electronics

MDPI

Article

Forward Learning of Large Language Models by Consumer Devices

Danilo Pietro Pau * and Fabrizio Maria Aymone

System Research and Applications, STMicroelectronics, Via C. Olivetti 2, 20864 Agrate Brianza, Italy; fabriziomaria.aymone@mail.polimi.it
* Correspondence: danilo.pau@st.com

Abstract: Large Language Models achieve state of art performances on a broad variety of Natural Language Processing tasks. In the pervasive IoT era, their deployment on edge devices is more compelling than ever. However, their gigantic model footprint has hindered on-device learning applications which enable AI models to continuously learn and adapt to changes over time. Back-propagation, in use by the majority of deep learning frameworks, is computationally intensive and requires storing intermediate activations into memory to cope with the model's weights update. Recently, "Forward-only algorithms" have been proposed since they are biologically plausible alternatives. By applying more "forward" passes, this class of algorithms can achieve memory reductions with respect to more naive forward-only approaches and by removing the need to store intermediate activations. This comes at the expense of increased computational complexity. This paper considered three Large Language Model: DistilBERT, GPT-3 Small and AlexaTM. It investigated quantitatively any improvements about memory usage and computational complexity brought by known approaches named PEPITA and MEMPEPITA with respect to backpropagation. For low number of tokens in context, and depending on the model, PEPITA increases marginally or reduces substantially arithmetic operations. On the other hand, for large number of tokens in context, PEPITA reduces computational complexity by 30% to 50%. MEMPEPITA increases PEPITA's complexity by one third. About memory, PEPITA and backpropagation, require a comparable amount of memory to store activations, while MEMPEPITA reduces it by 50% to 94% with the benefits being more evident for architectures with a long sequence of blocks. In various real case scenarios, MEMPEPITA's memory reduction was essential for meeting the tight memory requirements of 128 MB equipped edge consumer devices, which are commonly available as smartphone and industrial application multi processors.

Keywords: on-device learning; backpropagation; forward learning; PEPITA; MEMPEPITA; Large Language Models; Natural Language Processing

check for updates

Citation: Pau, D.P.; Aymone, F.M. Forward Learning of Large Language Models by Consumer Devices. *Electronics* **2024**, *13*, 402. https://doi.org/10.3390/electronics13020402

Academic Editors: Massimiliano Donati and Riccardo Berta

Received: 1 December 2023
Revised: 14 January 2024
Accepted: 16 January 2024
Published: 18 January 2024

1. Introduction

Since its introduction in 2017 [1], the Transformer architecture has revolutionized the field of Natural Language Processing (NLP) achieving unprecedented accuracy over a plethora of complex tasks such as translation [2], question answering [3], sentiment analysis [4,5], text classification [6], textual entailment [7] and summarization [8]. With the staggering growth of IoT devices, the deployment of such powerful models to the edge has become more challenging than ever. Nevertheless, it is known from theory [9] that the performance of a transformer-based language model scales as a power law with respect to the number of parameters. For this reason, recent years have featured a trend towards ever-increasing model footprint resulting in the notorious 175-billion parameters GPT-3 [10]. The consequent high memory requirements and heavy computational workloads are incompatible with micro-controllers (MCUs) and sensors embedded assets, which are severely resource-constrained. In order to mitigate such issue, a broad line of research [11]

has focused on reducing the computational complexity and memory usage running the inference by applying known techniques such as pruning [12,13], quantization [12,14] and knowledge distillation [15]. However, enabling only model's inference on the device is not enough. The performance of the AI models, in fact, deteriorates as time passes since the last training cycle; phenomenon known as concept drift [16], hence mandates model's parameter updates from time to time. The prominent field of on-device learning (ODL) [17] allows for machine learning (ML) models deployed on edge devices to adapt to the continuously changing data statistics, which are collected by the sensors, and performing model's parameter training. This paper explores the application of novel learning methods, PEPITA and MEMPEPITA, to Transformer-based Large Language Models (LLMs) for ODL on edge devices. It quantitatively analyzes the improvements brought by these methods in reducing computational complexity and memory usage compared to traditional backpropagation (BP), especially in resource-constrained environments. The memory and complexity performance of BP, PEPITA and MEMPEPITA on state-of-the-art models like GPT-3 Small, DistilBERT, and AlexaTM is evaluated and reported. The suitability of these training methods is also investigated in real-world edge devices, considering the constraints of memory and processing power. This work is organized as follows: Section 2 sets the research question that will be addressed in the subsequent sections; Section 3 reports the background literature on Transformers and Large Language Models to this day; Section 4 review relevant and related works in the known literature; Section 5 proposes the PEPITA [18] training procedure applied to the Transformer architecture; Section 6 refers to the method proposed in Section 5, and quantitatively analyze different learning algorithms on DistilBERT [19], GPT-3 Small [10] and AlexaTM [20], reporting as well the results of the analysis; Section 7 discusses and compares BP [21], PEPITA and MEMPEPITA [22] application to Large Language Models on the basis of the 6 results; Section 8 provides total RAM usage and latency estimations, based on candidate microprocessors, and comments the applicability to consumer edge devices; Section 9 concludes the paper and proposes further and future developments.

2. The Research Question

The application of ODL to LLMs deployed on edge devices represents the major challenge, given the computational complexity and high memory requirements of the BP algorithm when applied. PEPITA and MEMPEPITA represent compelling alternatives to BP, since they perform Forward-only passes. They remove the complexity associated to the backward pass and MEMPEPITA does not need to store the activations. The quantitative analysis [22] showed that PEPITA did not bring any advantages in terms of memory usage w.r.t. BP, while it introduced additional complexity. MEMPEPITA, instead, was capable of drastically reducing intermediate activations storage by introducing only a third more computational complexity. Unfortunately, such an analysis did not include the Transformer architecture which is the backbone of today's LLMs. Therefore, to progress in this respect, the question this paper set and tries to answer is: "How much would PEPITA and MEMPEPITA improve the peak memory and computational complexity w.r.t. BP-trained models when applied to Transformer-based LLMs processing?". The contributions, this paper proposes, can be summarized as follows:

- Formulation of PEPITA learning rule for transformers.
- Quantitative analysis of computational complexity and memory needs of PEPITA and MEMPEPITA in contrast with BP on state of the art LLMs such as GPT-3 Small, DistilBERT and AlexaTM.
- Study on the applicability, in terms of memory and latency constraints, of PEPITA and MEMPEPITA on existing hardware for enabling on-device learning of LLMs on edge consumer devices.

3. Background

Transformers and Large Language Models

When it comes to training a NLP model, the scarcity of task-specific labeled data has prompted researchers to leverage the vast unlabeled text corpora [23,24]. Initial approaches consisted in using, inside a specific model trained on a specific task, word embeddings that were previously trained in an unsupervised fashion [25]. Yet, knowledge that extends beyond the single word, such as high-level semantics, is not transferred to the final model. More sophisticated techniques [26] involve using the hidden representations of models trained on the unlabeled text collection as auxiliary features for the final model, requiring additional parameters and structural modifications. Eventually, the most effective paradigm has been proven to be training the model on the unlabeled text corpora (pre-training) and, in a second phase, training (fine-tuning) the same model on the final task starting from the previously trained parameters. Ref. [27] used for the first time unsupervised pre-training with the Transformer architecture and achieving SOTA results on a broad variety of NLP datasets. During the fine-tuning of the transformer, a last task-specific linear output layer was introduced and structured inputs were converted into a processable ordered sequence accordingly to the task. By marginally modifying the model architecture, knowledge acquired during pre-training was much exploited. This approach was vastly adopted and currently became the de-facto standard adopted by the NLP community.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

Its success is due to the Transformer capability to capture longer-range linguistic relations, as opposed to LSTM [28], and due to the attention mechanism modeled in (1) where Q, K and V are matrices of size $n_{ctx} \times d_k$ with n_{ctx} being the number of tokens in context. As it was first proposed [1], the transformer consisted in an encoder and decoder block and was used for machine translation. The main difference between the encoder and the decoder is that the former has got a bidirectional attention mechanism, while the latter features unidirectional attention (i.e., a token only attends to previous tokens). Most of later works used, depending on the considered task, only one of the two components, namely encoder-only (e.g., BERT [29]) and decoder-only (e.g., GPT [27]) models. Recent research, however, suggests that an encoder-decoder model, such as the 20 billions-parameters (20 B) AlexaTM [20], can achieve similar or higher performance w.r.t. decoder-only model (e.g., 540 B PaLM [30] or 175 B GPT-3 [10]) using far less parameters. In the academic literature, there is significant interest in the applications of LLMs at the edge [31], particularly in the context of mobile technologies [32,33]. Different techniques [34,35] have been experimented for compressing LLMs for inference optimization including DistilGPT2 and DistilBERT [19], MobileBERT [36] and TinyBERT [37]. The size of the context window n_{ctx} is a major issue for efficient inference, as the complexity of the attention mechanism grows as $\mathcal{O}(n_{ctx}^2)$. Ref. [38] proposes a dilated attention which reduces the complexity to linear, while preserving model expressivity. On the other hand, ref. [39] considers an efficient attention mechanism obtained by leveraging the associative property of matrix multiplication, reducing complexity to linear.

$$W_l = W_l - \eta \, \nabla_{z_l} \mathcal{L} \, a_{l-1}^T \tag{2}$$

4. Related Works

4.1. On-Device Learning

Compared to inference, training is by far more compute and memory intensive and therefore the major challenge to address in consumer devices. The training procedure of Neural Networks (NNs), in fact, relies on BP [21] which is memory-bounded [40]. This is due to the storage of NN's activations, resulting in slow and energy-consuming operations. Activations are mathematically required to compute the weight update as shown in (2), where η is the learning rate, $\nabla_{z_l} \mathcal{L}$ is the gradient of the loss function w.r.t. the pre-activations

of the l-th layer z_l and a_{l-1} the activations of the previous layer. For Convolutional Neural Networks (CNNs), a reduction of the number of trainable parameters does not affect total memory footprint since the activations footprint dominates [41]. Consequently, the field of ODL has focused on leveraging transfer learning by updating on-device only selected weights/layers of a pre-trained NN, with the scope of storing into memory only the corresponding activations. Since model's biases update does not depend on the activations, TinyTL [41] proposed to update only the former and, to compensate the loss in model capacity, introduced a memory-efficient bias module. It provided 7.3–12.9x memory savings, with the same accuracy of fine-tuning full models. MCUNetV3 [42] is the state of the art for ODL application to Visual Wake Words [43] within 256 KiB SRAM. It introduced several innovations such as Quantization-Aware Scaling (QAS), which controlled the quantization error during the training, and Sparse Update, to skip less important layers and sub-tensors. This approach pushed BP-based ODL to the limits of its efficiency. Unfortunately, ODL applied to Transformers topologies is still at its early stage, due to the daunting size of such models, and only few works have been published. Like TinyTL, BitFit [44] is a sparse-fine-tuning approach where only the bias terms are being modified. For small to medium training data, it achieved results which were competitive to fine-tuning the entire model, while for larger datasets it was competitive only with other sparse-fine-tuning methods. Ref. [45] proposed a training procedure for BERT-like models which diminished the memory consumption of activation maps by updating selectively only certain parameters. It reduced fine-tuning time of DistilBERT by 30% on CoLA dataset [46] and the time spent on memory operations by 47%, while achieving comparable accuracy. Ref. [47] did Transformers training with 4-bit integer with a specific Hadamard quantizer that suppresses activations' outliers and by leveraging the structural sparsity of gradients through bit splitting and score sampling techniques.

4.2. Alternatives to Back-Propagation

It is known that the human brain does not learn using BP [48,49]. The research community, therefore, has started investigating plausible alternatives for credit assignment [50,51], namely how much each weight would contribute to the loss function. Forward-Forward (FF) [52] and PEPITA credit assignment algorithms addressed the biological implausible aspects of BP by performing Forward-only passes, to remove the "weight transport" [53], the "update-locking" [54,55] and the "freezing activation" [56] problem. Ref. [22] proposed MEMPEPITA, a memory efficient version of PEPITA which reduced on average one third of the total RAM w.r.t. BP, and costing a third more complexity. By the same work, the computational complexity and memory usage of FF, PEPITA and MEMPEPITA were studied on the MLCommons-Tiny benchmarks [57]. The quantitative analysis revealed that PEPITA incremented computational complexity, while maintaining same peak memory usage as BP. On the other hand, FF and MEMPEPITA reduced activations memory by even ∼3x. During inference, however, FF introduced considerable computational overhead that grew linearly with the number of classes. This makes FF unsuitable for LLMs as the vocabulary is composed by thousands of tokens. On the other hand, MEMPEPITA followed the same inference procedure of BP, proposing itself as a candidate for NLP applications. A comparison of the learning procedures is reported in Table 1.

$$W_\ell = W_\ell - \eta(a_\ell - a_\ell^{err})(a_{\ell-1}^{err})^T \tag{3}$$

4.2.1. PEPITA

PEPITA [18] relies on two forward passes per each input sample. The first pass, named standard pass, calculates the error of the model with respect to the data labels. As the output and input dimensions are generally different, the error is projected onto the input through a fixed random matrix F, with zero mean and small standard deviation. The second pass, named modulated, transforms the input using the error calculated by the standard pass and computes the corresponding activations. The difference between the

activations of the two passes is then used to update the weights according to Equation (3). The weights of the last layer can be updated by the error at the output layer. It has also been experimentally analyzed that the convergence rate of PEPITA is lower than BP. However, this gap is narrowed when pre-mirroring technique [58] is adopted.

Table 1. Summary of the learning procedures.

Learning Methods	BP	PEP	MPE
Forward pass	1	2	3
Backward pass	1	0	0
Weight update	1	1	1
Activations	all	all	current

PEP stands for PEPITA and MPE for MEMPEPITA. MPE need to store only activation buffers for current layers, while BP and PEP for all the layers in the topology.

4.2.2. MEMPEPITA

Because PEPITA required to store the activations computed in the standard pass to calculate $(a_\ell - a_\ell^{err})$ in the modulated pass, it has the same memory costs of BP. To avoid this, MEMPEPITA added a second standard pass performed in parallel to the modulated pass, to recompute the activations needed for weight update. Nevertheless, it introduced additional computation workload.

5. Pepita Applied to Transformers

The steps implemented in the original PEPITA learning procedure are shown in the Algorithm 1 table, where σ_ℓ indicates the non-linearity of the layer ℓ. For transformers, it is not trivial to define an error projection matrix since input and output dimensions are not fixed, while vary depending on the length of the input sequence of tokens. In encoder-only and decoder-only architectures the number of tokens at the output is the same as at the input, a one-to-one correspondence. Therefore, the error has already the same dimension as the input and no error projection matrix multiplication is needed. In an encoder-decoder architecture, this is true only for the input to the decoder, while the input to the encoder has different dimensions.

$$T_{err}^{enc} = Attention(T^{enc}, T_{err}^{dec}, T_{err}^{dec}) \tag{4}$$

This paper suggests to apply an attention mechanism on the one-hot encoded tokens of the encoder input (before being multiplied for the embedding matrix, each token is represented as a one-hot encoded vector with length equal to the vocabulary size) and the error tokens at the end of the decoder to project the error, as illustrated in Equation (4). Depending on the task, the error could be calculated for each token in the sequence, e.g., Causal Language Modeling (CLM), or only for some of them, e.g., Masked Language Modeling (MLM). In the latter scenario, the error is defined at the tokens not considered to be simply zero and to adopt the same projection procedure described previously.

Algorithm 1: PEPITA

Given: Features(x) and label($target$)
Standard Pass
$a_0 = x$
for $\ell = 1, ..., L$ **do**
  $a_\ell = \sigma_\ell(W_\ell a_{\ell-1})$
end for
$e = a_L - target$
Modulated pass
$a_0^{err} = x + Fe$
for $\ell = 1, ..., L-1$ **do**
  $a_\ell^{err} = \sigma_\ell(W_\ell a_{\ell-1}^{err})$
  if $\ell < L$ **then**
    Weight update
    $W_\ell = W_\ell - (a_\ell - a_\ell^{err}) \cdot (a_{\ell-1}^{err})^T$
  end if
end for
$W_L = W_L - e \cdot (a_{L-1}^{err})^T$

6. Quantitative Analysis

The following sub-sections describe the metrics adopted for memory and computational complexity profiling, they describe the procedure implemented for the quantitative analysis and report the results obtained.

6.1. Memory and Computational Complexity

The objective of this analysis is to determine the memory usage and computational complexity of BP, PEPITA and MEMPEPITA ODL procedure applied to the Transformer-based LLMs. The peak memory usage is defined as the memory required by the device to run the training workload and it includes three main contributions: model parameters, activations and the scratch buffer to store the intermediate computations at run time. In this analysis, the latter will not be considered since it highly depends on the low-level implementation of the algorithm and the underlying hardware which executes it. Model parameters are also not being included as they do not vary across the learning algorithm, which set the purpose of the analysis as comparative. Furthermore, due to the enormous model size of LLMs, the parameters are stored in main memory and subject of multiple read/write write during training. Techniques for sparse-fine-tuning, which reduce the number of trainable parameters, are orthogonal to this work. For the previous reasons, the memory contribution on which our analysis will focus are the activations. In order to measure the computational complexity, two main metrics have been involved. Multiply and accumulates (MACCs) is a well-known metric, as the most intensive part of computations in ML consists of matrix multiplications. In a transformer, however, there are layers whose computation does not match them. To account for the latter, the more general floating-point operations (FLOPs) are also utilized. Following the same assumption used in [59], each mathematical operation (e.g., exponentiation, division, etc) is considered a FLOP, independently from the hardware implementation; therefore, one MACC corresponds to two FLOPs. The computational complexity estimation procedure adopts the conventions and the results discussed in [60].

6.2. Methodology

To compare BP, PEPITA and MEMPEPITA, LLMs named DistilBERT, GPT-3 Small and AlexaTM were chosen, as they represent respectively encoder-only, decoder-only and encoder-decoder architecture. The quantitative details for each them are reported in Table 2. To estimate computational complexity, as described in [60], the transformer's

topology was decomposed into five main blocks: Embedding Layer (EL), Attention Block (AB), Feed-Forward Network (FFN), Layer Normalization and Softmax (LNS). For each of them, MACCs and FLOPs were computed for the forward pass, the backward pass and the weights update. The total complexity is the sum of the complexity of the operations required by the blocks. Then, the workload of a certain learning procedure was estimated by adding all the macro operations involved in it as described in Table 1. Regarding the memory, activations for BP and PEPITA were calculated as the sum of the activations of all layers. In the case of MEMPEPITA, instead, the activations were the maximum value of the sum of the activation buffers of two consecutive layers and of the largest activation buffer between these two layers. Activations have been considered to be integer 8bits numbers (INT8) and have been measured in Megabytes (MB). As the input to the models has not a fixed dimension, the analysis has been conducted on values of n_{ctx}, the number of tokens in the input context, between 1 to 2048. For AlexaTM, there are two distinct token sequences, which are in input, respectively, to the encoder and to the decoder. The n_{ctx} relative to the encoder is assumed to be fixed and equal to 100, while the n_{ctx} of the decoder can vary between 1 and 2048. Varying the encoder n_{ctx}, instead of the decoder's one, does not affect the final results. For the analysis, a batch size of 1 has been considered. With increasing batch sizes, complexity and activations would scale linearly. The complexity and memory calculations are available in the github repository [61]. For further details concerning the complexity estimation procedure, refer to [60].

Table 2. Description of models analysed.

Model	Enc	Dec	# of Heads	d_{model}	# of Parameters
GPT-3 Small	0	12	12	768	125 M
distilBERT	6	0	12	768	66 M
AlexaTM	46	32	32	4096	19.75 B

Enc and Dec indicate respectively the number of encoder and decoder layers. d_{model} is the length of the embeddings.

6.3. Results

The results are reported in Figure 1. Plotted trends are consistent throughout the three different models. For low values of n_{ctx}, MEMPEPITA is the most complex in terms of MACCs and FLOPs, followed by PEPITA and BP which share similar values. About Distilbert, PEPITA is more complex than BP for n_{ctx} less than ∼680, about MACCs, and ∼280, about FLOPs. Considering GPT-3 Small, PEPITA is more complex than BP for n_{ctx} less than ∼600, regarding MACCs, while BP FLOPs are always greater than PEPITA's. About AlexaTM, BP achieves higher complexity than PEPITA in both MACCs and FLOPs. For mid values of n_{ctx} MEMPEPITA is the most complex followed by BP and PEPITA, while for higher values BP exceeds substantially MEMPEPITA. In particular, BP exceeds MEMPEPITA for n_{ctx} greater than: 1340 for MACCs and 1170 for FLOPs in DistilBERT; 1260 for MACCs and 1070 for FLOPs in GPT-3 Small; 2969 for MACCs and 1344 for FLOPs in AlexaTM. From Figure 1 about MACCs and FLOPs, BP has a higher second derivative w.r.t. n_{ctx} when compared to PEPITA and MEMPEPITA which seem to share the same complexity. About memory, for the three models, BP requires the largest amount of activations with PEPITA falling slightly behind. On the other hand, MEMPEPITA requires significantly less memory than BP and PEPITA. The percentage variations in the memory and computational complexity of PEPITA and MEMPEPITA w.r.t. BP for different values of n_{ctx} and for the different models are reported in Table 3. MEMPEPITA reduces the activations memory, depending on n_{ctx}, from 51% to 59% on DistilBERT, from 62% to 67% on GPT-3 Small and by 94% on AlexaTM. Memory reductions by PEPITA are marginal for low values of n_{ctx} and they increase with larger values up to 16% for $n_{ctx} = 2048$. Computational

complexity is increased by PEPITA and MEMPEPITA for low number of tokens in context and it reduces abruptly with growing values of n_{ctx}. In DistilBERT and GPT-3 Small, when $n_{ctx} = 2048$ PEPITA reduces overall computational workload by 50% and MEMPEPITA by approximately one third. In AlexaTM, PEPITA reduces MACCs by 14% and FLOPs by 30%, while MEMPEPITA increases MACCs by 14% and reduces FLOPs by 6%. Also PEPITA accounts for approximately a third less MACCs/FLOPs compared to MEMPEPITA.

$$\nabla_x \mathcal{L} = \nabla_s \mathcal{L} \begin{bmatrix} s_1 \cdot (1 - s_1) & \dots & -s_n \cdot s_1 \\ \vdots & \ddots & \vdots \\ -s_1 \cdot s_n & \dots & s_n \cdot (1 - s_n) \end{bmatrix} \tag{5}$$

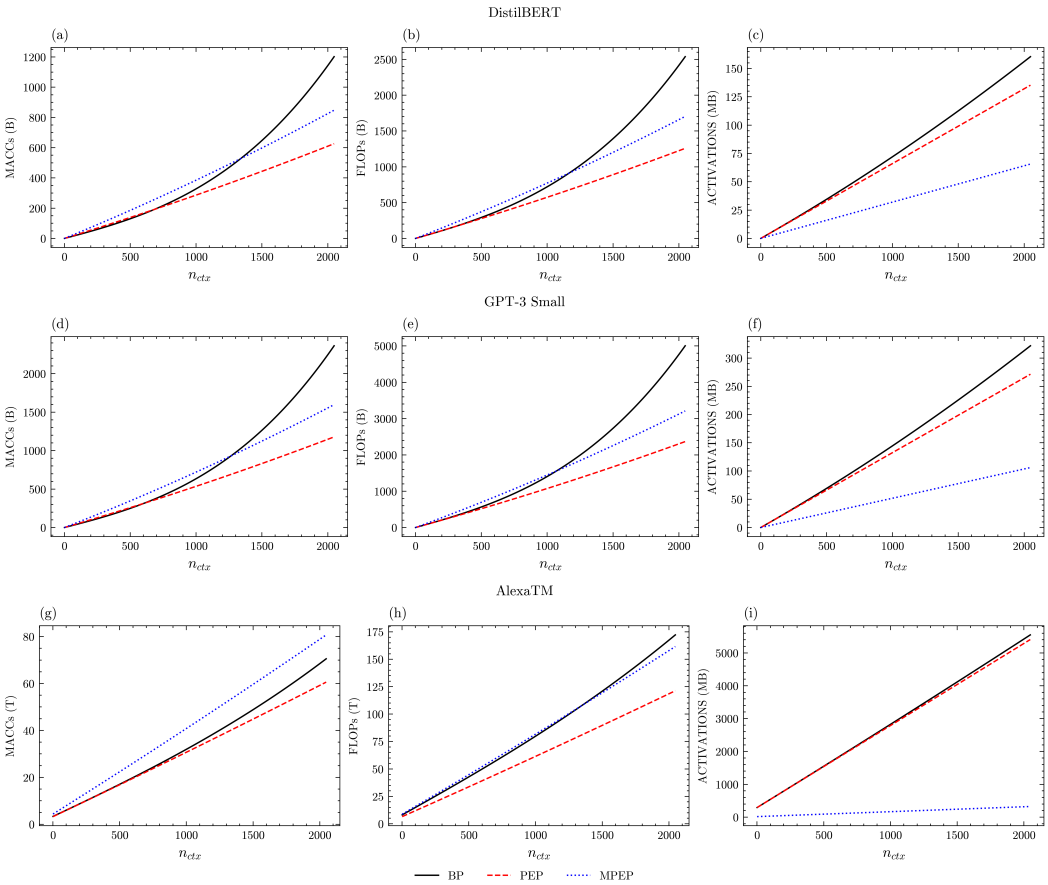

Figure 1. Computational and Memory complexity of the three LLMs analysed. Subfigures (**a**–**i**) depict respectively the MACCs, FLOPs and activations' footprint of DistilBert, GPT-3 Small and AlexaTM when trained with BP, PEP and MPEP.

Table 3. Variations w.r.t. BP.

Model	DistilBERT						GPT-3 Small						AlexaTM					
Metric	MACCs		FLOPs		ACT		MACCs		FLOPs		ACT		MACCs		FLOPs		ACT	
n_{ctx}	PEP	MPE	PEP	MPE	PEP	MPE	PEP	MPE	PEP	MPE	PEP	MPE	PEP	MPE	PEP	MPE	PEP	MPE
32	17%	56%	3%	37%	-0.3%	−51%	15%	52%	−0.5%	33%	−0.3%	−62%	−2%	31%	−20%	7%	−0.2%	−94%
128	16%	55%	2%	36%	−1%	−52%	13%	51%	−1%	32%	−1%	−62%	−1%	32%	−20%	6%	−0.2%	−94%
512	6%	42%	−5%	26%	−4%	−54%	3%	38%	−8%	22%	−4%	−63%	−1%	31%	−21%	5%	−0.6%	−94%
2048	**−48%**	**−29%**	**−50%**	**−32%**	−16%	−59%	**−50%**	**−32%**	**−53%**	**−36%**	−16%	−67%	−14%	14%	−30%	−6%	−2.5%	−94%

PEP stands for PEPITA, MPE stands for MEMPEPITA, ACT stands for ACTIVATIONS.

7. Discussion

Section 6 explained that forward learning algorithms are capable to reduce activations memory and complexity for large number of tokens in context. This is due to the BP's complexity that grows as $\mathcal{O}(n_{ctx}^3)$, while PEPITA's and MEMPEPITA's as $\mathcal{O}(n_{ctx}^2)$. An ablation study was conducted on the simulations and revealed that the different growth rates are due to the backward pass of the gradient associated to the softmax function in the Attention block, which requires $n_{ctx}^3 h$ MACCs with h number of heads. More in details, given a vector $x = \begin{bmatrix} x_1 & x_2 & \dots & x_{n_{ctx}} \end{bmatrix}$ and indicating its softmax as $s = \begin{bmatrix} s_1 & s_2 & \dots & s_{n_{ctx}} \end{bmatrix}$, the backward pass is formulated by (5) and its complexity is n_{ctx}^2 MACCs. In the attention layer, this operation is performed for each of the n_{ctx} tokens and for each head; hence, resulting in the complexity previously discussed. Furthermore, PEPITA's lower FLOPs complexity w.r.t. to BP for low values of n_{ctx} hints at a computational imbalance between the forward and the backward pass. The main reason for this difference is the differentiation procedures involved at some layers. The ones that contribute the most are the derivative at the *layernorm* layer and at the attention block. Such disparity does not emerge from the sole analysis of MACCs, as most of the computations considered are not traceable to matrix multiplications. MEMPEPITA consistently showed to be approximately one third more complex than PEPITA. As illustrated in Table 1, indeed, the former requires three forward passes and one weight update, while the latter only two forward passes and one weight update. MEMPEPITA's ability to save activations memory footprint compared to BP and PEPITA, seems to perform similarly in the DistilBERT and GPT-3 Small scenarios achieving about two thirds reductions, whereas in the case of AlexaTM they amount to the staggering value of 94%. This corresponds to a trend already mentioned in [22]. With the growing number of layers MEMPEPITA's reductions are emphasized as its activations memory requirements remain practically unaffected, while the ones of BP and PEPITA increase accordingly. As a matter of fact, DistilBERT and GPT-3 Small are characterized by the same number of encoder/decoder blocks, i.e., 12, while AlexaTM has got far more, i.e., 46 encoder blocks and 32 decoder blocks.

8. Applicability to Consumer Edge Devices

Over the last decade, supervised deep learning has proliferated across consumer edge devices, such as embedded and mobile processors, making them one of the preferred targets for supervised AI model deployment [62,63]. To investigate the applicability of ODL workloads to consumer edge devices, the memory and latency constraints for such consumer hardware shall be met. To quantify the memory requirements, activations are added to the model's parameters and both are assumed to be integer 8 bits (INT8). To compute latency, MACCs associated to a certain training procedure, are multiplied by the cycles needed to run each MACC and by the processor's frequency. FLOPs have not been taken into account, as MACCs are dominant and represent the most compute-intensive part of the workload. The analysis is performed for $n_{ctx} = 1024$, which are

approximately 800 words (equivalent to a five-minutes conversation) and a batch size of 1. More specifically, latency was calculated on DistilBERT and GPT-3 Small using the specifics of the ARM1176JZF-S processor featured by Raspberry Pi 1 and on AlexaTM considering Snapdragon 8 Gen 2 processor. The ARM1176JZF-S processor is operating at 700 MHz and is reported to spend 1 cycles per single precision MACC. This convention will be adopted, even if parameters and activations have been considered to be quantized INT8. On the other hand, Snapdragon 8 Gen 2 supports four CPU clusters, for a total of 8 cores (ARM Cortex-X3 3.2 GHz, 2 ARM Cortex-A715 2.8 GHz, 2 ARM Cortex-A710 2.8 GHz and 3 ARM Cortex-A510 2 GHz), making the latency estimation non-trivial. For computing the latency, it is assumed that the every core can perform simultaneously 1 MACC per cycle, resulting in 8 MACCs per cycle. The reference frequency corresponds to the lowest one, i.e., 2 GHz. In Figure 2, the results of the latency estimation are reported in minutes. About DistilBERT, PEPITA spends 7 min to perform a single weight update on one sample, while BP 8 min and MEMPEPITA 9.5 min. Similarly, in GPT-3 Small PEPITA takes 13 min compared to 15.5 and 17.5 min used by BP and MEMPEPITA respectively. For AlexaTM, PEPITA spends 32.5 min, BP 34 and MEMPEPITA 43.5. The latency proportions between the learning procedures reflect the trends already identified during the computational complexity analysis, being latency linearly proportional to the latter. In the case of very large models such as AlexaTM, however, the additional latency introduced by MEMPEPITA w.r.t. BP can be as much as 10 min, which in some scenarios can be a limiting factor. However, considering that memory access operations were not taken into account, MEMPEPITA's ability to avoid activations storage could lead to better performances. Regarding RAM, Figure 3 reports the memory footprint of the different learning procedures on the three LLMs evidencing the parameters' and the activations' contributions. For DistilBERT and GPT-3 Small, MEMPEPITA's activations reductions are critical to fit the Meta Training and Inference Accelerator (MTIA) [64] on-chip RAM requirements (128 MB). For AlexaTM, due to the large number of parameters (19.75 billions), activations savings introduced by MEMPEPITA do not affect total memory footprint as much as in the previous cases. All three training procedures fit the 24 GB Snapdragon 8 Gen 2 SDRAM.

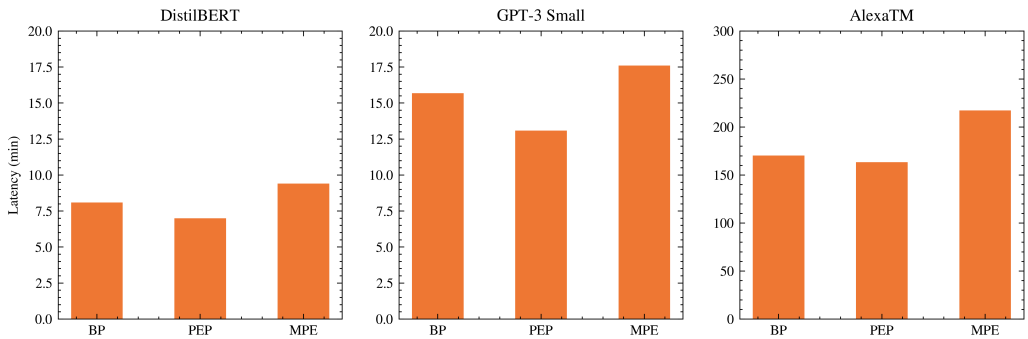

Figure 2. Latency estimation on the three LLMs.

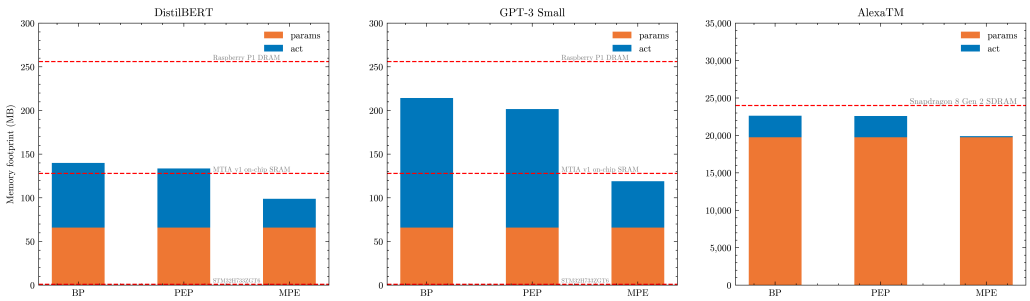

Figure 3. Memory footprint estimation on the three LLMs.

9. Conclusions and Future Works

This paper compared memory and computational complexity for "Forward-only" ODL learning workloads on three different Transformer models and against BP. For low number of tokens in context, PEPITA increases marginally or reduces substantially, depending on the model, arithmetic operations. On the other hand, for large number of tokens in context, PEPITA reduces drastically the computational complexity by 30% to 50%. MEMPEPITA increases consistently PEPITA's complexity by one third. About memory, PEPITA and BP require a similar amount of activations memory, while MEMPEPITA reduces it by 50% to 94% with its effect being more predominant for architectures with a long sequence of blocks. The study on the applicability on existing hardware shows that MEMPEPITA's activations reductions result to be critical for fitting the tight memory requirements of edge devices.

Future work will be aimed to prove implementations of PEPITA and MEMPEPITA on Transformers and will analyze its accuracy on different NLP datasets. PEPITA is not capable to support learning for NN topologies with more than three to fours layers [65]. Hence, the underlying learning mechanism of PEPITA should be closely inspected and appropriate adjustments to the algorithm should be made to improve applicability to deep NN. Also orthogonal memory reduction methods, like quantization and pruning shall be further investigated for PEPITA. The time-intensive nature of parameter update operations in LLMs remains a substantial challenge, especially when considering applications on consumer devices. Consequently, it is essential to investigate methods to decrease the duration of these updates and the associated memory consumption.

Author Contributions: Conceptualization, D.P.P. and F.M.A.; methodology, D.P.P. and F.M.A.; investigation, D.P.P. and F.M.A.; resources, D.P.P. and F.M.A.; writing—original draft preparation, writing—review and editing, D.P.P. and F.M.A.; supervision, D.P.P.; project administration, D.P.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: All the data is publicly available.

Conflicts of Interest: Authors Danilo Pau and Fabrizio Aymone were employed by the company STMicroelectronics Srl. They declare no conflict of interest.

References

1. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention is All you Need. In *Advances in Neural Information Processing Systems*; Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2017; Volume 30.
2. Team, N.; Costa-jussà, M.R.; Cross, J.; Çelebi, O.; Elbayad, M.; Heafield, K.; Heffernan, K.; Kalbassi, E.; Lam, J.; Licht, D.; et al. No Language Left Behind: Scaling Human-Centered Machine Translation. *arXiv* **2022**, arXiv:2207.04672.
3. Zhang, Z.; Yang, J.; Zhao, H. Retrospective Reader for Machine Reading Comprehension. *arXiv* **2020**, arXiv:2001.09694
4. Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; Liu, P.J. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *arXiv* **2020**, arXiv:1910.10683.
5. Jiang, H.; He, P.; Chen, W.; Liu, X.; Gao, J.; Zhao, T. SMART: Robust and Efficient Fine-Tuning for Pre-trained Natural Language Models through Principled Regularized Optimization. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Online, 5–10 July 2020; Association for Computational Linguistics: Pittsburgh, PA, USA, 2020. [CrossRef]
6. Yang, Z.; Dai, Z.; Yang, Y.; Carbonell, J.; Salakhutdinov, R.; Le, Q.V. XLNet: Generalized Autoregressive Pretraining for Language Understanding. *arXiv* **2020**, arXiv:1906.08237.
7. Wang, S.; Fang, H.; Khabsa, M.; Mao, H.; Ma, H. Entailment as Few-Shot Learner. *arXiv* **2021**, arXiv:2104.14690
8. Aghajanyan, A.; Shrivastava, A.; Gupta, A.; Goyal, N.; Zettlemoyer, L.; Gupta, S. Better Fine-Tuning by Reducing Representational Collapse. *arXiv* **2020**, arXiv:2008.03156.
9. Kaplan, J.; McCandlish, S.; Henighan, T.; Brown, T.B.; Chess, B.; Child, R.; Gray, S.; Radford, A.; Wu, J.; Amodei, D. Scaling Laws for Neural Language Models. *arXiv* **2020**, arXiv:2001.08361.
10. Brown, T.B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language Models are Few-Shot Learners. *arXiv* **2020**, arXiv:2005.14165.
11. Kim, S.; Hooper, C.; Wattanawong, T.; Kang, M.; Yan, R.; Genc, H.; Dinh, G.; Huang, Q.; Keutzer, K.; Mahoney, M.W.; et al. Full Stack Optimization of Transformer Inference: A Survey. *arXiv* **2023**, arXiv:2302.14017.

12. Han, S.; Mao, H.; Dally, W.J. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In Proceedings of the International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, 2–4 May 2016.
13. Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; Zhang, C. Learning Efficient Convolutional Networks through Network Slimming. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22 October 2017.
14. Wang, K.; Liu, Z.; Lin, Y.; Lin, J.; Han, S. HAQ: Hardware-Aware Automated Quantization with Mixed Precision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2019.
15. Hinton, G.; Vinyals, O.; Dean, J. Distilling the Knowledge in a Neural Network. *arXiv* **2015**, arXiv:1503.02531.
16. Bayram, F.; Ahmed, B.S.; Kassler, A. From Concept Drift to Model Degradation: An Overview on Performance-Aware Drift Detectors. *arXiv* **2022**, arXiv:2203.11070.
17. Pau, D.P.; Ambrose, P.K.; Aymone, F.M. A Quantitative Review of Automated Neural Search and On-Device Learning for Tiny Devices. *Chips* **2023**, *2*, 130–141. [CrossRef]
18. Dellaferrera, G.; Kreiman, G. Error-driven Input Modulation: Solving the Credit Assignment Problem without a Backward Pass. *arXiv* **2022**, arXiv:2201.11665.
19. Sanh, V.; Debut, L.; Chaumond, J.; Wolf, T. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In Proceedings of the NeurIPS EMC^2 Workshop, Vancouver, BC, Canada, 13 December 2019.
20. Soltan, S.; Ananthakrishnan, S.; FitzGerald, J.G.M.; Gupta, R.; Hamza, W.; Khan, H.; Peris, C.; Rawls, S.; Rosenbaum, A.; Rumshisky, A.; et al. AlexaTM 20B: Few-shot learning using a large-scale multilingual seq2seq model. *arXiv* **2022**, arXiv:2208.01448.
21. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. [CrossRef]
22. Pau, D.P.; Aymone, F.M. Suitability of Forward-Forward and PEPITA Learning to MLCommons-Tiny benchmarks. In Proceedings of the 2023 IEEE International Conference on Omni-Layer Intelligent Systems (COINS), Berlin, Germany, 23–25 July 2023; pp. 1–6. [CrossRef]
23. Ramachandran, P.; Liu, P.J.; Le, Q.V. Unsupervised Pretraining for Sequence to Sequence Learning. *arXiv* **2016**, arXiv:1611.02683.
24. Dai, A.M.; Le, Q.V. Semi-supervised sequence learning. *Adv. Neural Inf. Process. Syst.* **2015**, *28*. [CrossRef]
25. Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; Kuksa, P. Natural Language Processing (almost) from Scratch. *J. Mach. Learn. Res.* **2010**, *12*, 2493–2537.
26. Peters, M.E.; Ammar, W.; Bhagavatula, C.; Power, R. Semi-supervised sequence tagging with bidirectional language models. *arXiv* **2017**, arXiv:1705.00108.
27. Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I. Improving Language Understanding with Unsupervised Learning. 2018. Available online: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf (accessed on 6 September 2023).
28. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
29. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv* **2019**, arXiv:1810.04805.
30. Chowdhery, A.; Narang, S.; Devlin, J.; Bosma, M.; Mishra, G.; Roberts, A.; Barham, P.; Chung, H.W.; Sutton, C.; Gehrmann, S.; et al. PaLM: Scaling Language Modeling with Pathways. *arXiv* **2022**, arXiv:2204.02311.
31. Barbuto, V.; Savaglio, C.; Chen, M.; Fortino, G. Disclosing Edge Intelligence: A Systematic Meta-Survey. *Big Data Cogn. Comput.* **2023**, *7*, 44. [CrossRef]
32. Yuan, J.; Yang, C.; Cai, D.; Wang, S.; Yuan, X.; Zhang, Z.; Li, X.; Zhang, D.; Mei, H.; Jia, X.; et al. Rethinking Mobile AI Ecosystem in the LLM Era. *arXiv* **2023**, arXiv:2308.1436.
33. Alizadeh, K.; Mirzadeh, I.; Belenko, D.; Khatamifard, K.; Cho, M.; Mundo, C.C.D.; Rastegari, M.; Farajtabar, M. LLM in a flash: Efficient Large Language Model Inference with Limited Memory. *arXiv* **2024**, arXiv:2312.11514.
34. Li, T.; Mesbahi, Y.E.; Kobyzev, I.; Rashid, A.; Mahmud, A.; Anchuri, N.; Hajimolahoseini, H.; Liu, Y.; Rezagholizadeh, M. A short study on compressing decoder-based language models. arXiv **2021**, arXiv:2110.08460.
35. Ganesh, P.; Chen, Y.; Lou, X.; Khan, M.A.; Yang, Y.; Sajjad, H.; Nakov, P.; Chen, D.; Winslett, M. Compressing Large-Scale Transformer-Based Models: A Case Study on BERT. *Trans. Assoc. Comput. Linguist.* **2021**, *9*, 1061–1080. [CrossRef]
36. Sun, Z.; Yu, H.; Song, X.; Liu, R.; Yang, Y.; Zhou, D. MobileBERT: A Compact Task-Agnostic BERT for Resource-Limited Devices. *arXiv* **2020**, arXiv:2004.02984.
37. Jiao, X.; Yin, Y.; Shang, L.; Jiang, X.; Chen, X.; Li, L.; Wang, F.; Liu, Q. TinyBERT: Distilling BERT for Natural Language Understanding. *arXiv* **2020**, arXiv:1909.10351.
38. Ding, J.; Ma, S.; Dong, L.; Zhang, X.; Huang, S.; Wang, W.; Zheng, N.; Wei, F. LongNet: Scaling Transformers to 1,000,000,000 Tokens. *arXiv* **2023**, arXiv:2307.02486.
39. Shen, Z.; Zhang, M.; Zhao, H.; Yi, S.; Li, H. Efficient Attention: Attention with Linear Complexities. *arXiv* **2020**, arXiv:1812.01243.
40. Gómez-Luna, J.; Guo, Y.; Brocard, S.; Legriel, J.; Cimadomo, R.; Oliveira, G.F.; Singh, G.; Mutlu, O. An Experimental Evaluation of Machine Learning Training on a Real Processing-in-Memory System. *arXiv* **2023**, arXiv:2207.07886.
41. Cai, H.; Gan, C.; Zhu, L.; Han, S. TinyTL: Reduce Memory, Not Parameters for Efficient On-Device Learning. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 11285–11297.

42. Lin, J.; Zhu, L.; Chen, W.M.; Wang, W.C.; Gan, C.; Han, S. On-Device Training Under 256KB Memory. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 22941–22954.
43. Chowdhery, A.; Warden, P.; Shlens, J.; Howard, A.; Rhodes, R. Visual Wake Words Dataset. *arXiv* **2019**, arXiv:1906.05721.
44. Zaken, E.B.; Ravfogel, S.; Goldberg, Y. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv* **2021**, arXiv:2106.10199.
45. Vucetic, D.; Tayaranian, M.; Ziaeefard, M.; Clark, J.J.; Meyer, B.H.; Gross, W.J. Efficient Fine-Tuning of BERT Models on the Edge. In Proceedings of the 2022 IEEE International Symposium on Circuits and Systems (ISCAS), Austin, TX, USA, 27 May 2022–1 June 2022; pp. 1838–1842. [CrossRef]
46. Warstadt, A.; Singh, A.; Bowman, S.R. Neural Network Acceptability Judgments. *arXiv* **2018**, arXiv:1805.12471.
47. Xi, H.; Li, C.; Chen, J.; Zhu, J. Training Transformers with 4-bit Integers. *arXiv* **2023**, arXiv:2306.11987.
48. Crick, F. The recent excitement about neural networks. *Nature* **1989**, *337*, 129–132. [CrossRef]
49. Lillicrap, T.; Santoro, A.; Marris, L.; Akerman, C.; Hinton, G. Backpropagation and the brain. *Nat. Rev. Neurosci.* **2020**, *21*, 335–346.
50. Lillicrap, T.; Cownden, D.; Tweed, D.; Akerman, C. Random synaptic feedback weights support error backpropagation for deep learning. *Nat. Commun.* **2016**, *7*, 13276. [CrossRef]
51. Nøkland, A. Direct Feedback Alignment Provides Learning in Deep Neural Networks. In Proceedings of the Advances in Neural Information Processing Systems 29, Barcelona, Spain, 5–10 December 2016.
52. Hinton, G. The Forward-Forward Algorithm: Some Preliminary Investigations. *arXiv* **2022**, arXiv:2212.13345.
53. Burbank, K.S.; Kreiman, G. Depression-Biased Reverse Plasticity Rule Is Required for Stable Learning at Top-Down Connections. *PLoS Comput. Biol.* **2012**, *8*, e1002393. [CrossRef]
54. Jaderberg, M.; Czarnecki, W.M.; Osindero, S.; Vinyals, O.; Graves, A.; Silver, D.; Kavukcuoglu, K. Decoupled Neural Interfaces using Synthetic Gradients. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 11–15 August 2017; Precup, D., Teh, Y.W., Eds.; PMLR: London, UK, 2017; Volume 70, pp. 1627–1635.
55. Czarnecki, W.M.; Świrszcz, G.; Jaderberg, M.; Osindero, S.; Vinyals, O.; Kavukcuoglu, K. Understanding Synthetic Gradients and Decoupled Neural Interfaces. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; Precup, D., Teh, Y.W., Eds.; PMLR: London, UK, 2017; Volume 70, pp. 904–912.
56. Liao, Q.; Leibo, J.Z.; Poggio, T. How Important is Weight Symmetry in Backpropagation? *arXiv* **2016**, arXiv:1510.05067.
57. Banbury, C.; Reddi, V.J.; Torelli, P.; Holleman, J.; Jeffries, N.; Kiraly, C.; Montino, P.; Kanter, D.; Ahmed, S.; Pau, D.; et al. MLCommons Tiny Benchmark. *arXiv* **2021**, arXiv:2106.07597.
58. Akrout, M.; Wilson, C.; Humphreys, P.C.; Lillicrap, T.; Tweed, D. Deep Learning without Weight Transport. *arXiv* **2020**, arXiv:1904.05391.
59. Clark, K.; Luong, M.T.; Le, Q.V.; Manning, C.D. Pre-Training Transformers as Energy-Based Cloze Models. *arXiv* **2020**, arXiv:2012.08561.
60. Pau, D.P.; Aymone, F.M. Mathematical Formulation of Learning and Its Computational Complexity for Transformers' Layers. *Eng. Proc.* **2024**, *5*, 34–50. [CrossRef]
61. Forward Learning of Large Language Models by Consumer Devices Github Repository. Available online: https://github.com/fabrizioaymone/forward-learning-of-LLMs-to-consumer-devices (accessed on 6 September 2023).
62. Laskaridis, S.; Venieris, S.I.; Kouris, A.; Li, R.; Lane, N.D. The Future of Consumer Edge-AI Computing. *arXiv* **2022**, arXiv:2210.10514.
63. Morra, L.; Mohanty, S.P.; Lamberti, F. Artificial Intelligence in Consumer Electronics. *IEEE Consum. Electron. Mag.* **2020**, *9*, 46–47. [CrossRef]
64. Firoozshahian, A.; Coburn, J.; Levenstein, R.; Nattoji, R.; Kamath, A.; Wu, O.; Grewal, G.; Aepala, H.; Jakka, B.; Dreyer, B.; et al. MTIA: First Generation Silicon Targeting Meta's Recommendation Systems. In Proceedings of the 50th Annual International Symposium on Computer Architecture, Orlando, FL, USA, 17–21 June 2023; pp. 1–13.
65. Srinivasan, R.F.; Mignacco, F.; Sorbaro, M.; Refinetti, M.; Cooper, A.; Kreiman, G.; Dellaferrera, G. Forward Learning with Top-Down Feedback: Empirical and Analytical Characterization. *arXiv* **2023**, arXiv:2302.05440.