x^*(w)) + \lambda^t \cdot g(x^*(w), w) = f(x^*(w)) \tag{21}$$

Now, $x^*(w)$ is the minimum of $\phi(w)$ but not necessarily of $\phi(w + \Delta w)$, therefore

$$\phi(w + \Delta w) \leq f(x^*(w)) + \lambda^*(w)^t \cdot g(x^*(w), w + \Delta w) \tag{22}$$

Exploiting the assumption of linearity of g in w, the fact that its derivatives are 1, and that at $x^*(w)$ it equals 0, we have

$$g(x^*(w), w + \Delta w) = g(x^*(w), w) + \Delta w = \frac{\partial g_1}{\partial w}(x^*(w), w)\Delta w = \Delta w \tag{23}$$

Hence

$$\phi(w + \Delta w) \leq f(x^*(w)) + \lambda^*(w)^t \cdot \Delta w \tag{24}$$

namely

$$\phi(w + \Delta w) \leq \phi(w) + \lambda^*(w)^t \cdot \Delta w \tag{25}$$

i.e. $\lambda^*(w)$ is a subgradient for $\phi(w)$.

□

2.1 Analitic subproblem resolution

In this section, we solve the subproblem of (16) related to the variables c_b, that is:

$$\min\{c_b \log_2 c_b - \xi_b c_b\} \tag{26}$$

$$\frac{d}{dc_b}(c_b \log_2(c_b) - \xi_b c_b) = 0 \Longrightarrow \frac{d}{dc_b}\left(c_b \frac{\log(c_b)}{\log(2)} - \xi_b c_b\right) = 0 \Longrightarrow \tag{27}$$

$$\implies \frac{1}{\log(2)}\left(\log(c_b^*)+1\right)-\xi_b=0 \implies \log(c_b^*)=\log(2)\xi_b-1 \implies \tag{28}$$

$$\implies \log_2(c_b^*)=\frac{\log(2)\xi_b-1}{\log(2)}=\xi_b-\frac{1}{\log(2)} \implies c_b^*=2^{\xi_b-\frac{1}{\log(2)}} \tag{29}$$

or equivalently $c_b^*=e^{2\xi_b-1}$

As can be observed, the dependence of c_b^* on the multipliers ξ_b is exponential, which is why it is more appropriate to transform the problem (26) into

$$\min\{c_b\log_2 c_b-\xi_b c_b : c_b\in[\underline{c}_b,\overline{c}_b]\} \tag{30}$$

That is, truncate the solutions to a certain interval; those for which $c_b>\overline{c}_b$ are set to $\overline{c}_b$, while those for which $c_b<\underline{c}_b$ are set to $\underline{c}_b$.

The most natural choice for $\overline{c}_b$ is $\overline{c}_b=n$, since no more than n weights can end up in each bucket (the configuration where all weights are in bucket b). The choice for $\underline{c}_b$ is somewhat more delicate. It is certainly necessary to ensure that the logarithm exists, so $\underline{c}_b>0$ must hold. However, one must also avoid setting this value too close to 0, as it could significantly influence the optimal value of the objective function.

Initially, $\underline{c}_b=0.01$ could be a starting point, but it would be more appropriate to leave it as a hyperparameter to be chosen based on the type of network and the results obtained after various trials.

2.2 Knapsack subproblem

In this section, we discuss the solution of the subproblem of (16) related to the decision variables x. The problem is as follows:

$$\min_x \sum_{b\in\mathfrak{B}}\xi_b x_{i,b} \tag{31}$$

$$w_i=\sum_{b\in\mathfrak{B}}v_b x_{i,b} \tag{32}$$

$$\sum_{b\in\mathfrak{B}}x_{i,b}=1 \tag{33}$$

$$x_{i,b}=[0,1] \tag{34}$$

And it has to be solved for each weight $w_i, i\in I$.

In the remainder of this section, we will omit the indexing over i since the problems are independent of the network weight, and there is therefore no risk of ambiguity. We will write the sums in terms of dot products wherever possible.

The given problem has strong similarities to the knapsack problem, especially to the fractional variant. The goal is indeed to minimize or maximize a linear function involving the cost or value of the items, with constraints on the combinations of selected items.

The variables $x_{i,b}$ can be interpreted as a measure of how much to "include" each item. The main difference is the unique selection constraint (33), which does not appear in the classical version of the knapsack problem.

For solving the problem (31)-(34), we will assume that the vector ξ and the vector v are ordered. For the latter, the ordering property follows immediately from its construction, while for the former, it will need to be ensured at the implementation level.

2.2.1 `CVXPY` Solution

One way to solve the problem (31)-(34) is to rely on an optimization solver. In this work, we have chosen `CVXPY`, a Python library for modeling and solving mathematical optimization problems, particularly convex optimization problems.

`CVXPY` does not directly implement solving algorithms but relies on external solvers to solve optimization problems, which in turn use algorithms such as the simplex method (and variants), interior-point methods (and variants), etc. We refer to the official documentation for details [cvxpy.org].

The key point here is that this library solves the problem (31)-(34) but does not do so in the most efficient way possible. Indeed, since it is designed to solve a vast range of optimization problems, it does not provide specialized tools for a specific problem (as it clearly should).

For this reason, it was considered necessary, for the success of this work, to implement a specific algorithm for our needs, that is, to solve the problem (31)-(34).

Once this algorithm is presented, we will compare execution times for solving (31)-(34) to assess the effectiveness of implementing a specialized method.

2.2.2 Cutting Plane Algorithm

The starting point for implementing a specialized algorithm to solve (31)-(34) is Kelley's Cutting Plane algorithm [Kelley, 1960].

This algorithm is an iterative method used to solve convex optimization problems, specifically for approximating a convex programming problem through a finite number of iterations. It is one of the

"cutting plane" methods, which aim to progressively narrow down the search domain to converge toward an optimal solution.

The fundamental idea is to approximate the objective function using linear underestimations (tangent supports), relying on the convexity properties of the function.

In our case, let (P) be the following primal problem:[5]

$$\min_x \left\{ \xi^t \cdot x : w = v \cdot x, \sum_{b=1}^{C} x_b = 1, x_b \in [0,1], \ \forall b = 1, \ldots, C \right\} \tag{P}$$

Relaxing the constraint $w = v^t \cdot x$ yields a Lagrangian:

$$L(x, \lambda) = \xi^t \cdot x + \lambda(w - v^t \cdot x) \tag{35}$$

And therefore, we can define a dual function:

$$\phi(\lambda) = \min_{x \in X} L(x, \lambda) \tag{36}$$

where $X = \{x \in [0,1]^C : \sum_{b=1}^{C} x_b = 1\}$.

Th 2. *If x_1, $x_2 \in X$ then $\forall \theta \in [0,1]$, $\theta x_1 + (1-\theta)x_2 \in X$*

Dimostrazione.

$$\theta x_1 + (1-\theta)x_2 \in [0,1]^C \tag{37}$$

because for each component

$$(\theta x_1 + (1-\theta)x_2)_b = \theta x_{1b} + (1-\theta)x_{2b} \in [0,1] \tag{38}$$

Indeed, since $\theta x_{1b} + (1-\theta)x_{2b}$ is the sum of two non-negative terms, it will certainly be non-negative. Furthermore,

$$\theta x_{1b} \leq \theta \cdot 1 = \theta \tag{39}$$

and

$$(1-\theta)x_{2b} \leq (1-\theta) \cdot 1 = (1-\theta) \tag{40}$$

Therefore, summing the two inequalities, we obtain:

$$\theta x_{1b} + (1-\theta)x_{2b} \leq \theta + (1-\theta) = 1 \tag{41}$$

[5]The formulation is entirely equivalent to the problem (31)-(**??**) where the summation has been replaced with the dot product.

i.e.

$$\theta x_{1b} + (1-\theta)x_{2b} \in [0,1] \tag{42}$$

On the other hand

$$\sum_{b=1}^{C} \theta x_{1b} + (1-\theta)x_{2b} = \theta \sum_{b=1}^{C} x_{1b} + (1-\theta) \sum_{b=1}^{C} x_{2b} = \theta + (1-\theta) = 1 \tag{43}$$

And therefore

$$\theta x_1 + (1-\theta)x_2 \in X \tag{44}$$

□

The Dual Problem (D) is:

$$\max_{\lambda \in \mathbb{R}} \phi(\lambda) \tag{D}$$

To find a solution of (D) we use the cutting plane in this way:

1) We initialize $x_1, x_2 \in X$ such that $w \geq v^t \cdot x_1$ and $w < v^t \cdot x_2$. (For example, we can initialize x_1 as the vector with C components, all 0 except for a 1 in the component b_1 corresponding to the largest $v_{b_1} \leq w$, and x_2 as the vector with C components, all 0 except for a 1 in the component b_2 corresponding to the smallest $v_{b_2} > w$). At the implementation level, the assumption of ordering for v is crucial for identifying b_1 and b_2.

2) At this point, since (D) is equivalent to:

$$\max_{\phi,\lambda} \{\phi : \phi \leq L(x,\lambda), x \in X\} \tag{45}$$

and since there are only x_1 and x_2 in X, there are only 2 constraints, namely:

$$\phi \leq L(x_1,\lambda) \text{ e } \phi \leq L(x_2,\lambda) \tag{46}$$

Let (λ^+, ϕ^+) the intersection between the two curves:

$$\phi = L(x_1,\lambda) = \xi^t \cdot x_1 + \lambda(w - v^t \cdot x_1) \tag{47}$$

$$\phi = L(x_2, \lambda) = \xi^t \cdot x_2 + \lambda(w - v^t \cdot x_2) \tag{48}$$

in the (λ, ϕ) plane. We are certain that such an intersection exists because one curve has a positive slope and the other has a negative slope. By subtracting the second equation from the first and isolating λ^+, what we obtain is:

$$\lambda^+ = \frac{\xi^t \cdot (x_1 - x_2)}{(v^t \cdot x_1 - w)(v^t \cdot x_2 - w)} \tag{49}$$

and

$$\phi^+ = L(x_1, \lambda^+) = \xi^t \cdot x_1 + \lambda^+(w - v^t \cdot x_1) \tag{50}$$

or equivalently $(L(x_1, \lambda) = L(x_2, \lambda))$

$$\phi^+ = L(x_2, \lambda^+) = \xi^t \cdot x_2 + \lambda^+(w - v^t \cdot x_2) \tag{51}$$

Let x^+ also be the true solution of the relaxed problem, i.e.,

$$x^+ = \underset{x \in X}{\operatorname{argmin}}(\xi - \lambda^+ v)^t \cdot x \tag{52}$$

and since ξ (just like v) is ordered in increasing order, x^+ is a vector with 0 everywhere except for a 1 in the component where $\xi_b - \lambda v_b$ is minimum (i.e., for $b = 1$).

Case $\phi(\lambda^+) = \phi^+$.

When $\phi(\lambda^+) = \phi^+$, the point (λ^+, ϕ^+) lies exactly on the dual function $\phi(\lambda)$, i.e.,

$$\phi^+ = \min_{x \in X}(\xi^t \cdot x + \lambda^+(w - v^t \cdot x)) \tag{53}$$

By the Strong Duality Theorem, the optimal value of the primal problem (P) coincides with the optimal value of the dual problem (D):

$$\min_{x \in [0,1]^C} \xi^t \cdot x = \max_{\lambda \in \mathbb{R}} \phi(\lambda) = \phi^+ \tag{54}$$

Let us now define $x(\theta)$ as any convex combination of x_1 and x_2:

$$x(\theta) = \theta x_1 + (1 - \theta) x_2 \tag{55}$$

e let θ^* such that

$$v^t \cdot x(\theta^*) = w \tag{56}$$

$x(\theta^*)$ è soluzione di (P)

Dimostrazione. We need to prove that $x(\theta^{)}$ satisfies all the primal constraints and that the optimal value of the objective function (i.e., ϕ^+ in this case) coincides with $\xi^t \cdot x(\theta^{)}$.

As for the constraints, $x(\theta^*)$ satisfies (2) by construction, and satisfies (3) and (4) by Lemma 1.

As for the value of the objective function at $x(\theta^*)$, we have

$$\xi^t \cdot x(\theta^*) = \xi^t(\theta^* x_1 + (1-\theta^*)x_2) = \theta^* \xi^t \cdot x_1 + (1-\theta^*)\xi^t \cdot x_2 \tag{57}$$

Now, since

$$\phi^+ = \xi^t \cdot x_1 + \lambda^+(w - v^t \cdot x_1) \tag{58}$$

or equivalently

$$\phi^+ = \xi^t \cdot x_2 + \lambda^+(w - v^t \cdot x_2) \tag{59}$$

we can write

$$\phi^+ = \xi^t \cdot x_i + \lambda^+(w - v^t \cdot x_i),\ i = 1, 2 \tag{60}$$

from which we can obtain

$$\xi^t \cdot x_i = \phi^+ - \lambda^+(w - v^t \cdot x_i) \tag{61}$$

hence, by substituting in (28)

$$\xi^t \cdot x(\theta^*) = \theta^*(\phi^+ - \lambda^+(w - v^t \cdot x_1)) + (1-\theta^*)(\phi^+ - \lambda^+(w - v^t \cdot x_2)) \tag{62}$$

By doing some calculations, we can arrive at the expression

$$\xi^t \cdot x(\theta^*) = \phi^+ - \lambda^+[\theta^*(w - v^t \cdot x_1) + (1-\theta^*)(w - v^t \cdot x_2)] \tag{63}$$

But

$$0 = w - v^t \cdot x(\theta^*) = w - (\theta^* v^t \cdot x_1 + (1-\theta^*) v^t \cdot x_2) = \theta^*(w - v^t \cdot x_1) + (1-\theta^*)(w - v^t \cdot x_2) \quad (64)$$

Therefore

$$\xi^t \cdot x(\theta^*) = \phi^+ \quad (65)$$

□

Case $\phi(\lambda^+) \neq \phi^+$ and $w = v^t \cdot x^+$.

In this case, x^+ minimizes $L(x, \lambda^+)$ (see Equation (23)), but since it satisfies constraint (2) (as well as constraints (3) and (4)), it is also optimal with optimal value

$$\phi(\lambda^+) = \xi^t \cdot x^+ + \lambda^+(w - v^t \cdot x^+) = \xi^t \cdot x^+ \quad (66)$$

Case $w \neq v^t \cdot x^+$.

If $w > v^t \cdot x^+$, then x^+ does not yet satisfy constraint (2). Since the value of $v^t \cdot x^+$ is too small compared to w, it means that we need to create a new point x that gives a value of $v^t \cdot x$ closer to w. Consequently, we update the set X for the next iteration of the cutting plane algorithm by replacing x_1 with x^+ because x^+ is closer to the constraint than x_1, yet still maintains $v^t \cdot x^+ < w$. The new set X becomes $X = x^+, x_2$, and the procedure is repeated.

If $w < v^t \cdot x^+$, then $v^t \cdot x^+$ is too large compared to w, so we replace x_2 with x^+ since x^+ is closer to the constraint than x_2, but still maintains $v^t \cdot x^+ > w$. The new set X becomes $X = x_1, x^+$.

2.2.3 Specialized Algorithm

Finally, we are ready to construct a specific algorithm for solving the problem (31)-(34).

The basic idea is that when $\lambda \to -\infty$, the solution[6]

$$x^* = \arg\min_{x \in X} (\xi - \lambda v)^t \cdot x \quad (67)$$

The active component[7] of the solution is only the first component. This is because, as we stated earlier, both ξ and v are ordered in increasing order, and thus the minimum of the vector $\xi - \lambda v$

[6] See equation (52) where λ is left as a free variable.
[7] By active components of a vector, we mean its non-zero components.

corresponds to the minimum of the two vectors, that is, the first component.

Conversely, if $\lambda \to +\infty$, the solution to (**??**) has as the active component only the last component.

As λ moves from $-\infty$ to $+\infty$, the minimum of the vector $\xi - \lambda v$ moves to the right, transitioning from $b = 0$ to $b = C - 1$.

We will call the breakpoints the values of b where the minimum of $\xi - \lambda v$ switches from one component to another as λ moves from $-\infty$ to $+\infty$, and we will denote with $\overline{B}$ that C-dimensional vector with zeros everywhere and ones at the breakpoints. These breakpoints occur at the values λ such that $\xi_{b_i} - \lambda^v_{b_i} = \xi_{b_j} - \lambda^* v_{b_j}$, where $b_i, b_j = 0, ..., M$. It is evident that $M < C$ since at most we can have C breakpoints. Solving for λ^* gives:

$$\lambda^* = \frac{\xi_{b_i} - \xi_{b_j}}{v_{b_i} - v_{b_j}} \tag{68}$$

We have mentioned that two breakpoints are certainly $b = 0$ and $b = C - 1$; now we need to determine what the others are. To find them, we start with the first one ($b = 0$) and calculate the minimum of $\frac{\xi_b - \xi_0}{v_b - v_0}$ for $b = 1, \ldots, C - 1$. In other words, the minimum is calculated among the elements to the right of the last breakpoint found.

In Figure 3, we present a depiction of what a possible configuration of the breakpoints vector might look like.

Figura 3: An exemplary depiction of a possible configuration for the breakpoints vector with $C = 15$.

We observe that the vector $\overline{B}$ depends exclusively on ξ and not on the value of w (the vector v is fixed throughout the training). This property can be exploited to implement the entire algorithm in a parallelized manner.

Once the vector $\overline{B}$ is constructed, the solution to the problem (**??**) is determined by w.

First, we calculate the index ν such that $w = v_\nu$. Now, the index ν may or may not correspond to an active component of $\overline{B}$ (i.e., a breakpoint). In the first case, the solution to (67) is a C-dimensional vector with only one active component (with value 1) corresponding to the component ν. In the second case, the solution is the convex combination of the two breakpoints that enclose ν in $\overline{B}$.

As for the optimal multiplier λ^*, it is given by equation (68), where b_i is the smallest breakpoint larger than ν, and b_j is the largest breakpoint smaller or equal to ν.

If ν corresponds to the last component of $\overline{B}$, we would have $b_i = \nu$, and b_j would be the largest

breakpoint smaller than ν.

2.2.4 Algorithms comparison

The first test conducted was to compare the speed of the cutting plane algorithm against that of the CVXPY library. Using $C = 256$, 1000 instances of the problem (31)-(34) were run, and the execution time for the cutting plane algorithm was $0.20sec$, while for the CVXPY library it was $3.28sec$, resulting in an efficiency improvement of more than a factor of $16\times$.

For a comparison with the vectorized version of the cutting plane algorithm, 10000 instances of the problem (31)-(34) were run with $C = 5$, achieving an efficiency gain in terms of execution time of more than a factor of $3770\times$.

This result demonstrates how necessary it was to implement an alternative strategy to the standard CVXPY approach for solving the problem (31)-(34).

Finally, a comparison was also made between the vectorized cutting plane algorithm and the specialized algorithm (also in its specialized version). Since both are fast algorithms, 44000 instances of the problem were run 50 times (this value was chosen because it is of the order of the weights in LeNet-5) with $C = 32$. The execution time for the cutting plane algorithm was $1.47sec$, while the specialized algorithm took $0.27sec$, resulting in an efficiency gain of a factor of $5.4\times$.

The merit of these results can undoubtedly be attributed primarily to the structure of the refined algorithms discussed throughout this work; however, a crucial role was played by the ability to apply parallelization of the computations, enabling the simultaneous execution of multiple instances of the problem (31)-(34). In fact, for the specialized algorithm, the bulk of the complexity lies in constructing the vector $\overline{B}$; however, $\overline{B}$ is common to all w_i, so, provided we can run multiple instances of the problem for the different w_i simultaneously (which PyTorch handles very well as it was specifically designed for solving tensor-based problems), most of the work is done only once.

Next, we will discuss the complexity of the specialized algorithm.

As mentioned, the most computationally expensive part is calculating $\overline{B}$. Fortunately, this only needs to be done once for all the weights; however, this calculation involves finding a minimum that can potentially happen C times. The minimum is taken over a vector that gets progressively smaller, but still, the number of operations performed is $O(C^2)$. The worst-case scenario occurs when the breakpoints are exactly C, and thus for each breakpoint, a minimum of length $O(C)$ must be calculated.

We now provide a heuristic argument showing that the worst-case scenario practically never occurs if C is "large enough," also giving a characterization of the term "large enough."

To do this, it is sufficient to calculate the (empirical) probability of having a certain number of breakpoints.

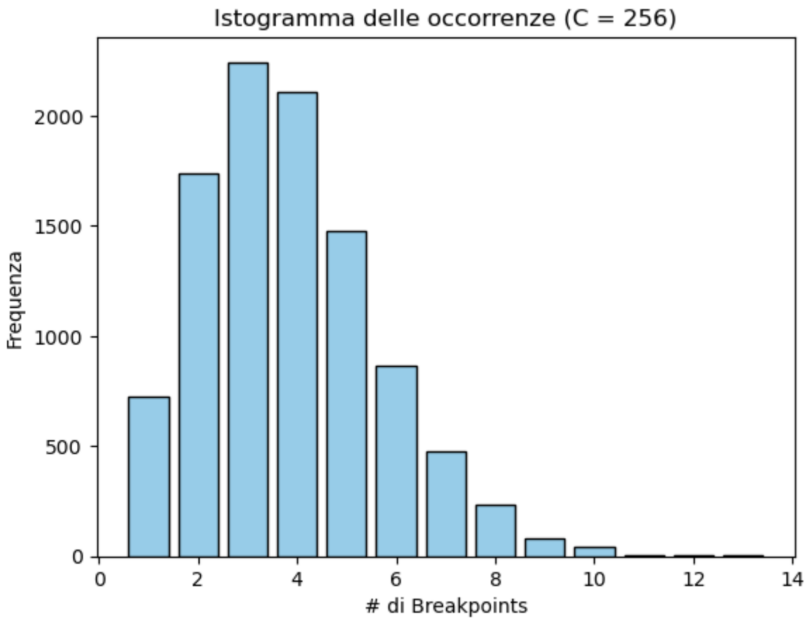

Figura 4: Histogram of the occurrences of having a certain number of breakpoints for a $\overline{B}$ with $C = 256$.

In Figure 4, we can see that for $C = 256$, meaning a $\overline{B}$ with 256 entries, out of 10,000 instances of the problem, no more than 13 breakpoints are ever obtained. Moreover, the probability seems to be modeled by a left-skewed Poisson distribution, peaking around a very low value for the most probable number of breakpoints.

2.3 Lagrangian Dual solution

At this point, we have found, for each weight of the network w at a fixed step of the training cycle, the optimal value for c_b (which we denoted as c_b) and the optimal value of the decision variables $x_{i,b}$ (which we denoted as $x_{i,b}$). Corresponding to these values, we have the optimum of the objective function $\phi_w(\xi)$, that is:

$$\phi_w(\xi) = \sum_b c_b^* \log_2(c_b^*) + \sum_b \xi_b (\sum_i x_{i,b}^* - c_b^*) \tag{69}$$

This function must be maximized in order to solve the Lagrangian dual (17), which we present here for the sake of readability.

$$\max\{\phi_w(\xi) : \xi \in \mathbb{R}^C\} \tag{70}$$

Below, we present some methods through which this problem can be solved. We focus on two families: the so-called subgradient methods and the bundle methods.

With reference to equation (69), we state the following theorem:

Th 3. *$g = \{\sum_{i=1}^{n} x_{i,b}^* - c_b^*\}_{b=1}^{C}$ is a super-gradient for $\phi_w(\xi)$*

Dimostrazione. From Equation (69), it follows that:

$$\sum_b c_b^* \log_2(c_b^*) = \phi_w(\xi) - \sum_b \xi_b(\sum_i x_{i,b}^* - c_b^*) \tag{71}$$

For any ξ', we have:

$$\phi_w(\xi') = \min(\sum_b c_b \log_2(c_b) + \sum_b \xi_b'(\sum_i x_{i,b} - c_b)) \tag{72}$$

And evaluating this expression at the optimum for ξ, we get:

$$\phi_w(\xi') \leq \sum_b c_b^* \log_2(c_b^*) + \sum_b \xi_b'(\sum_i x_{i,b}^* - c_b^*) = \phi_w(\xi) - \sum_b \xi_b(\sum_i x_{i,b}^* - c_b^*) + \sum_b \xi_b'(\sum_i x_{i,b}^* - c_b^*) \tag{73}$$

that is

$$\phi_w(\xi') \leq \phi_w(\xi) + \sum_b (\xi_b' - \xi_b) \sum_i (x_{i,b}^* - c_b^*),\ \forall \xi' \tag{74}$$

i.e. $\{\sum_i x_{i,b}^* - c_b^*\}_{b=1}^{C}$ is a supergradient for $\phi_w(\xi)$.

□

2.3.1 Supergradient Methods

To solve (17), we follow a strategy similar to the one used for differentiable problems, that is, we construct an iterative algorithm where at each iteration k the variable ξ (which we can index as ξ_k) is updated by moving it along the direction of the gradient (supergradient in our case) of the function. Let γ_k denote the step size by which this movement occurs along the gradient direction. We can then write the update for the next step as follows:

$$\xi_{k+1} = \xi_k + \gamma_k g_k \tag{75}$$

The choice of γ_k is crucial, and the simplest way to choose it is using the so-called diminishing square-summable (DSS) stepsize, that is,

$$\gamma_k = \frac{1}{k} \tag{76}$$

With this choice, the algorithm converges, in fact:

$$||\xi_{k+1} - \xi^*||^2 = ||\xi_k + \gamma_k g_k - \xi^*||^2 = ||\xi_k - \xi^*||^2 + (\gamma_k)^2||g_k||^2 + 2\langle \xi_k - \xi^*, \gamma_k g_k \rangle \tag{77}$$

which, by factoring out γ_k from the inner product, can be written as:

$$||\xi_{k+1} - \xi^*||^2 = ||\xi_k - \xi^*||^2 + (\gamma_k)^2||g_k||^2 + 2\gamma_k \langle \xi_k - \xi^*, g_k \rangle \tag{78}$$

Now, by the definition of a supergradient, we have:

$$f(\xi^*) \leq f(\xi_k) + \langle \xi^* - \xi_k, g_k \rangle \Rightarrow f(\xi^*) - f(\xi_k) \leq \langle \xi^* - \xi_k, g_k \rangle \tag{79}$$

That is, by changing the sign to make it explicit, we get $\langle \xi_k - \xi^*, g_k \rangle$:

$$\langle \xi^* - \xi_k, g_k \rangle \geq f(\xi^*) - f(\xi_k) \Rightarrow \langle \xi_k - \xi^*, g_k \rangle \leq f(\xi_k) - f(\xi^*) \leq 0 \tag{80}$$

In the last inequality, we used the fact that the optimal value $f(\xi^*)$ is a maximum. Therefore, what we have is that:

$$||\xi_{k+1} - \xi^*||^2 \leq ||\xi_k - \xi^*||^2 + (\gamma_k)^2||g_k||^2 + 2\gamma_k(f(\xi_k) - f(\xi^*)) \tag{81}$$

Therefore, by choosing γ_k as in equation (76), the term $(\gamma_k)^2||g_k||^2$ becomes negligible for large k compared to the term $2\gamma_k(f(\xi_k) - f(\xi^*))$, and since the latter term is negative, we have that with this choice of γ_k,

$$||\xi_{k+1} - \xi^*||^2 \leq ||\xi_k - \xi^*||^2 \tag{82}$$

That is, the iterates ξ_k converge to the optimal ξ^*.

Given its simplicity in implementation, the DSS stepsize method is very common in optimization practices. However, it has some limitations; it converges slowly and does not take into account information from previous iterations.

A more often considered alternative, which is generally more efficient, is the FISTA (Fast Iterative Shrinkage-Thresholding Algorithm) method, first introduced in 2009 by Amir Beck and Marc Teboulle in [Beck and Teboulle, 2009b]. FISTA is an evolution of gradient-based methods designed to accelerate convergence compared to the basic gradient method or methods with DSS stepsize. It combines the gradient method with an acceleration term based on an inertial term inspired by Nesterov's techniques

[Nesterov, 1983]. This term takes into account the "history" of previous iterations, helping to push the solution more quickly towards the optimal minimum.

Introducing a hyperparameter ζ as the convergence step, an auxiliary variable is used.

$$\sigma_k = \xi_k + \frac{t_{k-1} - 1}{t_k}(\xi_k - \xi_{k-1}) \tag{83}$$

where t_k is an increasing sequence that accelerates convergence, such as:

$$t_{k+1} = \frac{1 + \sqrt{1 + 4t_k^2}}{2} \tag{84}$$

and the iterate ξ_k is updated using the auxiliary variable σ_k as follows:

$$\xi_{k+1} = \sigma_k + \frac{1}{\zeta} g_k \tag{85}$$

2.3.2 Bundle Methods

In this section, we discuss bundle methods, a class of optimization algorithms (both linear and nonlinear) widely used to tackle non-differentiable problems or those with objective functions that exhibit non-smooth behaviors. These methods are a generalization of subgradient methods, improving their stability and convergence speed.

The main idea behind bundle methods is to construct an approximation of the objective function based on a set $B^{(k)}$ (called the bundle) of local information, such as function evaluations and subgradients, which are then used to iteratively generate a direction of improvement. This approach proves particularly useful in convex (or concave) non-differentiable optimization problems, where the objective function is not sufficiently smooth for classical gradient-based methods to be applied.

In this family of methods, we will discuss the Proximal Bundle Method (PBM), a variant of bundle methods that introduces a regularization term to make the optimization more stable and effectively handle non-differentiability. The main idea is to minimize an aggregated approximation of the objective function, balanced with a penalization term that accounts for the distance between the current point and the next candidate point.

The PBM approach is based on the iterative solution of the regularized problem:

$$\xi^{(k+1)} = \arg\max_{\xi}(\phi_w^{(k)}(\xi) - \frac{\zeta}{2}||\xi - \xi^{(k)}||^2) \tag{86}$$

where

- $\underline{\phi}_w^{(k)}(\xi)$ è una sottostima modellata della funzione obiettivo costruita utilizzando il bundle (supergradienti e valori della funzione in punti precedenti);
- $||\xi - \xi^{(k)}||^2$ è un termine di penalizzazione che incoraggia le iterazioni a rimanere vicine al punto corrente $\xi^{(k)}$;
- $\zeta > 0$ è un parametro di regolarizzazione, che bilancia il peso tra l'approssimazione della funzione e il termine di penalizzazione.

At each iteration, the supergradients $g^{(k)}$ and the function values $\phi_w(\xi^{(k)})$ are collected at a series of $B_s = |B^{(k)}|$ previous points $\xi_j^{(k)},\ j = 1, \ldots, B_s$.

The parameter B_s represents the bundle size and is a key hyperparameter of the algorithm. If the bundle size is too small, the algorithm may not have enough historical information to construct an accurate approximation of the objective function, risking suboptimal choices. In such cases, bundle methods behave similarly to subgradient methods, losing the advantage of constructing the bundle. On the other hand, if the bundle size is too large, the algorithm may become more complex and computationally expensive because the number of points to be considered grows.

Thus, at each iteration k, the goal is to solve the subproblem among the elements of the bundle $\xi^{(k)}j, \phi w, j^{(k)}, g^{(k)}jj = 1^{B_s}$

$$j^* = \arg\min_{j \in B^{(k)}} \phi_{w,j}^{(k)} + g_j^{(k)t} \cdot (\xi - \xi_j^{(k)}) \tag{87}$$

and the update of the iterates is performed through:

$$\xi^{k+1} = \xi_{j^*}^{(k)} + \frac{1}{\zeta} g_{j^*}^{(k)} \tag{88}$$

3 NETWORK COMPRESSION

Once the problem (17) has been solved, or at least an approximate solution $\overline{\xi}$ has been found, we have also solved, in correspondence with this solution, the problem (31)-(34), and thus we have found the corresponding optimal multipliers β (or at least their approximation $\overline{\beta}^*$), which, as we have seen, correspond to the (sub)gradient of $\phi(w)$ to be used in training the network and thus solving the METaQ Problem (8).

This training will aim to push the network weights w not only towards minimizing the loss function and their magnitude but also towards minimizing entropy. Once the training is complete, the weights will be arranged in such a way as to satisfy all three of these requirements.

However, at the end of the training, the weights will not be exactly in the quantized buckets, and each weight will need to be assigned to the corresponding bucket. This means that if, at the end of the training, the weight w_i lies within the interval $[w_0 - r + b\frac{2r}{C}, w_0 - r + (b+1)\frac{2r}{C})$, it will be assigned the central value $w_0 - r + b\frac{r}{C}$, i.e., it will be quantized to that value.

This will further decrease the entropy without compromising the model's predictive capability.

Once this procedure is carried out, the network is ready to be compressed. It is, in fact, a sequence of weights (numerical values) for which there are numerous compression techniques to represent it with a smaller size in terms of occupied bytes. For this work, Huffman Coding and Arithmetic Coding were used. The latter algorithm, although much slower than the former, achieves entropy in the encoding, and it is the main reason why we have used this metric as a representative of the network size up to this point.

4 RESULTS

In this chapter, we present the results achieved by testing METaQ.

4.1 LeNet-5 & MNIST

The tests were conducted by training LeNet-5, a pioneering convolutional neural network (CNN) designed by Yann LeCun in 1998 [LeCun et al., 1998], primarily used for image recognition and, in particular, for classifying handwritten digits such as those in the MNIST dataset (short for Modified National Institute of Standards and Technology), a dataset widely used in the fields of machine learning and deep learning, especially for image classification.

MNIST is one of the most well-known datasets for evaluating optical character recognition (OCR) algorithms and contains images of handwritten digits representing the numbers 0 to 9. Each image is grayscale, with a size of 28x28 pixels (a total of 784 pixels), and is accompanied by a label indicating the represented digit (for example, an image of a "5" has the label 5). The training set consists of 60,000 images for training the models, while the test set contains 10,000 images to evaluate their performance.

LeNet-5 consists of a combination of convolutional layers and pooling (or subsampling) layers, followed by fully connected layers. Its architecture is relatively simple compared to modern CNNs, but it was revolutionary at the time of its creation. Here is a summary of its structure:

- Input Layer: Accepts grayscale images of size 32x32 (a larger format than MNIST, which is 28x28, so the images are usually resized). Each pixel of the image represents one unit in the input layer.

- Convolutional Layer 1: Applies 6 convolutional filters (kernels) of size 5x5. Output: 6 feature maps of size 28x28. The filters allow for the extraction of local features such as edges and textures.
- Pooling Layer 1: Applies a subsampling operation (average pooling) with 2x2 windows and a stride of 2. Output: 6 feature maps of size 14x14. Reduces the dimensionality and makes the features more robust to translations.
- Convolutional Layer 2 (C3): Applies 16 convolutional filters of size 5x5, with partial connections between feature maps (not all filters are connected to all input maps, reducing complexity). Output: 16 feature maps of size 10x10.
- Pooling Layer 2 (S4): Applies another layer of average pooling with 2x2 windows. Output: 16 feature maps of size 5x5.
- Fully Connected Layers: Layer 5: A fully connected layer with 120 neurons. Layer 6: Another fully connected layer with 84 neurons.
- Output Layer: Uses a softmax function to classify the image into 10 classes (from 0 to 9, for digit classification).

4.2 METaQ Test

Several tests of the training and compression algorithm presented in this work have been conducted. We refer to Appendix 1 for a summary of all the model's hyperparameters.

Here, we present one particular configuration that led to interesting results. With reference to the problem (8), the following was used:

$$L(w) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{C} y_{i,j}\log(\hat{y}_{i,j}(w)) \tag{89}$$

and

$$R(w) = ||w||_2 \tag{90}$$

That is, the cross-entropy was chosen as the loss function and the l_2 norm as the standard regularization term.

We configured an Adam optimizer to train the network with a learning rate $l_r = 0.0007$. The specific parameters of the Adam algorithm that control the exponential decay coefficients for the

first-order moments (mean of the gradients) and second-order moments (variance of the gradients) were set to the standard values of 0.9 for the first and 0.999 for the second.

In equation (8), the coefficient of the standard regularization term is given by $\lambda\alpha$, while the coefficient of the entropy regularization term is given by $\lambda(1-\alpha)$. To configure these two parameters, we first studied the behavior of the network without entropy regularization to understand the most appropriate value for $\lambda\alpha$, and then we added a small perturbation and gradually adjusted its intensity.

Indeed, if we denote by $L_1 = \lambda\alpha$ and $L_2 = \lambda(1-\alpha)$, we find that once the value for L_1 is determined without perturbation, we can keep it fixed while varying L_2 to progressively increase the entropy regularization. α and λ are given by $\lambda = L_1 + L_2$ and $\alpha = L_1/\lambda$.

The optimal values for these parameters that we report here are $\lambda = 0.0015$ and $\alpha = 0.533$.

As for the parameter C, it is one of the most important parameters in the entire model but also one of the ones that most significantly impacts the execution time of the whole algorithm. For this reason, we chose to keep it at low values ($C = 6$) with the idea of adjusting the actual number of quantization buckets in the post-training phase.

For the weight initialization, we also observed how the weights were distributed in the network without entropy regularization and tried to set the parameters w_0 and r accordingly. These parameters are characteristic of each network and tell us within which range the weights should be initialized for optimal convergence. The LeNet-5 network for MNIST predictions without perturbation tends to converge its weights approximately in the range [-1, 1] ($w_0 = 0, r = 1$), and it is around these values that we explored the configuration of these two hyperparameters, obtaining $w_0 = -0.11$ and $r = 1.114$.

With these parameter configurations, by the 144th epoch, we achieved a model with 98.95% accuracy and an entropy of 114,234 bits.

Quantization was performed by varying C within the range [1, 1000], and the best result was for $C = 140$, achieving 99.01% accuracy and an entropy of 59,601 bits.

Considering that the initial entropy of the network weights starts at 1,421,632 bits, this means that the model manages to train the network while compressing it by a factor of 23.85× (compression ratio of 4.19%) without losing predictive capability.

For comparison with the state of the art in compression strategies for LeNet-5 on MNIST, we report the results from [Han et al., 2016], where a compression ratio of 39× (compression ratio of 2.56%) was achieved, from [Basha et al., 2024] where 2.02% was achieved, and from [Jin et al., 2019], where a compression ratio of 56× (1.79%) was achieved.

5 CONCLUSIONS

In this thesis, we developed and analyzed METaQ, an innovative algorithm designed to integrate entropy minimization into the training phase of a neural network, combining advanced optimization techniques and tailored approaches. METaQ was tested on LeNet-5 and MNIST, a relatively simple architecture and dataset, and the applicability of the algorithm to larger networks or complex datasets remains to be explored. It will be essential to test METaQ on more advanced network architectures, such as ResNet, Transformer, and Large Language Models, to evaluate its effectiveness in modern contexts and on more challenging datasets, such as ImageNet or CIFAR-10.

The presence of several hyperparameters (such as λ, α, or the number of buckets C) requires careful tuning to achieve optimal results. This sensitivity makes the algorithm subject to revision to adapt from one network to another. However, the description of how this tuning was performed to train LeNet-5 should serve as a useful guide should the application target change. Given the large number of hyperparameters, some of these were not fully explored due to computational costs and available time.

Although Lagrangian relaxation techniques and specialized algorithms provided an effective solution for computing the subgradient of $\phi(w)$, the approximations introduced could compromise accuracy in cases where the data structure or the network configuration do not perfectly align with the underlying assumptions.

A crucial area for development could be the search for more advanced optimization algorithms to solve the problem (17), or the integration of METaQ with other compression strategies, such as knowledge distillation or structured pruning, which could lead to even more computationally efficient models and better compression results.

Another critical point on which the entire analysis in this work was based is the form of entropy. One could consider the problem of minimizing higher-order entropy (instead of zero-order entropy), treating the network weights as not independent. A promising direction could be the introduction of compression metrics that exploit the dependency between weights, for example, using FM-index or higher-order entropy algorithms such as gzip-9 or zstd-22.

Studying alternative regularizations and evaluating other regularization functions that reduce entropy, exploring the possibility of models that more naturally balance compression and accuracy, could also bring some benefits.

The adoption of METaQ could have a significant impact on the efficiency of artificial intelligence models, especially in resource-constrained environments such as mobile or embedded devices. However, further studies are needed to address the identified limitations and fully explore its potential.

6 THANKSGIVING

I would like to thank my Professors, who gave me the opportunity, through their ideas and insights, to be able to work on this project, which is as innovative and topical as it is stimulating and engaging, allowing me to interface with cutting-edge research work and technologies in a rapidly changing world.

But most of all, I thank my mom, who gave me the opportunity to graduate and build the person I am.

Appendix 1: Algorithm hyperparameters.

We summarize in this Appendix all the hyperparameters called out in the text that are necessary for configuring the algorithm with a brief description of them.

Hyperparameter	**Description**
l_r	learning rate in the training optimizer
β_1	moving average of the gradient in ADAM for training
β_2	moving average of the gradient square in ADAM for training
C	number of buckets
λ	coefficient of the total regularization in METaQ
α	percentage of standard regularization versus total regularization
ζ	parameter for adjusting the convergence pitch in FISTA and PBM
w_0	central value around which the weights are initialized
r	radius of initialization of weights
$\overline{c}_b$	upper bound for c_b^*
$\underline{c}_b$	lower bound for c_b^*
n_{epochs}	number of epochs used to train the network

Tabella 1: Hyperparameters of which METaQ is composed

BIBLIOGRAPHY

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. *Gpt-4 technical report*. arXiv preprint arXiv:2303.08774, 2023.

SH Shabbeer Basha, Mohammad Farazuddin, Viswanath Pulabaigari, Shiv Ram Dubey, and Snehasis Mukherjee. *Deep model compression based on the training history*, volume 573, page 127257. Neurocomputing, Elsevier, 2024.

Amir Beck and Marc Teboulle. *A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems*, volume 2, pages 183–202. SIAM J. IMAGING SCIENCES, 2009a. doi: 10.1137/080716542.

Amir Beck and Marc Teboulle. *A fast iterative shrinkage-thresholding algorithm for linear inverse problems*, volume 2, pages 183–202. SIAM journal on imaging sciences, 2009b.

Matteo Cacciola, Antonio Frangioni, and Andrea Lodi. *Structured pruning of neural networks for constraints learning*, volume 57, page 107194. Operations Research Letters, 2024. doi: https://doi.org/10.1016/j.orl.2024.107194. URL `https://www.sciencedirect.com/science/article/pii/S0167637724001305`.

Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. *A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations*. IEEE, 2024.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. *Palm: Scaling language modeling with pathways*, volume 24, pages 1–113. Journal of Machine Learning Research, 2023.

cvxpy.org. Cvxpy api documentation. URL `https://www.cvxpy.org/api_reference/cvxpy.html`.

Giacomo D'Antonio and Antonio Frangioni. *Convergence Analysis of Deflected Conditional Approximate Subgradient Methods*, volume 20, pages 357–386. SIAM J. Optim., 2009. doi: 10.1137/080718814.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. *Parameter-efficient fine-tuning of large-scale pre-trained language models*, volume 5, pages 220–235. Nature Publishing Group UK London, 2023.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. *The llama 3 herd of models*. arXiv preprint arXiv:2407.21783, 2024.

Moe Elbadawi, Hanxiang Li, Abdul W Basit, and Simon Gaisford. *The role of artificial intelligence in generating original scientific research*, volume 652, page 123741. International Journal of Pharmaceutics, Elsevier, 2024.

Maxim Enis and Mark Hopkins. *From LLM to NMT: Advancing Low-Resource Machine Translation with Claude.* arXiv preprint arXiv:2404.13813, 2024.

Salvatore Claudio Fanni, Maria Febi, Gayane Aghakhanyan, and Emanuele Neri. *Natural language processing*, pages 87–99. Springer, 2023.

Andrea Farruggia, Paolo Ferragina, Antonio Frangioni, and Rossano Venturini. *Bicriteria Data Compression*, volume 48, pages 1603–1642. SIAM Journal on Computing, 2019. doi: 10.1137/17M1121457. URL `https://doi.org/10.1137/17M1121457`.

Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. *An alphabet-friendly FM-index*, pages 150–160. International Symposium on String Processing and Information Retrieval, 2004.

Antonio Frangioni. *About Lagrangian Methods in Integer Optimization*, volume 139, pages 163–193. Annals of Operations Research, 2005. doi: https://doi.org/10.1007/s10479-005-3447-9.

Antonio Frangioni. *Standard Bundle Methods: Untrusted Models and Duality*, pages 61–116. Springer International Publishing, Cham, 2020. doi: 10.1007/978-3-030-34910-3_3.

Antonio Frangioni and Enrico Gorgone. *A library for continuous convex separable quadratic knapsack problems*, volume 229, pages 37–40. European Journal of Operational Research, 2013. doi: https://doi.org/10.1016/j.ejor.2013.02.038. URL `https://www.sciencedirect.com/science/article/pii/S0377221713001719`.

Elias Frantar and Dan Alistarh. *Sparsegpt: Massive language models can be accurately pruned in one-shot*, pages 10323–10337. International Conference on Machine Learning, 2023.

Zihao Fu, Haoran Yang, Anthony Man-Cho So, Wai Lam, Lidong Bing, and Nigel Collier. *On the effectiveness of parameter-efficient fine-tuning*, volume 37, pages 12799–12807. Proceedings of the AAAI conference on artificial intelligence, 2023.

Song Han, Huizi Mao, and William J. Dally. *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding.* 2016. URL `https://arxiv.org/abs/1510.00149`.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction.* Springer, 2017.

Simon Haykin. *Neural networks and learning machines*. Pearson Education India, 2009.

Sian Jin, Sheng Di, Xin Liang, Jiannan Tian, Dingwen Tao, and Franck Cappello. *DeepSZ: A Novel Framework to Compress Deep Neural Networks by Using Error-Bounded Lossy Compression*, page 159–170. HPDC '19. Association for Computing Machinery, New York, NY, USA, 2019. ISBN 9781450366700. doi: 10.1145/3307681.3326608. URL https://doi.org/10.1145/3307681.3326608.

J. E. Kelley, Jr. *The Cutting-Plane Method for Solving Convex Programs*, volume 8, pages 703–712. Journal of the Society for Industrial and Applied Mathematics, 1960. doi: 10.1137/0108053. URL https://doi.org/10.1137/0108053.

Will Knight. *Nvidia's* $3,000$*'Personal AI Supercomputer' Will Let You Ditch the Data Center*. *Wired*, 2025.

Akshit Kurani, Pavan Doshi, Aarya Vakharia, and Manan Shah. *A comprehensive comparative study of artificial neural network (ANN) and support vector machines (SVM) on stock forecasting*, volume 10, pages 183–208. Annals of Data Science, Springer, 2023.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. *Gradient-based learning applied to document recognition*, volume 86, pages 2278–2324. Proceedings of the IEEE, 1998.

Xiaoya Li, Xiaofei Sun, Yuxian Meng, Junjun Liang, Fei Wu, and Jiwei Li. *Dice loss for data-imbalanced NLP tasks*. arXiv preprint arXiv:1911.02855, 2019.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. *Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning*, volume 35, pages 1950–1965. Advances in Neural Information Processing Systems, 2022.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. *Llm-qat: Data-free quantization aware training for large language models*. arXiv preprint arXiv:2305.17888, 2023.

Zhenhua Liu, Yunhe Wang, Kai Han, Wei Zhang, Siwei Ma, and Wen Gao. *Post-training quantization for vision transformer*, volume 34, pages 28092–28103. Advances in Neural Information Processing Systems, 2021.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. *Llm-pruner: On the structural pruning of large language models*, volume 36, pages 21702–21720. Advances in neural information processing systems, 2023.

LE Melkumova and S Ya Shatskikh. *Comparing Ridge and LASSO estimators for data analysis*, volume 201, pages 746–755. Procedia engineering, Elsevier, 2017.

Tom M. Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.

Yurii Nesterov. *A method for solving the convex programming problem with convergence rate O (1/k2)*, volume 269, page 543. 1983.

Massimo Pappalardo and Mauro Passacantando. *Ricerca operativa*. Didattica e ricerca. Manuali. Pisa University Press, 2012. ISBN 9788867411542. URL `http://digital.casalini.it/9788867411542`. Casalini id: 2935851.

Md Iqbal Quraishi, J Pal Choudhury, and Mallika De. *Image recognition and processing using artificial neural network*. 2012 1st international conference on recent advances in information technology (RAIT), (pp. 95-100). IEEE., 2012.

Michael Santacroce, Zixin Wen, Yelong Shen, and Yuanzhi Li. *What matters in the structured pruning of generative language models?* arXiv preprint arXiv:2302.03773, 2023.

Anton Maximilian Schäfer and Hans Georg Zimmermann. *Recurrent neural networks are universal approximators*, pages 632–640. Artificial Neural Networks–ICANN 2006: 16th International Conference, Athens, Greece, September 10-14, 2006. Proceedings, Part I 16, 2006.

Aaquib Syed, Phillip Huang Guo, and Vijaykaarti Sundarapandiyan. *Prune and tune: Improving efficient pruning techniques for massive language models*. The First Tiny Papers Track at ICLR 2023, 2023.

Hongzhi Tong. *Functional linear regression with Huber loss*, volume 74. Journal of Complexity, 2023.

Tim Van Erven and Peter Harremos. *Rényi divergence and Kullback-Leibler divergence*, volume 60, pages 3797–3820. IEEE Transactions on Information Theory, 2014.

Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. *A comprehensive survey of loss functions in machine learning*, pages 1–26. Annals of Data Science, 2020.

Mingyang Zhang, Hao Chen, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, and Bohan Zhuang. *Loraprune: Pruning meets low-rank parameter-efficient fine-tuning*. arXiv preprint arXiv:2305.18403, 2023.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. *A Survey on Model Compression for Large Language Models*, volume 12, pages 1556–1577. Transactions of the Association for Computational Linguistics, 2024. doi: 10.1162/tacl_a_00704. URL `https://doi.org/10.1162/tacl_a_00704`.