UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

IMPLEMENTATION AND STUDY OF ADAM-BASED ALGORITHMS

Relatore:
prof. Loris Nanni

Laurenado:
Lorenzo Giannini

ANNO ACCADEMICO: 2021/2022
Data di laurea: 22 Settembre 2022

Abstract

The following thesis proposes an analysis of some variants of the ”Adam” optimizer (adaptive moment estimation) used mainly in the training of convolutional neural networks, also known as ”CNN”. This algorithm is derived from ”SGD” (stochastic gradient descent) based optimizer, a method which, to decrease the error function of the classifier, updates the weights of the various neurons in the network using the gradient of a ”Loss Function”. Starting with the implementation and analysis of ”AngularGrad” and ”AdaInject”, recently idealized variants of the ”Adam” algorithm, they will then be used to create ensembles using variants already proposed in the literature.

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Artificial Intelligence

A good AI definition can be:

> ”Artificial intelligence (AI) systems are software (and possibly also hardware) systems designed by humans that, given a complex goal, act in the physical or digital dimension by perceiving their environment through data acquisition, interpreting the collected structured or unstructured data, reasoning on the knowledge, or processing the information, derived from this data and deciding the best action(s) to take to achieve the given goal [1]”.

AI, in fewer words, can also be defined as the partial reproduction of some of the intellectual activity of man (mainly learning, recognition, choice) through the construction of ideal mathematical models with the use of electronic computers and programming techniques.

Some fields in which this discipline focuses are knowledge rapresentation, communication and planning. It can be totally software-based, like a voice assistant, or embedded in a hardware device, like an autonomous car.

1.2 Machine Learning

> ”Today, intelligent systems that offer artificial intelligence capabilities often rely on machine learning. Machine learning describes the capacity of systems to learn from problem-specific training data to automate the process of analytical model building and solve associated tasks [2]”.

As mentioned in the article just cited, in the last decade in the field of ML there have been notable developments in the search for sophisticated learning algorithms and efficient pre-processing techniques. In tasks related to high-dimensional data, such as classification, clustering and regression, ML tries to overcome the difficulty of humans to explain all of the tacit knowledge that is required, automating the task of analytical model building. This can be achieved by applying

algorithms that iteratively learn from problem-specific training data, which allows computers to find complex patterns without explicitly being programmed.

Based on the type of problem and the amount of data available, ML can be divided into three categories: supervised learning, unsupervised learning, and reinforcemement learning. Finally, depending on the type of task, there are numerous algorithms available, each with different specifications and variants.

Below, in Figure 1.1, a diagram on the various concepts of Machine Learning.

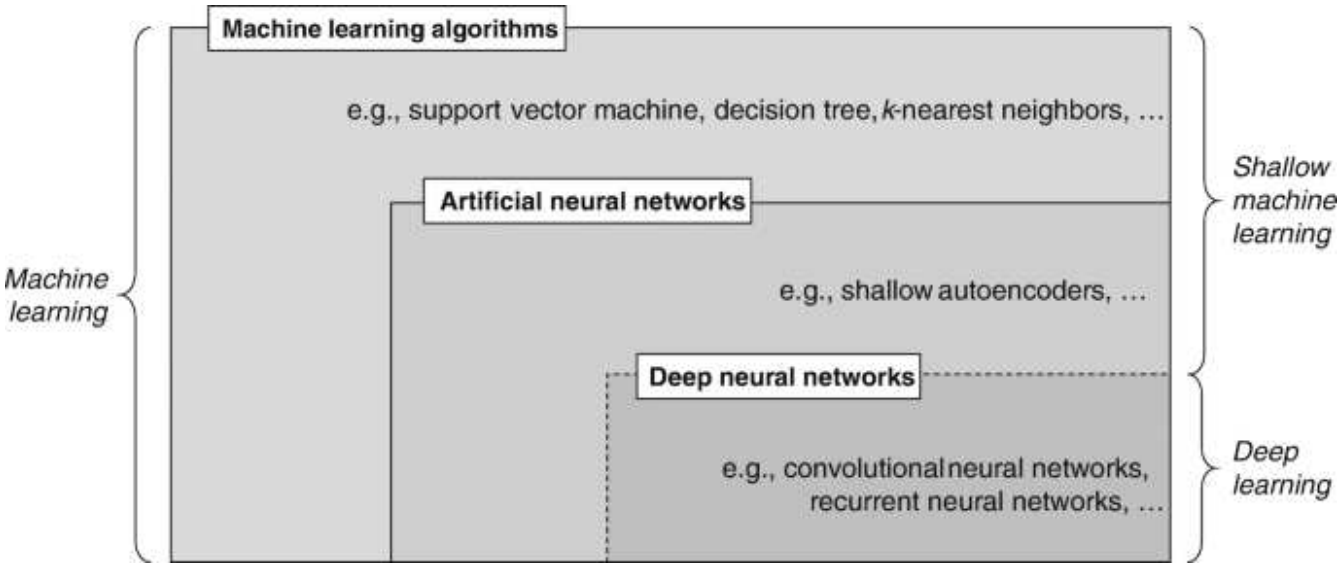

Figure 1.1: Venn diagram of machine learning concepts and classes (inspired by Goodfellow et al. 2016, p. 9).

1.3 Artificial Neural Networks

An artificial neural network (ANN) is a computational model composed of interconnected processing units, called ”artificial neurons,” inspired by its biological counterpart. Introduced by McCulloch and Pitts (M&P) in 1943 [3], the first artificial neuron model functioned as a logic gate, with only two internal states. These ANNs could then only perform basic logic functions, such as AND, OR and NOT. Neurons receive data as input and then aggregate it and make it available to a given activation function. The model proposed by M&P could have only Boolean values as inputs, all of which had equal weight and contributed with equal importance to the decision, and the activation function was a simple decision threshold, which had to be predetermined. These limitations meant that the only functions that could be represented were linear ones while nonlinear ones, such as XOR, could not.

A new model of artificial neuron called ”perceptron” was proposed in 1957 by F. Rosenblatt [4]. This new neuron is capable of ”learning” from given data. Unlike the M&P model, each input has a distinct weight and these inputs can take on real values instead, the activation function is a heaviside step function as seen in Figure 1.2.

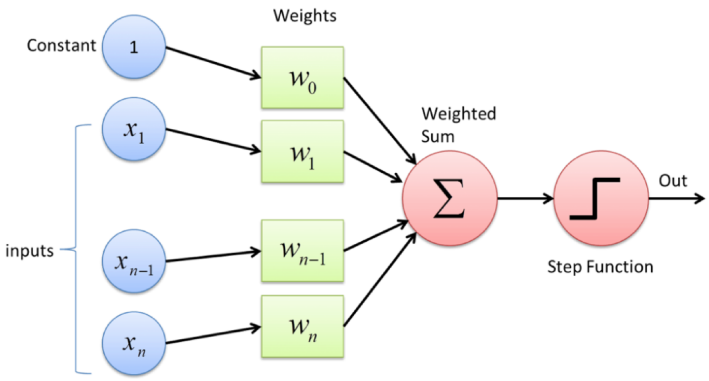

Figure 1.2: Perceptron scheme

Thanks to these new features, the perceptron becomes optimal for linear classification since it identifies the hyperplane of class separation given by the formula:

$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + w_0 = 0$$

For the following decades the networks remained with a single layer of perceptrons, thus limiting them to learning only linear functions, until Multilayer Perceptron (MLP) networks were introduced in the 1980s. In these networks neurons are organized in different layers. An input layer usually receives the data input, and an output layer produces the ultimate result. In between, there are zero or more hidden layers as can be seen in Figure 1.3, responsible for learning a non-linear mapping between input and output. In these models there are HyperParameters, such as learning rate or activation function, which are parameters that cannot be determined by training but set manually or determined by an optimization routine [2].

These ANNs are feedforward networks, that is, neurons in a certain layer are connected only with neurons in the next layer. The activation function most historically used by MLP networks is the sigmoid function. These new networks are capable of learning any continuous function in a compact set on $\mathbb{R}$.

The more complex the functions to be approximated the more exponentially the number of hidden layers must be increased.

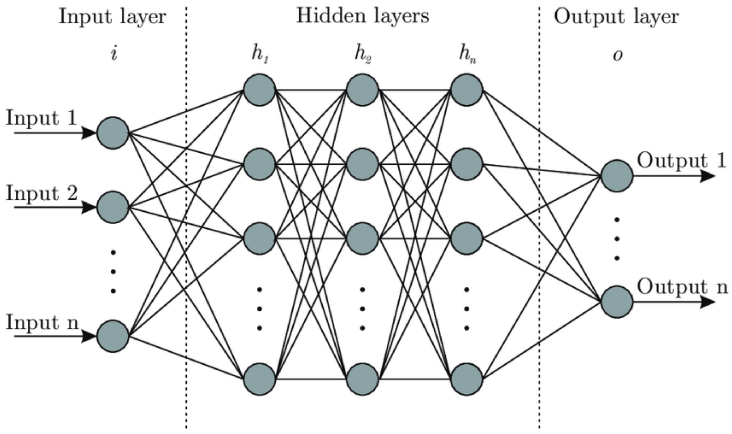

Figure 1.3: An example of an MLP network

1.4 Deep Learning

In recent decades there have been many developments in the field of ML, including the evolution of ANNs into Deep Neural Networks (DNNs) that enhance their learning capabilities, such as the example demonstrated in Figure 1.4. The field in which these DNNs are studied is called Deep Learning (DL). Development in this field is mainly due to the creation of vast datasets for training and the increasing computing power of available hardware.

For many applications, deep learning models outperform shallow machine learning models and traditional data analysis approaches [2].

A formal definition of DNNs might be:

> ”A Deep Neural Network (DNN) is defined to be an Artificial Neural Network (ANN) with at least one hidden layer of units between the input and output layers. The extra layers give it added levels of abstraction, thus enhancing it’s modelling capability” [5].

In addition, the neurons of which they are formed are generally more complex than those found in ANNs. They generally perform either more complex operation, such as convolution, or multiple activations per single neuron. These features make the core capability of DNNs to directly receive raw input data and automatically discover the corresponding learning task.

DL is great in case you need to handle a large amount of high dimensional data, such as text, image, video and audio processing [2].

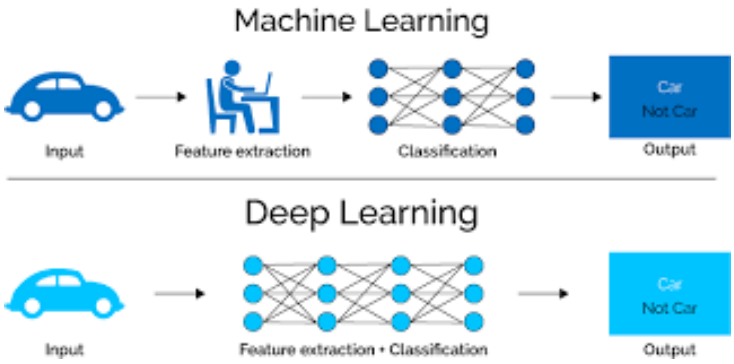

Figure 1.4: An example of the difference between ANNs and DNNs

However, there are also disadvantages with DNNs, such as overfitting and the computational time required. Overfitting happens when the network tends to ”memorize” details instead of ”learning” them, causing it to perform poorly when the input data are different [5]. This problem can be solved by monitoring the progress of iterative training and stopping it at the ideal point. For the computational time required instead, a good solution is to use GPUs so as to take advantage of their enormous processing power for the matrix and vector computations required.

1.5 CNN

One of the most popular types of DNNs are convolutional neural networks (CNNs). This type of network has been introduced since 1998 with Yann LeCun's proposal of the network "LeNet-5" [6], whose architecture is shown in Figure 1.5, after a study on back-propagation started in 1988 [7].

CNNs are variants of MLP networks that exploit certain new features to reduce their overall complexity. The first is local processing, i.e., each neuron is connected only locally to neurons in the next layer thus leading to a strong reduction in the number of connections. The second is group shared weights , that is, different neurons of the same level perform the same type of processing on different portions of input, trivially leading to a total reduction in the number of weights. Another difference with MLP networks are the activation functions used, generally the sigmoid function, which causes the vanishing (or exploding) gradient problem in deep networks [8], has been replaced since 2011 by the "Rectified Linear Units"(ReLu) activation function [9].

At last the architecture of CNNs is extremely different from MLP networks. A CNN is composed of a hierarchy of layers, the last ones are generally fully-connected and operate like an MLP classifier, while at intermediate levels the features described earlier are used. They are also divided into blocks that exploit convolutional, pooling, and activation layers.

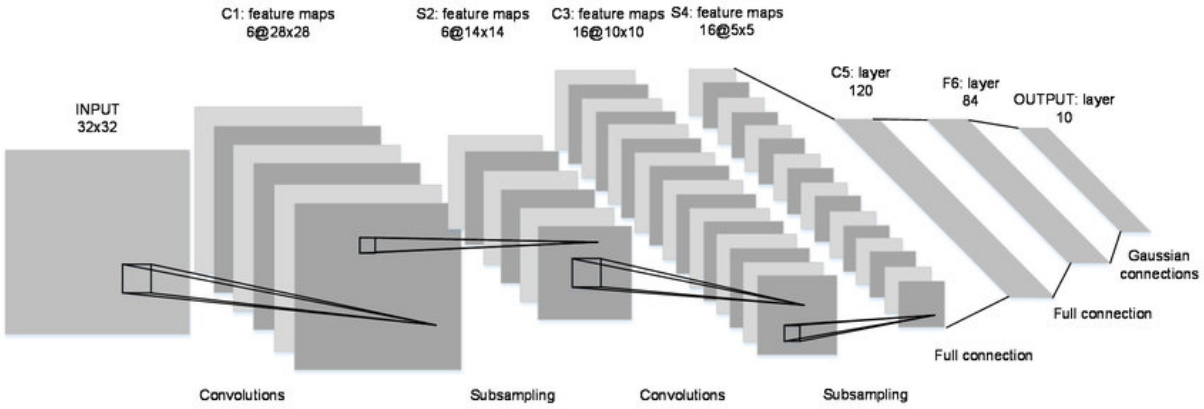

Figure 1.5: LeNet-5 architecture

1.6 Training

In this section we will briefly discuss supervised training. First the network is presented with data whose class is already known, after which the output produced z by the network for each pattern x is compared with the desired output t through a loss function, one of which is the sum of squares of the errors:

$$J(w,x) = \frac{1}{2} \sum_{c=1...s} (t_c - z_c)^2$$

which represents the difference between the two outcomes. The overall error $J(w)$ is the average of all $J(w,x)$. To reduce $J(w)$ we modify the weights w by following, through the geometric

interpretation of the gradient theorem, the opposite direction of the gradient of J. This algorithm, based on the backpropagation concept first announced by Rumelhalt, Hinton and Williams [10], is named *gradient descent*, the graphical interpretation of which can be seen in Figure 1.6.

For a deeper understanding of this algorithm it is recommended to read "An overview of gradient descent optimization algorithms" [11] by Sebastian Ruder, for this thesis we will stop at a general formula concerning the updating of parameters viz:

$$\theta = \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$

where η is the learning rate that is, the size of the steps taken to reach the minimum.

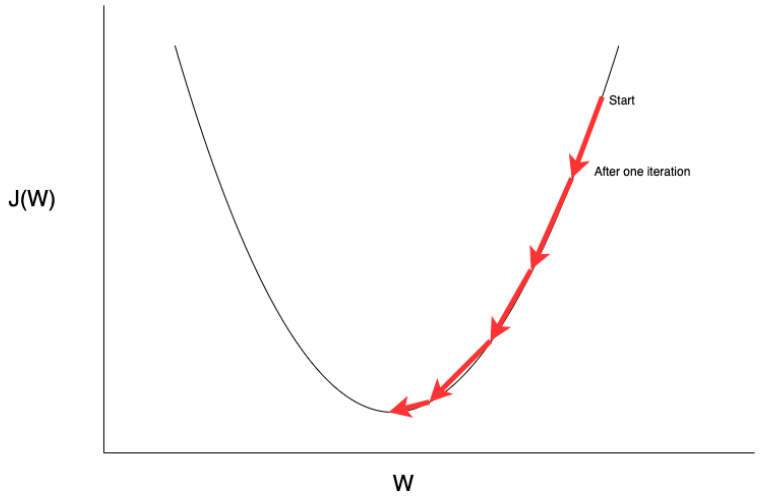

Figure 1.6: Graphical interpretation of gradient descent operation

Chapter 2

Optimizers

In this chapter we are going to cover the topic of optimizers, starting with what they are and ending with an explanation of some of Adam's variants.

Optimizers are methods or algorithms that reduce the loss function by updating network parameters. As one can realize from this definition, the gradient descent algorithm, discussed in the previous chapter, can also be considered an optimizer. In fact, the latter is the basis for numerous optimization algorithms.

2.1 Stochastic gradient descent

The *Stochastic gradient descent* (SGD) algorithm, compared to gradient descent, instead of calculating the gradient of the entire dataset for each step, performs an update by calculating the gradient relative to one parameter at a time. That is, the update function becomes:

$$\theta = \theta - \eta \nabla J(\theta; x_i, y_i)$$

with x_i and y_i a given pattern and its label, respectively. This means that for each epoch, which is the presentation of all n patterns of the training set to the network, there will be n updates. SGD therefore requires less space to perform operations even though since these are smaller they are more frequent.

SGD performs frequent updates with high variance that heavily fluctuate the objective function. These fluctuations allow it to move much more than before and perhaps reach new, potentially better, local minima. On the other hand, however, these fluctuations complicate the convergence of the loss function as it may overshoot. This phenomenon can be partly counteracted by decreasing the learning rate [11].

Another variant called *mini-batch gradient descent* involves dividing the training set into batches, and updating the parameters occurs once each pattern in the batch has been visited.

$$\theta = \theta - \eta \nabla J(\theta; x_{1:k}, y_{1:k})$$

with k the number of patterns in a batch.

The difference between these three variants is represented graphically in Figure 2.1

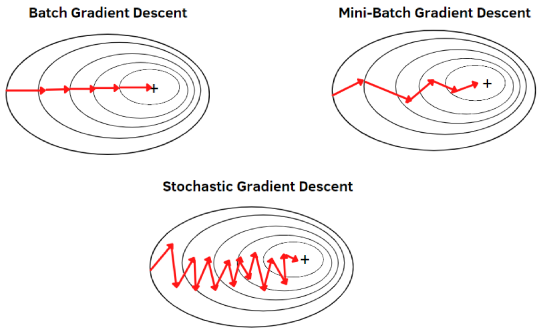

Figure 2.1: The difference beetween gradient descent, SGD and mini-batch gradient descent

2.2 Adam

In 2014 the *Adam* algorithm, which stands for *ADAptive Moment estimation*, was proposed by Diederik P. Kingma and Jimmy Lei Ba [12]. Ispired by previous algorithms like *Adadelta* [13] and *RMSprop*[1], it combines the ideas of momentum and the adaptive gradient. In addition to maintaining exponential moving average (EMA) of past squared gradients v_t, like the algorithms just mentioned, it also maintains an EMA of past gradients m_t similar to momentum [14].

Henceforth m_t will be referred to as *first moment* while v_t as *second moment* and represent the mean and the uncentered variance of the gradients respectively. Adam defines his EMAs as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

where β_1 and β_2 are hyperparameters representing the exponential decay rate for the two Ema (generally set at 0.9 and 0.999 respectively), while g_t represents the gradient. The moments are initialized as vectors of 0's. It was noticed that in the first iterations these tend to zero, so to prevent this, the authors developed a bias-correction to be performed:

$$\widehat{m_t} = \frac{m_t}{1 - \beta_1^t}$$

$$\widehat{v_t} = \frac{v_t}{1 - \beta_2^t}$$

Finally, the updating of the parameters is done according to this formula:

$$\theta_t = \theta_{t-1} - \eta \frac{\widehat{m_t}}{\sqrt{\hat{v}_t} + \epsilon}$$

where ϵ is a really small positive real number placed to avoid a fraction by zero.

[1]RMSprop is an unpublished, adaptive learning rate method proposed by Geoff Hinton in Lecture 6e of his Coursera Class. URL: 12http://www.cs.toronto.edu/ tijmen/csc321/slides/lecture$_s lides_l ec6.pdf$

2.3 DiffGrad

Proposed in 2019 [15] diffGrad is an optimizer that, unlike Adam, dynamically adjusts the learning rate. One of the problems of Adam is with controlling friction for the first moment in order to avoid overshooting near to an optimum solution. To solve it diffGrad takes into account the change in short-term gradients by inserting a new coefficient in the parameter update formula.

The first step is to compute the absolute difference of the gradients of two consecutive steps:

$$\Delta g_t = g_{t-1} - g_t$$

The new coefficient is defined as:

$$\xi_t = \frac{1}{1 + e^{-|\Delta g_t|}}.$$

DiffGrad does not change how the moments m_t and v_t are computed, and like Adam, it uses their bias-corrected versions $\widehat{m_t}$ and $\widehat{v_t}$ in the parameter update formula. The formula then becomes:

$$\theta_t = \theta_{t-1} - \eta \frac{\xi_t \cdot \widehat{m_t}}{\sqrt{\hat{v}_t} + \epsilon}$$

The importance of having an adequate learning rate (lr) is demonstrated graphically in Figure 2.2. The three cases are: A) Case where the lr is too small, the process does not reach the local minimum. B) Case in which the lr) is too large, it risks overshooting the local minimum.
C) Case in which lr adjusts dynamically.

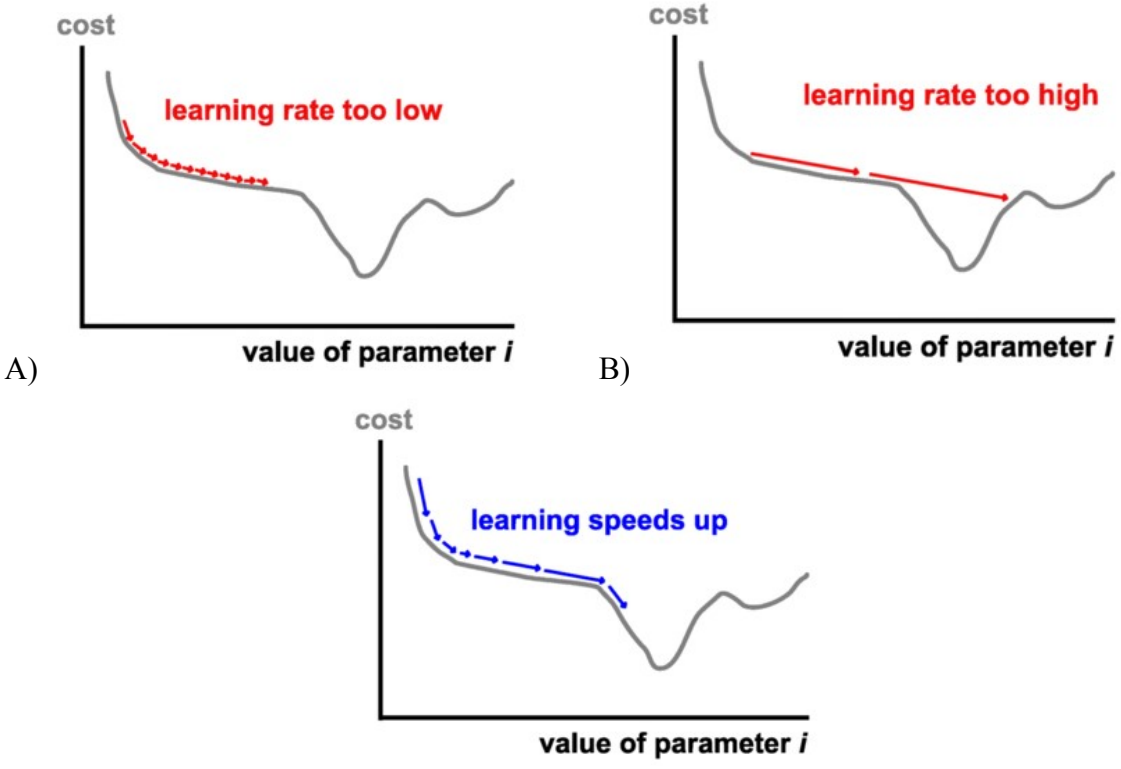

Figure 2.2: Different learning rates

2.4 AngularGrad

AngularGrad is one of Adam's newly published variants ispired by diffGrad, developed mainly to reduce the zig-zag effect in the optimization trajectory due to the high gradient variation that penalize the final result [16]. To do this, the algorithm also takes into account the information from the angle/direction of the gradient vector instead of just the magnitude of it.

To exploit the change of gradients during optimization steps a new angular coefficent was introduced. The latter is defined as follows:

$$\phi_t = \tanh\left(\angle(|A_{min}|)\right) \cdot \lambda_1 + \lambda_2$$

where λ_1 and λ_2 two hyperparameters that empirically perform best when set at 0.5 both.

The $\angle$, on the other hand, can take on two values. In fact, this optimizer arrives with two variants, in one case $\angle$ is the function cos while in the other the function tan . The coefficent A_t represents the angle between the current gradient and that of the previous iteration. A_{t-1} as can be guessed is the angle between the gradients of the previous iterations, namely g_{t-1} and g_{t-2}. Hence the term A_{min} is defined as:

$$A_{min} = \min\left(A_{t-1}, A_t\right).$$

The coefficent ϕ_t dynamically adjusts the learning rate by causing the parameter update to be lower in low-changing gradient regions and vice-versa. AngularGrad computes the first and second moments as Adam and diffGrad but in updating the parameters it inserts ϕ as follows:

$$\theta_t = \theta_{t-1} - \eta \frac{\phi_t \cdot \widehat{m_t}}{\sqrt{\hat{v}_t} + \epsilon}$$

As a result, due to these new changes, fluctuations in the trajectory are significantly smoothed out by plotting a more direct path toward the minimum of the cost function. Another advantage gained is the reduction in computational power required.

2.5 AdaInject

Like AngularGrad, AdaInject is one of the latest Adam variants released [17]. This variant injects the second order momentum into the weight upgrade formula. The curvature information is obtained through short-term parameters.

Unlike the variants seen before, AdaInject is not a stand-alone approach, but can be integrated with any existing adaptive momentum stochastic gradient descent approach.

In Adam the first-order moment m_t is used to update the parameters, while the second-order moment v_t is used to control the learning rate. From this it can be seen that Adam mainly relies on gradients.

AdaInject instead proposes to inject the curvature information weighted second order mo-

mentum into first order momentum. To do this a new EMA is presented:

$$s_t = \beta_1 s_{t-1} + (1 - \beta_1) \cdot \frac{(g_t + \Delta\theta \cdot g_t^2)}{k}$$

where $\Delta\theta$ represents the difference at short-term of the parameters, thus $\theta_{t-2} - \theta_{t-1}$, while k is a real-valued hyperparameter that empirically is set to 2.

This new approach keeps the second-order moment v_t the same as Adam, but replaces the first-order moment m_t with the *inject moment* s_t just introduced. A bias-correction is also performed in AdaInject, respectively:

$$\widehat{s_t} = \frac{s_t}{1 - \beta_1^t}$$

$$\widehat{v_t} = \frac{v_t}{1 - \beta_2^t}$$

Finally, the parameter update rule is given as:

$$\theta_t = \theta_{t-1} - \eta \frac{\widehat{s_t}}{\sqrt{\widehat{v_t}} + \epsilon}.$$

These new features mean that AdaInject helps optimizers make larger upgrades to parameters in scenarios where, typically, there is a low gradient, such as flat regions, or where there are areas of low curvature, such as monotonous increase/decrease regions.

In the same article in which it was presented, AdaInject shows a boost in the performance of the optimizers to which it was applied.

2.6 Other Adam Variants

In this new section some variants of Adam and diffGrad previously seen will be analyzed. These are the approaches from which inspiration was drawn to develop the new variants proposed in this thesis. These algorithms were presented and studied in Bassani Matteo's bachelor's thesis, a computer engineering student at the "Department of Information Engineering - DEI" in Padua, Italy.[2].

2.6.1 Hyp

This approach follows diffGrad's philosophy in trying to dynamically update the learning rate. Unlike the old variant, instead of using the absolute sigmoid function, it uses the hyperbolic function, hence where the name (Hyperbolic) comes from.

[2]Thesis title: "Exploiting Adam-like Optimization Algorithms to Improve Performance of Multi-label Classification," year of publication: 2020, URL: http://hdl.handle.net/20.500.12608/5001

The revised parameter ξ_t is calculated as follows:

$$\xi_t = -\frac{1}{a \cdot \Delta g_t + b} + c$$

where a,b and c are initialized to 10, 2/3 and 3/2 respectively.

2.6.2 CLRW

The name of this variable stands for Cyclic Learning Rate with Warm restarts, and as it suggests it exploits the idea of a cyclic learning rate with a certain number of restarts being performed relative to the current number of epochs. The learning rate starts with its maximum value and then decays until it restarts. A graphical visualization of the Learning Rate trend by adopting this approach can be seen in Figure 2.3

This approach comes with three variations:

$$\xi_{1,t} = 0.5 + 0.5 \cos\left(\frac{\text{mod}(e, period)}{period} \cdot \pi\right) \tag{2.1}$$

$$\xi_{2,t} = \cos\left(\frac{\text{mod}(e, period)}{period} \cdot \frac{\pi}{2}\right) \tag{2.2}$$

$$\xi_{3,t} = 0.5 + 0.5 \cos\left(\frac{\text{mod}(e, period)}{period} \cdot \pi\right) \cdot exp^{-drop \cdot e} \tag{2.3}$$

where e is the current epoch, period=(number of epochs) / (number of restarts).

Whichever variant is chosen in each case the final step consists of:

$$X = 4 \cdot \xi_{i,t} \cdot \Delta \widehat{g_t}$$

$$\xi_t = \frac{1}{1 + exp^{-}X}$$

where $\Delta \widehat{g_t}$ represents the normalization of the short-term difference of the gradients and ξ_t is the coefficient that will be placed in the update function.

The purpose of this approach is to improve convergence in cases of ill-conditioned functions and to avoid ending up in suboptimal local minima.

The variant described in formula (2.3) decreases at each warm restart the maximum and minimum learning rate so as to control and contain the divergence phenomenon.

2.6.3 Linear CLRW

As the name suggests, this approach is nothing more than a linear version of CLRW that instead of degree returns values in radians. It is executed as follows:

$$\xi_{4,t} = \arccos\left(\cos\left(\frac{\text{mod}(e, period)}{period} \cdot \frac{\pi}{2}\right)\right)$$

2.6.4 CLRWDecreasing

This approach is also a variant of CLRW although more complex than the previous one.

At each warm restart, the first iteration is executed as the function (2.2), and the rest as follows:

$$\xi_{4,t} = \frac{y}{\frac{e}{period}} \cdot \left(2 + 0.5 \cos\left(\frac{\text{mod } (e, period)}{period} \cdot \pi\right)\right)$$

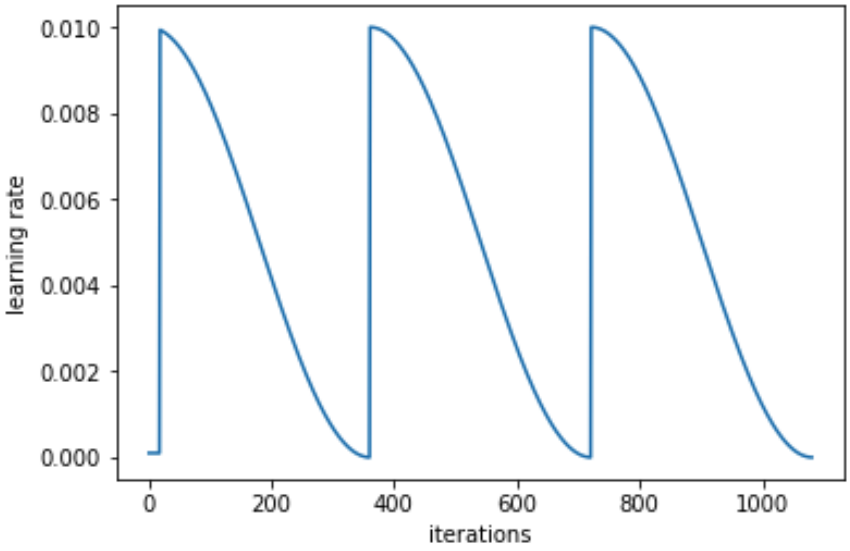

Figure 2.3: An example of cyclic warm restart

Chapter 3

New Optimizers Approach

This chapter will look at five new variants of the Adam optimizer. These have all been designed by building on some features found in ”AngularGrad” and ”AdaInject” and combining them with other existing approaches.

- **AngularHypGrad**: union of ”AngularGrad” and ”Hyp”;
- **HypAngularGrad**: another union of the concepts of ”AngularGrad” and ”Hyp”, variant of ”AngularHypGrad”;
- **AngularCLRWDecreasing**: merging of the concepts of ”AngularGrad” and ”CLRWDecreasing”;
- **LinearAngularCLRW**: merging of the concepts of ”AngularGrad” and ”LinearCLRW”;
- **AdaDoubleInject**: expanding the features of ”AdaInject”.

3.1 AngularHypGrad

This variant retains the main structure of the ”Hyp” approach, set forth in the previous chapter, except that it no longer uses the mere difference between gradients as the variable of the hyperbolic function. The new coefficient is obtained by applying the first step of ”AngularGrad,” that is, taking the lesser of A_t, which is the angle between the current gradient vector g_t and that of the previous iteration g_{t-1}, and A_{t-1}, which is the angle between g_{t-1} and that of the still previous iteration g_{t-2}.

A_t and A_{t-1} are computed according to this formula:

$$A_t = \arctan\left(\frac{|g_t - g_{t-1}|}{(1 + g_t \cdot g_{t-1})}\right)$$

$$A_{t-1} = \arctan\left(\frac{|g_{t-1} - g_{t-2}|}{(1 + g_{t-1} \cdot g_{t-2})}\right)$$

$$A_{min} = \min\left(A_t, A_{t-1}\right)$$

Recalling that the hyperbolic function is computed as follows:

$$\xi_t = -\frac{1}{a \cdot A_{min} + b} + c$$

where a,b and c retain their previous values, namely 10, 2/3 and 3/2 respectively.

The coefficient before being entered into the parameter update formula is normalized.

3.2 HypAngularGrad

This approach is very similar to AngularHypGrad but swaps the order of operations. In this variant, the hyperbola function is executed as many as twice per iteration. The variable for the first is the angle between g_t and g_{t-1}, while for the second is the angle A_{t-1}.

The results of these two functions is then compared and the lesser is chosen as the new coefficient ξ_t. As before, the coefficient is normalized before being used.

Other variants of these two approaches were tested, including trying to integrate more of the features of "AngularGrad" but none of them gave good results except for "AngularHypGrad" and "HypAngularGrad". The results will be presented in the next chapter.

3.3 AngularCLRWDecreasing

The combination of the "CLRWDecresing" and "AngularGrad" approaches led to the creation of this variant. Like the original approach, the latter applies the same operations except that, instead of using the simple difference between gradients (the current one and the immediate previous one), it follows the approach of "AngularGrad," i.e., the minor angle A_{min}. The only thing that changes then is the calculation of the variable X:

$$X = 4 \cdot \xi_{i,t} \cdot A_{min}.$$

Recalling that the remaining operations are carried out as follows:

$$\xi_{4,t} = \frac{y}{\frac{e}{period}} \cdot \left(2 + 0.5 \cos\left(\frac{\text{mod } (e, period)}{period} \cdot \pi\right)\right)$$

$$\xi_t = \frac{1}{1 + exp^{-}X}$$

3.4 LinearAngularCLRW

As easily guessed from the name, this approach is the union of the "AngularGrad" and "Linear CLRW" methods. Like the previous method exhibited, the only real variance from the original approach is in the calculation of the variable X. Again, the latter is calculated with A_{min} instead of the difference of gradients.

3.5 AdaDoubleInject

This approach is an expansion of the ”AdaInject” approach. Instead of exploiting the curvature information only to replace the first momentum m_t, it also uses the short-term parameter history to create a new EMA to replace the second momentum v_t.

The new EMAs thus become:

$$s_t = \beta_1 s_{t-1} + (1 - \beta_1) \cdot \frac{(g_t + \Delta\theta \cdot g_t^2)}{k}$$

$$p_t = \beta_1 p_{t-1} + (1 - \beta_2) \cdot \frac{(g_t + \Delta\theta \cdot g_t^2)}{k}.$$

remembering that $\Delta\theta$ is the difference between the parameters of the previous iteration with the current one.

As with the original approach, a bias-correction is computed at the EMAs:

$$\widehat{s_t} = \frac{s_t}{1 - \beta_1^t}$$

$$\widehat{p_t} = \frac{p_t}{1 - \beta_2^t}$$

Finally, following hand in hand with the ”parent,” ”AdaDoubleInject” is a general approach that can be applied to any pre-existing method by causing performance to be boosted.

Chapter 4

Experimental phase

This chapter will first expose the working environment and then turn to the results obtained from studying the approaches presented in the previous chapters.

4.1 Working environment

The analysis work in this thesis was done on the matlab platform, which is a programming and numerical computing platform used by millions of engineers and scientists for data analysis, algorithm development and model building [18]. The version of matlab used is R2022a using mainly the ”Deep Learning Toolbox” functions.

The basis for this thesis was the work done in Bassani Matteo’s thesis mentioned in the previous chapters. From there the program was modified so that the new ”AngularGrad” and ”AdaInject” approaches could be implemented.

It was only after the implementation of the two approaches just mentioned that research was devoted to finding new variants that had these approaches as their initial cues. The first tests carried out were performed with ”AlexNet” [19] as a neural network and using a subset of the ”LAR” dataset. This subset contains 1320 elements of which 880 were used as a training set and 440 as a test set. These specific tools were used for time reasons.

The pseudocode of the program is presented in the next page.

4.1.1 Pseudocode

Choose the boost method
load dataset
load network
for all fold in dataset **do**
    make data compatible with network input
    create training and test sets
    create graphic
    for all methods in approaches **do**
        set the parameters
        train the network
        classification of test patterns
        calculation of accuracy
    end loop
end loop

4.2 Experimental results

This section will proceed to expose and then analyze the results obtained in the work done in this thesis under the conditions and using the tools described above. All the results that are exposed are an average of the results, i.e. the accuracy, that each approach obtained in each fold. Existing methods are first exposed, while, later new methods are exposed, including the two versions of ”AngularGrad.” Below is the Table 4.1 with the percentage results obtained with the working environment described above. The table is divided into four columns. The first shows the approach used, the second shows the result obtained by following the directions of ”Adam,” and the third and fourth show the results obtained by the boost methods ”AdaInject” and ”AdaDoubleInject.”

Optimizer	Base	AdaInject	AdaDoubleInject
Adam	0.3954	0.64697	0.68483
Hyp	0.6106	0,7523	0.7682
CLRWDecreasing	0.5909	0.7167	0.7932
LinearCLRW	0.5212	0.6962	0.7364
AngularGrad			
AngularGrad(tan)	0.6212	0.7182	0.7393
AngularGrad(cos)	0.4682	0.6303	0.7061
New Approaches			
AngularHypGrad	0.6023	0.7387	0.7076
HypAngularGrad	0.6795	0.7053	0.73563
AngularCLRWDecreasing	0.6000	0.71817	0.7265
LinearAngularCLRW	0.5800	0.67347	0.7280

Table 4.1: Experimental results with AlexNet and subset

”AngularGrad” seems to work markedly better in its variant that uses the tangent function instead of the cosine. This fact is quite strange since in the article in which it was presented [16] they seemed to be quite similar. The implementation was performed following the creators' guidelines, nevertheless this result is probably due to the dataset and neural network used.

As for ”AdaInject,” this boost approach greatly increases the performance of all methods to which it has been applied, as actually specified in [17]. In some cases the performance increases by as much as 10%, while in the base case it presents a huge improvement.

”AdaDoubleInject,” like the previous method analyzed, boosts the accuracy of the methods to which it was applied. In most cases ”AdaDoubleInject” performs as well as ”AdaInject” if not better. Only in one case of those tested does this not happen, namely for the ”AngularHypGrad” approach.

As can be seen from the table, the accuracy of the new approaches is quite similar, ”HypAngularGrad” mainly stands out, showing slightly higher than average results. As for the variants of ”Hyp” these do not show improvement over the parent, except in the base case for ”HypAngularGrad.”

As for the other two variants, these seem to have an improvement over their basic CLRW counterparts, but not over the same boosted methods and the pure ”AngularGrad” method.

From here on, the Figures 4.1, 4.2, 4.3, 4.4, 4.5 and 4.6 represent some graphs of the loss function obtained during network training with the new approaches are shown. The first will always be the one obtained with the EMAs of ”Adam,” and the second and third with those of ”AdaInject” and ”AdaDoubleInject.”

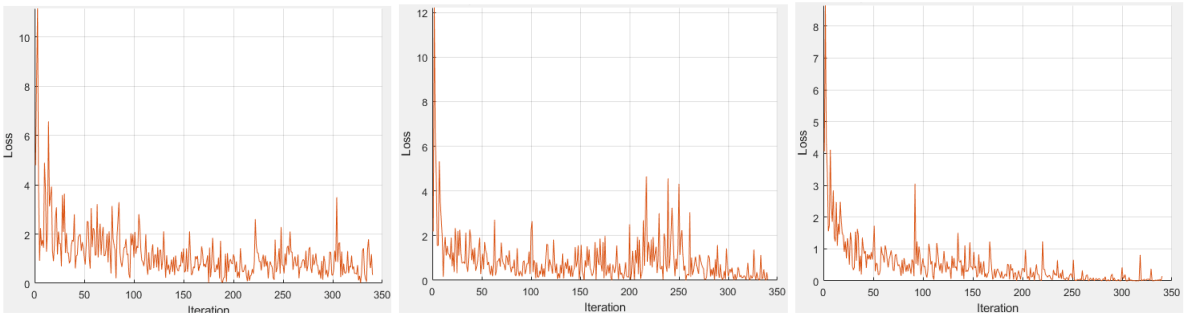

Figure 4.1: Graphs of the three versions of the AngularGrad^{tan} method

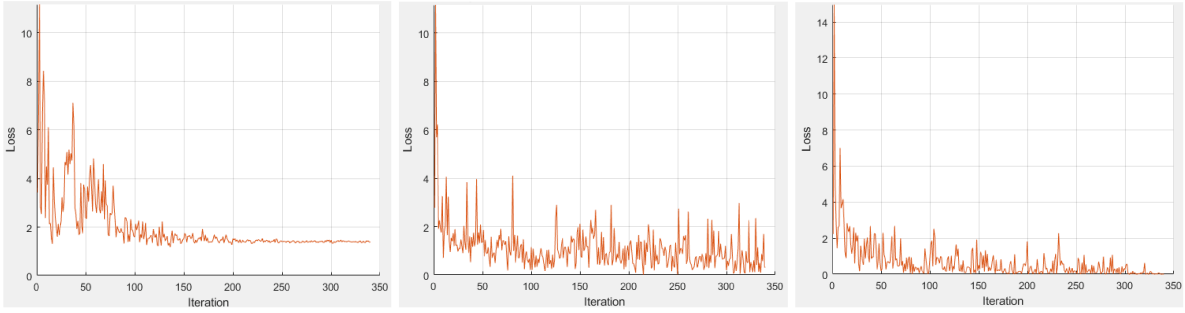

Figure 4.2: Graphs of the three versions of the AngularGrad^{cos} method

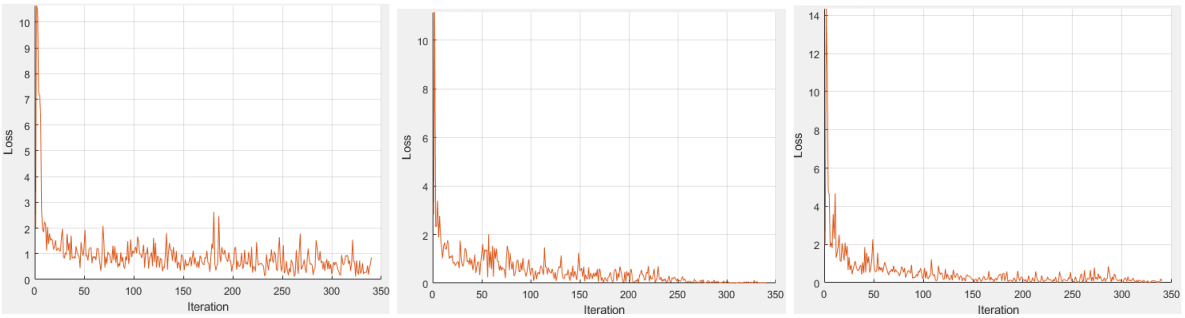

Figure 4.3: Graphs of the three versions of the AngularHypGrad method

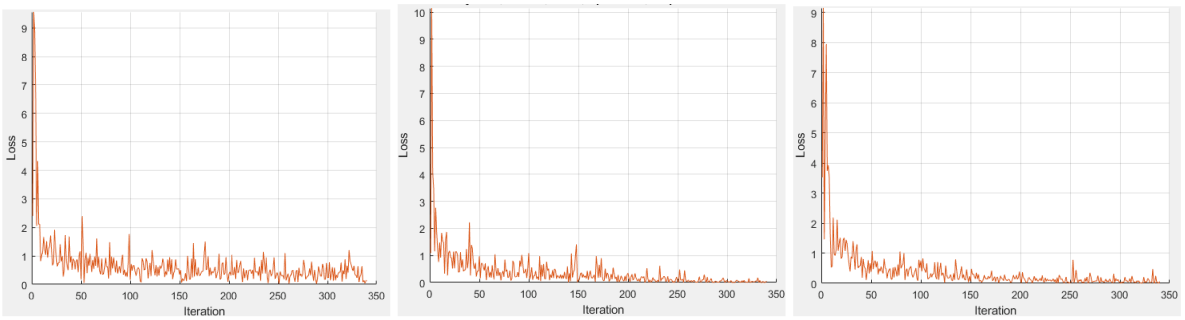

Figure 4.4: Graphs of the three versions of the HypAngularGrad method

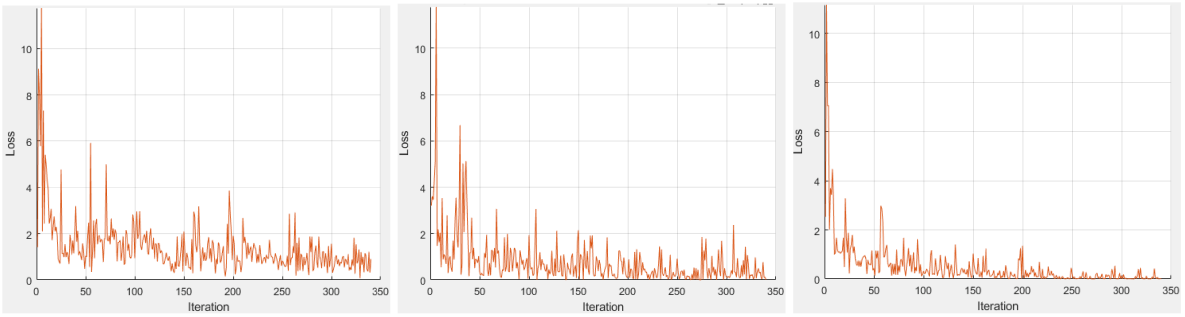

Figure 4.5: Graphs of the three versions of the AngularCLRWDecreasing method

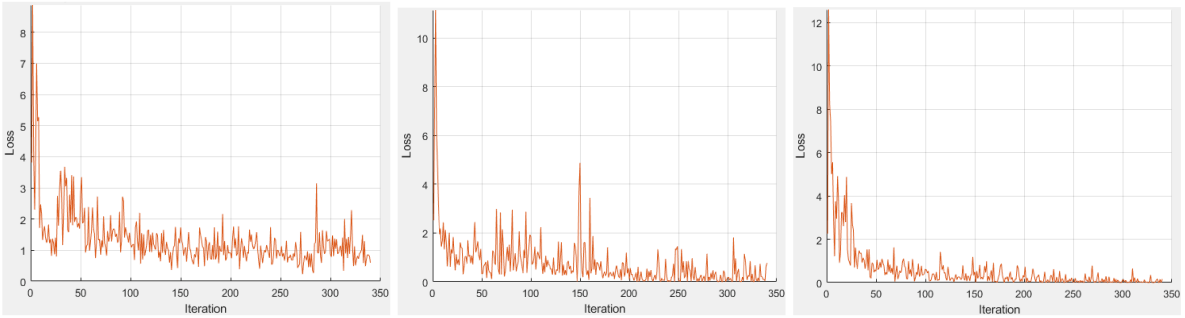

Figure 4.6: Graphs of the three versions of the LinearAngularCLRW method

4.2.1 Additional experimental results

For the remaining time, only a series of more extensive tests were carried out. These involved using a new network, ”ResNet50” [20], and the entire ”LAR” dataset [21]. The methods tested here are just a few of those mentioned above. The boost approach used is that of ”AdaInject.” The table 4.2 shows the percentage results obtained in these tests. The first column shows the name of the method tested, the second the average of seven trainings, and the third the result obtained from the seven-network sum rule. The first item in the table, ”Adam(base),” was included for comparison and represents the results obtained with this work environment using the EMAs of the classical ”Adam” optimizer, not those of ”AdaInject” as for the remaining entries.

Optimizer	Average	Sum rule
Adam(base)	0.9215	0.9629
Adam	0.9372	0.9583
Hyp	0.9381	0.9682
AngularGrad(cos)	0.9382	0.9583
AngularHypGrad	0.9396	0.9644
HypAngularGrad	0.9453	0.9652
AngularCLRWDecreasing	0.9433	0.9629
LinearAngularCLRW	0.9511	0.9652

Table 4.2: Experimental results with ResNet50 and LAR dataset

In this working environment and with the tests performed, it can be seen that the results are about in the same range. The boost brought by ”AdaInject” is not as visible as it was before. The results of the new variants are slightly better than the boosted approaches of ”Adam” and ”Hyp”.

In any case experimenting on one dataset is by no means sufficient to reach definite conclusions. That is why other tests are already planned and in progress.

Chapter 5

Conclusions

The objective of this thesis was to implement and study the two new approaches ”AngularGrad” and ”AdaInject,” and then move on to study possible new variants of the ”Adam” algorithm. The analysis of the first two approaches was quite in line with the results proposed in the papers in which they were presented, except for the case of ”AngularGrad^{cos}” which turned out to perform less well than expected. The new variants, achieve on average the same performance as the approaches they were inspired by. The ”AdaDoubleInject” method, on the other hand, seems to perform better than ”AdaInject,” or at worst, like it.

That said, the results obtained seem promising, although a study on only one dataset is not enough to reach definitive conclusions, and that is why new tests, on different networks and datasets, are already planned and underway.

Bibliography

[1] H.-L. E. Group, "A definition of ai: Main capabilities and scientific disciplines," *European commission*, 2019.

[2] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning," *Electronic Markets*, vol. 31, pp. 685–695, apr 2021.

[3] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, 1943.

[4] F. Rosenblatt, "Perceptron–a perceiving and recognizing automaton," *Report 85-460-1, Cornell Aeronautical Laboratory*, 1957.

[5] A. Simon, M. Deo, V. Selvam, and R. Babu, "An overview of machine learning and its applications," *International Journal of Electrical Sciences & Engineering*, vol. Volume, pp. 22–24, 01 2016.

[6] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[7] Y. Lecun, "A theoretical framework for back-propagation," in *Proceedings of the 1988 Connectionist Models Summer School, CMU, Pittsburg, PA* (D. Touretzky, G. Hinton, and T. Sejnowski, eds.), pp. 21–28, Morgan Kaufmann, 1988.

[8] Y. Bohra, "The challenge of vanishing/exploding gradients in deep neural networks," *Data Science Blogathon*, 2021.

[9] G. E. Hinton, "Rectified linear units improve restricted boltzmann machines vinod nair," 2010.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature 323, 533–536*, 1986.

[11] S. Ruder, "An overview of gradient descent optimization algorithms," 2016.

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[13] M. D. Zeiler, "Adadelta: An adaptive learning rate method," 2012.

[14] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145–151, 1999.

[15] S. R. Dubey, S. Chakraborty, S. K. Roy, S. Mukherjee, S. K. Singh, and B. B. Chaudhuri, "diffgrad: An optimization method for convolutional neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 11, pp. 4500–4511, 2020.

[16] S. K. Roy, M. E. Paoletti, J. M. Haut, S. R. Dubey, P. Kar, A. Plaza, and B. B. Chaudhuri, "Angulargrad: A new optimization technique for angular convergence of convolutional neural networks," arXiv, 2021.

[17] S. R. Dubey, S. H. S. Basha, S. K. Singh, and B. B. Chaudhuri, "Curvature injected adaptive momentum optimizer for convolutional neural networks," 2021.

[18] The Mathworks, Inc., Natick, Massachusetts, *MATLAB version 9.12.0.1884302 (R2022a)*, 2022.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[21] G. M. e. a. Moccia S, De Momi E, "Confident texture-based laryngeal tissue classification for early stage diagnosis support," *J Med Imaging (Bellingham). 4(3):034502.*, 2017.