algorithms

Article

Combining Optimization Methods Using an Adaptive Meta Optimizer

Nicola Landro [†], **Ignazio Gallo** [*,†,‡] **and Riccardo La Grassa** [†]

Department of Theoretical and Applied Sciences, University of Insubria, 21100 Varese, Italy; nlandro@uninsubria.it (N.L.); rlagrassa@uninsubria.it (R.L.G.)
* Correspondence: ignazio.gallo@uninsubria.it
† These authors contributed equally to this work.
‡ Current address: Dipartimento di Scienze Teoriche e Applicate, DISTA, Via O. Rossi, 9, Padiglione Rossi, 21100 Varese, Italy.

Abstract: Optimization methods are of great importance for the efficient training of neural networks. There are many articles in the literature that propose particular variants of existing optimizers. In our article, we propose the use of the combination of two very different optimizers that, when used simultaneously, can exceed the performance of the single optimizers in very different problems. We propose a new optimizer called ATMO (AdapTive Meta Optimizers), which integrates two different optimizers simultaneously weighing the contributions of both. Rather than trying to improve each single one, we leverage both at the same time, as a meta-optimizer, by taking the best of both. We have conducted several experiments on the classification of images and text documents, using various types of deep neural models, and we have demonstrated through experiments that the proposed ATMO produces better performance than the single optimizers.

Keywords: deep learning; optimization algorithm; optimizers; text classification; image classification

check for updates

Citation: Landro, N.; Gallo, I.; La Grassa, R. Combining Optimization Methods Using an Adaptive Meta Optimizer. *Algorithms* **2021**, *14*, 186. https://doi.org/10.3390/a14060186

Academic Editor: Akemi Galvez Tomida

Received: 24 May 2021
Accepted: 17 June 2021
Published: 19 June 2021

1. Introduction

Stochastic Gradient Descent [1] (SGD) is the dominant method for solving optimization problems. SGD iteratively updates the model parameters by moving them in the direction of the negative gradient calculated on a mini-batch scaled by the step length, typically referred to as the learning rate. It is necessary to decay this learning rate as the algorithm proceeds to ensure convergence. Manually adjusting the learning rate decay in SGD is not easy. To address this problem, several methods have been proposed that automatically reduce the learning rate. The basic intuition behind these approaches is to adaptively tune the learning rate based on only recent gradients, therefore limiting the reliance on the update to only a few past gradients. ADAptive Moment estimation [2] (ADAM) is one of several methods based on this update mechanism [3]. On the other hand, adaptive optimization methods such as ADAM, even though they have been proposed to achieve a rapid training process, are observed to generalize poorly with respect to SGD or even fail to converge due to unstable and extreme learning rates [4]. To try to overcome the problems of both of these types of optimizers and at the same time try to exploit their advantages, we propose an optimizer that combines them in a new meta-optimizer.

As depicted in Figure 1, the basic idea of the ATMO optimizer proposed here is to combine two different known optimizers and automatically go quickly towards the direction of both on the surface of the loss function when the two optimizers agree (see geometric example in Figure 2a). When the two optimizers used in the combination do not agree, our solution always goes towards the predominant direction between the two but slowing down the speed (see example of Figure 2b).

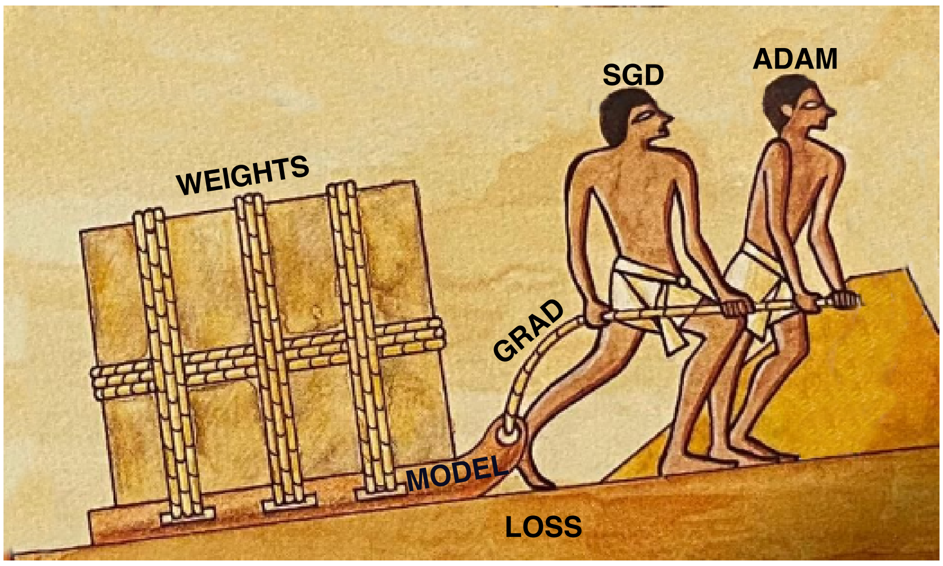

Figure 1. Intuitive representation of the idea behind the proposed ATMO that Mix ADAM and SGD optimizers: the weights are modified simultaneously by both the optimizers.

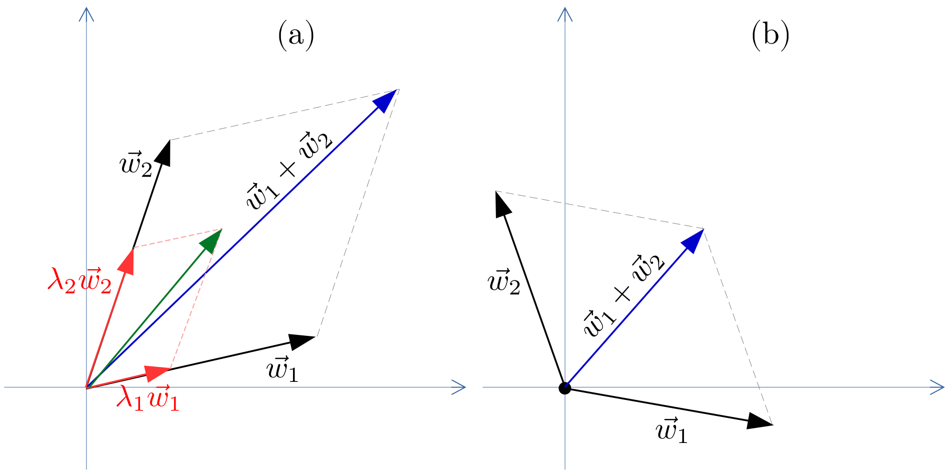

Figure 2. Graphical representation of the basic idea for the proposed ATMO optimizer. In (**a**), if the two translations $\vec{w}_1$ and $\vec{w}_2$ obtained from two different optimizers are similar, then the resulting translation $\vec{w}_1 + \vec{w}_2$ is boosted. In (**b**), if the translations $\vec{w}_1$ and $\vec{w}_2$ go in two different directions, then the resulting translation is smaller. We also use two hyper-parameters λ_1 and λ_2 to weigh the contribution of the two optimizers.

In the literature, there are many papers that compare neural models trained with the use of different optimizers [5–8] or that propose modifications for existing optimizers [4,9,10], always aimed at improving the results on a subset of problems. Each paper demonstrates that an optimizer is better than the others, but as the problem changes, this type of result is no longer valid and we have to start from scratch. Our method can be combined with other methods like Genetically Trained DNN [11], which combines learning using gradient descent with genetic algorithms. The genetic part, after a selected number of epochs, selects a new population through three states called selection, crossover, and manipulation. In general, a Genetically Trained DNN is very different from our proposal, which combines two gradient descent methods together. However, the genetic method can also be used with ATMO.

In our paper, we propose combining two different optimizers like SGD and ADAM to overcome the performances of the single optimizers in very different problems.

Below are the main contributions of this paper:

- We show experimentally that the combination of two different optimizers in a new meta-optimizer leads to a better generalization capacity in different contexts.
- We describe ATMO using Adam and SGD but show experimentally that other types of optimizers can be profitably combined.
- We release the source code and setups of the experiments [12].

2. Related Work

In the literature, there are not many papers that try to combine different optimizers together. In this section, we report some of the more recent papers that in some ways use different optimizers in the same learning process.

In [13], the authors investigate a hybrid strategy, called **SWATS** (SWitching from Adam To Sgd), which starts training with an adaptive optimization method and switches to SGD when appropriate. This idea starts from the observation that despite superior training results, adaptive optimization methods such as ADAM generalize poorly compared to SGD because they tend to work well in the early part of the training but are overtaken by SGD in the later stages of training. In concrete terms, SWATS is a simple strategy that goes from Adam to SGD when an activation condition is met. The experimental results obtained in this paper are not so different from ADAM or SGD when used individually, so the authors concluded that using SGD with perfect parameters is the best idea. In our proposal, we want to combine two well-known optimizers to create a new one that simultaneously uses two different optimizers from the beginning to the end of the training process.

ESGD is a population-based Evolutionary Stochastic Gradient Descent framework for optimizing deep neural networks [14]. In this approach, individuals in the population optimized with various SGD-based optimizers using distinct hyper-parameters are considered competing species in the context of coevolution. The authors experimented with optimizer pools consisting of SGD and ADAM variants, where it is often observed that ADAM tends to be aggressive early on but stabilizes quickly, while SGD starts slowly but can reach a better local minimum. ESGD can automatically choose the appropriate optimizers and their hyper-parameters based on the fitness value during the evolution process so that the merits of SGD and ADAM can be combined to seek a better local optimal solution to the problem of interest. In the method we propose, we do not need another approach, such as the evolutionary one, to decide which optimizer to use and with which hyper-parameters, but it is the same approach that decides the contribution of SGD and that of ADAM at each step.

In this paper, we also compare our ATMO optimizer with **ADAMW** [15,16] (ADAM with decoupled Weight decay regularization), which is a version of ADAM in which weight decay is decoupled from L_2 regularization. This optimizer offers good generalization performance, especially for text analysis, and since we also perform some experimental tests on text classification, then we also compare our optimizer with ADAMW. In fact, ADAMW is often used with BERT [17] applied to well-known datasets for text classification.

Padam [18] (Partially ADAM) is one of the recent Adam derivates that achieves very interesing results. It bridges the generalization gap for adaptive gradient methods by introduceing a partial adaptive parameter to control the level of adaptiveness of the optimization procedure. We principally use ATMO with a combination of ADAM and SGD, but we test the generalization of this method also by combining Padam and SGD [12] to compare with many other optimizers (Table 1).

3. Preliminaries

Training neural networks is equivalent to solving the following optimization problem:

$$\min_{w \in \mathbb{R}^n} \mathcal{L}(w) \tag{1}$$

where $\mathcal{L}$ is a loss function and w are the weights.

The iterations of an **SGD** [1] optimizer can be described as:

$$w_{k+1} = w_k - \eta \cdot \nabla \mathcal{L}(w) \tag{2}$$

where w_k denotes the weights w at the k-th iteration, η denotes the learning rate, and $\nabla \mathcal{L}(w)$ denotes the stochastic gradient calculated at w_k. To propose a stochastic gradient that is calculated as generically as possible, we introduce the **weight decay** [19] strategy, often used in many SGD implementations. The weight decay can be seen as a modification of the $\nabla \mathcal{L}(w)$ gradient, and in particular, we describe it as follows:

$$\widehat{\nabla} \mathcal{L}(w_k) = \nabla \mathcal{L}(w_k) + w_k \cdot \gamma \tag{3}$$

where γ is a small scalar called weight decay. We can observe that if the weight decay γ is equal to zero; then $\widehat{\nabla} \mathcal{L}(w) = \nabla \mathcal{L}(w)$. Based on the above, we can generalize Equation (2) to the following one that includes weight decay:

$$w_{k+1} = w_k - \eta \cdot \widehat{\nabla} \mathcal{L}(w) \tag{4}$$

The SGD algorithm described up to here is usually used in combination with **momentum**, and in this case, we refer to it as **SGD(M)** [20] (Stochastic Gradient Descend with Momentum). SGD(M) almost always works better and faster than SGD because the momentum helps accelerate the gradient vectors in the right direction, thus leading to faster convergence. The iterations of SGD(M) can be described as follows:

$$v_k = \mu \cdot v_{k-1} + \widehat{\nabla} \mathcal{L}(w) \tag{5}$$

$$w_{k+1} = w_k - \eta \cdot v_k \tag{6}$$

where $\mu \in [0, 1)$ is the momentum parameter and for $k = 0$, v_0 is initialized to 0. The simpler methods of momentum have an associated **damping** coefficient [21], which controls the rate at which the momentum vector decays. The dampening coefficient changes the momentum as follows:

$$v_{d_k} = \mu \cdot v_{k-1} + \widehat{\nabla} \mathcal{L}(w) \cdot (1 - d) \tag{7}$$

where $0 \leq d < 1$ is the dampening value, so the final SGD with momentum and dampening coefficients can be seen as follows:

$$w_{k+1} = w_k - \eta \cdot v_{d_k} \tag{8}$$

Nesterov momentum [22] is an extension of the moment method that approximates the future position of the parameters that takes into account the movement. The SGD with nesterov transforms again the v_k of Equation (5); more precisely:

$$v_{n_k} = \widehat{\nabla} \mathcal{L}(w) + v_{d_k} \cdot m \tag{9}$$

$$w_{k+1} = w_k - \eta \cdot v_{n_k} \tag{10}$$

The complete SGD algorithm, used in this paper, is shown in Algorithm 1.

Algorithm 1 Stochastic Gradient Descent (SGD).

Input: the weights w_k, learing rate η, weight decay γ, momentum μ, dampening d, boolean *nesterov*

```
$v_0 = 0$
function $\Delta_{\text{SGD}}(w_k, \nabla, \gamma, \mu, d, nesterov)$
    $\widehat{\nabla} = \nabla + w_k \cdot \gamma$
    if $m \neq 0$ then
        if $k = 0$ then
            $v_k = \widehat{\nabla}$
        else
            $v_k = v_{k-1} \cdot \mu + \widehat{\nabla} \cdot (1 - d)$
        end if
        if $nesterov = True$ then
            $v_k = \widehat{\nabla} + v_k \cdot \mu$
        end if
    end if
return $v_k$
end function
for batches do
    $w_{k+1} = w_k - \eta_s \cdot \Delta_{\text{SGD}}(w_k, \nabla, \gamma, \mu, d, nesterov)$
end for
```

ADAM [2] (ADAptive Moment estimation) optimization algorithm is an extension to SGD that has recently seen broader adoption for deep learning applications in computer vision and natural language processing. ADAM's equation for updating the weights of a neural network by iterating over the training data can be represented as follows:

$$m_k = \beta_1 \cdot m_{k-1} + (1 - \beta_1) \cdot \widehat{\nabla}\mathcal{L}(w_k) \tag{11}$$

$$v_k^a = \beta_2 \cdot v_{k-1} + (1 - \beta_2) \cdot \widehat{\nabla}\mathcal{L}(w_k)^2 \tag{12}$$

$$w_{k+1} = w_k - \eta \cdot \frac{\sqrt{1 - \beta_2}}{1 - \beta_1} \cdot \frac{m_k}{\sqrt{v_k^a} + \epsilon} \tag{13}$$

where m_k and v_k^a are estimates of the first moment (the mean) and the second moment (the non-centered variance) of the gradients respectively; hence the name of the method. β_1, β_2 and ϵ are three new introduced hyper-parameters of the algorithm. **AMSGrad** [23] is a stochastic optimization method that seeks to fix a convergence issue with Adam based optimizers. AMSGrad uses the maximum of past squared gradients v_{k-1} rather than the exponential average to update the parameters:

$$\widehat{v}_k = \max(\widehat{v}_{k-1}, v_k^a) \tag{14}$$

$$w_{k+1} = w_k - \eta \cdot \frac{\sqrt{1 - \beta_2}}{1 - \beta_1} \cdot \frac{m_k}{\sqrt{\widehat{v}_k} + \epsilon} \tag{15}$$

The complete ADAM algorithm used in this paper is shown in Algorithm 2.

Algorithm 2 ADAptive Moment estimation (ADAM).

```
Input: the weights $w_k$, learing rate $\eta$, weight decay $\gamma$, $\beta_1$, $\beta_2$, $\epsilon$, boolean $amsgrad$
  $m_0 = 0$
  $v^a_0 = 0$
  $\widehat{v}_0 = 0$
  function $\Delta_{\text{ADAM}}(w_k, \nabla, \eta, \gamma, \beta_1, \beta_2, \epsilon, amsgrad)$
    $\widehat{\nabla} = \nabla + w_k \cdot \gamma$
    $m_k = m_{k-1} \cdot \beta_1 + \widehat{\nabla} \cdot (1 - \beta_1)$
    $v^a_k = v^a_{k-1} \cdot \beta_2 + \widehat{\nabla} \cdot \widehat{\nabla} \cdot (1 - \beta_2)$
    if $amsgrad = True$ then
      $\widehat{v}_k = \max(\widehat{v}_{k-1}, v^a_k)$
      $denom = \frac{\sqrt{\widehat{v}_k}}{\sqrt{1-\beta_2+\epsilon}}$
    else
      $denom = \frac{\sqrt{v^a_k}}{\sqrt{1-\beta_2+\epsilon}}$
    end if
    $\eta_a = \frac{\eta}{1-\beta_1}$
    $d_k = \frac{m_k}{denom}$
    return $d_k, \eta_a$
  end function
  for batches do
    $d_k, \eta_a = \Delta_{\text{ADAM}}(w_k, \nabla, \eta, \gamma, \beta_1, \beta_2, \epsilon, amsgrad)$
    $w_{k+1} = w_k - \eta_a \cdot d_k$
  end for
```

4. Proposed Approach

In this section, we develop the proposed new optimization method called ATMO. Our goal is to propose a strategy that automatically combines the advantages of an adaptive method like ADAM with the advantages of SGD throughout the entire learning process. This strategy can by applied to every combination of optimizer, but we focused on ADAM and SGD combination. This combination of optimizers is summed, as shown in Figure 2, where w_1 and w_2 represent the displacements on the ADAM and SGD on the surface of the loss function, while $w1 + w2$ represents the displacement obtained thanks to our optimizer. Below, we explain each line of the ATMO algorithm represented in Algorithm 3.

The ATMO optimizer has only two hyper-parameters which are λ_a and λ_s, used to balance the contribution of ADAM and SGD, respectively. It also uses all the hyper-parameters inherited from SGD and ADAM. In this paper, we assume the use of the most common implementation of gradient descent used in the field of deep learning, namely mini-batch gradient descent, which divides the training dataset into small batches that are used to calculate the model error and update the model coefficients w_k. For each mini-batch, we calculate the contribution derived from the two components ADAM and SGD and then update all the coefficients as described in the three following subsections.

4.1. ADAM Component

The complete ADAM algorithm is defined in Algorithm 2. In order to use ADAM in our optimizer, we have extracted the Δ_{ADAM} function, which calculates and returns the increments d_k for the coefficients w_k, as defined in Equation (16).

$$d_k = \frac{\sqrt{1-\beta_2} \cdot m_k}{\sqrt{\widehat{v}_k} + \epsilon} \tag{16}$$

Note that if the components of the two vectors $\widehat{v}_k$ and m_k are not all equal, then the direction has changed with respect to the natural gradient.

The same Δ_{ADAM} function also returns the new learning rate η_a defined in Equation (17), useful when a variable learning rate is used. In this last case, ATMO uses η_a to calculate a new learning rate at each step.

$$\eta_a = \frac{\eta}{1-\beta_1} \tag{17}$$

Now, having η_a and d_k, we can directly modify the weights w_k exactly as done in the ADAM optimizer and described in Equation (18).

4.2. SGD Component

As for the ADAM component, the SGD component, defined in Algorithm 1, has also been divided into two parts: the Δ_{SGD} function, which returns the increment to be given to the weight w_k, and the formula to update the weights as defined in Equation (10). The v_{n_k} value returned by the Δ_{SGD} function is exactly the value defined in Equation (9), which we use directly for our ATMO optimizer.

4.3. The ATMO Optimizer

The proposed approach can be summarized as follows:

$$w_{k+1} = w_k - (\lambda_s \cdot \eta + \lambda_a \cdot \eta_a) \cdot (\lambda_s \cdot v_{n_k} + \lambda_a \cdot d_k) \tag{18}$$

where λ_s is a scalar for the SGD component and λ_a is another scalar for the ADAM component used for balancing the two contributions of the two optimizers. η is the learning rate of the proposed ATMO optimizer, while η_a is the learning rate of ADAM defined in Equation (17). d_k and v_{n_k} are the two increments define in Equations (16) and (9), respectively.

Equation (18) can be expanded in the following Equation (19) to make explicit what elements are involved in the weights update step used by our ATMO optimizer.

$$\begin{aligned} w_{k+1} = w_k - (\lambda_s \cdot \eta + \lambda_a \cdot \frac{\eta}{1-\beta_1}) \cdot \\ \cdot (\lambda_s \cdot v_{n_k} + \lambda_a \cdot \frac{\sqrt{1-\beta_2} \cdot m_k}{\sqrt{\hat{v}_k} + \epsilon}) \end{aligned} \tag{19}$$

where β_1 and β_2 are two parameters of the ADAM optimizer, v_k^a is defined in Equation (12), and m_k is defined in Equation (11).

The ATMO algorithm can be easily implemented by the following pseudo code defined in Algorithm 3 and by calling the two functions Δ_{ADAM} defined in Algorithm 2 and Δ_{SGD} defined in Algorithm 1. We can also show that convergence is guaranteed for the ATMO optimizer if we assume that convergence has been guaranteed for the two optimizers SGD and ADAM.

Algorithm 3 ATMO on mixing ADAM and SGD.

Input: the weights w_k, λ_a, λ_s, learing rate η, weight decay γ, other SGD and ADAM parameters ...

```
for batches do
    $d_k, \eta_a = \Delta_{\text{ADAM}}(w_k, \nabla, \eta, \gamma, \ldots)$
    $v_{n_k} = \Delta_{\text{SGD}}(w_k, \nabla, \gamma, \ldots)$
    $merged = \lambda_s \cdot v_{n_k} + \lambda_a \cdot d_k$
    $\eta_m = \lambda_s \cdot \eta + \lambda_a \cdot \eta_a$
    $w_{k+1} = w_k - \eta_m \cdot merged$
end for
```

Theorem 1 (ATMO Cauchy necessary convergence condition)**.** *If ADAM and SGD are two optimizers whose convergence is guaranteed, then the Cauchy necessary convergence condition is also true for ATMO.*

Proof. Under the conditions in which the convergence of ADAM and SGD is guaranteed [23,24], we can say that $\sum_{k=0}^{p} \eta \cdot v_{n_k}$ and $\sum_{k=0}^{p} \eta_a \cdot d_k$ converge at ∞ . That implies the following:

$$\lim_{p \to \infty} \eta \cdot v_{n_p} = \lim_{p \to \infty} \eta_a \cdot d_p = 0 \tag{20}$$

We can observe that $\lim_{p \to \infty} \sum_{k=0}^{p} \eta = \lim_{p \to \infty} \sum_{k=0}^{p} \eta_a = \infty$, so we can obtain the following:

$$\lim_{p \to \infty} v_{n_p} = \lim_{p \to \infty} d_p = 0 \tag{21}$$

The thesis is that $\sum_{k=0}^{p} (\lambda_s \cdot \eta + \lambda_a \cdot \eta_a) \cdot (\lambda_s \cdot v_{n_k} + \lambda_a \cdot d_k)$ respects the Cauchy necessary convergence condition, so: $\lim_{p \to \infty} (\lambda_s \cdot \eta + \lambda_a \cdot \eta_a) \cdot (\lambda_s \cdot v_{n_p} + \lambda_a \cdot d_p) = 0$ and for Equation (21), this last equality is trivially true:

$$\begin{aligned} &\lim_{p \to \infty} (\lambda_s \cdot \eta + \lambda_a \cdot \eta_a) \cdot (\lambda_s \cdot v_{n_p} + \lambda_a \cdot d_p) = \\ &(\lambda_s \cdot \eta + \lambda_a \cdot \eta_a) \cdot \lim_{p \to \infty} (\lambda_s \cdot 0 + \lambda_a \cdot 0) = 0 \end{aligned} \tag{22}$$

□

Theorem 2. *(ATMO convergence) If for $p \to \infty$ it is valid that $\sum_{k=0}^{p} \eta \cdot v_{n_k} = \eta \cdot m_1$ and $\sum_{k=0}^{p} \eta_a \cdot d_k = \eta_a \cdot m_2$ where $m_1 \in \mathbb{R}$ and $m_2 \in \mathbb{R}$ are two finite real values, then $ATMO = \sum_{k=0}^{p} (\lambda_s \cdot \eta + \lambda_a \cdot \eta_a) \cdot (\lambda_s \cdot v_{n_k} + \lambda_a \cdot d_k) = \lambda_s^2 \cdot \eta \cdot m_1 + \lambda_a^2 \cdot \eta_a \cdot m_2 + \lambda_s \cdot \lambda_a \cdot \eta_a \cdot m_1 + \lambda_s \cdot \lambda_a \cdot \eta \cdot m_2$.*

Proof. We can write ATMO series as:

$$\begin{aligned} ATMO = \lambda_s^2 \cdot \eta \cdot \sum_{k=0}^{p} v_{n_k} + \lambda_a^2 \cdot \eta_a \cdot \sum_{k=0}^{p} d_k + \\ + \lambda_s \cdot \lambda_a \cdot \eta_a \cdot \sum_{k=0}^{p} v_{n_k} + \lambda_s \cdot \lambda_a \cdot \eta \cdot \sum_{k=0}^{p} d_k \end{aligned} \tag{23}$$

This can be rewritten for $p \to \infty$ as:

$$\begin{aligned} ATMO = \lambda_s^2 \cdot \eta \cdot m_1 + \lambda_a^2 \cdot \eta_a \cdot m_2 + \\ + \lambda_s \cdot \lambda_a \cdot \eta_a \cdot m_1 + \lambda_s \cdot \lambda_a \cdot \eta \cdot m_2 \end{aligned} \tag{24}$$

□

This last theorem does not exclude the possibility that ATMO converges even if ADAM or SGD or both do not converge, for example, due to some unsuitable parameters. In this paper, this aspect is not proven.

4.4. Geometric Explanation

We can see optimizers as two explorers w_1 and w_2 who want to explore an environment (the surface of a loss function). If the two explorers agree to go in a similar direction, then they quickly go in that direction ($w_1 + w_2$). Otherwise, if they disagree and each prefers a different direction than the other, then they proceed more cautiously and slower ($w_1 + w_2$). As we can see in Figure 2a, if the directions of the displacement of w_1 and w_2 are similar then the amplitude of the resulting new displacement $w_1 + w_2$ is increased, however, as shown in Figure 2b, if the directions of the two displacements w_1 and w_2 are not similar, then the amplitude of the new displacement $w_1 + w_2$ us decreased.

In our approach, the sum $w_1 + w_2$ is weighted (see red vectors in Figure 2a), so one of the two optimizers SGD or ADAM can become more relevant than the other in the choice of direction for ATMO; hence, the direction resultant may tend towards one of the two. In ATMO, we set the weight of the two contributions so as to have a sum $\lambda_1 + \lambda_2 = 1$ in order to maintain a learning rate of the same order of magnitude.

Another important component that greatly affects the ATMO shift module at each training step is its learning rate, defined in Equation (18), which combines η and η_a. The shifts are scaled using the learning rate, so there is a situation where ATMO gets more thrust than the ADAM and SGD starting shifts. In particular, we can imagine that the displacement vector of ADAM has a greater magnitude than SGD and the learning rate of SGD is greater than that of ADAM. In this case, the ATMO shift has a greater vector magnitude than SGD and a higher ADAM learning rate, which can cause a large increase in the ATMO shift towards the search of a minimum.

4.5. Toy Examples

To better understand our proposal, we built a toy example where we highlight the main behaviour of ATMO. The toy examples, even if they are not a true example of a deep learning model, can be easily visualized because the exploration surface can be plotted in three dimensions.

More precisely, we consider the following example:

$$x = [1, 2], \ y = [2, 4] \tag{25}$$

$$p_i = w_1 \cdot (w_2 \cdot x_i) \tag{26}$$

$$\mathcal{L}(w_1, w_2) = \sum_{i=0}^{1} (p_i - y_i^2) \tag{27}$$

We set $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $amsgrad = False$, dampening $d = 0$, $nesterov = False$ and $\mu = 0$. As we can see in Figure 3a, our ATMO optimizer goes faster towards the minimum value after only two epochs, and SGD is fast at the first epoch; however, it decreases its speed soon after and comes close to the minimum after 100 epochs, and ADAM instead reaches its minimum after 25 epochs. Our approach can be fast when it gets a large v_k from SGD and a large η_a from ADAM.

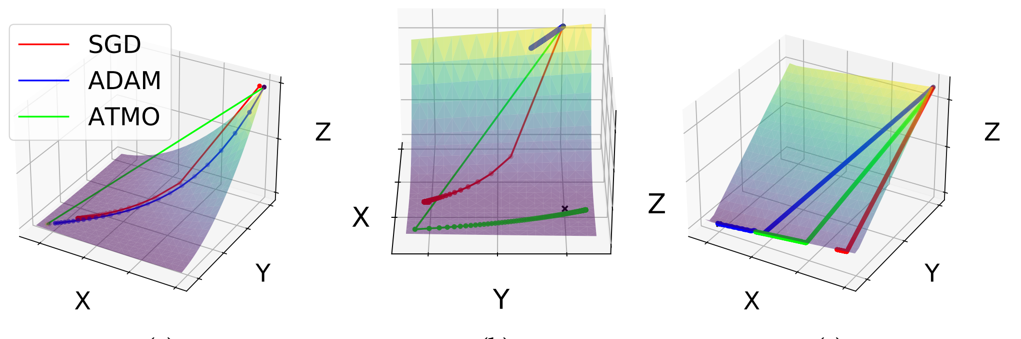

Figure 3. The figures show the behavior of the three optimizers ATMO, ADAM and SGD on different surfaces. The subfigure (**a**) describes the surface defined in Equation (27). For better visualization in this figure the SGD was shifted on X axis of 0.1. The subfigure (**b**) describes the Rosenbrook's surface with $a = 1$ and $b = 100$. The subfigure (**c**) describes the surface $z = \frac{|x|}{10} + |y|$.

Another toy example can be done with the benchmark Rosenbrook [25] function:

$$z = (a - y)^2 + b \cdot (y - x^2)^2 \tag{28}$$

We set $a = 1$ and $b = 100$, weight $x = 3$ and weight $y = 1$, $lr = 0.0001$, $epochs = 1000$, and default parameter for ADAM and SGD. The ATMO optimizer sets $\lambda_s = \lambda_a = 0.5$. The comparative result for the minimization of this function is shown in Figure 3b. In this experiment, we can see how, by combining the two optimizers ADAM and SGD, we can obtain a better result than the single optimizers. For this function, going from the starting point towards the direction of the maximum slope means moving away from the minimum, and therefore it takes many training epochs to approach the minimum.

Let us use a final toy example to highlight the behavior of the ATMO optimizer. In this case we look for the minimum of the function $z = \frac{|x|}{10} + |y|$. We set the weights $x = 3$ and $y = 2$, $lr = 0.01$, $epochs = 400$ and use all the default parameters for ADAM and SGD. ATMO assigns the same value 0.5 for the two lambdas hyper-parameters. In Figure 3b, we can see how not all the paths between the paths of ADAM and SGD are the best choice. Since ATMO, as shown in Figure 2, goes towards an average direction with respect to that of ADAM and SGD, then in this case ADAM arrives first at the minimum point.

4.6. Dynamic λ

To avoid selecting the best two lambdas of Equation (18) for each experiment, we introduce an approach that automatically changes the two lambdas during training. Motivated by experiments showing how selecting the correct lambdas can greatly affect the results, we studied a solution to avoid having to find the best two hyper-parameters for each experiment. In fact, experiments show that ADAM is usually better than SGD at the beginning of training while SGD performs better in the final phase [14] (see also the example in Figure 4). Following the idea of SWAT [13] to hard-switch from an optimizer to another, we introduce an approach that linearly changes the two lambdas from $\lambda_a = 1$, $\lambda_s = 0$ to $\lambda_a = 0$, $\lambda_s = 1$, in order to exploit the peculiarities of ADAM and SGD. This approach changes the two hyper-parameters at each epoch, and in particular we have $\lambda_a = 1 - \lambda_s$ with $\lambda_s(ep) = ep/P$, where P is the maximum number of epochs and ep is the current epoch.

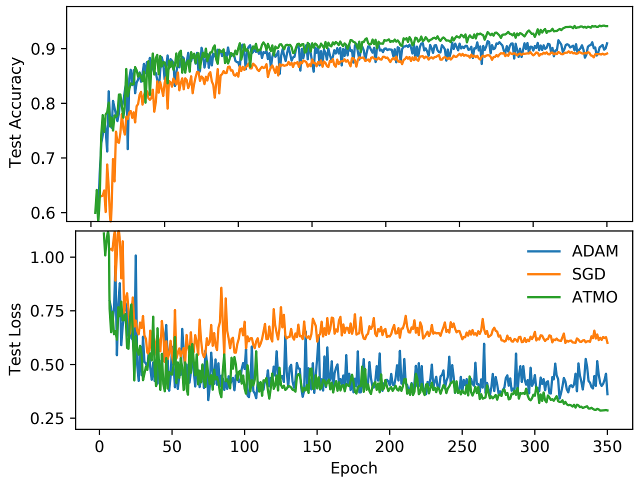

Figure 4. Resnet18 test accuracies with the best parameters of the best results obtained on Cifar10.

5. Datasets

In this section, we briefly describe the datasets used in the experimental phase.

The **Cifar10** [26] dataset consists of 60,000 images divided into 10 classes (6000 per class) with a training set size and test set size of 50,000 and 10,000, respectively. Each input sample is a low-resolution color image of size 32×32. The 10 classes are airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

The **Cifar100** [26] dataset consist of 60,000 images divided into 100 classes (600 per classes) with a training set size and test set size of 50,000 and 10,000, respectively. Each input sample is a 32×32 colour image with a low resolution.

The **Corpus of Linguistic Acceptability** (CoLA) [27] is another dataset that contains 9594 sentences belonging to training and validation sets and excludes 1063 sentences belonging to a set of tests kept out. In our experiment, we only used the training and test sets.

The **AG's news corpus** [28,29] is the last dataset used in our experiments. It is a dataset that contains news articles from the web subdivided into four classes. It has 30,000 training samples and 1900 test samples.

6. Experiments

The optimizer ATMO proposed is a generic solution not oriented exclusively to image analysis, so we conducted experiments on both image classification and text document classification. By doing so, we are able to give a clear indication of the behavior of the proposed optimizer in different contexts, also bearing in mind that many problems, such as audio recognition, can be traced back to image analysis. In all the experiments, unless differently specified, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $amsgrad = False$, dampening $d = 0$, $nesterov = False$ and a batch size near to the maximum our hardware can support. The hyper-parameters are set to obtain good results without trying to maximize accurac; this is because if the loss function is the same, all well-set optimizers find the same minimum in the long run.

In Table 1, we apply Dynamic ATMO method by combining Padam [18] with SGD to compare it with other recently proposed solutions, and it shows that many other optimizers can be combined with our proposed method. In this experiment, λ_a for Padam changes from 1 to 0 in the first 100 epochs. We trained ATMO for 200 epochs in total with $\eta = 0.1$, which is multiplied by 0.1 at epoch 100 and 150 and $\mu = 0.9$, $\gamma = 5 \cdot 10^{-4}$. The partial adaptive parameter for Padam was set to $\frac{1}{8}$.

Table 1. Best accuracies in percentage to compare ATMO (that combines Padam and SGD) and the results published in [18] using a Resnet18 on the Cifar10 dataset.

SGD-Momentum	ADAM	Amsgrad	AdamW	Yogi	AdaBound	Padam	Dynamic ATMO
95.00	92.89	93.53	94.56	93.92	94.16	94.94	**95.27**

6.1. Experiments with Images

In this first group of experiments, we used two well-known image datasets for: (1) conducting an analysis of the two main parameters of ATMO, λ_a and λ_s; (2) comparing the performance of ATMO with respect to the two starting optimizers SGD and ADAM; (3) analyzing the behavior of ATMO with different neural models. The datasets used in this first group of experiments are Cifar10 and Cifar100, and and the results are summarized in Tables 2 and 3. The neural models we compared are Resnet18 and Resnet34 [30,31]. For each model, we used, respectively, 1024 and 512 as batch size. We analyzed different values of λ_a and λ_s and also Dynamic ATMO.

For both Cifar10 and Cifar100 experiments, we used the following parameters: 350 epoch, Momentum 0.95, weight decay 0.0005 and learning rate 0.001, with Cosine Annealing for learning rate reduction. We performed simple data augmentation with random horizontal flip and random crops. Dynamic ATMO linearly changes the optimizer lambda from $\lambda_a = 1$, $\lambda_s = 0$ and $\lambda_a = 0$, $\lambda_s = 1$.

To better understand what happens during the training phase, in Figure 4, we represent the accuracy of the test and the corresponding loss values of the experiments that produced the best results with Resnet18 on Cifar10. We can see also the effectivenes of Dynamic ATMO without epoch noise in Figure 5. As we can see in the first part of training, ADAM has a good growth, so Dynamic ATMO inherits this trend. As we can see, Dynamic ATMO overcomse others optimizers because it has the convergence speed of ADAM in the early stages of the learning process, while in the late stage, it benefits more and more from SGD. Therefore, in general, we can say that the ATMO optimizer leads to better generalization than the other optimizers used. In addition, looking at the results obtained with the Resnet34, we can say that all configuration of ATMO exceeds the average and the maximum accuracies of SGD and ADAM.

Table 2. Mean accuracies (avg. acc) and maximum accuracies (acc max) in percentage on Cifar10, after 7 runs and 350 epochs for each run. Dyn. ATMO shows the accuracies obtained with the use of dynamic lambdas.

Name	λ_a	λ_s	**avg acc.**	**acc max**
Resnet18				
Adam	1	0	91.42	91.55
SGD	0	1	89.12	89.52
ATMO	0.5	0.5	92.07	92.40
ATMO	0.4	0.6	92.12	92.35
ATMO	0.6	0.4	91.84	91.99
ATMO	0.7	0.3	91.87	92.14
ATMO	0.3	0.7	91.99	92.17
Dyn. ATMO	$[1, 0]$	$1 - \lambda_a$	**94.02**	**94.22**
Resnet34				
Adam	1	0	91.39	91.68
SGD	0	1	90.94	91.43
ATMO	0.5	0.5	92.30	92.46
ATMO	0.4	0.6	92.24	92.63
ATMO	0.6	0.4	92.21	92.34
ATMO	0.7	0.3	92.01	92.18
ATMO	0.3	0.7	92.26	92.84
Dyn. ATMO	$[1, 0]$	$1 - \lambda_a$	**93.88**	**94.11**

Table 3. Accuracy results on Cifar100, after 3 runs of 350 epochs.

Name	**avg acc.**	**acc max**
Resnet18		
Adam	67.52	68.11
SGD	63.26	63.74
Dynamic ATMO	**74.27**	**74.48**
Resnet34		
Adam	68.26	68.64
SGD	66.38	67.26
Dynamic ATMO	**73.85**	**73.89**

In conclusion, as we have seen from the results shown in Tables 2 and 3, the proposed method leads to a better generalization than the other optimizers used in each experiment. We get better results both by setting λ_a and λ_s well, and also even when we do not use the best set of parameters.

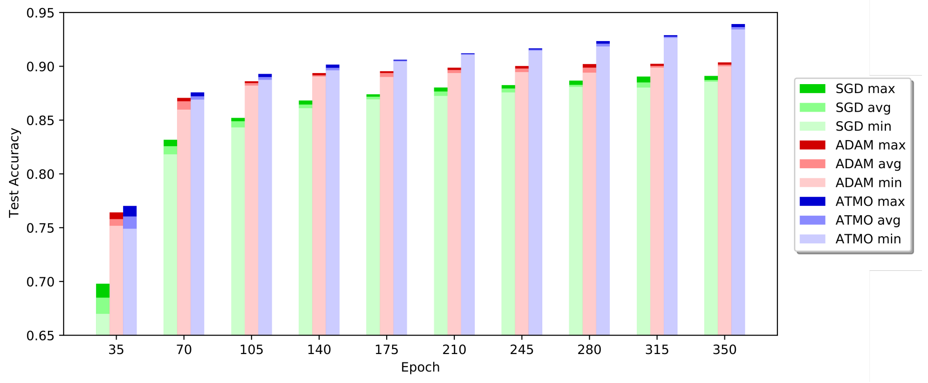

Figure 5. Resnet18 test accuracies min, max, and avg computed every 35 epochs using all experiments.

6.2. Experiments with Text Documents

In this last group of experiments, we used the two datasets of text documents: CoLA and AG's News. As a neural model, we used a model based on BERT [17], which is one of the best techniques for working with text documents. To run fewer epochs, we used a pre-trained version [32] of BERT. In these experiments, we also introduced the comparison with the AdamW optimizer, which is usually the optimizer used in BERT-based models.

For the CoLA dataset, we set $\eta = 0.0002$, momentum $\mu = 0.95$, and batch size equal to 100. We ran the experiments five times for 50 epochs. For the AG's News dataset, we set the same parameters used for CoLA, but we only ran it for 10 epochs because it achieved good results in the firsts epochs and also because the dataset was very large and therefore took more time. In these experiments with text analysis, we did not use the Dynamic ATMO approach because we used a very small number of epochs. We can see all the results in Table 4. Even for text analysis problems, we can confirm the results of the experiments done on images: although AdamW sometimes has better performances than ADAM, ATMO performs better.

Table 4. Accuracy results of BERT pre-trained on CoLA (50 epochs) and AG's news (10 epochs) after 5 runs.

Name	λ_a	λ_s	**avg acc.**	**acc max**
CoLA				
AdamW	-	-	78.59	85.96
Adam	1	0	79.85	83.30
SGD	0	1	81.48	81.78
ATMO	0.5	0.5	85.92	86.72
ATMO	0.4	0.6	86.18	**87.66**
ATMO	0.6	0.4	85.45	86.34
ATMO	0.7	0.3	84.66	85.78
ATMO	0.3	0.7	**86.34**	86.91
AG's News				
AdamW	-	-	92.62	92.93
Adam	1	0	92.55	92.67
SGD	0	1	91.28	91.39
ATMO	0.5	0.5	93.72	93.80
ATMO	0.4	0.6	93.82	93.98
ATMO	0.6	0.4	93.55	93.67
ATMO	0.7	0.3	93.19	93.32
ATMO	0.3	0.7	**93.86**	**93.99**

6.3. Time Analysis

We also provide a study about the average computational time of ATMO compared with other optimizers. We computed the mean computational time for one epoch in seconds. The experiments were conducted on different datasets as well as different neural models. We conducted all the experiments on a Nvidia 1080 with 8 GB of RAM.

We report the results in Table 5. Considering each row of the table, we can conclude that the computation time is almost the same for all optimizers, and the differences depend on the operating system overhead. We can therefore conclude that our approach does not add computation time overhead.

Table 5. Mean computational time for one epoch in seconds (s) for each experiment. We computed the train and test time together.

Dataset	Model	SGD	ADAM	ADAMW	ATMO
Cifar 10					
	Resnet18	32.85 s	33.05 s	-	32.34 s
	Resnet34	52.54 s	52.47 s	-	52.07 s
Cifar 100					
	Resnet18	33.83 s	34.42 s	-	33.56 s
	Resnet34	52.72 s	53.17 s	-	52.17 s
CoLA					
	BERT	64.05 s	63.67 s	60.34 s	62.6 s
AG's News					
	BERT	916.54 s	898.89 s	867.98 s	901.19 s

7. Conclusions

In this paper, we introduced ATMO (AdapTive Meta Optimizer), which is a new combined optimization method that combines the capability of two different optimizers into one. We demonstrated through experiments that our ATMO meta-optimizer can outperform the performance of individual optimizers introducing a negligible time complexity. To balance the contribution of the optimizers used within ATMO, we introduced two new hyperparameters λ_a, λ_s and showed experimentally that, using ADAM and SGD, the combination of these two hyperparameters can be set automatically without having to manually configure them. In the present work, we also tried to combine different optimizers such as Padam and SGD, obtaining also in this case the best accuracy compared to the accuracies present in the literature.

Author Contributions: Conceptualization, N.L. and R.L.G.; methodology, N.L., I.G. and R.L.G.; software, N.L.; validation, N.L., I.G. and R.L.G.; formal analysis, N.L., I.G. and R.L.G.; resources, N.L. and R.L.G.; writing—original draft preparation, N.L. and I.G.; writing—review and editing, N.L. and I.G.; visualization, N.L. and R.L.G.; supervision, N.L. and I.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Datasets used in this paper are in the public domain and can be found in the following links: CiFAR https://www.cs.toronto.edu/~kriz/cifar.html; CoLA https://nyu-mll.github.io/CoLA/; AGnews http://groups.di.unipi.it/~gulli/\AG_corpus_of_news_articles.html (accessed on 15 October 2020).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Robbins, H.; Monro, S. A stochastic approximation method. *Ann. Math. Stat.* **1951**, *22*, 400–407. [CrossRef]
2. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.

3. Zaheer, M.; Reddi, S.; Sachan, D.; Kale, S.; Kumar, S. Adaptive methods for nonconvex optimization. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2018; pp. 9793–9803.
4. Luo, L.; Xiong, Y.; Liu, Y.; Sun, X. Adaptive Gradient Methods with Dynamic Bound of Learning Rate. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
5. Bera, S.; Shrivastava, V.K. Analysis of various optimizers on deep convolutional neural network model in the application of hyperspectral remote sensing image classification. *Int. J. Remote Sens.* **2020**, *41*, 2664–2683. [CrossRef]
6. Graves, A. Generating sequences with recurrent neural networks. *arXiv* **2013**, arXiv:1308.0850.
7. Duchi, J.; Hazan, E.; Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.* **2011**, *12*, 2121–2159.
8. Zeiler, M.D. Adadelta: an adaptive learning rate method. *arXiv* **2012**, arXiv:1212.5701.
9. Kobayashi, T. SCW-SGD: Stochastically Confidence-Weighted SGD. In Proceedings of the 2020 IEEE International Conference on Image Processing (ICIP), Abu Dhabi, United Arab Emirates, 25–28 October 2020; pp. 1746–1750.
10. Zhang, Z. Improved adam optimizer for deep neural networks. In Proceedings of the 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), Banff, AB, Canada, 4–6 June 2018; pp. 1–2.
11. Pawełczyk, K.; Kawulok, M.; Nalepa, J. Genetically-trained deep neural networks. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, Kyoto, Japan, 15–19 July 2018; pp. 63–64.
12. Landro Nicola, G.I.; Riccardo, L.G. Mixing ADAM and SGD: A Combined Optimization Method with Pytorch. 2020. Available online: https://gitlab.com/nicolalandro/multi_optimizer (accessed on 18 June 2021).
13. Keskar, N.S.; Socher, R. Improving generalization performance by switching from adam to sgd. *arXiv* **2017**, arXiv:1712.07628.
14. Cui, X.; Zhang, W.; Tüske, Z.; Picheny, M. Evolutionary stochastic gradient descent for optimization of deep neural networks. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2018; pp. 6048–6058.
15. Loshchilov, I.; Hutter, F. Decoupled weight decay regularization. *arXiv* **2017**, arXiv:1711.05101.
16. Loshchilov, I.; Hutter, F. Fixing Weight Decay Regularization in Adam. 2018. Available online: https://openreview.net/forum?id=rk6qdGgCZ (accessed on 18 June 2020).
17. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
18. Chen, J.; Zhou, D.; Tang, Y.; Yang, Z.; Gu, Q. Closing the generalization gap of adaptive gradient methods in training deep neural networks. *arXiv* **2018**, arXiv:1806.06763.
19. Krogh, A.; Hertz, J.A. A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 1992; pp. 950–957.
20. Sutskever, I.; Martens, J.; Dahl, G.; Hinton, G. On the importance of initialization and momentum in deep learning. In Proceedings of the International Conference on Machine Learning, Atlanta, GA, USA, 16–21 June 2013; pp. 1139–1147.
21. Damaskinos, G.; Mhamdi, E.M.E.; Guerraoui, R.; Patra, R.; Taziki, M. Asynchronous Byzantine machine learning (the case of SGD). *arXiv* **2018**, arXiv:1802.07928.
22. Liu, C.; Belkin, M. Accelerating SGD with momentum for over-parameterized learning. *arXiv* **2018**, arXiv:1810.13395.
23. Reddi, S.J.; Kale, S.; Kumar, S. On the convergence of adam and beyond. *arXiv* **2019**, arXiv:1904.09237.
24. Lee, J.D.; Simchowitz, M.; Jordan, M.I.; Recht, B. Gradient descent only converges to minimizers. In Proceedings of the Conference on Learning Theory, New York, NY, USA, 23–26 June 2016; pp. 1246–1257.
25. Rosenbrock, H. An automatic method for finding the greatest or least value of a function. *Comput. J.* **1960**, *3*, 175–184. [CrossRef]
26. Krizhevsky, A.; Hinton, G. *Learning Multiple Layers of Features from Tiny Images*; Citeseer: University Park, PA, USA, 2009.
27. Warstadt, A.; Singh, A.; Bowman, S.R. Neural Network Acceptability Judgments. *arXiv* **2018**, arXiv:1805.12471.
28. Gulli, A. AG's Corpus of News Articles. 2005. Available online: http://groups.di.unipi.it/~gulli/\AG_corpus_of_news_articles.html (accessed on 15 October 2020).
29. Zhang, X.; Zhao, J.; LeCun, Y. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2015; pp. 649–657.
30. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. *arXiv* **2015**, arXiv:1512.03385.
31. Targ, S.; Almeida, D.; Lyman, K. Resnet in resnet: Generalizing residual architectures. *arXiv* **2016**, arXiv:1603.08029.
32. Huggingface.co. Bert Base Uncased Pre-Trained Model. Available online: https://huggingface.co/bert-base-uncased (accessed on 15 October 2020).