Politecnico di Torino

Politecnico di Torino

Master's Degree in Computer Engineering: Artificial Intelligence and Data Analytics
July 2024

The Multiply-And-Max/min Neural Paradigm as a Pruning and Training Accelerator

Supervisors:

Prof. Fabio PARESCHI
Prof. Gianluca SETTI
Dr. Luciano PRONO

Candidate:

Lorenzo NIKIFOROS

Abstract

Neural networks have revolutionized the field of artificial intelligence, enabling machines to perform complex tasks once exclusive to human cognition. However, large-scale neural networks present significant computational challenges, particularly during training on servers and deployment on embedded devices. The high computational cost and resource demands impede their practical application in low-resources/energy devices. To address this issue, pruning is introduced, which is a technique that systematically removes redundant parameters and has emerged as a promising solution to reduce computational complexity while maintaining performance.

This Master's thesis explores the effectiveness of a novel layer, Multiply-And-Max/min (MAM), introduced as an alternative to the classical Multiply and Accumulate (MAC) approach, wherein the reduction function is not the sum of all elements but only of the largest and the smallest.

Experimental results demonstrate the efficacy of the MAM-based approach in significantly sparsifying matrices through different pruning techniques, particularly the Global Gradient Pruning (GGP), which achieved, e.g. on ViT trained on ImageNet-1K, an accuracy drop less than 3% while removing 99.93% of weights.

In particular, this study highlights novel properties of the MAM neurons. Since the MAM layer's ability is to identify essential interconnection, it is possible to reintroduce the MAC layer post-pruning, thereby reducing numbers of FLOPs from 3 to 2 for each weight.

The validity of this transition is supported by empirical evidence showing that the strength of this layer lies in identifying crucial interconnections. As an example, as observed with ViT trained on ImageNet-1K dataset, moving from a deeply pruned MAM structure to a deeply pruned MAC structure keeps the accuracy unaltered, that goes from 79.70% for MAM to 78.95%.

A final experiment involves pruning the DNN layers before the convergence of the training process, demonstrating two important properties of the MAM neural paradigm. Firstly, MAM is capable of identifying the crucial interconnections prior to convergence. Secondly, by leveraging this, it is possible to introduce significant FLOPs savings during the training on server, reducing the energy consumption. As an example, in the case of AlexNet

trained on CIFAR-10, 99.8% of FLOPs can be theoretically saved for the pruned layers during training with MAM.

Acknowledgements

I want to thank my family, especially my grandmother and my sister for their support throughout these long years, despite the difficulties.

I also want to thank my friends from Ovada (and Costa): Lollo, Davi, Ila, and Ele who have always been part of my life since the first year of high school and whom I know I can always count on.

Other thanks go to Samu, who have always supported me even in the most difficult moments, as well as my other friends from Turin: Rob, Martino, Cappellino, Daniè, Ometto, Rosy, Angelo, Miri, Claudia, Cuda, Angeluzzo and Pascuzzi.

Also I want to send my regards to Messrs. Gabriele, Peppe and Francesco for treating me like part of the family.

A special thanks to Gaia who is always there for me and encourages me to be a better person.

I would also like to thank my supervisors for their guidance in completing my thesis, particularly Luciano for his constant availability and help.

In memory of my mother and my grandfather.

Table of Contents

List of Tables

List of Figures

Acronyms

AI
Artificial Intelligence

IoT
Internet of Things

TinyML
Tiny Machine Learn- ing

LLMs
Large Language Models

MAC
Multiply and Accumulate

MAM
Multiply and Max/Min

DNN
Deep Neural Network

CNN
Convolutional Neural Network

ViT
Vision Transformer

LMP

Local Magnitude Pruning

GMP

Global Magnitude Pruning

LGP

Local Gradient Pruning

GGP

Global Gradient Pruning

LPSP

Local Probability of Selection Pruning

GPSP

Global Probability of Selection Pruning

DST

Dynamic Sparse Training

LTH

Lottery Ticket Hypothesis

FLOP

Floating Point Operation

Chapter 1

Introduction

Neural networks have emerged as a cornerstone of modern artificial intelligence, revolutionizing various fields including computer vision, natural language processing, and reinforcement learning. These powerful computational models, inspired by the biological structure of the human brain, are adept at learning complex patterns and relationships from data, enabling machines to perform tasks that were once thought to be exclusive to human cognition.

Neural networks are very important in solving many real-world problems. They are used for things like identifying images and recognizing speech, as well as for self-driving cars and natural language processing. Neural networks can automatically find complex patterns in raw data and do well with new, unseen data. This has made them a key part of AI research and applications.

For example, in image classification, neural networks like U-Net [1] help doctors by identifying diseases from X-rays and MRIs, often as accurately as human experts [2]. In speech recognition, they power virtual assistants like Siri and Alexa, helping them understand and respond to what people say [3]. Self-driving cars use neural networks to process data from cameras and sensors to drive safely and smoothly [4].

Large Language Models (LLMs), a type of neural network, are used in natural language processing (NLP) to perform a wide range of language tasks. These include generating human-like text, translating languages, summarizing documents, and more. Models like GPT-4 [5] are examples of LLMs that perform a wide range of language-related tasks.

However, the widespread adoption of neural networks is not without challenges. One significant impediment is their substantial computational

cost, both in terms of memory and processing power. Training and deploying large-scale neural networks demand formidable computational resources, often necessitating high-performance computing infrastructure and energy-intensive computations. This issue becomes even more pronounced when considering the integration of neural networks with the Internet of Things (IoT) and embedded systems.

In the context of IoT, devices are typically characterized by their limited computational capacity, restricted memory, and power constraints. These limitations present a formidable challenge when attempting to implement neural networks on such devices [6]. For instance, smart sensors and IoT devices designed for monitoring environmental conditions, industrial equipment, or health metrics need to operate continuously on battery power, making energy efficiency a critical concern. Implementing traditional large-scale neural networks in these scenarios would be impractical due to their heavy computational and power requirements.

To address these challenges, the concept of Tiny Machine Learning (TinyML) has emerged. TinyML focuses on developing and deploying machine learning models on resource-constrained devices, by optimizing algorithms and models to run efficiently on low-power microcontrollers, enabling the integration of neural networks into IoT devices [7]. This opens up numerous applications where intelligent processing is performed at the edge, reducing the need for constant communication with centralized cloud servers and thereby saving bandwidth and energy.

Modern architecture, for exemple Large Language Models like GPT-4, represent the other end of the spectrum in terms of computational demand. GPT-4, with its billions of parameters, requires extensive computational resources for both training and inference [8]. Training such models involves vast amounts of data and high-performance GPUs or TPUs, consuming significant electrical power. Deploying these models also necessitates considerable computational infrastructure to ensure low-latency responses and high availability, which is why they are typically hosted on powerful cloud servers rather than on local devices.

Integrating these huge models into the context of TinyML poses a significant challenge due to their contrasting computational requirements. Addressing these computational needs has become a focal point for researchers in the field. One promising technique that has garnered significant attention is pruning [9] [10]. Pruning involves systematically removing redundant or less influential parameters, connections, or neurons from a trained neural

network, thereby reducing its size and computational complexity.

By selectively pruning components while preserving model performance, one can achieve more efficient models that are easier to deploy on resource-constrained devices. This approach not only reduces the computational workload but also facilitates faster inference times and more energy-efficient operation.

In recent years, pruning techniques have evolved considerably, offering increasingly sophisticated methods for optimizing neural network architectures. These advancements have led to simpler and more efficient AI systems.

1.1 Thesis Objective

The thesis objective is based on the pruning of fully-connected layers in neural networks for image classification tasks. The focus of this work is to explore a novel pruning method, centered around a new layer called Multiply-And-Max/min (MAM) [11][12], that replaces the traditional Multiply-and-ACumulate (MAC) paradigm by altering the accumulate phase. Instead of adding together all the elements (i.e., the inputs multiplied by the weights), this neural model sums only the maximum and minimum values.

Previous work on this layer has demonstrated its effectiveness in making weight matrices highly sparsifiable through pruning. Specifically, the initial studies employed it in small neural networks composed of fully connected and convolutional layers, targeting applications in the IoT field. More recent researches show its efficacy for state-of-the-art neural networks in image classification, such as Vision Transformers.

The first part of this thesis focuses on replicating the results obtained in these studies. Specifically, the models and datasets used are ViT trained on Imagenet-1K and AlexNet trained on CIFAR-10, achiving results comparable with the previous studies.

Additionally, specific of this work, an analysis is conducted to understand why Gradient Pruning (GP) usually performs much better compared to other techniques. This is done by creating a custom score, multiplying the absolute value of the weight by the number of times it has been selected as the maximum or minimum, in an attempt to replicate the functionality of Gradient Pruning. The results show that the effectiveness of GP may be linked to its correlation with the number of times a weight is chosen.

However, despite the effectiveness of this layer, it still had some drawbacks.

Firstly, by altering the layer's structure, the number of FLOPs required per weight theoretically increases from 2 (as required by a MAC layer) to 3. While this is already a disadvantage, it is also important to consider that MAC layers benefit from widely optimized and prevalent hardware architectures, ensuring high efficiency. In contrast, MAM layers lack such optimized hardware, furthermore, creating an optimization customized for this layer would not only need to be specifically implemented in the hardware architecture, but also would require two branch instructions (for the maximum and minimum), which tend to disrupt the normal pipeline structure, further slowing down the process.

Therefore, a central contribution I made in this work was to address the problem at its root. I found that after pruning a MAM layer, it can be reverted to a MAC layer while maintaining the same sparse matrices previously produced. This process enables the creation of a training pipeline that ends with the network's structure identical to its starting point, thus allowing the use of well-known, highly optimized hardware architectures.

For instance, when ViT is trained on the ImageNet-1K dataset, the accuracy remains almost unchanged when switching from a deeply pruned MAM structure to a deeply pruned MAC structure.

Another issue related to the MAM layer concerns its training. During the training of neural networks, especially the newer large models, a vast amount of computational resources is required. Therefore, in my thesis, I advanced the pruning phase to an earlier stage, before the network converges. This strategic adjustment could potentially lead to significant server-side savings in terms of FLOPs during the training process if sparse training were used.

Furthermore, for these models, it is essential to use hardware accelerators like GPUs, which require specially programmed kernels to achieve high efficiency. Currently, despite the existence of a kernel specifically created for MAM optimization, its performance is still about three times worse than a MAC kernel. Therefore, building on the previously obtained results, this work explores further optimization of the training process as follows: once again, before the network converges, it is pruned. After this, the MAC layer is directly reinserted, thus achieving optimization through both the more efficient kernel and potentially, as before, sparse training.

The results obtained demonstrate that with these strategies, it is possible to achieve models with accuracy and size reduction comparable to those obtained with the standard training pipeline, but with significant resource savings.

Chapter 2

Deep Neural Networks

Neurons are the basic computational units of neural networks, functioning similarly to biological neurons in the human brain. These units receive input signals, process them, and produce output signals [13]. The simplest and most commonly used type of neuron is found in Fully Connected layers, also called MAC layers.

Each neuron computes the weighted sum of its inputs. This computation involves multiplying each input by its corresponding weight and summing up these weighted inputs. Additionally, a bias term is added to the weighted sum to account for the neuron's baseline activation level. Mathematically, the weighted sum z can be expressed as:

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{2.1}$$

Following the computation of the weighted sum, the neuron applies an activation function to introduce non-linearity into its output. Activation functions play a crucial role in enabling neural networks to learn complex patterns and relationships in data. Various activation functions are available, each with its unique properties. Commonly used activation functions include sigmoid, hyperbolic tangent (tanh), rectified linear unit (ReLU), and softmax.

The output of the neuron is determined by the result of the activation function applied to the weighted sum of inputs. This output, often denoted as y, represents the neuron's activation level and serves as input to subsequent neurons or output layers in the network.

$$y = f(z) \tag{2.2}$$

Where $f(\cdot)$ is the activation function.

Figure 2.1 shows an overview of a MAC neuron.

Neurons are organized into layers within a feed-foward neural network, with each layer serving a specific purpose in the information processing pipeline.

- Input Layer: The input layer serves as the entry point for raw data or features into the neural network. Each neuron in the input layer corresponds to a distinct feature or attribute of the input data. The number of neurons in the input layer is determined by the dimensionality of the input data.

- Hidden Layers: Hidden layers are intermediary layers situated between the input and output layers. These layers perform complex transformations on the input data, progressively extracting higher-level features and representations. Deep neural networks may contain multiple hidden layers, enabling them to learn hierarchical representations of the input data.

- Output Layer: The output layer produces the final output of the neural network, which is typically used to make predictions or decisions based on the input data. The structure and function of the output layer depend on the specific task the neural network is designed to solve. For instance, in classification tasks, the output layer may employ a softmax activation function to produce probability distributions over different classes.

In figure 2.2 is shown a classical feed-forward neural network.

The connections between neurons in adjacent layers form the synaptic connections of the neural network. Each connection is associated with a weight, which determines the strength of the connection and influences the neuron's activation. During the training process, these connection weights are adjusted through a process known as backpropagation [14], allowing the network to learn from examples and improve its performance over time.

Training a neural network involves iteratively adjusting the weights and biases of its connections to minimize a predefined loss function. This process aims to optimize the network's parameters to make accurate predictions on unseen data.

While neural networks offer remarkable capabilities in various domains, they often come with significant computational costs, especially for training large models on extensive datasets. The training process typically requires

substantial computational resources, including high-performance GPUs. As a result, there's growing interest in developing efficient algorithms and hardware architectures to reduce the computational burden of neural networks, particularly for deployment on edge devices with limited computational capabilities and power constraints.

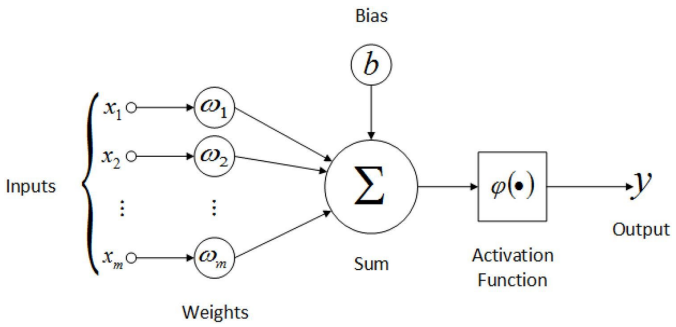


Figure 2.1: MAC neuron: representing the weights, the bias, the summation, and the activation function to produce the output [15].

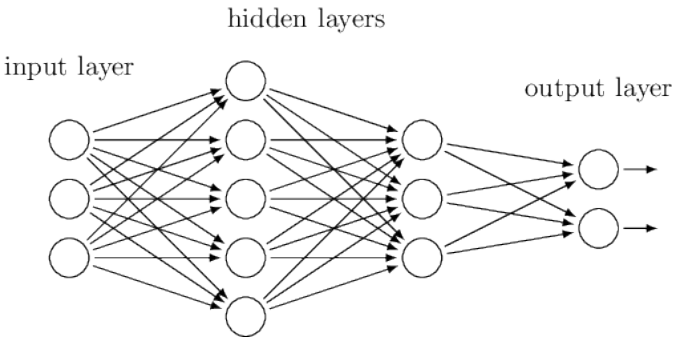


Figure 2.2: Topology of a Neural Network [16].

2.1 Convolutional Neural Networks

Convolutional layers are fundamental building blocks in convolutional neural networks (CNNs), specifically designed to extract and learn spatial hierarchies of features from input data such as images [17].

Convolutional layers were introduced to address the limitations of fully connected layers in processing spatial data efficiently. Traditional neural

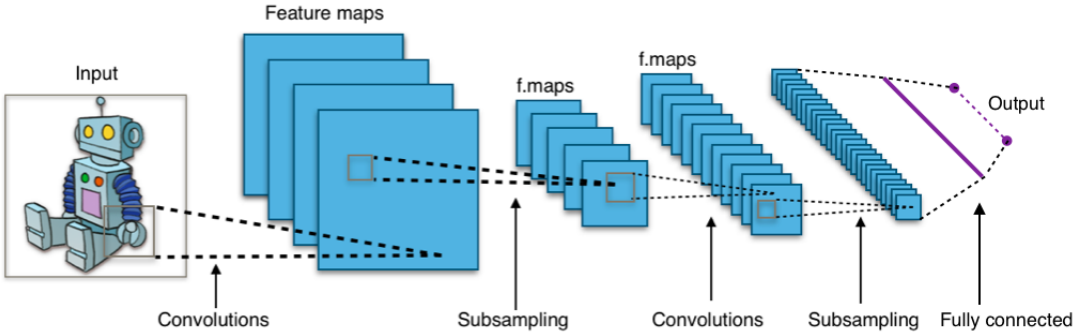


Figure 2.3: CNN composed by convolutions, pooling and a fully-connected layers[18].

networks struggle with images due to the vast number of parameters involved in processing high-dimensional data. Convolutional layers use shared weights and sparse connections to reduce the computational complexity while preserving spatial information.

The operation of a convolutional layer involves applying a series of learnable filters or kernels to the input data through a convolution operation. Each filter is a window that slides over the input data, computing element-wise multiplications and summations to produce feature maps. The convolution operation can be expressed as:

$$Y = X * W + b \tag{2.3}$$

Where Y is the output map, X is the input, W is the filter, b is the bias and $*$ denotes the convolution operator.

This operation is performed at every spatial location of the input data, resulting in a set of feature maps capturing different aspects of the input.

A key aspect of convolutional layers is the concept of shared weights and sparse connectivity. Each filter is shared across the entire input space, enabling the network to learn local patterns that are invariant to translation. Additionally, convolutional layers exhibit sparse connectivity, where each output feature map is only connected to a small local region of the input. This reduces the number of parameters in the network and promotes feature reuse, making convolutional networks more parameter-efficient and robust to overfitting.

In conjunction with convolutional layers, pooling layers are often used to downsample feature maps, reducing their spatial dimensions while preserving

important features. Common pooling operations include max pooling and average pooling, which extract the maximum or average value from local regions of the feature maps, respectively. Pooling layers help to capture the most salient features while discarding redundant information, further enhancing the efficiency and robustness of convolutional networks.

Following the convolution and pooling operations, each feature map typically undergoes a non-linear activation function like the Feed Foward layers, figure 2.3 provides an overview of a simple convolutional neural network.

2.1.1 AlexNet

AlexNet was introduced in [19] and marked a significant breakthrough in the field of computer vision. Its architecture played a pivotal role in the resurgence of interest in deep learning and convolutional neural networks by achieving unprecedented performance on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC).

This achievement underscored the power of deep learning for image recognition and paved the way for the adoption of CNNs in various applications, including image classification, object detection, and segmentation. AlexNet demonstrated the feasibility of training deep neural networks on large-scale datasets and highlighted the importance of model architecture, data augmentation, and regularization techniques in achieving superior performance. In this work, the PyTorch implementation of AlexNet based on [20] was utilized: The initial convolutional layer convolves the 224×224×3 input image using 64 kernels, each sized 11×11×3, with a stride of 4.

Subsequently, the second convolutional layer processes the pooled output from the first convolutional layer, applying 192 kernels sized 5×5×64.

Following a sequential pattern, the third, fourth, and fifth convolutional layers directly connect to one another without any intervening pooling. The third convolutional layer features 384 kernels sized 3×3×256, which are linked to the outputs of the second convolutional layer.

Similarly, the fourth and the fifth convolutional layer comprises 256 kernels sized 3×3×256.

After a pooling stage, two fully connected layers with 4096 neurons each are present, interspersed with a Dropout layer. A fully connected classifier forms the concluding stage of the network's architecture. Overall, it contains more than 60 million learnable parameters stored as float32 occuping approximately 228.88 MB of memory.

The image 2.4 shows the implementation of the network.

The performance achieved by this network had a significant impact, leading to a substantial improvement over the state-of-the-art of those years. Specifically, for the ILSVRC-2010 competition, the best published results were a 45.7% top-1 error rate and a 25.7% top-5 error rate. In contrast, AlexNet achieved 37.5% and 17% respectively. AlexNet also attained first place at the ILSVRC-2012 with a top-5 error rate of 15.3%, surpassing the second-place finisher by 10.8%.

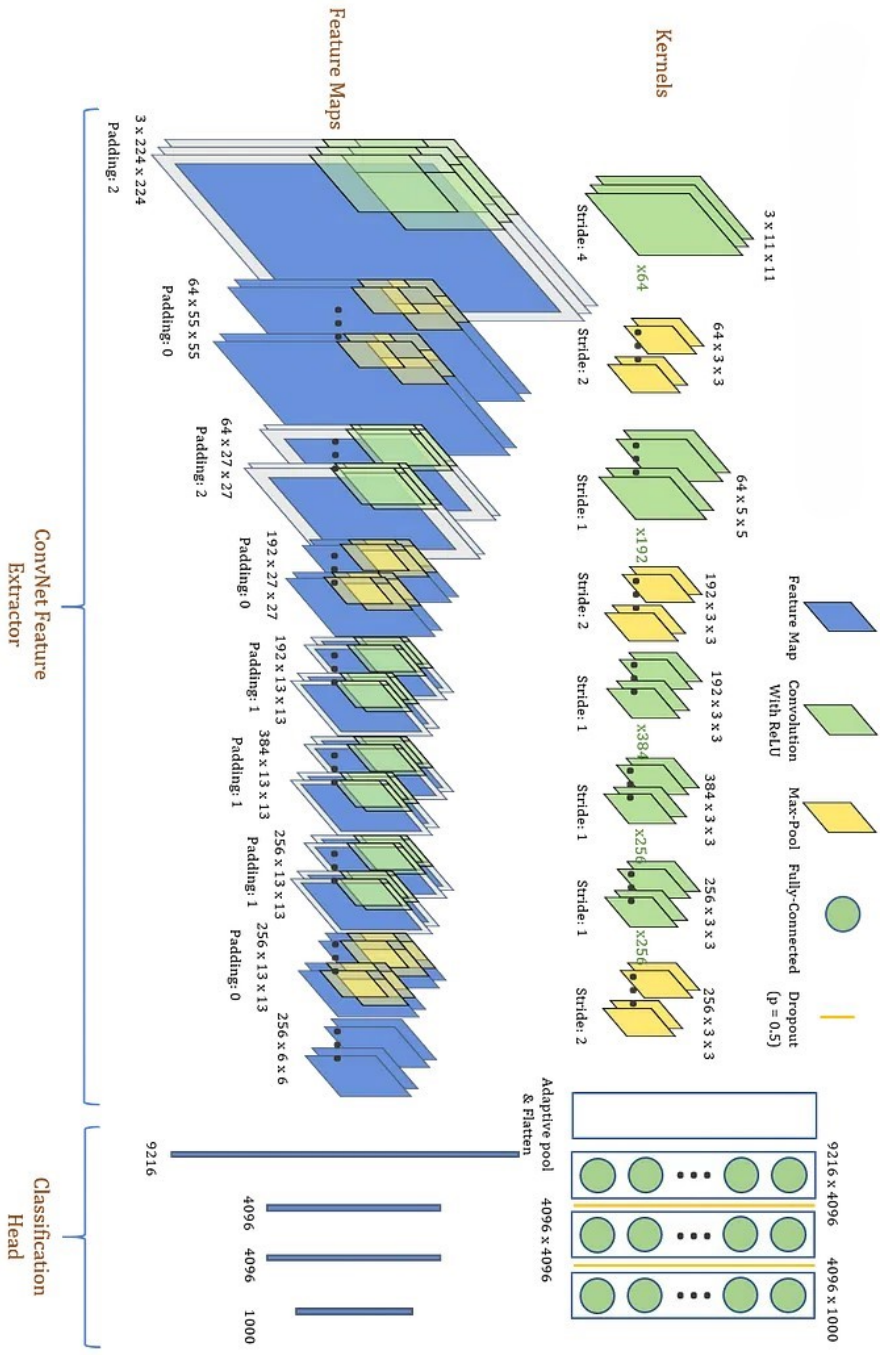


Figure 2.4: The PyTorch's AlexNet implementation[21].

2.2 Transformers

The Transformer architecture, introduced in the groundbreaking paper [22], represents a paradigm shift in sequence modeling. Transformers are a type of neural network architecture designed to handle sequential data by leveraging self-attention mechanisms, which allow the model to weigh the importance of different elements in a sequence, regardless of their position. This contrasts with previous architectures like Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, which process data sequentially and can struggle with long-range dependencies.

Transformers have gained widespread adoption due to their ability to scale effectively with the size of the network. As the number of layers and parameters in the network increases, transformers continue to improve their performance, enabling them to handle larger and more complex datasets with greater accuracy. This scalability has made them the architecture of choice for a variety of tasks across different domains, from language processing and image recognition to time series prediction and beyond.

One of the most notable applications of transformer architecture is in the development of Large Language Models (LLMs) [8] for Natural Language Processing (NLP) tasks. These models, such as GPT [5], has demonstrated exceptional capabilities in generating coherent and contextually relevant text.

More specific this architecture consists of an encoder and a decoder, each comprising multiple layers of self-attention mechanisms and feedforward neural networks.

The self-attention mechanism is central for this architecture, which enables the model to capture global dependencies within input sequences. Given an input sequence $X = \{x_1, x_2, ..., x_n\}$ the self-attention mechanism computes attention weights α_{ij} for each pair of tokens x_i and x_j, indicating the relative relevance.

These attention weights are calculated as a softmax function applied to the scaled dot-product of query, key, and value vectors:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.4}$$

represents the compatibility score between tokens.

The query, key, and value vectors are obtained by projecting the input embeddings into query, key, and value spaces, respectively by FC layers.

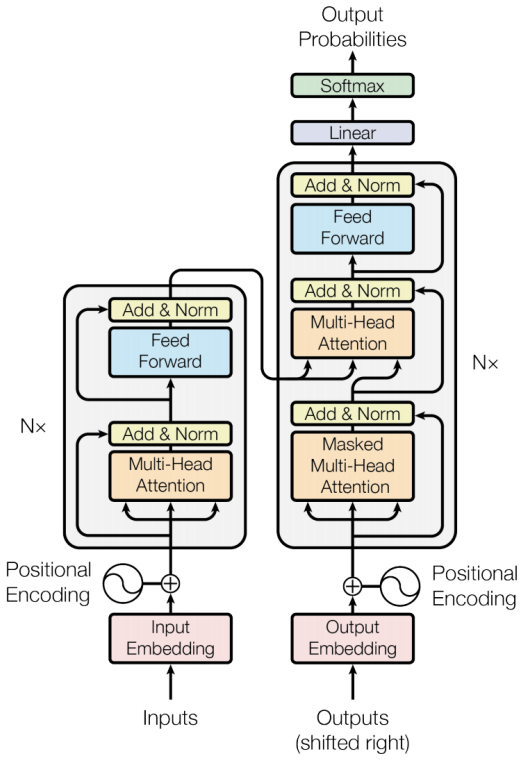


Figure 2.5: Transformer architecture[22].

To incorporate positional information into the model, positional encoding vectors are added to the input token embeddings. These vectors encode the position of each token in the sequence using sine and cosine functions of different frequencies and phases:

$$\begin{aligned} \mathrm{PE}(\mathrm{pos},2i) &= \sin\left(\frac{\mathrm{pos}}{10000^{\frac{2i}{d_{\mathrm{model}}}}}\right) \\ \mathrm{PE}(\mathrm{pos},2i+1) &= \cos\left(\frac{\mathrm{pos}}{10000^{\frac{2i}{d_{\mathrm{model}}}}}\right) \end{aligned} \tag{2.5}$$

where pos rappresents the position and d_{model} is the dimensionality of the input embeddings.

In tasks such as machine translation, the Transformer architecture employs an encoder-decoder architecture, in 2.5 the left side is the encoder and the decoder is on the right. The encoder processes the input sequence and

generates context-aware representations, while the decoder utilizes these representations to generate the output sequence.

2.2.1 ViT

The Vision Transformer (ViT) is a transformative architecture designed to address computer vision tasks by adapting the Transformer model to image processing. Proposed in [23] ViT marks a departure from traditional convolutional neural networks (CNNs) by treating images as sequences of tokens and leveraging self-attention mechanisms for global context aggregation.

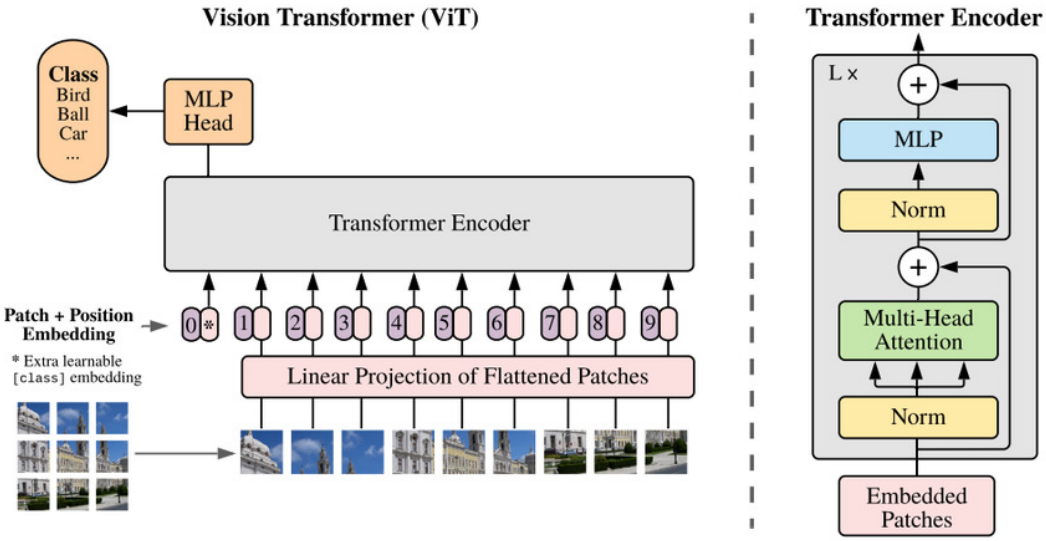


Figure 2.6: The general framework of ViT[23].

ViT begins by dividing the input image into fixed-size patches, typically 16×16 pixels, resulting in a grid of image patches. Each patch is then linearly projected into a lower-dimensional embedding space, yielding a sequence of image tokens that serves as the model's input.

The core of ViT consists of a stack of Transformer encoder layers, mirroring the architecture of the original Transformer model. Each encoder layer incorporates self-attention mechanisms to capture dependencies between image tokens and position-wise feedforward neural networks for feature transformation.

In ViT, the multi-head self-attention mechanism enables the model to attend to different parts of the image simultaneously. Given the sequence

of image tokens, self-attention computes attention weights for each pair of tokens, allowing the model to focus on relevant regions while aggregating global context information.

To encode spatial information into the model, positional embeddings are added to the token embeddings. These embeddings convey the spatial relationships between image patches and enable the model to differentiate between tokens based on their positions within the image grid.

Following the Transformer encoder, ViT utilizes a classification head to predict the class labels or perform other downstream tasks. Typically, the output token representation of the [CLS] token, which summarizes the entire image, is fed into the classification head for task-specific processing.

In the original paper three models were proposed, in this work the ViT-Base model is used composed by 12 Layers, an Hidden Size D equal to 768, an MLP size of 3072 and 12 Heads; consisting in total of 86 milion parameters, equivalent to 320.31 MB if saved as float32.

The results of ViT depend on the implementation used; the larger the model, the better it performs. For example, taking the two extreme cases, ViT-Base (86M parameters) and ViT-Huge (632M parameters), achieved a top-1 accuracy of 84% and 88.55% on ImageNet, respectively.

Chapter 3

Pruning

Neural network pruning stands as a pivotal technique in optimizing trained neural networks by strategically reducing parameters or computational resources while upholding predictive accuracy. It involves the systematic identification and elimination of less influential components, such as certain weights or neurons, thereby streamlining the network's complexity and size.

This method finds particular relevance in scenarios where large, over-parameterized models exhibit effectiveness but face computational constraints or limitations in deployment due to their size. By selectively discarding components that contribute minimally to output predictions, pruning facilitates the creation of swifter models that retain high accuracy levels.

In essence, pruning offers a pathway to enhance the efficiency and scalability of neural networks, enabling them to be deployed more effectively across various applications and platforms. Through the judicious removal of redundant or less impactful components, neural networks can achieve improved performance metrics, such as reduced memory footprint, faster inference times, and enhanced energy efficiency.

Neural network pruning encompasses diverse techniques, including unstructured pruning and structured pruning. Unstructured pruning entails the selective removal of individual connections within the network by setting corresponding weights to zero, yielding sparse weight matrices. Structured pruning involves the removal of entire neurons along with their connections, resulting in a more compact and simplified network architecture.

The pruning process typically unfolds through a structured pipeline, which can span multiple stages to achieve optimal results. Initially, a baseline neural network model is trained on a relevant dataset to establish a performance

benchmark. Subsequently, the less influential parameters or connections are identified and pruned away.

Following, the pruned model undergoes fine-tuning or retraining to restore or enhance its performance on the given task. This fine-tuning stage aims to compensate for any performance degradation resulting from the removal of parameters or connections during pruning.

Moreover, the pruning process may incorporate multiple stages of pruning and fine-tuning, each refining the network's architecture further. Through iterative cycles of pruning and fine-tuning, the network's architecture evolves to strike an optimal balance between model size reduction and preserved performance.

3.1 Unstructured Pruning

Unstructured pruning offers the benefit of achieving a very high compression rate, as it can remove individual weights without considering their arrangement within the network. In contrast, structured pruning, due to stricter constraints, to maintain high accuracy, achieves a lesser reduction in network size. However, structured pruning has the advantage of requiring fewer modifications to the original network structure, facilitating easier and more efficient implementation. This is not the case with unstructured pruning, as it leads to sparse weight matrices that necessitate more complex hardware implementation and, currently, exhibit lower efficiency compared to dense matrices.

In essence, both pruning methods aim for high efficiency but through different approaches: unstructured pruning excels in compression efficiency but requires sacrificing hardware optimization, whereas structured pruning offers easier deployment and optimization at the cost of lower compression rates.

We can describe unstructured pruning as a minimization problem as follows:

$$\begin{aligned} &\min_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}; \mathcal{D}) = \min_{\boldsymbol{w}} \frac{1}{N} \sum_{i=1}^{N} l(\boldsymbol{w}; (\boldsymbol{x}_i, y_i)) \\ &\text{s.t. } \|\boldsymbol{w}\|_0 \leq k. \end{aligned} \tag{3.1}$$

Where $\mathcal{D}$ id the dataset, $\mathcal{L}$ is the loss function on the entire dataset, N is the number of samples of the dataset, $\boldsymbol{x}_i$ is the i-th sample assosciated with

is label y_i, l is the loss function on a single sample, $\|\cdot\|_0$ denotes the zero norm, so how many weights are non-zero, $\boldsymbol{w}$ are the weights to prune and k is how many weights we want to maintain.

It can also be viewed as applying a binary mask $\boldsymbol{m}$ to the original weights:

$$\begin{aligned} &\min_{\boldsymbol{w'}} \mathcal{L}(\boldsymbol{w'}; \mathcal{D}) \\ &\text{s.t. } \boldsymbol{w'} = \boldsymbol{w} \odot \boldsymbol{m} \\ &\quad \|\boldsymbol{m}\|_0 \leq k \end{aligned} \tag{3.2}$$

Where $\odot$ is the Hadamard product and $\boldsymbol{w'}$ are the new weights.

In the literature, numerous methods for unstructured pruning have been proposed [9] [10]. Particularly, this work will only analyze some of them.

In general, each method seeks to associate a score with each weight, representing its importance. During the pruning phase, weights with lower scores are removed first. Additionally, pruning can occur at various granularities, specifically, after assigning a score to each weight, one can choose to eliminate the lower scores by separately analyzing each layer (Local Pruning) or by directly applying it to all the layers intended for pruning (Global Pruning).

Below, I will present and explain several pruning methods, starting with the simplest one, Random Pruning, progressing to those used in this work, Magnitude and Gradient Pruning, and culminating with a more intricate technique that shares similarities with the MAM layer, that will be later analyzed, namely Dynamic Sparse Training.

3.1.1 Random Pruning

Random pruning is perhaps the simplest method employed in neural network pruning. As the name suggests, it involves randomly selecting weights or neurons to prune from the network. While seemingly elementary, random pruning serves as a fundamental baseline for evaluating the efficacy of more sophisticated pruning algorithms [24]. This comparison helps assess whether the additional complexity of sophisticated techniques significantly improves model efficiency and performance, otherwise, if advanced pruning methods are unnecessary, it implies that all the weights to prune already contribute little to the network's accuracy.

3.1.2 Magnitude Based Pruning

These methods prioritize the removal of weights with smaller magnitudes, under the assumption that they contribute less to the network's overall performance. One common approach is to rank the weights based on their absolute values, considering larger weights as more significant and smaller weights as less significant [25]. The corresponding score can be expressed as:

$$g(\iota) = |w_\iota| \tag{3.3}$$

Magnitude-based scoring methods are straightforward and computationally efficient. However, they may not always capture the true importance of weights, especially in cases where small weights are still influential in the network's performance. Additionally, they do not consider the interactions between weights or their context within the network architecture.

3.1.3 Gradient Based Pruning

Another class of pruning methods aims to identify the importance of weights based on the change obtained when they are eliminated. One approach is through the approximation of the change in loss following a perturbation for weights $\boldsymbol{w}$ via the first-order Taylor expansion as follow:

$$\Delta\mathcal{L} = \nabla_{\boldsymbol{w}}\mathcal{L}\Delta\boldsymbol{w} + O(\|\delta\boldsymbol{w}\|^2) \tag{3.4}$$

And so it's possible to express the scores as:

$$g(\iota) = \mathbf{E}\left[\left|\frac{\partial\mathcal{L}(\boldsymbol{y}, DNN(\boldsymbol{x}, \boldsymbol{w}))}{\partial w_\iota} w_\iota\right|\right] \tag{3.5}$$

This method, proposed in [26], allows for the identification of the importance of weights based on dataset information, enabling a more accurate evaluation. However, it requires both a dedicated pruning dataset with a distribution similar to that of the training and test sets, as well as increased computational demand, as gradients need to be computed for each weight and input.

3.1.4 Dynamic Sparse Training

Dynamic sparse training is a technique aimed at enhancing the pruning by dynamically adjusting the sparsity of network parameters during the training

process. Unlike the previous pruning techniques that remove connections or parameters after the training, dynamic sparse training adapts the sparsity pattern throughout training based on various criteria.

In particular, the work done in [27] proposes an adaptive threshold that is trained during the network's training process.

It can be formulated as follows:

$$\boldsymbol{M}_{i,j} = S(|\boldsymbol{W}_{i,j}| - \boldsymbol{t}_i) \text{ with } 1 \leq i \leq M, 1 \leq j \leq N \tag{3.6}$$

Where $\boldsymbol{W} \in \mathbb{R}^{M \times N}$ is the weight matrix, $\boldsymbol{t} \in \mathbb{R}^{M}$ is the threshold vector, $S(\cdot)$ is the Step Function and $\boldsymbol{M} \in \mathbb{R}^{M \times N}$ is the mask.

During the inference the masked weight matrix $\boldsymbol{M} \odot \boldsymbol{W}$ is used, therefore, a threshold is added to the weights that should be used. However, since the Step Function has an impulse function as its derivative during the BackPropagation phase, an estimator of the derivative is introduced that behaves similarly to the original derivative but is suitable for gradient propagation:

$$\frac{d}{dx}S(x) \approx H(x) = \begin{cases} 2 - 4|x|, & -0.4 \leq x \leq 0.4 \\ 0.4, & 0.4 < |x| \leq 1 \\ 0, & \text{otherwise} \end{cases} \tag{3.7}$$

Additionally, during the training phase, a regularization term is added to encourage the thresholds not to be too small while simultaneously preventing them from becoming too large:

$$R = \sum_{i=1}^{M} e^{-\boldsymbol{t}_i} \tag{3.8}$$

It is also worth noting that the unused weights are not zeroed out during the process, thus preserving the learned information.

This method yields excellent results; some of the most notable examples include training WideResNet-16-8 on CIFAR-10 achieving an accuracy of 94.73% with 4.64% remaining weights, and training ResNet-50 on ImageNet achieving 72.78% top-1 accuracy with 9.87% remaining weights.

3.2 Lottery Ticket Hypothesis

The Lottery Ticket Hypothesis [28] represents a groundbreaking concept in the realm of deep learning, offering profound insights into the structure and

optimization of neural network architectures. This hypothesis challenges conventional knowledge regarding the necessity of large, over-parameterized networks by suggesting that within these complex structures lie sparse, trainable subnetworks capable of achieving comparable performance with significantly fewer parameters.

At the core of the LTH lies the premise that during the training process of deep neural networks, there exist subnetworks, or "winning tickets" that, when initialized appropriately, can achieve high accuracy with a sparse set of weights. This notion challenges the prevailing belief that the huge number of parameters in large networks is essential for achieving state-of-the-art performance. Instead, the hypothesis says that only a small subset of connections is critical for successful learning and generalization.

The LTH draws inspiration from the concept of network sparsity, which suggests that neural networks can function effectively with a reduced number of connections. By identifying and preserving these crucial connections, one can potentially train more compact models without sacrificing performance.

Empirical studies have provided evidence in support of the Lottery Ticket Hypothesis across various deep learning architectures and tasks. Researchers have demonstrated that by iteratively pruning connections based on their importance, sparse subnetworks emerge, which exhibit comparable or even superior performance to the original dense networks. These winning tickets often converge faster during training and generalize better to unseen data, indicating that they capture essential patterns and relationships within the data more efficiently.

For example, some of the most notable results include ResNet-18 trained on CIFAR-10 achieving an accuracy of 89.5% when only the subnetwork was trained (the entire network achieved 90.5% accuracy), with 21.9% of weights remaining.

Furthermore, experiments have shown that these sparse subnetworks are not merely artifacts of the training process but rather inherent properties of the initial network architecture. Even when reinitialized with different random seeds, the same winning tickets can be identified, suggesting robustness and reproducibility across different training runs.

The significance of this study, unlike those previously described concerning pruning, lies not in finding an effective methodology for pruning a network, but rather in empirically demonstrating that only a small fraction of weights is crucial for achieving good performance. Identifying this subset allows for initializing the entire network by pruning it beforehand, resulting in a

remarkable increase in training speed due to significantly fewer parameters. While it has limited practical impact for networks already trained and pruned, which would achieve comparable accuracy and pruning rates, it has sparked extensive research interest: if one can identify this winning ticket before training, it is possible to train only this subnetwork, conserving substantial resources. This approach, known as Pruning at Initialization, is currently a focal area of interest among researchers specializing in neural network optimization [29].

Chapter 4

Multiply and Max/Min Layer

Traditional pruning techniques in Deep Neural Networks (DNNs) involve selecting specific connections or entire neurons without altering the network's fundamental structure, which relies on the Multiply-and-Accumulate (MAC) paradigm. This can be seen as a map-reduce approach, in which the mapping function is represented by multiplication and the reducing function by accumulation, namely summation.

It has been observed that the summation operation in MAC neurons can be considered a specific instance of a more general aggregation:

$$o = \left(\sum_{i=1}^{n} v_i^p \right)^{\frac{1}{p}} \tag{4.1}$$

where the resulting output is determined by a parameter, p, controlling the influence of individual values. This realization has prompted the consideration of replacing the traditional sum-based , $p = 1$, reduce operation with a max/min-based one, $p \to +\infty / p \to -\infty$.

It is worth noting that in magnitude-based pruning, only weights with the largest absolute values are selected. This process aligns with the concept being pursued here, further emphasizing the direction of this approach.

This innovation has led to the development of a new neuron model based on the Multiply-And-Max/min (MAM) paradigm [11] [12]. Unlike traditional MAC neurons, MAM neurons explicitly identify which inputs contribute to the output, thus distinguishing themselves from previous methods. Most

importantly, this represents a novel approach to non-structured pruning, as it renders individual neuron behavior less sensitive to pruning while remaining hardware-friendly.

Although the functional equivalence between original and alternative neurons cannot be guaranteed on a neuron-by-neuron basis beforehand, empirical evidence supports the idea that substituting MAC-based neurons with MAM-based ones across the entire network preserves the overall behavior of the DNN.

The MAM paradigm has been found to naturally lend itself to pruning, as neurons consistently select a few connections during maximum and minimum reduction operations. Moreover, most existing pruning algorithms applicable to MAC-based layers can be adapted to MAM-based ones with minimal modifications, thereby enhancing performance without introducing significant computational overhead.

4.1 MAM description

To enter into the specifics of the MAM layer, it's better to first examine separately how the map and reduce phases behave. During the mapping phase, the weighted input matrix $\boldsymbol{V} \in \mathbb{R}^{M \times N}$ is defined as follow:

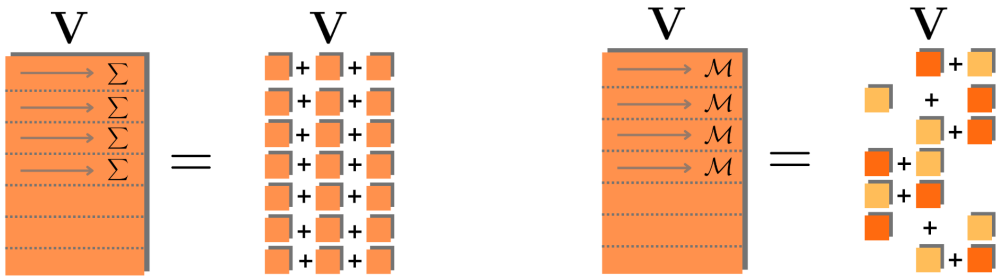
$$\boldsymbol{V} = \begin{bmatrix} \boldsymbol{w_1} \odot \boldsymbol{x}^T \\ \boldsymbol{w_2} \odot \boldsymbol{x}^T \\ \vdots \\ \boldsymbol{w_M} \odot \boldsymbol{x}^T \end{bmatrix} \tag{4.2}$$

Where $\boldsymbol{x} \in \mathbb{R}^N$ is the input vector, $\boldsymbol{w_i}$ is the i-th row of the weights matrix $\boldsymbol{W} \in \mathbb{R}^{M \times N}$ and $\odot$ is the Hadamard product. It also can be view more explicitally as:

$$v_{i,j} = w_{i,j} x_j \text{ with } i \in \{1, .., M\}, j \in \{1, .., N\} \tag{4.3}$$

This map operation is shared by both the MAM layer and the MAC layer. In the MAC layer, the reduce operation is the sum of the elements row by row of the matrix $\boldsymbol{V}$:

$$z_i = \sum_{j=1}^{N} v_{i,j} + b_i \text{ with } i \in \{1, .., M\} \tag{4.4}$$



(a) **(b)**

Figure 4.1: A comparison between the two reduction phases:the sum operation indiscriminately retains all values of $\boldsymbol{V}$ (a), whereas the Max/min operation selectively chooses only two weights per row (b).

To observe the reduce operation in the MAM layer, it is necessary to first introduce the operator $\mathcal{M}$ as follows:

$$\mathcal{M}_{j=1}^{N} v_{i,j} \triangleq \max_{j\in\{1,..,N\}} v_{i,j} + \min_{j\in\{1,..,N\}} v_{i,j} \tag{4.5}$$

which selects the maximum and minimum of each row of the weighted input matrix. The reduce operation so can be defined as:

$$z_i = \mathcal{M}_{j=1}^{N} v_{i,j} + b_i \text{ with } i \in \{1, .., M\} \tag{4.6}$$

There are several important points to note. First, this operation does not perform pruning but rather redefines how the reduce phase should be carried out. In fact, the unselected weights are not zeroed out but simply do not contribute to interference. Furthermore, each time a new input is passed through the neural network, the selected maximum and minimum values may differ.

The second point to note is that empirically, it is often observed that the selected maximum is associated with a positive weight value while the minimum is associated with a negative one and this enhances the neuron's stability. This can be explained by the fact that the ReLU activation function is commonly used in architectures, rendering the activation map non-negative.

Lastly, it is useful to observe that through the comparison between formulas (4.4) and (4.6), it is evident that the MAM layer serves as an approximation of the MAC layer, attempting to utilize the most influential information.

4.1.1 Probability of Selection Pruning

The MAM layer introduces a pruning method based on the probability of a weight being selected, specifically, considering formula (4.6), only two values are chosen for each input, reflecting the use of only two weights from the weight matrix $\boldsymbol{W}$, hence, an empirical probability can be associated with each weight based on how often it is selected out of the total number of input passes. Therefore, it is necessary to introduce, as in (3.1.3), a pruning dataset to compute this score. This can be more rigorously defined as follows:

$$g(\iota) = \frac{\sum_{k=1}^{N} \mathbb{I}_{w_\iota}^k}{N} \tag{4.7}$$

Where N is the cardinality of the pruning dataset, and $\mathbb{I}_{w_\iota}^k$ is equal to 1 if w_ι is selected, regardless of whether it is the maximum or minimum, for sample k, and 0 otherwise.

It is noteworthy that the gradient score is intrinsically linked to this score; indeed, when a weight is never selected, it will have a score of 0 for both.

4.2 Backpropagation and vanishing contributions method

During the training of the neural network, directly employing the formulation in (4.6) yields unsatisfactory results. To comprehend the reason behind this, we must analyze the gradient of (4.6):

$$\frac{\partial z_i}{\partial v_{i,j}} = \delta_{i,j} \text{ with } \delta_{i,j} = \begin{cases} 1 & \text{if } v_{i,j} = \max_{j \in \{1,..,N\}} v_{i,j} \\ 1 & \text{if } v_{i,j} = \min_{j \in \{1,..,N\}} v_{i,j} \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

From empirical findings, we are aware of certain properties of the MAM layer, specifically, only a subset of weights from the matrix is effectively utilized, this is also the reason why the MAM layer exhibits good pruning capabilities. By employing formula (4.8) and leveraging insights from the lottery ticket hypothesis, in particular the experimental evidence suggests that the selection of interconnections is crucial for achieving good performance, we can infer that directly utilizing the MAM layer may lead to premature convergence of crucial interconnections within the network. This occurs

because weights with already high absolute values are updated from the outset, inhibiting the network's thorough exploration of solution space.

For these reasons, a gradual transition between the MAC layer and the MAM layer has been proposed. This can be achieved through an affine combination of formulas (4.4) and (4.6):

$$z_i = \beta \sum_{j=1}^{N} v_{i,j} + (1-\beta)\mathcal{M}_{j=1}^{N} v_{i,j} + b_i \text{ with } i \in \{1,..,M\} \text{ and } \beta \in [0,1] \quad (4.9)$$

The formula (4.9) can be seen as a bridge between the MAC and MAM layers: when $\beta = 1$, it is equivalent to (4.4), while when $\beta = 0$, it is equivalent to (4.6). To perform this gradual transition between the two layers, β is initialized to 1 at the beginning of training then decremented until it reaches 0. From that point onwards, for the remainder of the training, it remains at 0. Experimental results show that immediately after the transition from MAC to MAM, i.e β is 0, there is a drop in performance. For this reason, it is necessary to continue training the network for several epochs after this happens.

This strategy allows a more effective exploration and leading to significantly improved performance. Beta can be decremented according to various strategies, which however seem to lead to the same final performance, so a linear decay is usually employed. However, in this work, for specific applications, a parabolic decay has also been utilized [1].

[1]Further details of linear and parabolic decay can be found in Appendix A

Chapter 5

MAM as a pruning tool for MAC

The MAM layer modifies the architecture of the neural network, introducing computational overhead compared to the traditional MAC layer. Changing the layer's structure increases the number of FLOPs per weight from 2 (as needed by a MAC layer) to 3. This is already a disadvantage, but it's also important to note that MAC layers benefit from widely optimized and common hardware architectures, ensuring high efficiency. MAM layers, on the other hand, do not have such optimized hardware support. Additionally, creating optimization specifically for MAM layers would not only require special implementation in the hardware architecture but also involve two branch instructions (for maximum and minimum), which disrupt the normal pipeline structure and slow down the process further.

To address these issues, a key contribution of my work was to tackle the problem at its core. My hypothesis was that the importance of MAM lay solely in its ability to identify which connections are important. In previous work, it was believed that for MAM to function effectively, it needed to maintain its structure of selecting maximum and minimum values. It was thought that this competition, which generates different subnetworks for each input, was the reason for the good performance of this layer.

My hypothesis was based on the experimental evidence that the MAM layer retains very few active connections after the pruning phase. This may suggest that non-linearities (max and min) do not primarily account for the layer's success, but rather serve as a necessary mechanism to ensure that the network focuses only on a few connections.

Based on this hypothesis, I proposed that after pruning a MAM layer, the network can be reverted by reintroducing MAC layers while retaining the same sparse matrices produced by pruning the MAM layer.

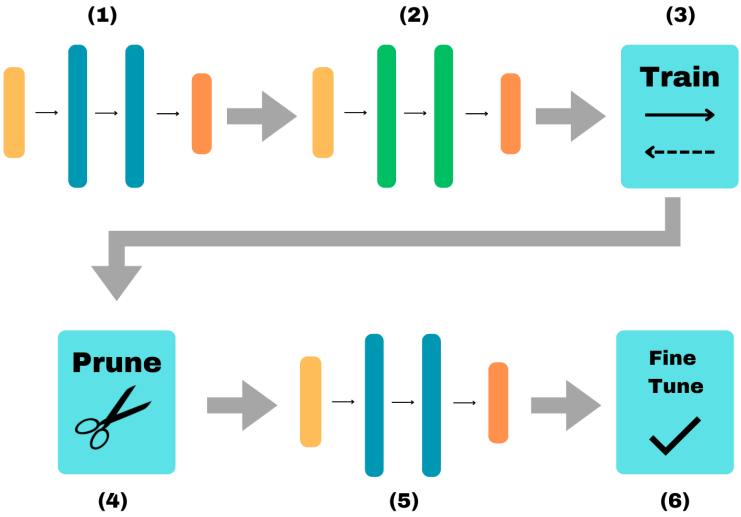


Figure 5.1: Pipeline for pruning MAC layers: (1) DNN with MAC layers (2) Replace MAC with MAM (3) Train the network (4) Prune the DNN (5) Reinsert MAC layers (6) Fine-tune the network.

Following Figure 5.1, the approach used can be analyzed: first, the MAM layers are inserted in place of the MAC layers that are to be pruned (steps 1 and 2). Then, the network is trained until reaching convergence as described in 4.2 and pruned (steps 3 and 4). At this point, the previously replaced MAC layers are reintroduced, retaining the weight matrices of the MAM layers, and the network is fine-tuned (steps 5 and 6).

This process enables the creation of a training pipeline that ends with the starting network's structure, but with deeply pruned layers. Thus allows the use of well-known, highly optimized hardware architectures.

5.1 Pruning Before Convergence

In recent years, the development and deployment of large neural network models have posed significant challenges related to resource requirements. These models, for exemple Large Language Models like GPT-4, require vast

amounts of computational power, memory, and storage. The training process itself involves massive datasets and long durations leading to high energy consumption.

To address this problem, one technique used is pruning the network before it converges [30]. This allows for the removal of unimportant weights during an intermediate training phase, thereby saving computational resources on the server side. However, this process has its own challenges. Hardware accelerators used for training neural networks, such as GPUs and TPUs, heavily rely on the continuity of weight matrices in memory to achieve high efficiency. A sparse weight matrix does not have this property, so, as of now, the gains obtained are not proportional to the compression rate, therefore, to achieve computational gains, the pruning must be extremely high.

In this work, being aware of the high prunability of the MAM layer, I adopted this strategy. Specifically, I investigated how long the network needs to be trained after setting β to zero in (4.9) to determine at what point the crucial interconnections are identified. However, the savings from this strategic anticipation of the pruning phase are assessed only in terms of theoretical FLOPs saved, and an actual sparse training is not implemented due to the difficulties mentioned above.

Additionally, as mentioned earlier, it is crucial to use hardware accelerators like GPUs for these models, which require specially programmed kernels to achieve high efficiency. Despite the existence of a kernel specifically designed for MAM optimization, its performance is still about three times worse than that of a MAC kernel. Building on previous concepts, this work investigates further optimization of the training process as follows: before the network converges, it is pruned. Then, the MAC layer is directly reinserted, allowing for optimization through both the more efficient kernel and potentially, as before, sparse training.

5.2 Dynamic Sparse Training Comparison

Despite the MAM layer having a formulation 4.6 that may appear quite different from that of dynamic sparse training 3.6 at first, once it is noted that after pruning, the MAM layer can revert to a MAC layer, certain parallels can be observed. This layer can be viewed as a particular case of DST, with two substantial differences: firstly, instead of the weight matrix $\boldsymbol{W}$, the weighted input matrix $\boldsymbol{V}$ is used; secondly, rather than having a threshold

based on the minimum value a weight should have, there is a threshold based on how many are selected, namely 2 (i.e., the largest and smallest values).

From this perspective, it is also noteworthy that the option of choosing only two values is due to the use of the matrix $\boldsymbol{V}$ (moreover, maintaining more than two values in this setting would not be feasible, as it would involve excessively high computational and memory cost). If this were not the case and the weight matrix were used instead, the same interconnections would always be selected, leading to significantly inferior performance.

Similarities can also be observed during the initial training phase; indeed, in both methods, one starts with a regular MAC layer, which is then transformed into a sparse structure. However, regarding DST, this occurs gradually by eliminating some interconnections, whereas with MAM, it happens through a gradual reduction in the contribution of the unselected interconnections.

Chapter 6

Experiments

In this section, detailed descriptions of all datasets, architectures, hyperparameters and all the information needed to ensure the replicability of the experiments are provided. The investigation begins by replicating the findings from [12] on the efficacy of pruning the MAM layer. It then explores the feasibility of using MAM to prune a MAC layer. Lastly, the study involves pruning the MAM layer before the network converges, examining scenarios where the MAM layer remains and where the MAC layer is reintroduced post-pruning.

6.1 Dataset

Two different computer vision datasets have been used for this work: CIFAR-10 [31] and ImageNet-1K [32].

1. **CIFAR-10:** is a widely used benchmark dataset in the field of computer vision. It consists of 60,000 color images, each belonging to one of ten classes, including common objects such as airplanes, cars, birds, cats, and dogs. The dataset is divided into 50,000 training images and 10,000 test images, with each image having a size of 32x32 pixels. CIFAR-10 serves as a standard dataset for training and evaluating image classification algorithms and models. Consistent data augmentation is applied, including image rotation, random cropping, adjustments in brightness, contrast, and saturation, horizontal flipping, and normalization.
2. **Imagenet-1K:** is a very popular dataset, consisting of over 1 281 167 high-resolution images across 1,000 object categories. The validation

and test set have 50000 images. These categories cover a wide range of objects, animals, scenes, and more, making ImageNet-1K one of the largest and most diverse datasets available for object recognition and classification tasks. Data augmentation is performed by random horizontal flip, random saturation and random change in contrast and brightness.

Where the pruning set is required, the validation set is utilized, and evaluation is conducted on the test set.

6.2 MAM training and pruning methodology

The initial experiments conducted concern the prunability of the MAM layer. For each network, MAM is inserted in place of certain MAC layers present in the original network. Subsequently, the training phase occurs, when the network converges, its performance is evaluated, followed by a one-shot pruning (that is a single pruning phase rather than iteratively) on the MAM layers using methodologies outlined in 3.1.2, 3.1.3 and 4.1.1. The network's performance is then assessed after weight removal.

Specifically, the networks are trained as follows:

1. **AlexNet:** The MAM layers were inserted in place of the last two fully connected layers before the classifier, where 54.5 million weights are present. The network was trained on CIFAR-10 starting from a model pre-trained on ImageNet-1k. A batch size of 256 was used, along with Adam with an initial learning rate of 0.0001, trained for a total of 70 epochs with the initial 35 epochs of vanishing contribution.

2. **ViT:** In this work a pretrained MAM-model is used, trained in this way [12]: the MAM layers replace the fully connected layers present in the intermediate and output states of the encoders, containing 28.2 million total interconnections. During the training on ImageNet Adam was used with a learning rate of 0.0005, a batch size of 128 and trained for 50 epochs with 12 epochs of vanishing contribution. A model pre-trained on Imagenet-21k[32] was used.

Pruning is then performed at different granularities with the aim of not dropping below 3% of the network's initial accuracy.

6.3 MAM to prune MAC

The approach to assess the feasibility of pruning a MAC layer using MAM employs the following pipeline:

- Train the model with MAM until convergence and prune it as described in 6.2.
- Revert the model to the original network with fully connected layers, while retaining the same pruned weight matrices obtained with MAM.
- Fine-tuning the network with the same parameters used previously for training.

The fine-tuning is done for 30 epochs for AlexNet CIFAR-10 and only one epoch for ViT ImageNet-1K (due to computational resources).

The next experiment, instead, concerns the prunability of the MAM layer before convergence, specifically to observe if it maintains previous performance levels and, if so, at what epoch crucial interconnections are learned. The experiment follows these specified phases:

- Train the model with MAM for a few epochs beyond those of vanishing gradient.
- Prune the network with the same strategy and threshold that obtained the best results in the first experiment.
- Fine-tuning the network with the same parameters used previously for training.

This experiment exclusively focuses on AlexNet trained on CIFAR-10, using 1, 3, 5, 10, and 15 epochs after the gradient vanishing contribution. The network is trained using both linear and parabolic β decay.

Finally, the experiment is repeated, with a small adjustment: after pruning the network, the MAC layer is reintroduced exploiting the sparse matrices achieved by MAM.

Chapter 7

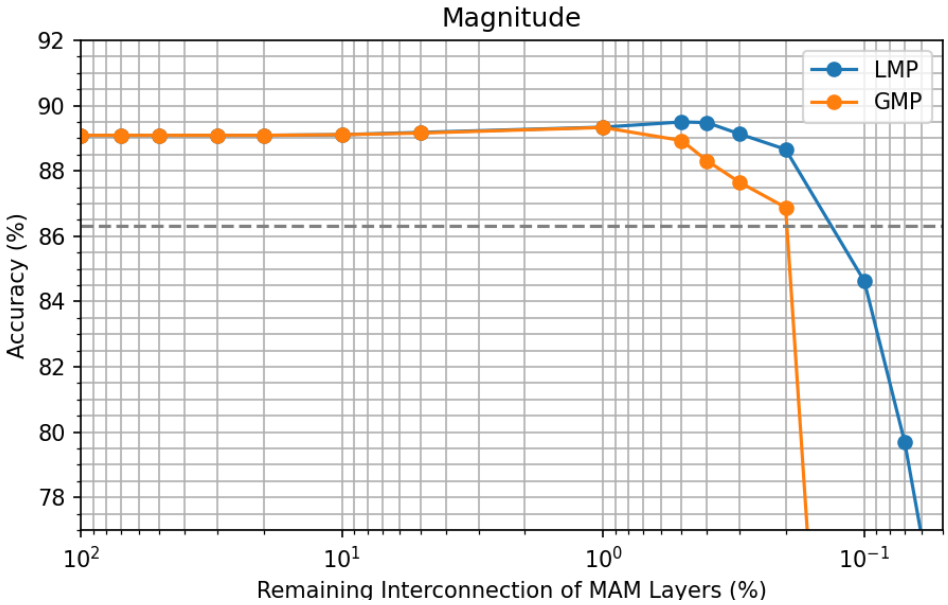
Results

7.1 Pruning Performance

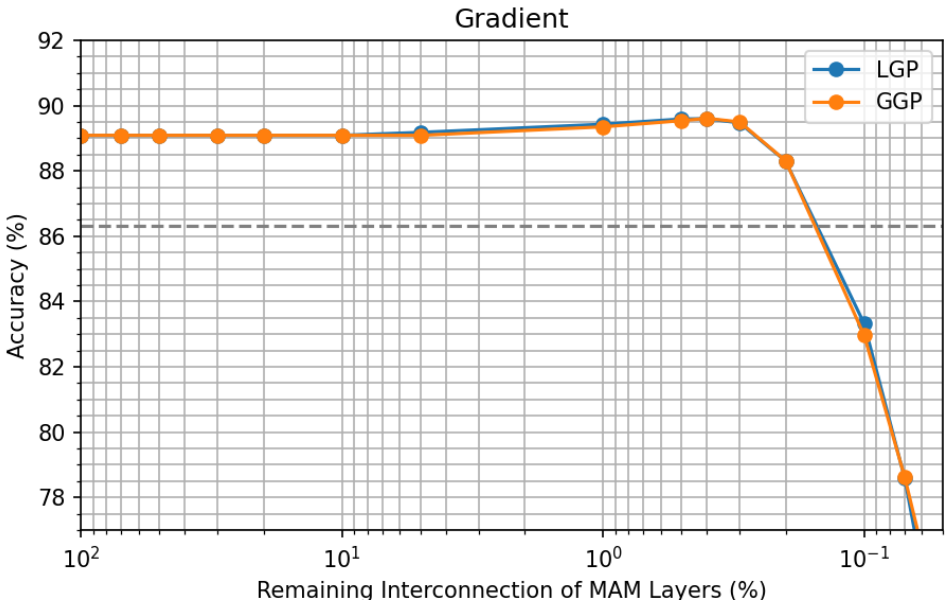
This section is dedicated to demonstrating the prunability of the MAM layer, employing various pruning strategies and thresholds to identify the most efficient approach. The goal is to replicate results comparable to those achieved in [12]. Explicit comparison with the accuracy and prunability of the MAC layer is not provided, as the findings from the previous works already extensively confirm that MAM achieves comparable accuracy and much greater prunability.

In Figure 7.1, the results of pruning AlexNet on CIFAR-10 using LMP, GMP, LGP, GGP, LPSP, GPSP are reported.

All the proposed methods exhibit highly performance of pruning, managing to prune about 99.8% of the present interconnections, thus demonstrating the remarkable effectiveness of this approach. In particular, gradient pruning, both local and global, as well as local magnitude pruning, appear to be the most promising methods. An important point to note is the performance of the probability of selection pruning. It maintains excellent results up to a certain point, after which there is an immediate and drastic drop. One explanation for this phenomenon could be that initially, weights that are never selected, and thus have no impact on performance, are pruned. Subsequently, weights that are rarely activated are pruned, sometimes leading to a slight improvement, likely because it enhances the network's generalization capabilities. The performance drop occurs when the remaining weights are all frequently activated, resulting in very similar scores. At this stage, the importance assigned to each weight is approximately equal and not very

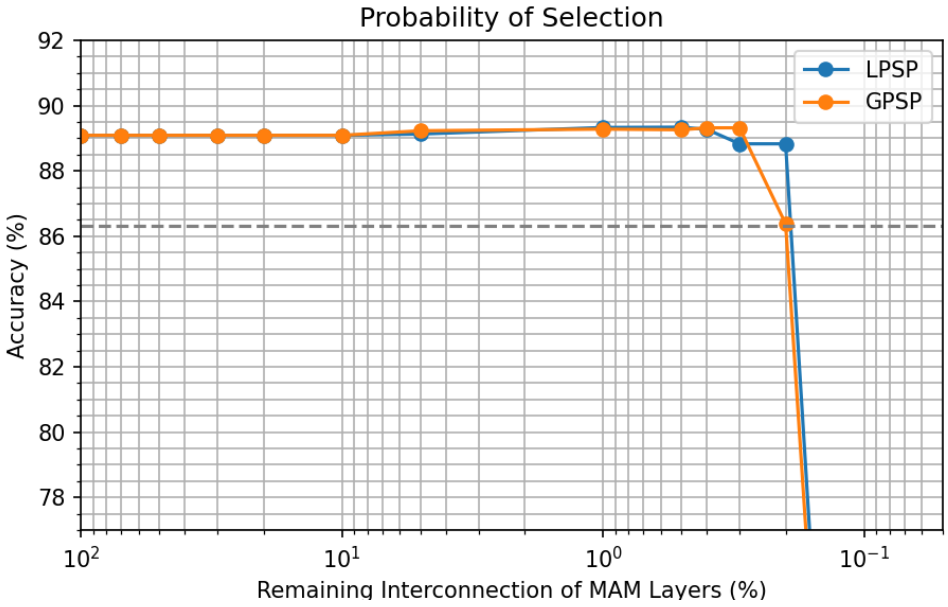


(a) LMP - GMP



(b) LGP - GGP

Figure 7.1: AlexNet CIFAR-10 Pruning

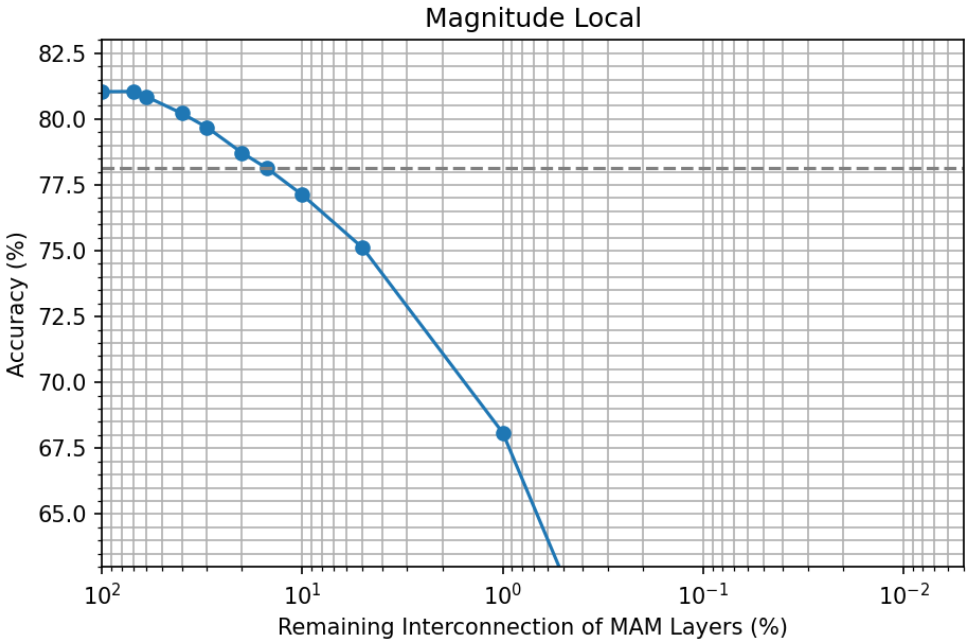


(c) LPSP - GPSP

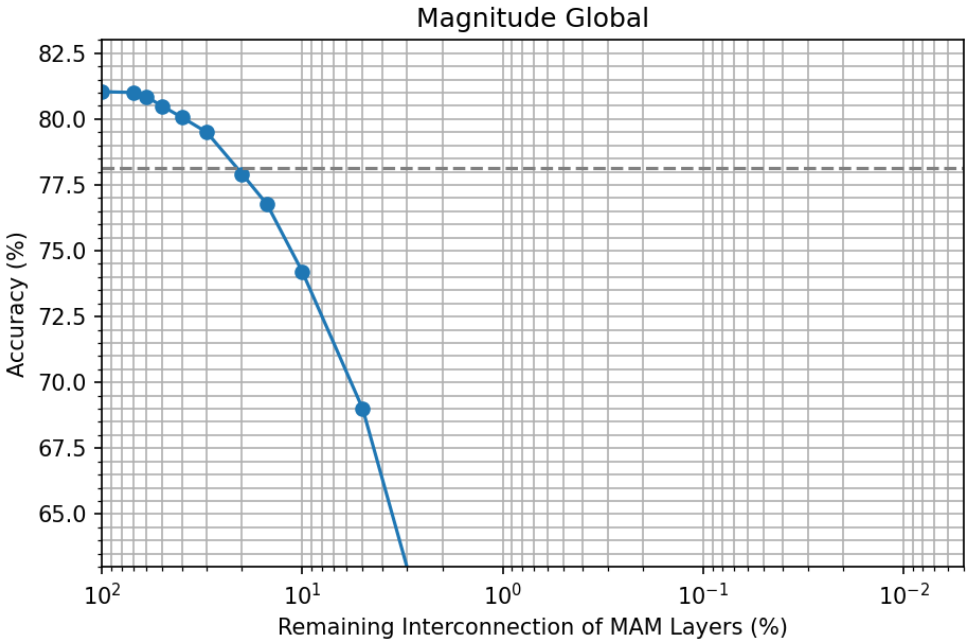
Figure 7.1: AlexNet CIFAR-10 Pruning

significant.

The Figures 7.2, 7.3 and 7.4 show, instead, the results of all the mentioned pruning methods for ViT trained on ImageNet. Once again, even for this complex architecture and dataset, it can be observed that the prunability of the layer is extremely high, retaining only 0.07% of the present weights for GGP, and demonstrate how Gradient pruning exhibits significantly better effectiveness compared to other methods. In this case, a notable difference from before is observed in the behavior of Magnitude pruning, which does not stand up to comparison with Gradient pruning. For the PSP, similar observations apply as those discussed earlier.

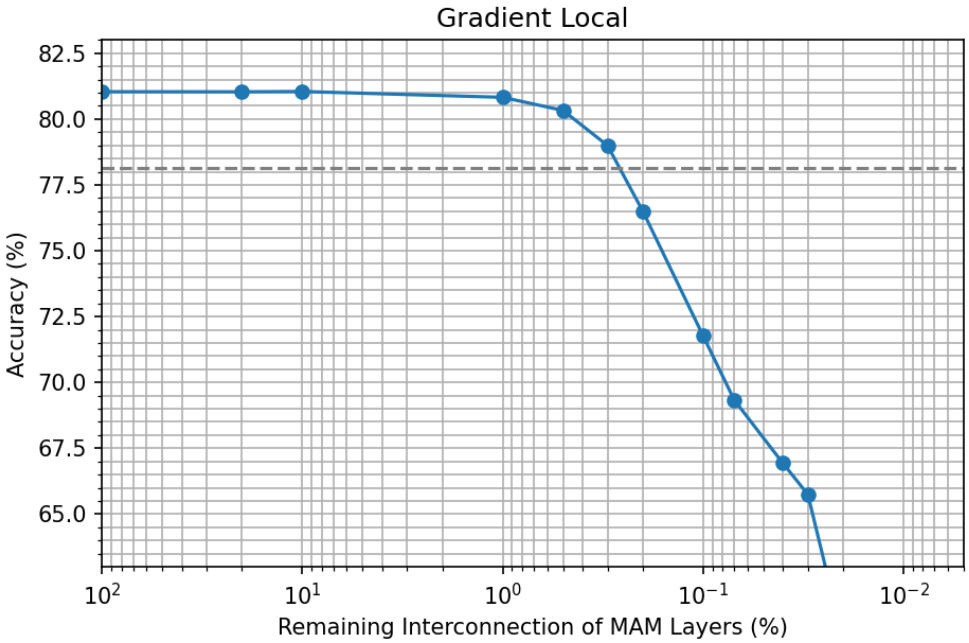


(a) LMP

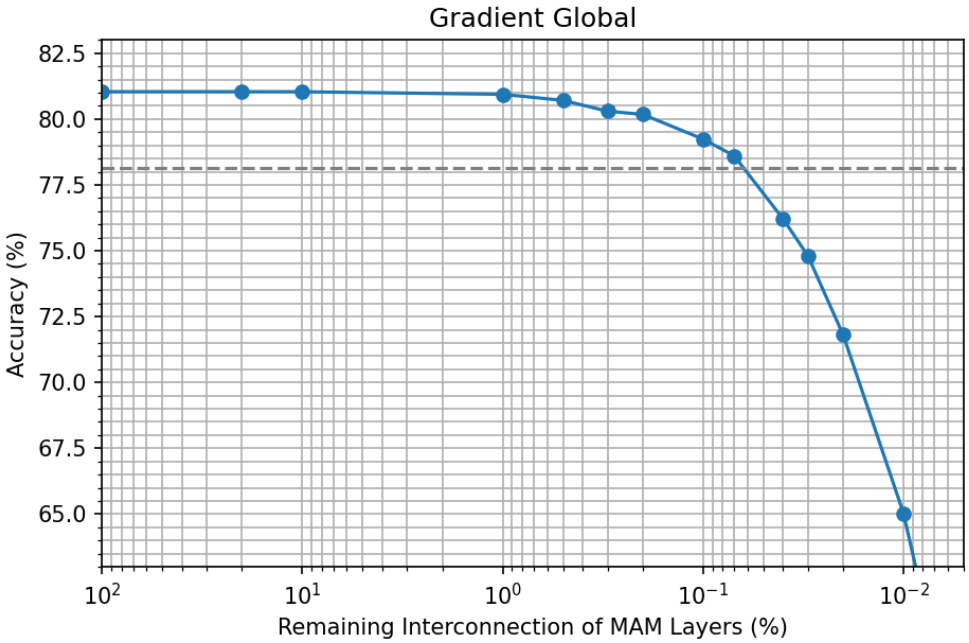


(b) GMP

Figure 7.2: ViT ImageNet-1K Magnitude Pruning

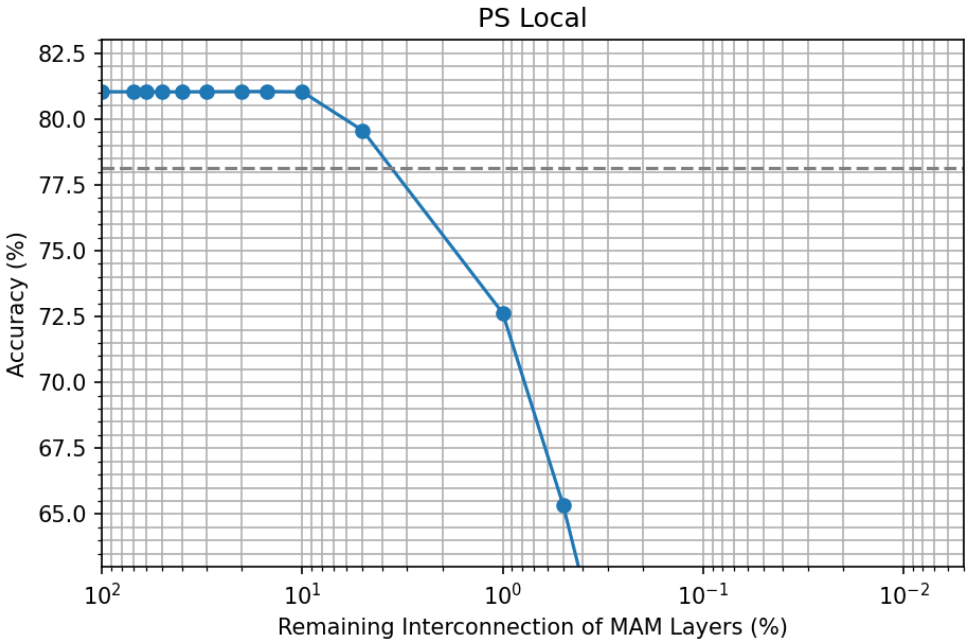


(a) LGP

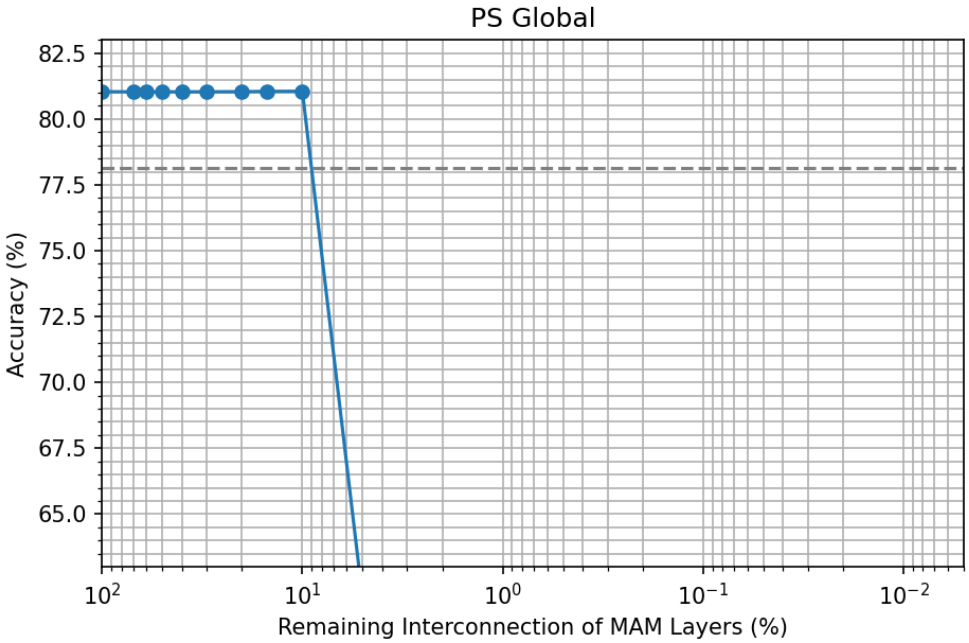


(b) GGP

Figure 7.3: ViT ImageNet-1K Gradient Pruning



(a) LPSP



(b) GPSP

Figure 7.4: ViT ImageNet-1K Probability of Selection Pruning

7.1.1 Analysis of Gradient Pruning

In this study, a more specific analysis was conducted to examine why Gradient Pruning is the method that has yielded the best overall results, particularly for ViT. One hypothesis for this observation lies in the correlation between this method and Probability of Selection Pruning. These two methodologies assign an equal score, namely 0, when a weight is never selected. When a weight is selected, their scores diverge, yet there exists a correlation: the gradient accumulates for each weight whenever it is selected, accumulating as many times as the PSP's assigned score. Considering the earlier discussion on PSP's potential functioning, a new score can be defined as the product of the Magnitude score and the Probability of Selection score. This approach combines PSP's properties during initial pruning phases and associates a score proportional to the numbers of activations, but not solely dependent on them; it also considers magnitude, aiming to replicate behavior of the Gradient Pruning. This could serve to provide experimental evidence supporting these hypotheses.

Figure 7.5 demonstrates how this method, despite still performing less effectively than LGP, provides evidence that the hypotheses mentioned above may be realistic. Indeed, it outperforms both individual LMP and LPSP.

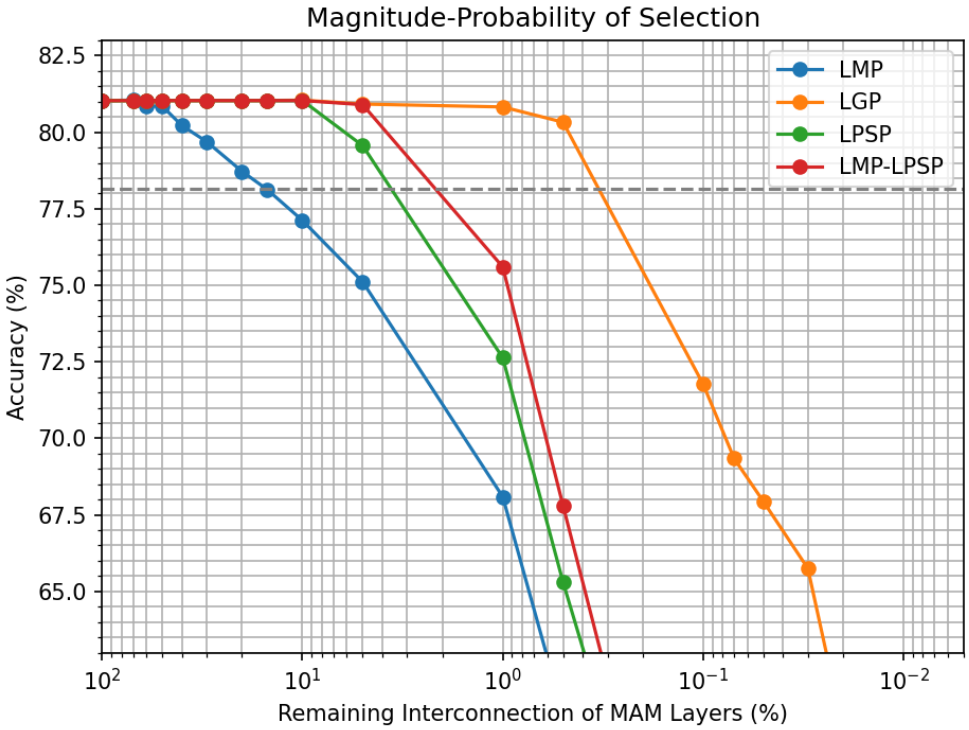


Figure 7.5: ViT ImageNet-1K parallelism between LGP and a mixed LMP-LPSP

7.2 Analysis of MAC Pruning Results with MAM Approach

7.2.1 Exploiting MAM to prune MAC

In this section, I present the experiment to evaluate the feasibility of using MAM for pruning MAC layers. Recalling the process pipeline: after the training and pruning process with MAM, the MAC layers are reintroduced, and the network is fine-tuned.

	Pruning	**MAM**	**MAC**
AlexNet CIFAR-10	GGP 99.8%	89.46%	89.91%
ViT ImageNet-1K	GGP 99.93%	79.70%	78.95%

Table 7.1: MAM to MAC performance after Pruning

In Table 7.1, the results of this approach are shown once the network has reached convergence; a comparison is also provided to illustrate how the MAM layer performs with fine-tuning after pruning. It is worth noting that the results on ViT trained with ImageNet could potentially be better, as it was only fine-tuned for one epoch.

The results demonstrate that the hypothesis suggesting the MAM layer serves to identify crucial interconnections is indeed correct, and that the non-linearity introduced is not necessary to achieve these performances with strongly sparse weight matrices. In fact, the result using MAC are comparable with that using MAM.

7.2.2 Early pruning of MAM

In this following, is demonstrated the possibility of pruning MAM before the network reaches convergence. After the vanishing contribution, non-essential weights are removed using GGP with a compression rate of 99.8%. The network is then further trained with this leaner structure.

The results presented in table 7.2 demonstrate how the crucial interconnections are indeed learned in a phase preceding convergence. Achieving an even higher top-1 accuracy compared to pruning alone in the final phase, with a theoretical savings of 99.8% FLOPs for each epoch for the pruned

After Epochs [1]	Linear β decay	Parabolic β decay
1	87.82%	88.87%
3	88.26%	88.64%
5	87.85%	88.95%
10	87.76%	89.33%
15	88.38%	**89.46%**

Table 7.2: AlexNet CIFAR-10 MAM pruning before convergence

MAM layers if training with sparse matrices were used. his result is achieved by considering that only 0.2% of the weights remain, while the others have been removed. Consequently, computational resources will only need to be used for this small number of weights.

The results also suggest that there is a performance increase when using a parabolic beta decay. Moreover, utilizing more after epochs indicates a potential small increase in accuracy, although less than 1%.

Now the experiment is repeated, but MAC is introduced in the place of the MAM layers after the pruning.

After Epochs	Linear β decay	Parabolic β decay
1	88.99%	89.33%
3	89.37%	89.75%
5	89.42%	89.46%
10	89.38%	89.53%
15	89.51%	**89.79%**

Table 7.3: AlexNet CIFAR-10 MAM to MAC before convergence

Table 7.3 presents the results of this approach. It can be observed that the results are even better than those previously shown, demonstrating not only that it is possible to prune the MAM layer before convergence, but also confirming that the crucial interconnections are already identified at this stage and that is the fundamental property of MAM that makes it extremely

[1]"After Epochs" is refered to those epochs that are conducted after the vanishing contributions phase.

efficient. Similar analyses to the previous ones are confirmed in this case, potentially saving 99.87% of the FLOPs compared to normal training without early pruning of the MAM layers. The savings are greater because the MAC layer requires only 2 FLOPs per weight compared to the 3 required by MAM.

Chapter 8

Conclusions

Throughout this research, the Multiply-And-Max/min (MAM) approach has been evaluated as a novel technique for pruning and accelerating the training of neural networks, as well as a pruning tool for MAC.

The initial experiments focused on replicating the previous work concerning the prunability of the MAM layer within neural networks. This is done by replacing specific MAC layers with MAM layers, training the network and conducting one-shot pruning using various methodologies.

The results confirmed that the MAM layer significantly sparsified matrices, showing its efficacy in reducing model complexity while maintaining high accuracy. Notably, the experiments on ViT trained on ImageNet-1K revealed an accuracy drop of less than 3% while removing 99.93% of weights using Gradient Global Pruning (GGP).

Specifically this work also provides mathematically insights, experimentally confirmed, into why Gradient Pruning outperforms other techniques. This is attributed to its inherent correlation with Probability of Selection Pruning, demonstrated through the development of a customized scoring mechanism that mimics the characteristics of Gradient Pruning.

The central contribution of this work was founding out that after pruning a MAM layer, it can be reverted to a MAC layer while preserving the same sparse matrices previously generated. This approach facilitates a training pipeline that concludes with the network retaining its original structure. This approach is adopted because the MAM layer introduces computationally challenges. Altering the structure of the net increases the theoretical FLOPs per weight from 2 (typical for a MAC layer) to 3. Additionally, MAC layers benefit from widely optimized hardware, ensuring efficient performance. In

contrast, MAM layers lack such hardware optimization and would require specialized hardware adaptations.

The transition from deeply pruned MAM structures to deeply pruned MAC structures achived an accuracy of 78.95% for ViT trained on ImageNet-1k, obtaining an incredible compression for the MAC layer.

Another important contribution concern experiments involving pruning DNN layers before convergence, showing the MAM layer's capability to identify crucial interconnections early in the training process. The reason for this is because in recent years, the development and deployment of large neural network models have posed significant challenges related to resource requirements during the training. This early pruning potentially leads to significant server-side savings up to 99.8% FLOPs for the pruned layers during the final stages of training for AlexNet trained on CIFAR-10.

These aspects find further confirmation through parallels with dynamic sparse training, as shown in paragraph 5.2. It can thus be concluded that the MAM layer can be utilized as a method of dynamic pruning based on the provisional modification of the neural network's structure.

Appendix A

Linear and Parabolic Beta Decay

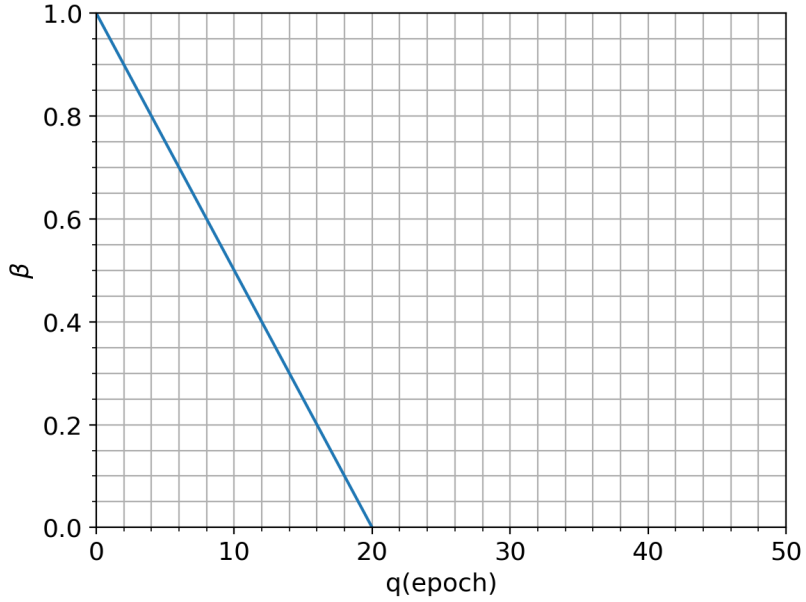
In transitioning from the MAC to the MAM layer, β must decrease from 1 to 0. To achieve this, various monotonically decreasing functions can be defined to exhibit this behavior. Particularly, an analysis on:

1. **Linear:**
$$\beta = -\frac{1}{Q}q + 1 \tag{A.1}$$

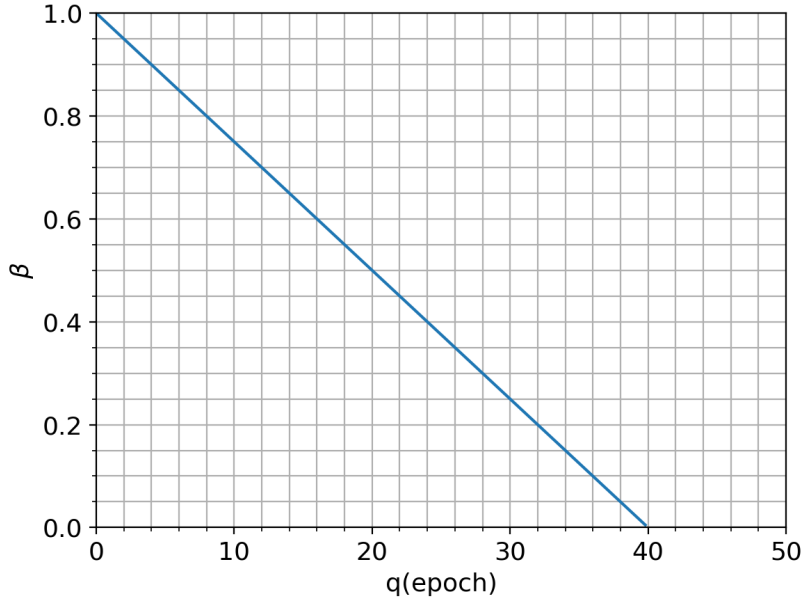
2. **Parabolic:**
$$\beta = \frac{1}{Q^2}q^2 - \frac{2}{Q}q + 1 \tag{A.2}$$

Where Q is the number of epoch used for the transition and q is the current epoch.

The profiles of functions A.1 and A.2 can be seen in Figures A.1 and A.2 for different values of Q. Empirically, it is observed that both lead to comparable results. However, during training, near $\beta = 0$, there is a performance decrease that is less pronounced when using parabolic decay rather than linear decay.

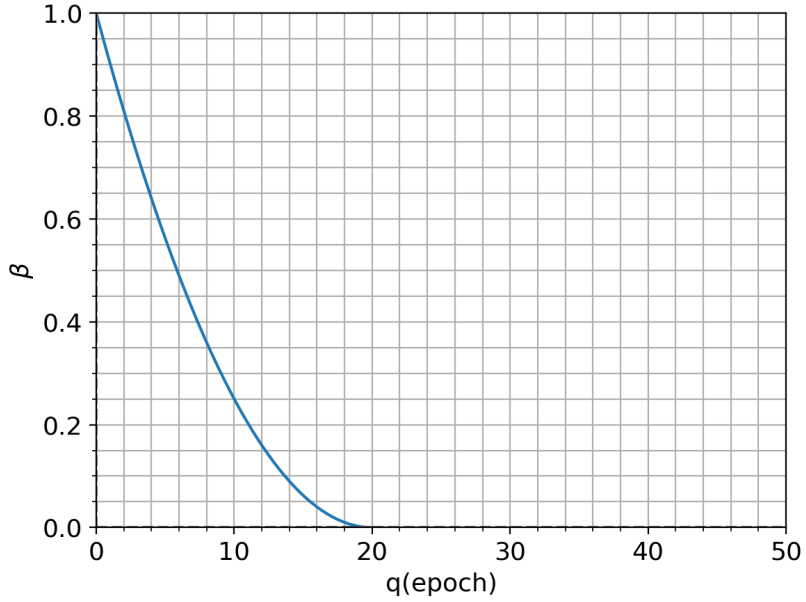


(a)

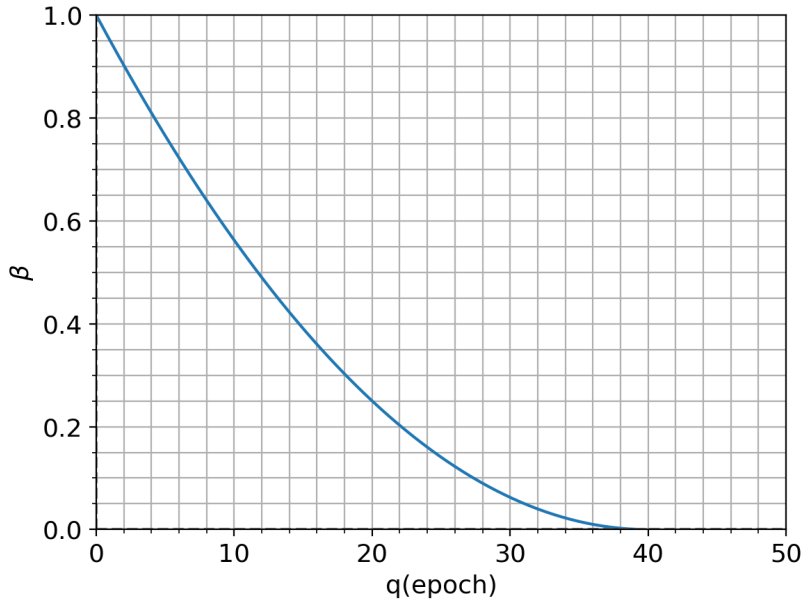


(b)

Figure A.1: Linear Decay with $Q = 20$(b), 40(d).



(a)



(b)

Figure A.2: Parabolic Decay with $Q = 20$(a), 40(b).

Bibliography

[1] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV]. URL: https://arxiv.org/abs/1505.04597 (cit. on p. 1).

[2] Pawan Kumar Mall, Pradeep Kumar Singh, Swapnita Srivastav, Vipul Narayan, Marcin Paprzycki, Tatiana Jaworska, and Maria Ganzha. «A comprehensive review of deep neural networks for medical image processing: Recent developments and future opportunities». In: *Healthcare Analytics* 4 (2023), p. 100216. ISSN: 2772-4425. DOI: https://doi.org/10.1016/j.health.2023.100216. URL: https://www.sciencedirect.com/science/article/pii/S2772442523000837 (cit. on p. 1).

[3] Ali Bou Nassif, Ismail Shahin, Imtinan Attili, Mohammad Azzeh, and Khaled Shaalan. «Speech Recognition Using Deep Neural Networks: A Systematic Review». In: *IEEE Access* 7 (2019), pp. 19143–19165. DOI: 10.1109/ACCESS.2019.2896880 (cit. on p. 1).

[4] Pranav Singh Chib and Pravendra Singh. *Recent Advancements in End-to-End Autonomous Driving using Deep Learning: A Survey*. 2023. arXiv: 2307.04370 [cs.RO]. URL: https://arxiv.org/abs/2307.04370 (cit. on p. 1).

[5] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL]. URL: https://arxiv.org/abs/2303.08774 (cit. on pp. 1, 12).

[6] Fangxin Wang, Miao Zhang, Xiangxiang Wang, Xiaoqiang Ma, and Jiangchuan Liu. «Deep Learning for Edge Computing Applications: A State-of-the-Art Survey». In: *IEEE Access* 8 (2020), pp. 58322–58336. DOI: 10.1109/ACCESS.2020.2982411 (cit. on p. 2).

[7] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, and Song Han. «Tiny Machine Learning: Progress and Futures [Feature]». In: *IEEE Circuits and Systems Magazine* 23.3 (2023), pp. 8–34. ISSN: 1558-0830. DOI: 10.1109/mcas.2023.3302182. URL: http://dx.doi.org/10.1109/MCAS.2023.3302182 (cit. on p. 2).

[8] Guangji Bai et al. *Beyond Efficiency: A Systematic Survey of Resource-Efficient Large Language Models*. 2024. arXiv: 2401.00625 [cs.LG]. URL: https://arxiv.org/abs/2401.00625 (cit. on pp. 2, 12).

[9] Vadera; Sunil and Ameen; Salem. «Methods for Pruning Deep Neural Networks». In: *IEEE Access* 10 (2022), pp. 63280–63300. DOI: 10.1109/ACCESS.2022.3182659 (cit. on pp. 2, 18).

[10] Hongrong Cheng; Miao Zhang; Javen Qinfeng Shi. «A Survey on Deep Neural Network Pruning-Taxonomy, Comparison, Analysis, and Recommendations». In: (2023). DOI: https://doi.org/10.48550/arXiv.2308.06767 (cit. on pp. 2, 18).

[11] Philippe Bich, Luciano Prono, Mauro Mangia, Fabio Pareschi, Riccardo Rovatti, and Gianluca Setti. «Aggressively prunable MAM²-based Deep Neural Oracle for ECG acquisition by Compressed Sensing». In: *2022 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. 2022, pp. 163–167. DOI: 10.1109/BioCAS54905.2022.9948676 (cit. on pp. 3, 23).

[12] Luciano Prono, Philippe Bich, Chiara Boretti, Mauro Mangia, Fabio Pareschi, Riccardo Rovatti, and Gianluca Setti. «A Multiply-And-Max/min Neuron Paradigm for Aggressively Prunable Deep Neural Networks». In: (Feb. 2024). DOI: 10.36227/techrxiv.22561567.v2. URL: http://dx.doi.org/10.36227/techrxiv.22561567.v2 (cit. on pp. 3, 23, 32, 33, 35).

[13] Ian Goodfellow; Yoshua Bengio; Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on p. 5).

[14] Rumelhart; Hinton; Williams. «Learning representations by back-propagating errors». In: 323 (6088) (1986), pp. 533–536. DOI: doi:10.1038/323533a0 (cit. on p. 6).

[15] URL: https://www.developersmaggioli.it/blog/le-reti-neurali-ricorrenti/ (cit. on p. 7).

[16] URL: https://www.ce.unipr.it/people/medici/geometry/node112.html (cit. on p. 7).

[17] Ragav Venkatesan; Baoxin Li. *Convolutional Neural Networks in Visual Computing*. First. CRC Press, 2017. DOI: https://doi.org/10.4324/9781315154282 (cit. on p. 7).

[18] URL: https://it.wikipedia.org/wiki/Rete_neurale_convoluzionale#/media/File:Typical_cnn.png (cit. on p. 8).

[19] Alex Krizhevsky;Ilya Sutskever;Geoffrey E. Hinton. «ImageNet classification with deep convolutional neural networks». In: (2012). DOI: https://doi.org/10.1145/3065386 (cit. on p. 9).

[20] Alex Krizhevsky. «One weird trick for parallelizing convolutional neural networks». In: (2014). DOI: https://doi.org/10.48550/arXiv.1404.5997 (cit. on p. 9).

[21] URL: https://medium.com/swlh/scratch-to-sota-build-famous-classification-nets-2-alexnet-vgg-50a4f55f7f56 (cit. on p. 11).

[22] Ashish Vaswani; Noam Shazeer; Niki Parmar; Jakob Uszkoreit; Llion Jones; Aidan N. Gomez; Lukasz Kaiser; Illia Polosukhin. «Attention Is All You Need». In: (2017). DOI: https://doi.org/10.48550/arXiv.1706.03762 (cit. on pp. 12, 13).

[23] Alexey Dosovitskiy; Lucas Beyer; Alexander Kolesnikov; Dirk Weissenborn; Xiaohua Zhai; Thomas Unterthiner; Mostafa Dehghani; Matthias Minderer; Georg Heigold; Sylvain Gelly; Jakob Uszkoreit; Neil Houlsby. «An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale». In: (2020). DOI: https://doi.org/10.48550/arXiv.2010.11929 (cit. on p. 14).

[24] Davis Blalock; Jose Javier Gonzalez Ortiz; Jonathan Frankle; John Guttag. «What is the State of Neural Network Pruning?» In: (2020). DOI: https://doi.org/10.48550/arXiv.2003.03033 (cit. on p. 18).

[25] Song Han; Huizi Mao; William J. Dally. «Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding». In: (2015). DOI: https://doi.org/10.48550/arXiv.1510.00149 (cit. on p. 19).

[26] Zhonghui You; Kun Yan; Jinmian Ye; Meng Ma; Ping Wang. «Gate Decorator: Global Filter Pruning Method for Accelerating Deep Convolutional Neural Networks». In: (2019). DOI: https://doi.org/10.48550/arXiv.1909.08174 (cit. on p. 19).

[27] Junjie Liu, Zhe Xu, Runbin Shi, Ray C. C. Cheung, and Hayden K. H. So. *Dynamic Sparse Training: Find Efficient Sparse Network From Scratch With Trainable Masked Layers*. 2020. arXiv: 2005.06870 [cs.LG] (cit. on p. 20).

[28] Jonathan Frankle; Michael Carbin. «The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks». In: (2019). DOI: https://doi.org/10.48550/arXiv.1803.03635 (cit. on p. 20).

[29] Huan Wang, Can Qin, Yue Bai, Yulun Zhang, and Yun Fu. *Recent Advances on Neural Network Pruning at Initialization*. 2022. arXiv: 2103.06460 [cs.LG]. URL: https://arxiv.org/abs/2103.06460 (cit. on p. 22).

[30] Maying Shen, Pavlo Molchanov, Hongxu Yin, and Jose M. Alvarez. *When to Prune? A Policy towards Early Structural Pruning*. 2021. arXiv: 2110.12007 [cs.CV]. URL: https://arxiv.org/abs/2110.12007 (cit. on p. 30).

[31] Alex Krizhevsky. «Learning Multiple Layers of Features from Tiny Images». In: 2009. URL: https://api.semanticscholar.org/CorpusID:18268744 (cit. on p. 32).

[32] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. «ImageNet: A large-scale hierarchical image database». In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848 (cit. on pp. 32, 33).