

## Article

# Mathematical Formulation of Learning and Its Computational Complexity for Transformers' Layers

Danilo Pietro Pau \* and Fabrizio Maria Aymone 

Department of Systems Research and Applications, STMicroelectronics, 20864 Agrate Brianza, Italy;  
fabrizio.aymone@gmail.com

\* Correspondence: danilo.pau@st.com

**Abstract:** Transformers are the cornerstone of natural language processing and other much more complicated sequential modelling tasks. The training of these models, however, requires an enormous number of computations, with substantial economic and environmental impacts. An accurate estimation of the computational complexity of training would allow us to be aware in advance about the associated latency and energy consumption. Furthermore, with the advent of forward learning workloads, an estimation of the computational complexity of such neural network topologies is required in order to reliably compare backpropagation with these advanced learning procedures. This work describes a mathematical approach, independent from the deployment on a specific target, for estimating the complexity of training a transformer model. Hence, the equations used during backpropagation and forward learning algorithms are derived for each layer and their complexity is expressed in the form of MACCs and FLOPs. By adding all of these together accordingly to their embodiment into a complete topology and the learning rule taken into account, the total complexity of the desired transformer workload can be estimated.

**Keywords:** complexity; transformers; backpropagation; PEPITA; large language models



**Citation:** Pau, D.P.; Aymone, F.M. Mathematical Formulation of Learning and Its Computational Complexity for Transformers' Layers. *Eng* **2024**, *5*, 34–50. <https://doi.org/10.3390/eng5010003>

Academic Editor: Jordan Hristov

Received: 20 October 2023

Revised: 26 November 2023

Accepted: 7 December 2023

Published: 21 December 2023



**Copyright:** © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Transformers [1] have revolutionized the field of artificial intelligence (AI) by achieving unprecedented accuracy results over a broad variety of complex tasks, including natural language processing (NLP). Their performance, however, has been proven analytically to scale as a power law of the number of parameters of the model and the dataset size [2]. The inevitable consequences of such dependence are bigger model footprints, larger datasets and an increasing number of gradient descent iterations. The associated enormous number of computations and memory usage has critical economic, time and environmental impacts. As an example, the recent 175-billion-parameter GPT-3 [3] was trained for 300 billion tokens [4] with a total compute of ~3640 petaflop/s-days. According to [5], this process emitted 502 tonnes of carbon dioxide and cost 1.8 million USD. Given the pivotal role computational complexity plays when training a transformer, it is fundamental to provide for it an accurate estimation. Moreover, a variety of alternative learning rules to backpropagation (BP) were recently proposed [6,7]. A precise complexity analysis would allow one to reliably compare these algorithms and quantifying the advantages that forward learning would bring with respect to BP. Ideally, the training complexity should be obtained by considering the single operations performed at the hardware level. This is not convenient for several reasons. First of all, it strictly depends upon the device used, whose knowledge is generally limited, restricting the generalizability of the prediction. Secondly, it is not trivial to obtain the low-level implementation of the training algorithm, which in turn depends on the AI runtime library being used. This paper is organized as follows: Section 2 describes the objectives of this work and its contributions; Section 3 cites the main related works in the known literature; Section 4 describes the notation and conventions adopted

during the quantitative analysis of the learning algorithms; Section 5 reports the equations for learning with BP, PEPITA and MEMPEPITA and estimates their complexity for all transformer layers; Section 6 presents an example application of the results obtained, and Section 7 concludes the paper.

## 2. Key Contributions of This Work

In order to eliminate the dependence from the deployment device, this study introduces a mathematical approach that assesses the complexity of layers in a transformer topology solely based on mathematical expressions, rather than considering hardware operations. In this respect, the contributions brought by this paper can be summarized as follows:

- A description of the equations implemented in BP, PEPITA and MEMPEPITA for all transformer layers;
- A mathematical derivation of the weights' and activations' gradient with respect to the loss function;
- Quantitative complexity analysis in terms of multiply and accumulate (MACCs) and floating point operations (FLOPs) of each layer for the forward pass, backward pass and weight updates.

## 3. Related Works

### 3.1. Automatic Differentiation

Currently, most AI algorithms are implemented in two major libraries: Tensorflow [8] and Pytorch [9]. Both of the aforementioned frameworks use reverse-mode automatic differentiation (i.e., autodiff) [10], namely BP. Autodiff creates a computational graph from the mathematical expression considered, where each node describes an operation and each edge a variable. During the forward pass, intermediate variables are populated, and each node is complemented with the derivatives of the outputs with respect to the inputs. During the backward pass, the gradient is obtained by leveraging the chain rule of differential calculus to compute partial derivatives of the objective with respect to the weights. The operations involved during training can be referred to as forward pass, backward pass (gradient of the loss with respect to the activations) and weight update (gradient of the loss with respect to the weights). In several works [3,11], the complexity of the backward pass and weight update is assumed to be  $2\times$  that of the forward pass, which is true only in certain specific cases such as fully connected and convolutional layers. Regarding transformers, Ref. [2] empirically identified the relations between performance, training time, dataset size, number of parameters and amount of computation. Moreover, also in this case, the computational complexity of training was approximated to be  $3\times$  that of a forward pass. To the best of the authors' knowledge there is no work that has analytically described the computational complexity of the transformer topology.

### 3.2. Alternatives to Backpropagation

It is well known from theory that BP is not describing the learning process happening in the human brain [12,13]. There are four main aspects considered to be in contrast with neuro-biological observations. Firstly, during the backward pass, weights previously used during the forward pass are utilized to backpropagate the error. Considering that synapses in the brain are unidirectional, this characteristic of BP gives rise to the "weight symmetry" problem [14]. Secondly, when calculating the error gradient, the activities of the neurons computed during the forward pass are left unaffected. The freezing of the activities during the backward pass is incompatible with the behaving of feedback connections in neural circuits, through which the signal travels via modulating activities [15]. Thirdly, the modification of synaptic weights is influenced by downstream neurons and synapses, whereas synaptic learning in the brain is predominantly governed by localized signals that are contingent upon the activity of the interconnected neurons [16]. Lastly, in order to update the weights of the  $l$ -th layer, the forward pass has to end, and the backward pass

has to arrive at such layer. This means that learning cannot happen in an online fashion, contrarily to biological evidence. Such problem is referred to as “update-locking” [17,18]. With the hope of creating a correspondence between deep learning and brain nature, a vast research field focused on finding biologically plausible alternatives to BP has emerged. Addressing the “weight symmetry” issue, learning has been found to happen even when the error is backpropagated with matrices which only share the sign with the forward weights [19] or that are random and fixed like in feedback alignment (FA) [20]. The latter can be modified by propagating directly the error from the output layer to each layer through random connectivity matrices. Such technique is denoted as direct feedback alignment [21]. A broad variety of other algorithms have been proposed in literature [22–26]; however, their knowledge goes beyond the scope of this work.

### 3.3. Forward Learning

Recently, two promising bioplausible learning algorithms were proposed: forward-forward [6] and PEPITA [7]. The use of forward-only passes solves several implausible aspects of BP. Their effect on memory usage and computational complexity has been studied by [27] on the MLCommons/Tiny industrial benchmarks [28], suggesting that FF is unsuitable for multiclass classification [27]. Moreover, Ref. [27] proposed MEMPEPITA, a memory-efficient version of PEPITA, which introduces an additional forward pass saving on average a third of RAM at the expense of a third more complexity.

#### 3.3.1. PEPITA

PEPITA [7] performs two forward passes. The first pass, named standard pass, calculates the error of the model’s output with respect to the ground truth. 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 a small standard deviation (e.g.,  $0.05\sqrt{\frac{2}{FANIN}}$  [29]). The second pass, named modulated, transforms the input by adding to it the projected 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. The weights of the last layer can be updated by the error at the output layer as in BP without compromising accuracy [7]. Algorithm 1 illustrates the procedure implemented in PEPITA, where  $a_0$ ,  $a_l$  and  $a_L$  are the activations of the first,  $l$ -th and last layer, respectively, during the standard pass,  $a_0^{err}$ ,  $a_l^{err}$  and  $a_L^{err}$  are the activations of the first,  $l$ -th and last layer, respectively, during the modulated pass,  $\sigma_l$  is the nonlinearity of the  $l$ -th layer, and  $W_l$  are the weights of the  $l$ -th layer. A theoretical analysis of the learning dynamics of PEPITA was performed in [29]. By observing that the perturbation is small compared to the input  $\|Fe\| \ll \|x\|$ , it was possible to perform a Taylor expansion of the presynaptic term  $a_l - a_l^{err}$  thus obtaining the update rule for the first layer, as described in Equation (1). It was considered that  $W(t+1) = W(t) - \eta \Delta W$ , with  $\eta$  symbolizing the learning rate, and  $x$  was used instead of  $(x - Fe)$  since the small perturbation was determined to have a negligible impact on performance.

$$\Delta W_1 \approx \left[ (W_1 Fe) \odot a_1' \right] x^T \quad (1)$$

PEPITA essentially adopts an update similar to DFA and equivalent to FA in two-layer networks. However, it uniquely employs an adaptive feedback matrix (AF) in which the network weights modulate the random component. In such a way, the learning effect of PEPITA found experimentally was justified theoretically.

**Algorithm 1: PEPITA**


---

**Given:** Features( $x$ ) and label( $target$ )

**Standard Pass**

$$a_0 = x$$

**for**  $\ell = 1, \dots, L$  **do**

$$a_\ell = \sigma_\ell(W_\ell a_{\ell-1})$$

**end for**

$$e = a_L - target$$

**Modulated pass**

$$a_0^{err} = x + Fe$$

**for**  $\ell = 1, \dots, L$  **do**

$$a_\ell^{err} = \sigma_\ell(W_\ell a_{\ell-1}^{err})$$

**Weight update**

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

**end for**

---

**3.3.2. MEMPEPITA**

In its original form, the PEPITA algorithm [7] necessitates retaining activations calculated during the standard computational pass for the subsequent evaluation of  $(a_\ell - a_\ell^{err})$  in the modulated pass. This requirement unfortunately aligns with the memory demands characteristic of backpropagation (BP). To circumvent this memory constraint, one could introduce a concurrent secondary standard pass alongside the modulated pass. This approach enables the recalculation of necessary activations for the weight update process. However, this solution does introduce an additional computational overhead. This variant of the original algorithm, termed MEMPEPITA, is presented in [27] and significantly enhances memory efficiency by avoiding the intermediate activations' storage, which is detrimental in deep neural networks (DNNs). This variant, detailed in Algorithm 2, while maintaining the core principles of PEPITA, offers a more resource-conscious alternative, particularly in scenarios where memory resources are a critical constraint.

**Algorithm 2: MEMPEPITA**


---

**Given:** Features( $x$ ) and label( $target$ )

**Standard Pass**

$$a_0 = x$$

**for**  $\ell = 1, \dots, L$  **do**

$$a_\ell = \sigma_\ell(W_\ell a_{\ell-1})$$

**end for**

$$e = a_L - target$$

**Modulated + 2nd Standard pass**

$$a_0^{err} = x + Fe$$

**for**  $\ell = 1, \dots, L$  **do**

**Standard Pass**

$$a_\ell = \sigma_\ell(W_\ell a_{\ell-1})$$

**Modulated pass**

$$a_\ell^{err} = \sigma_\ell(W_\ell a_{\ell-1}^{err})$$

**Weight update**

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

**end for**

---

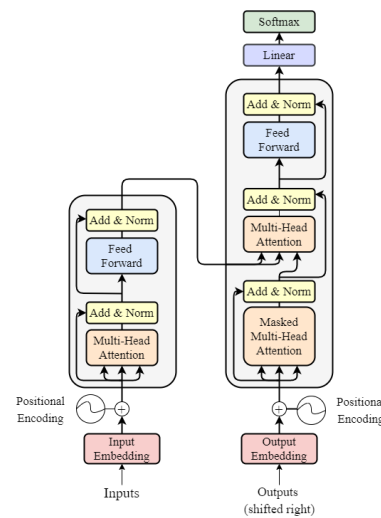
**4. Notation and Conventions**

The objective of the quantitative analysis in this paper is to accurately model the mathematical equations behind BP, PEPITA and MEMPEPITA for estimating the computational complexity [30–32] of training the transformer architecture. Therefore, it is necessary to

clearly define beforehand the notations and conventions used in the proposed analysis. Each “mathematical” operation (e.g., exponentiation, sum, product, division) is considered a FLOP of 32 bits even if the underlying hardware may require performing more operations. Hence, each MACC is equivalent to two FLOPs, one ADD and one MULTIPLY [33]. Even if a MULTIPLY operation is more complex than an ADD operation when implemented on hardware, this work considers them to be both equivalent to one FLOP as they both consist in one mathematical operation.

For the sake of a clear and lean notation, the symbol  $\frac{\partial y}{\partial x}$  (i.e., partial derivative) is used to indicate the gradient, whose adequate symbol would be  $\nabla_x y$ . Such a choice was determined by the fact that  $\frac{\partial y}{\partial x}$  highlights the target with respect to which the gradient is computed. Moreover, for each layer, the total number of MACCs and FLOPs estimated for the macro-operations (forward pass, backward pass, weight update, etc.) are framed in a box to highlight them. Lastly, a new operator indicated with  $\times_{slice}$  is introduced. This operator receives a 2d matrix of size  $N \times M$  as a left operand and a 3d matrix of size  $N \times M \times K$  as a right operand and outputs a 2d matrix of size  $N \times K$ . The operator multiplies the first row of the left operand by the first 2d matrix in the 3d matrix’s right operand, obtaining a row vector which corresponds to the first row of the output 2d matrix. Then, it obtains the second row of the output matrix by multiplying the second row of the left operand by the second 2d matrix of the 3d matrix’s right operand. This process is iterated  $N$  times for each row in the 2d matrix’s left operand.

The transformer is composed of an encoder and a decoder, and its architecture is reported in Figure 1 [1]. Given such structure, there is a collection of hyperparameters needed to uniquely identify a specific architecture embodiment. The latter are reported in Table 1 and they are used as parameters throughout the analysis.



**Figure 1.** Transformer architecture.

**Table 1.** Architecture hyperparameters.

Name	Description
$voc_{size}$	Number of word/tokens in the corpus
$d_{model}$	Dimension of embeddings
$d_k$	Dimension of the single attention head
$d_{ff}$	Dimension of the first layer in the feed forward network
$n_{enc}$	Number of encoder layers
$n_{dec}$	Number of decoder layers
$max_{len}$	Maximum number of tokens in the context

## 5. Complexity Analysis

The method adopted for estimating the complexity of a specific learning procedure involves subdividing the latter in a series of macro-operations (e.g., forward pass, backward pass, weight update, error projection), as reported in Table 2. The total complexity of a macro-operation is obtained by calculating the complexity of performing such a macro-operation at each single layer of the transformer and adding it for all layers. In the following paragraphs, the structure and functionality of each layer is described and their complexity for the forward pass, backward pass and weight update is computed.

**Table 2.** 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
Error projection	0	1	1

PEP stands for PEPITA and MPE for MEMPEPITA.

### 5.1. Embedding Layer

The embedding Layer consists of a matrix of  $W_{emb}$  size  $voc_{size} \times d_{model}$ , where each row corresponds to the embedding of a token.

#### 5.1.1. Forward Pass

Given a sequence of  $N$  tokens, these are represented as a matrix  $T$  of size  $M \times voc_{size}$ , where each row is a one-hot-encoded representation of the token. By multiplying the token matrix with the embedding matrix, a matrix  $E = TW_{emb}$  of size  $M \times d_{model}$  is obtained, where each row corresponds to the embedding representation of the original token in the sequence. Hence, the complexity of an embedding layer for a forward pass is, as in [11],

$$\text{MACCs} = M \times voc_{size} \times d_{model}$$

$$\text{FLOPs} = 2M \times voc_{size} \times d_{model}$$

#### 5.1.2. Weight Update (Only PEPITA and MEMPEPITA)

In BP, the gradient of the loss function with respect to the input tokens is directly calculated during the backward pass of the next layer, without involving the embedding matrix. Such gradient is directly used for updating the rows of the embedding layer corresponding to the tokens considered. Therefore, in BP no computation is needed by the embedding layer for the backward pass and weight update. On the other hand, PEPITA updates the embedding layer as for other layers, by performing a matrix multiplication. The resulting complexity of the weight update is the same as that of the forward pass.

$$\text{MACCs} = M \times voc_{size} \times d_{model}$$

$$\text{FLOPs} = 2M \times voc_{size} \times d_{model}$$

### 5.2. Position Embeddings

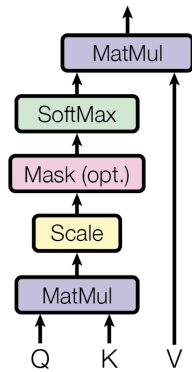
A positional embedding matrix  $P$  of size  $max_{len} \times d_{model}$  is used to store the positional embeddings for each position up to the maximum number of tokens in the context. To encode positional information, the first  $M$  rows of the positional matrix are added to  $E$ . The obtained matrix of size  $M \times d_{model}$  is indicated with  $X$ . A positional embedding matrix can be learned or it can be already assigned following the sinusoidal positional encoding

proposed in [1]. Being a simple addition operation, its complexity is negligible and can be considered to be already incorporated in the MACCs/FLOPs of the embedding layer.

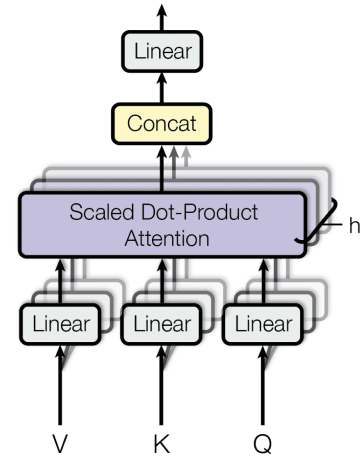
### 5.3. Multihead Attention

This is the most important block in the transformer, and it occupies the first stage of the encoder layer and the first two stages of the decoder layer. Its structure is reported in Figure 2 [1].

Scaled Dot-Product Attention



Multi-Head Attention



**Figure 2.** Structure of the attention block.

#### 5.3.1. Forward Pass

The first step consists in identifying the input query, key and value matrices  $X^Q$  of size  $M \times d_{model}$  and  $X^K, X^V$  of size  $N \times d_{model}$ . Then, for each different head  $i$ ,  $Q_i, K_i$  and  $V_i$  are computed where the number of heads  $h$  is determined by dividing  $d_{model}$  by  $d_k$ . This is achieved by multiplying the inputs with appropriate weight matrices of size  $d_{model} \times d_k$ .

$$(Q_i, K_i, V_i) = (X^Q W_i^Q, X^K W_i^K, X^V W_i^V) \quad (2)$$

$$\text{MACCs} = M \times d_{model} \times d_{model} + 2 \times N \times d_{model} \times d_{model}$$

$$\text{FLOPs} = 2M \times d_{model} \times d_{model} + 4 \times N \times d_{model} \times d_{model}$$

Then, the  $Q_i, K_i$  and  $V_i$  matrices are fed into the attention mechanism of each head. It is assumed, in accordance with the conventions adopted in [11], that computing the softmax of an array of size  $N$  requires  $5N$  FLOPs.

$$\text{Attention}(Q_i, K_i, V_i) = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i \quad (3)$$

$$\text{MACCs} = 2M \times N \times d_{model}$$

$$\text{FLOPs} = 4M \times N \times d_{model} + M \times 5N \times h + M \times N \times h$$

The output of the attention of each head is concatenated and multiplied by a weight matrix  $W^O$  of size  $d_{model} \times d_{model}$ . Summing up, the output of a multihead attention block is

$$\text{Multihead}(X^Q, X^K, X^V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) W^O \quad (4)$$

where  $\text{head}_i = \text{Attention}(Q_i, K_i, V_i)$

$$\text{MACCs} = M \times d_{model} \times d_{model}$$

$$\text{FLOPs} = 2M \times d_{model} \times d_{model}$$



The multihead attention block in the decoder builds the query matrix  $Q_i$  starting from the output of the previous masked multihead attention block in the decoder  $X_{dec}$ , while the key and value matrices are obtained from the output of the encoder stack  $X_{enc}$ . Namely,

$$(Q_i, K_i, V_i) = (X_{dec}W_i^Q, X_{enc}W_i^K, X_{enc}W_i^V) \quad (5)$$

The masked multihead attention puts a mask on the softmax output, during training, in order for tokens not to look for a correlation with the next tokens in the sequence. The complexity of this operation is not considered, as it consists in putting to  $-\inf$  certain cells of the matrix.

The total number of MACCs are

$$2M \times d_{model} \times d_{model} + 2M \times N \times d_{model} + 2N \times d_{model} \times d_{model}$$

The total number of FLOPs are

$$4M \times d_{model} \times d_{model} + 4M \times N \times d_{model} + 4N \times d_{model} \times d_{model} + 6M \times N \times h$$

### 5.3.2. Backward Pass and Weight Update

The learnable parameters in the multiheaded attention block are the weight matrices  $W_i^Q$ ,  $W_i^K$ ,  $W_i^V$  and  $W_O$ . During backpropagation, the derivatives of the output of the block with respect to these matrices and with respect to the inputs  $X^Q$ ,  $X^K$  and  $X^V$  should therefore be calculated. If it is a multihead encoder or masked multihead decoder:  $X^Q = X^K = X^V = X$ ; if it is a multihead decoder:  $X^Q = X_{dec}$  and  $X^K = X^V = X_{enc}$ . Let us denote  $f(X^Q, X^K, X^V) = \text{multiheadattention}(X^Q, X^K, X^V)$  and indicate with  $\mathcal{L}$  the loss function.

$$f = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O \quad (6)$$

$$\frac{\partial \mathcal{L}}{\partial W^O} = \text{Concat}(\text{head}_1, \dots, \text{head}_h)^T \frac{\partial \mathcal{L}}{\partial f} \quad (7)$$

$$\frac{\partial \mathcal{L}}{\partial \text{Concat}(\text{head}_1, \dots, \text{head}_h)} = \frac{\partial \mathcal{L}}{\partial f} (W^O)^T \quad (8)$$

Then, the derivatives of the attention module, considering the different heads, result in the following:

$$\text{Attention}(Q_i, K_i, V_i) = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i \quad (9)$$

$$\frac{\partial \mathcal{L}}{\partial \text{softmax}} = \frac{\partial \mathcal{L}}{\partial \text{Attention}} V_i^T \quad (10)$$

Every row of the softmax is independent from the other rows. Given a row  $[x_1 \ x_2 \ x_3 \ \dots \ x_n]$  and the softmax of that row  $[s_1 \ s_2 \ s_3 \ \dots \ s_n]$  the jacobian of the softmax with respect to the row is

$$\begin{bmatrix} s_1 \cdot (1 - s_1) & -s_2 \cdot s_1 & -s_3 \cdot s_1 & \dots & -s_n \cdot s_1 \\ -s_1 \cdot s_2 & s_2 \cdot (1 - s_2) & -s_3 \cdot s_2 & \dots & -s_n \cdot s_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -s_1 \cdot s_n & -s_2 \cdot s_n & -s_3 \cdot s_n & \dots & s_n \cdot (1 - s_n) \end{bmatrix} \quad (11)$$

It is possible to define a 3-dimensional matrix composed of the Jacobian of each row of the softmax layer, denoted as  $\frac{\partial \text{softmax}}{\partial X}$ , where  $X$  is a 2-dimensional matrix in  $\text{softmax}(X)$ . In order to maintain a concise notation, we also introduce a new type of matrix product  $\times_{\text{slice}}$



which multiplies each row of the first 2-dimensional matrix by each slice of the 3d matrix. The complexity of this product is the same as the regular matrix product.

$$\frac{\partial \mathcal{L}}{\partial Q_i K_i^T} = \frac{\partial \mathcal{L}}{\partial \text{softmax}} \times_{\text{slice}} \left( \frac{\partial \text{softmax}}{\partial \frac{Q_i K_i^T}{\sqrt{d_k}}} \right)^T \frac{1}{\sqrt{d_k}} \quad (12)$$

Now, the derivative of the loss function with respect to  $Q_i$ ,  $K_i$ ,  $V_i$  is obtained.

$$\frac{\partial \mathcal{L}}{\partial Q_i} = \frac{\partial \mathcal{L}}{\partial Q_i K_i^T} K_i \quad (13)$$

$$\frac{\partial \mathcal{L}}{\partial K_i} = \left( \frac{\partial \mathcal{L}}{\partial Q_i K_i^T} \right)^T Q_i \quad (14)$$

$$\frac{\partial \mathcal{L}}{\partial V_i} = \text{softmax}^T \frac{\partial \mathcal{L}}{\partial \text{Attention}} \quad (15)$$

The derivatives with respect to the inputs  $X^Q$ ,  $X^K$  and  $X^V$  for each head  $i$  are the following:

$$\left( \frac{\partial \mathcal{L}}{\partial X^Q} \right)_i = \frac{\partial \mathcal{L}}{\partial Q_i} (W_i^Q)^T \quad (16)$$

$$\left( \frac{\partial \mathcal{L}}{\partial X^K} \right)_i = \frac{\partial \mathcal{L}}{\partial K_i} (W_i^K)^T \quad (17)$$

$$\left( \frac{\partial \mathcal{L}}{\partial X^V} \right)_i = \frac{\partial \mathcal{L}}{\partial V_i} (W_i^V)^T \quad (18)$$

Then, the derivatives with respect to the weight matrix are computed:

$$\frac{\partial \mathcal{L}}{\partial W_i^Q} = (X^Q)^T \frac{\partial \mathcal{L}}{\partial Q_i} \quad (19)$$

$$\frac{\partial \mathcal{L}}{\partial W_i^K} = (X^K)^T \frac{\partial \mathcal{L}}{\partial K_i} \quad (20)$$

$$\frac{\partial \mathcal{L}}{\partial W_i^V} = (X^V)^T \frac{\partial \mathcal{L}}{\partial V_i} \quad (21)$$

To obtain the derivative with respect to  $X^Q$ ,  $X^K$  and  $X^V$ , it is required to add all heads. As  $X^Q$ ,  $X^K$  and  $X^V$  are the same for the encoder, they are all added together. In the decoder, only the derivatives with respect to  $X^K$  and  $X^V$  are added together as they come from the encoder. Let  $N$  be the dimension of  $X^K$  and  $X^V$  and  $M$  be the dimension of  $X^Q$ .

The complexity for each operation during the backward pass can be calculated as follows:

$$\frac{\partial \mathcal{L}}{\text{Concat}(\text{head}_1, \dots, \text{head}_h)} = \frac{\partial \mathcal{L}}{\partial f} (W^O)^T \quad (22)$$

$$\text{MACCs} = M \times d_{\text{model}} \times d_{\text{model}}$$

$$\text{FLOPs} = 2M \times d_{\text{model}} \times d_{\text{model}}$$

$$\frac{\partial \mathcal{L}}{\partial W^O} = \text{Concat}(\text{head}_1, \dots, \text{head}_h)^T \frac{\partial \mathcal{L}}{\partial f} \quad (23)$$

$$\text{MACCs} = d_{\text{model}} \times M \times d_{\text{model}}$$

$$\text{FLOPs} = 2d_{\text{model}} \times M \times d_{\text{model}}$$

In order to compute the derivative of the loss function with respect to  $X^V$  and  $W_i^V$ , the following complexities are obtained:

$$\frac{\partial \mathcal{L}}{\partial V_i} = \text{softmax}^T \frac{\partial \mathcal{L}}{\partial \text{Attention}_i} \quad (24)$$

$$\text{MACCs} = N \times M \times d_k \times h = N \times M \times d_{\text{model}}$$

$$\text{FLOPs} = 2N \times M \times d_{\text{model}}$$

$$\left( \frac{\partial \mathcal{L}}{\partial X^V} \right)_i = \frac{\partial \mathcal{L}}{\partial V_i} (W_i^V)^T \quad (25)$$

$$\frac{\partial \mathcal{L}}{\partial X^V} = \sum_{i=1}^h \left( \frac{\partial \mathcal{L}}{\partial X^V} \right)_i \quad (26)$$

$$\text{MACCs} = N \times d_k \times d_{\text{model}} \times h = N \times d_{\text{model}} \times d_{\text{model}}$$

$$\text{FLOPs} = 2N \times d_{\text{model}} \times d_{\text{model}}$$

$$\frac{\partial \mathcal{L}}{\partial W_i^V} = (X^V)^T \frac{\partial \mathcal{L}}{\partial V_i} \quad (27)$$

$$\text{MACCs} = d_{\text{model}} \times N \times d_k \times h = d_{\text{model}} \times N \times d_{\text{model}}$$

$$\text{FLOPs} = 2d_{\text{model}} \times N \times d_{\text{model}}$$

In order to compute the derivative of the loss function with respect to  $X^Q$ ,  $X^K$ ,  $W_i^Q$  and  $W_i^K$ , the following complexities are obtained:

$$\frac{\partial \text{softmax}}{\partial \frac{Q_i K_i^T}{\sqrt{d_k}}} \quad (28)$$

$$\text{MACCs} = 0$$

$$\text{FLOPs} = N \times N \times M \times h$$

$$\frac{\partial \mathcal{L}}{\partial Q_i K_i^T} = \frac{\partial \mathcal{L}}{\partial \text{Attention}} V_i^T \times_{\text{slice}} \left( \frac{\partial \text{softmax}}{\partial \frac{Q_i K_i^T}{\sqrt{d_k}}} \right)^T \frac{1}{\sqrt{d_k}} \quad (29)$$

$$\text{MACCs} = M \times d_{\text{model}} \times N + M \times N \times M \times h$$

$$\text{FLOPs} = 2M \times d_{\text{model}} \times N + 2M \times N \times M \times h + M \times N \times h$$

$$\frac{\partial \mathcal{L}}{\partial Q_i} = \frac{\partial \mathcal{L}}{\partial Q_i K_i^T} K_i \quad (30)$$

$$\text{MACCs} = M \times N \times d_{\text{model}}$$

$$\text{FLOPs} = 2M \times N \times d_{\text{model}}$$

$$\left( \frac{\partial \mathcal{L}}{\partial X^Q} \right)_i = \frac{\partial \mathcal{L}}{\partial Q_i} (W_i^Q)^T \quad (31)$$

$$\frac{\partial \mathcal{L}}{\partial X^Q} = \sum_{i=1}^h \left( \frac{\partial \mathcal{L}}{\partial X^Q} \right)_i \quad (32)$$

$$\text{MACCs} = M \times d_{\text{model}} \times d_{\text{model}}$$

$$\text{FLOPs} = 2M \times d_{\text{model}} \times d_{\text{model}}$$

$$\frac{\partial \mathcal{L}}{\partial W_i^Q} = (X^Q)^T \frac{\partial \mathcal{L}}{\partial Q_i} \quad (33)$$

$$\begin{aligned}
\text{MACCs} &= d_{\text{model}} \times M \times d_{\text{model}} \\
\text{FLOPs} &= 2d_{\text{model}} \times M \times d_{\text{model}} \\
\frac{\partial \mathcal{L}}{\partial K_i} &= \left( \frac{\partial \mathcal{L}}{\partial Q_i K_i^T} \right)^T Q_i
\end{aligned} \tag{34}$$

$$\begin{aligned}
\text{MACCs} &= N \times M \times d_{\text{model}} \\
\text{FLOPs} &= 2N \times M \times d_{\text{model}} \\
\left( \frac{\partial \mathcal{L}}{\partial X^K} \right)_i &= \frac{\partial \mathcal{L}}{\partial K_i} (W_i^K)^T
\end{aligned} \tag{35}$$

$$\frac{\partial \mathcal{L}}{\partial X^K} = \sum_{i=1}^h \left( \frac{\partial \mathcal{L}}{\partial X^K} \right)_i \tag{36}$$

$$\begin{aligned}
\text{MACCs} &= N \times d_{\text{model}} \times d_{\text{model}} \\
\text{FLOPs} &= 2N \times d_{\text{model}} \times d_{\text{model}} \\
\frac{\partial \mathcal{L}}{\partial W_i^K} &= (X^K)^T \frac{\partial \mathcal{L}}{\partial K_i}
\end{aligned} \tag{37}$$

$$\text{MACCs} = d_{\text{model}} \times N \times d_{\text{model}}$$

$$\text{FLOPs} = 2d_{\text{model}} \times N \times d_{\text{model}}$$

In the case of the multihead attention encoder and of the masked multihead attention decoder, it should be considered that  $X^Q = X^K = X^V$  ( $M = N$ ), and the derivatives with respect to  $X^Q$ ,  $X^K$  and  $X^V$  are added. Conversely, in the multi-head attention decoder, it holds that  $X^Q \neq X^K = X^V$  ( $M \neq N$ ), and only the derivatives with respect to  $X^K$  and  $X^V$  are added. The complexity of these operations is marginal and can be considered to be already integrated in previous MACCs/FLOPs.

In conclusion, the **backward pass** has a complexity of

$$\begin{aligned}
&\text{MACCs} = \\
&2M \times d_{\text{model}} \times d_{\text{model}} + 2N \times d_{\text{model}} \times d_{\text{model}} + 4M \times N \times d_{\text{model}} + M \times N \times M \times h
\end{aligned}$$

$$\begin{aligned}
&\text{FLOPs} = 4M \times d_{\text{model}} \times d_{\text{model}} + 4N \times d_{\text{model}} \times d_{\text{model}} + 8M \times N \times d_{\text{model}} + 2M \times N \times \\
&M \times h + M \times N \times h
\end{aligned}$$

and the **weight update** has a complexity of

$$\text{MACCs} = 2M \times d_{\text{model}} \times d_{\text{model}} + 2N \times d_{\text{model}} \times d_{\text{model}}$$

$$\text{FLOPs} = 4M \times d_{\text{model}} \times d_{\text{model}} + 4N \times d_{\text{model}} \times d_{\text{model}}$$

#### 5.4. Feed-Forward Network

The feed-forward network (FFN) is a 2-layer neural network, where the first layer is of size  $M \times d_{\text{ff}}$  and the second  $M \times d_{\text{model}}$ . Only the first layer uses an activation function.

$$\text{FFN}(x) = \text{GELU}(xW_1 + b_1)W_2 + b_2 \tag{38}$$

##### 5.4.1. Forward Pass

$$\text{MACCs} = 2M \times d_{\text{model}} \times d_{\text{ff}}$$

The FLOPs for the GeLU activation are assumed to be 8 FLOPs in the forward pass and 13 FLOPs for computing the derivative.

The FLOPs accounting for bias and NL are

$$\text{FLOPs} = 4M \times d_{\text{model}} \times d_{ff} + 9M \times d_{ff} + M \times d_{\text{model}}$$

#### 5.4.2. Backward Pass

The backward pass is characterized by the following complexity.

$$\text{MACCs} = 2M \times d_{ff} \times d_{\text{model}}$$

$$\text{FLOPs} = 4M \times d_{\text{model}} \times d_{ff} + 13M \times d_{ff}$$

#### 5.4.3. Weight Update

The weight update requires the following complexity.

$$\text{MACCs} = 2M \times d_{ff} \times d_{\text{model}}$$

$$\text{FLOPs} = 4M \times d_{ff} \times d_{\text{model}}$$

#### 5.5. Add and Norm

After each multihead attention and feed-forward block, the input to the block, indicated as sublayer, is added, and a layer normalization is applied.

$$\text{LayerNorm}(x + \text{Sublayer}(x)) \quad (39)$$

Layer The normalization normalizes the features across each token, multiplies the results by  $\gamma$  and adds  $\beta$ , where  $\gamma$  and  $\beta$  are learnable parameters.

$$\text{LayerNorm}(x) = \frac{x - E[x]}{\sqrt{\text{Var}[x]}} \gamma + \beta \quad (40)$$

##### 5.5.1. Forward Pass

Such operation does not properly constitute a MACC.

$$\text{MACCs} = 0$$

Operations that are performed for each neuron are considered. The square root is only performed once. The other operations are addition for the mean, subtract, square and addition for the variance, subtract, divide, bias (add) and scale (multiply). It results in 8 FLOPs per neuron. Furthermore, the FLOPs relative to the addition between  $x$  and the sublayer should also be considered:

$$\text{MACCs} = 0$$

$$\text{FLOPs} = 9M \times d_{\text{model}}$$

##### 5.5.2. Backward Pass and Weight Update

To train the parameters  $\gamma$  and  $\beta$ , we first need to compute the derivative of layernorm. The layer-normalized activation matrix is denoted as  $z$ .

$$\frac{\partial \mathcal{L}}{\partial \gamma_j} = \sum_i^M \frac{\partial \mathcal{L}}{\partial \text{LayerNorm}_{ij}} z_{ij} \quad (41)$$

$$\text{MACCs} = M \times d_{\text{model}}$$

$$\text{FLOPs} = 2M \times d_{\text{model}}$$

$$\frac{\partial \mathcal{L}}{\partial \beta_j} = \sum_i^M \frac{\partial \mathcal{L}}{\partial \text{Layernorm}_{ij}} \quad (42)$$

$$\text{MACCs} = 0$$

$$\text{FLOPs} = M \times d_{\text{model}}$$

$$\frac{\partial \mathcal{L}}{\partial z_{ij}} = \frac{\partial \mathcal{L}}{\partial \text{Layernorm}_{ij}} \gamma_{ij} \quad (43)$$

$$\text{MACCs} = 0$$

$$\text{FLOPs} = M \times d_{\text{model}}$$

The Jacobian of activation vector  $z_i$  with respect to  $x_i$  is defined as  $\text{jac}_i = \left\{ \frac{\partial z_{ij}}{\partial x_{ik}} = \frac{1}{\sigma} \left( \delta_{jk} - \frac{1}{d_{\text{model}}} - \frac{(x_j - \mu)(x_k - \mu)}{d_{\text{model}} \sigma^2} \right) \right\}_{jk}$ . By combining the various Jacobians for each row  $z_i$ , a 3d matrix  $\text{jac}$  is obtained, where each slice corresponds to the Jacobian of a row vector. The operations involved in such a computation do not properly constitute MACCs. The FLOPs to calculate the Jacobian is 3 muls, 2 divisions, 4 adds, namely 9 FLOPs for each element.

$$\text{MACCs} = 0$$

$$\text{FLOPs} = 9 \times M \times d_{\text{model}} \times d_{\text{model}}$$

To obtain the derivative

$$\frac{\partial \mathcal{L}}{\partial X} = \frac{\partial \mathcal{L}}{\partial z} \times_{\text{slice}} \text{jac} \quad (44)$$

$$\text{MACCs} = M \times d_{\text{model}} \times d_{\text{model}}$$

$$\text{FLOPs} = 2M \times d_{\text{model}} \times d_{\text{model}}$$

Then, FLOPs used to add the derivative for the skip layer should also be taken into account.

$$\text{MACCs} = 0$$

$$\text{MACCs} = M \times d_{\text{model}}$$

The total number of MACCs for **backward** are

$$\boxed{\text{MACCs} = M \times d_{\text{model}} \times d_{\text{model}}}$$

$$\boxed{\text{FLOPs} = 11M \times d_{\text{model}} \times d_{\text{model}} + 2M \times d_{\text{model}}}$$

The total number of MACCs for **weight update** are

$$\boxed{\text{MACCs} = M \times d_{\text{model}}}$$

$$\boxed{\text{FLOPs} = 3M \times d_{\text{model}}}$$

### 5.6. Softmax Layer

At the end of the transformer, there is a softmax layer where the weight matrix  $W^S$  is of size  $d_{\text{model}} \times \text{voc}_{\text{size}}$ .

$$\text{Softmax}(XW^S) \quad (45)$$

#### 5.6.1. Forward Pass

The forward pass requires

$$\boxed{\text{MACCs} = M \times d_{\text{model}} \times \text{voc}_{\text{size}}}$$

The softmax function requires  $5N$  FLOPs for an array of  $N$  elements. Hence, the number of FLOPs are

$$\text{FLOPs} = 2M \times d_{\text{model}} \times \text{voc}_{\text{size}} + M \times 5\text{voc}_{\text{size}}$$

### 5.6.2. Backward Pass and Weight Update

The derivative of the loss with respect to  $Z$ , with  $Z$  being the product of  $X$  with  $W^s$  is  $\frac{\partial \mathcal{L}}{\partial z} = \text{target} - s$ . As the target is usually a one hot encoded vector, it is assumed that such operation has no FLOPs or MACCs.

$$\frac{\partial \mathcal{L}}{\partial X} = \frac{\partial \mathcal{L}}{\partial Z} (W^s)^T \quad (46)$$

$$\text{MACCs} = M \times \text{voc}_{\text{size}} \times d_{\text{model}}$$

$$\text{FLOPs} = 2M \times \text{voc}_{\text{size}} \times d_{\text{model}}$$

$$\frac{\partial \mathcal{L}}{\partial W^s} = X^T \frac{\partial \mathcal{L}}{\partial Z} \quad (47)$$

$$\text{MACCs} = d_{\text{model}} \times M \times \text{voc}_{\text{size}}$$

$$\text{FLOPs} = 2d_{\text{model}} \times M \times \text{voc}_{\text{size}}$$

The total number of MACCs for **backward** are

$$\text{MACCs} = M \times \text{voc}_{\text{size}} \times d_{\text{model}}$$

$$\text{FLOPs} = 2M \times \text{voc}_{\text{size}} \times d_{\text{model}}$$

The total number of MACCs for **weight update** are

$$\text{MACCs} = d_{\text{model}} \times M \times \text{voc}_{\text{size}}$$

$$\text{FLOPs} = 2d_{\text{model}} \times M \times \text{voc}_{\text{size}}$$

### 5.7. Error Projection (Only PEPITA and MEMPEPITA)

The output error has dimensionality  $M \times \text{voc}_{\text{size}}$ , which is the same dimensionality as the decoder input. Therefore, a projection matrix to project it to the decoder input is not needed. On the other hand, an attention mechanism is used to project the error of dimensionality  $M \times d_{\text{model}}$  to the dimensionality of the input to the encoder  $N \times d_{\text{model}}$ .

$$T_{\text{err}}^{\text{enc}} = \text{Attention}(T^{\text{enc}}, T_{\text{err}}^{\text{dec}}, T_{\text{err}}^{\text{dec}}) \quad (48)$$

$$\text{MACCs} = 2N \times M \times \text{voc}_{\text{size}}$$

$$\text{FLOPs} = 4N \times M \times \text{voc}_{\text{size}} + 6N \times M$$

## 6. Exemplary Application

To explain better the applicability of the proposed mathematical formulation, the complexity estimation in terms of MACCs for a one-block encoder-only simplified architecture trained with BP, PEPITA or MEMPEPITA is reported in this section. The layers involved in the architecture are the following (sections): embedding layer (Section 5.1), multihead attention (Section 5.3), add and norm (Section 5.5), feed-forward network (Section 5.4) and softmax (Section 5.6). To compute the number of MACCs required for a forward pass, its complexity at each layer is added together.

$$\text{MACCs}_{\text{forward}} = M \times \text{voc}_{\text{size}} \times d_{\text{model}} + 2M \times d_{\text{model}}^2 + 2M \times N \times d_{\text{model}} + 2N \times d_{\text{model}}^2 + 0 + 2M \times d_{\text{model}} \times d_{\text{ff}} + 0 + M \times d_{\text{model}} \times \text{voc}_{\text{size}}$$

Analogously, the total number of MACCs for the backward pass and the weight update are the following:

$$\begin{aligned} \text{MACCs}_{\text{backward}} = \\ 0 + 2M \times d_{\text{model}} \times d_{\text{model}} + 2N \times d_{\text{model}} \times d_{\text{model}} + 4M \times N \times d_{\text{model}} + M \times N \times M \times h + \\ M \times d_{\text{model}} \times d_{\text{model}} + 2M \times d_{\text{model}} \times d_{\text{ff}} + M \times d_{\text{model}} \times d_{\text{model}} + d_{\text{model}} \times M \times \text{voc}_{\text{size}} \end{aligned}$$

$$\text{MACCs}_{\text{weight-update}} = M \times \text{voc}_{\text{size}} \times d_{\text{model}} + 2M \times d_{\text{model}}^2 + 2N \times d_{\text{model}}^2 + M \times d_{\text{model}} + 2M \times d_{\text{model}} \times d_{\text{ff}} + M \times d_{\text{model}} + M \times \text{voc}_{\text{size}} \times d_{\text{model}}$$

The first term of the sum in the weight-update MACC estimation shall be discarded when considering BP. As the output dimension is the same as the input dimension for an encoder-only architecture, the error projections for PEPITA and MEMPEPITA are not required. Referring to Table 2, the total numbers of MACCs for training a one-block encoder-only transformer and adopting the different learning procedures are the following:

$$\text{MACCs}_{\text{BP}} = \text{MACCs}_{\text{forward}} + \text{MACCs}_{\text{backward}} + \text{MACCs}_{\text{weight-update}}$$

$$\text{MACCs}_{\text{PEPITA}} = 2\text{MACCs}_{\text{forward}} + \text{MACCs}_{\text{weight-update}}$$

$$\text{MACCs}_{\text{MEMPEPITA}} = 3\text{MACCs}_{\text{forward}} + \text{MACCs}_{\text{weight-update}}$$

## 7. Conclusions

In this work, the equations behind BP (reverse-mode autodiff), PEPITA and MEMPEPITA for the layers of a generic transformer architecture were derived and described. The computational complexity of the forward pass, backward pass and weight update were expressed in terms of MACCs and FLOPs for each layer, using the mathematical formulas previously obtained. An exemplary application for the computation of the complexity in the case of a one-block encoder-only transformer was also reported for illustration purposes. The method proposed in this work combines the advantages of being device-agnostic with mathematical rigour, providing a robust estimation of complexity independent of the specific target. By taking advantage of the results of this paper, the reader can easily provide a reliable estimation of the computational complexity involved in training a transformer architecture of their choice using BP and forward learning procedures.

**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.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Data are contained within the article.

**Conflicts of Interest:** Authors Danilo Pietro Pau and Fabrizio Maria Aymone were employed by the company STMicroelectronics. The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential 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 Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; Curran Associates, Inc.: New York, NY, USA, 2017; Volume 30.
2. 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.
3. 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.
4. Mielke, S.J.; Alyafeai, Z.; Salesky, E.; Raffel, C.; Dey, M.; Gallé, M.; Raja, A.; Si, C.; Lee, W.Y.; Sagot, B.; et al. Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP. *arXiv* **2021**, arXiv:2112.10508.
5. Maslej, N.; Fattorini, L.; Brynjolfsson, E.; Etchemendy, J.; Ligett, K.; Lyons, T.; Manyika, J.; Ngo, H.; Niebles, J.C.; Parli, V.; et al. *The AI Index 2023 Annual Report*; Technical report; AI Index Steering Committee, Institute for Human-Centered AI, Stanford University: Stanford, CA, USA, 2023.
6. Hinton, G. The Forward-Forward Algorithm: Some Preliminary Investigations. *arXiv* **2022**, arXiv:2212.13345.
7. Dellaferriera, G.; Kreiman, G. Error-driven Input Modulation: Solving the Credit Assignment Problem without a Backward Pass. *arXiv* **2022**, arXiv:2201.11665.
8. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv* **2016**, arXiv:1603.04467.
9. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv* **2019**, arXiv:1912.01703.
10. Baydin, A.G.; Pearlmutter, B.A.; Radul, A.A.; Siskind, J.M. Automatic Differentiation in Machine Learning: A Survey. *J. Mach. Learn. Res.* **2017**, *18*, 5595–5637.
11. Clark, K.; Luong, M.T.; Le, Q.V.; Manning, C.D. Pre-Training Transformers as Energy-Based Cloze Models. In Proceedings of the EMNLP, Online, 16–20 November 2020.
12. Crick, F. The recent excitement about neural networks. *Nature* **1989**, *337*, 129–132. [\[CrossRef\]](#)
13. Lillicrap, T.; Santoro, A.; Marris, L.; Akerman, C.; Hinton, G. Backpropagation and the brain. *Nat. Rev. Neurosci.* **2020**, *21*, 335–346. [\[CrossRef\]](#)
14. 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. [\[CrossRef\]](#) [\[PubMed\]](#)
15. Liao, Q.; Leibo, J.Z.; Poggio, T. How Important is Weight Symmetry in Backpropagation? *arXiv* **2016**, arXiv:1510.05067. [\[CrossRef\]](#)
16. Baldi, P.; Sadowski, P. A theory of local learning, the learning channel, and the optimality of backpropagation. *Neural Netw.* **2016**, *83*, 51–74. [\[CrossRef\]](#) [\[PubMed\]](#)
17. 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, 6–11 August 2017; Volume 70, pp. 1627–1635.
18. 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; Volume 70, pp. 904–912.
19. Xiao, W.; Chen, H.; Liao, Q.; Poggio, T. Biologically-plausible learning algorithms can scale to large datasets. *arXiv* **2018**, arXiv:1811.03567.
20. Lillicrap, T.; Cownden, D.; Tweed, D.; Akerman, C. Random synaptic feedback weights support error backpropagation for deep learning. *Nat. Commun.* **2016**, *7*, 13276. [\[CrossRef\]](#) [\[PubMed\]](#)
21. Nøklund, A. Direct Feedback Alignment Provides Learning in Deep Neural Networks. In Proceedings of the 30th International Conference on Neural Information Processing Systems, NIPS'16, Barcelona, Spain, 5–10 December 2016; pp. 1045–1053.
22. Akrou, M.; Wilson, C.; Humphreys, P.; Lillicrap, T.; Tweed, D.B. Deep Learning without Weight Transport. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019; Curran Associates, Inc.: New York, NY, USA, 2019; Volume 32.
23. Frenkel, C.; Lefebvre, M.; Bol, D. Learning Without Feedback: Fixed Random Learning Signals Allow for Feedforward Training of Deep Neural Networks. *Front. Neurosci.* **2021**, *15*, 629892. [\[CrossRef\]](#) [\[PubMed\]](#)
24. Xie, X.; Seung, H. Equivalence of Backpropagation and Contrastive Hebbian Learning in a Layered Network. *Neural Comput.* **2003**, *15*, 441–454. [\[CrossRef\]](#) [\[PubMed\]](#)
25. Scellier, B.; Bengio, Y. Equilibrium Propagation: Bridging the Gap between Energy-Based Models and Backpropagation. *Front. Comput. Neurosci.* **2017**, *11*, 24. [\[CrossRef\]](#)
26. Clark, D.; Abbott, L.; Chung, S. Credit Assignment Through Broadcasting a Global Error Vector. In Proceedings of the Advances in Neural Information Processing Systems 34—35th Conference on Neural Information Processing Systems, NeurIPS 2021, Virtual, 6–14 December 2021; pp. 10053–10066.
27. 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. [\[CrossRef\]](#)

28. 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. In Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks, Virtual, 6–14 December 2021.
29. 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.
30. Justus, D.; Brennan, J.; Bonner, S.; McGough, A.S. Predicting the Computational Cost of Deep Learning Models. *arXiv* **2018**, arXiv:1811.11880.
31. Zargar, B.; Ponci, F.; Monti, A. Evaluation of Computational Complexity for Distribution Systems State Estimation. *IEEE Trans. Instrum. Meas.* **2023**, *72*, 9001512. [[CrossRef](#)]
32. Muhammad, N.; Bibi, N.; Jahangir, A.; Mahmood, Z. Image denoising with norm weighted fusion estimators. *Form. Pattern Anal. Appl.* **2018**, *21*, 1013–1022. [[CrossRef](#)]
33. Getzner, J.; Charpentier, B.; Günnemann, S. Accuracy is not the only Metric that matters: Estimating the Energy Consumption of Deep Learning Models. *arXiv* **2023**, arXiv:2304.00897.

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.