Heavy Ball Gradient and CFM Deflected Subgradient for Neural Networks Training

Gabriele Marino*

Pisa, Sep 2, 2022

Abstract

This project explores the use of Heavy Ball Gradient and Camerini-Fratta-Maffioli subgradient for training neural networks using L1 (Lasso) regularization. First, the learning problem of neural networks is formalized in a general optimization framework. Next, the two algorithms are analyzed extensively from a theoretical point of view, especially in terms of their convergence complexity. Finally, many experiments are run to appreciate the differences between them, discussing in depth their strengths and weaknesses in light of the obtained results.

1 Notation

1.1 Multi Layer Perceptrons (MLP)

We consider a Multi Layer Perceptron (MLP) with L layers. We will denote by $\mathbf{W}_l$ and by $\mathbf{b}_l$ respectively the weights matrix and the bias vector of layer l, so that the parameters matrix of the same layer, i.e. the concatenation of its bias vector with its weights matrix, will be $\theta_l = [\mathbf{b}_l, \mathbf{W}_l]$. The activation function of layer l will be instead denoted by f_l, so that the output of layer l will then be: $\mathbf{o}_l = f_l(\mathbf{h}_l)$, where $\mathbf{h}_l = \mathbf{W}_l\mathbf{o}_{l-1} + \mathbf{b}_l$ is the net input of the layer. Finally, we will denote the parameters vector of the MLP, i.e. the concatenation of the flattened parameters matrices of its layers, as $\theta = \left[flat(\theta_1)^T, \ldots, flat(\theta_L)^T\right]^T$.

1.2 Activation function

We will consider two different activation functions. The first one is the Rectified Linear Unit (ReLU):

$$R(x) = \max(0, x),$$

and the second one is the Sigmoid, or logistic function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

Both of them are continuous, but while the Sigmoid is differentiable, ReLU is not in $x = 0$; still, as usual when dealing with this activation function, we will assume $R'(0) = 0$ and ignore the issue.

*Master's Degree student in Artificial Intelligence, g.marino37@studenti.unipi.it

1.3 Objective function

Consider a dataset $\mathcal{D} = (\mathbf{X}, \mathbf{Y})$ consisting of S samples, with input data $\mathbf{x}_i \in \mathbf{X} \subseteq \mathbb{R}^m$ and targets $\mathbf{y}_i \in \mathbf{Y} \subseteq \mathbb{R}^n$, for $i = 1, 2, \ldots, S$. A MLP with m input units and n output units defines a prediction function $\mathcal{M}_\theta : \mathbb{R}^m \to \mathbb{R}^n$, parametric in θ. The training of a MLP aims to minimize an objective loss function

$$J(\theta) = \mathcal{L}\left(\mathbf{Y}, \hat{\mathbf{Y}}\right) + \lambda\Omega(\theta),$$

wher $\mathcal{L}\left(\mathbf{Y}, \hat{\mathbf{Y}}\right)$ is the error function between the true targets $\mathbf{Y}$ and the predicted targets $\hat{\mathbf{Y}} = \mathcal{M}_\theta(\mathbf{X})$, and $\Omega(\theta)$ is the regularization term, which is weighted by the regularization hyperparameter λ.

As error function we will consider the Mean Squared Error (MSE):

$$\mathcal{L}\left(\mathbf{Y}, \hat{\mathbf{Y}}\right) = \frac{1}{2S}\left\|\mathbf{Y} - \hat{\mathbf{Y}}\right\|_2^2 = \frac{1}{2S}\sum_{i=1}^{S}\left(\mathbf{y}_i - \hat{\mathbf{y}}_i\right)^2,$$

where $\hat{\mathbf{y}}_i = \mathcal{M}_\theta(\mathbf{x}_i)$.

As regularization function we will consider the L1-regularization:

$$\Omega(\theta) = \sum_{l=1}^{L}\|\mathbf{W}_l\|_1$$

Note that a loss function such defined is neither convex nor differentiable: non convexity is caused by the non linearity of the activation functions of the layers, non differentiability is caused by the 1-norm in the L1-regularization term.

2 Optimization algorithms

2.1 Forward and backward propagation

Forward and backward propagation are used to compute $\nabla_\theta J(\theta)$ for a given θ. This is needed by the optimization algorithms used to adjust the parameters of the network to minimize J.

Note that J may be not differentiable: if this is the case, for the sake of simplicity, the very same notation $\nabla_\theta J(\theta)$ will denote any of the subgradients of $J(\theta)$ with respect to θ, and we will continue to refer to it as to the gradient of the objective function, without loss of generality.

In algorithm 1 we present the pseudocode of forward and backward propagation for a single sample; for a detailed description of these algorithms refer to [5].

2.2 Heavy Ball Gradient (HBG)

The Heavy Ball Gradient (HBG) is a deflected gradient optimization method widely used in neural networks training; its pseudocode is presented in algorithm 2.

Algorithm 1 Forward and Backward Propagation
Requires: the input data $\mathbf{x}$, the target output $\mathbf{y}$.

```
$\mathbf{o}_0 = \mathbf{x}$
for $l = 1, \dots, L$ do                                    ▷ Forward propagates the input
    $\mathbf{h}_l = \mathbf{W}_l \mathbf{o}_{l-1} + \mathbf{b}_l$
    $\mathbf{o}_l = f_l(\mathbf{h}_l)$
end for
$\hat{\mathbf{y}} = \mathbf{o}_L$
$J = \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) + \lambda\Omega(\theta)$                        ▷ Computes the loss

$\mathbf{g} \leftarrow \nabla_{\hat{\mathbf{y}}} J = \nabla_{\hat{\mathbf{y}}} \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$           ▷ Computes the gradient of the loss
for $l = L, \dots, 1$ do                                    ▷ Backward propagates the gradient
    $\mathbf{g} \leftarrow \nabla_{\mathbf{h}_l} J = \mathbf{g} \odot f'(\mathbf{h}_l)$
    $\nabla_{\mathbf{b}_l} J = \mathbf{g} + \lambda \nabla_{\mathbf{b}_l} \Omega(\theta)$
    $\nabla_{\mathbf{W}_l} J = \mathbf{g}\mathbf{o}_{l-1}^T + \lambda \nabla_{\mathbf{W}_l} \Omega(\theta)$
    $\mathbf{g} \leftarrow \nabla_{\mathbf{o}_{l-1}} J = \mathbf{W}_l^T \mathbf{g}$
end for
```

The algorithm iteratively updates the parameters of the network until a stopping condition is satisfied. The update direction at epoch i is a linear combination of the negative gradient direction at that epoch with the update direction at epoch $i-1$, where the gradient is computed by forward and backward propagation. The coefficients of the linear combination are the hyperparameters α and β, which are called respectively learning rate and momentum.

As stopping condition we chose the reaching of a max number of epochs, given by `MAX_EPOCHS`. This is very typical in Machine Learning, but many other stopping conditions are possible: for instance, the algorithm may be designed to stop as soon as the gradient norm drops below a certain threshold.

Algorithm 2 Heavy Ball Gradient (HBG)
Requires: the function $J(\theta)$ to optimize, the starting point θ_0, the learning rate hyperparameter α, the momentum hyperparameter β, the max number of epochs `MAX_EPOCHS`.

```
$\theta \leftarrow \theta_0$
for $i = 1, \dots,$ MAX_EPOCHS do
    if $i = 1$ then
        $\Delta\theta \leftarrow -\alpha \nabla_\theta J(\theta)$
    else
        $\Delta\theta \leftarrow -\alpha \nabla_\theta J(\theta) + \beta \mathbf{d}$
    end if
    $\theta \leftarrow \theta + \Delta\theta$
    $\mathbf{d} \leftarrow \Delta\theta$
end for
```

2.3 Camerini-Fratta-Maffioli subgradient (CFM)

Camerini-Fratta-Maffioli (CFM) is a deflected subgradient optimization method; its pseudocode is presented in algorithm 3. For a wide introduction to subgradient methods refer to [1]; for an in-depth analysis of deflected subgradient methods refer to [4]; for a detailed description of this specific algorithm refer to [3].

Algorithm 3 Camerini-Fratta-Maffioli (CFM) with exact Polyak step size
Requires: the function $J(\theta)$ to optimize, the starting point θ_0, the momentum hyperparameter γ, the optimal value J^* of $J(\theta)$ (or an estimate of it), the max number of epochs `MAX_EPOCHS`.

```
$\theta \leftarrow \theta_0$
for $i = 1, \dots,$ MAX_EPOCHS do
    $\mathbf{g} \leftarrow \nabla_\theta J(\theta)$
    if $i = 1$ then
        $\mathbf{d} \leftarrow \mathbf{g}$
    else
        $\beta \leftarrow \max\left(0, -\gamma \frac{\mathbf{d}^T \mathbf{g}}{\|\mathbf{d}\|_2^2}\right)$
        $\mathbf{d} \leftarrow \mathbf{g} + \beta \mathbf{d}$
    end if
    $\alpha \leftarrow \frac{J(\theta) - J^*}{\|\mathbf{d}\|_2^2}$
    $\theta \leftarrow \theta - \alpha \mathbf{d}$
end for
```

The algorithm iteratively updates the parameters of the network until a stopping condition is satisfied, exactly like HBG. The update direction at each epoch i depends not only on the negative gradient direction at that epoch, but also on the update direction at epoch $i-1$. The gradient is still computed by forward and backward propagation, and the momentum hyperparameter is $\gamma \in [0, 2]$.

The step size α along the update direction is called Polyak step size[1], and is a function of J^*, the optimal value of the objective function. In general, this is hardly available at optimization time, so an estimate of it may be needed.

In algorithm 4, a common technique called target following is used to provide a moving estimate of J^*. Here, J^* is estimated by the target level $J_{\text{best}}^{(i)} - \delta^{(i)}$, where the reference value $J_{\text{best}}^{(i)}$ is the best objective value up to epoch i, and the threshold $\delta^{(i)}$ can be interpreted as an estimate of how suboptimal the current value is.

3 Convergence results

3.1 HBG

HBG is not a descent algorithm, as the objective function does not necessarily decrease at each step. Still, many convergence results can be proven.

[8] shows that for L-smooth and τ-convex functions, the best possible α and β can be computed analytically. This gives the following optimal values:

$$\alpha^* = \left(\frac{2}{\sqrt{L} + \sqrt{\tau}}\right)^2, \qquad \beta^* = \left(\frac{\sqrt{L} - \sqrt{\tau}}{\sqrt{L} + \sqrt{\tau}}\right)^2.$$

[1]Many other step sizes are worthy of consideration when considering deflected subgradient methods. For a detailed description of the alternatives refer to [4].

Algorithm 4 Camerini-Fratta-Maffioli (CFM) with target level
Requires: the function $J(\theta)$ to optimize, the starting point θ_0, the momentum hyperparameter γ, the target level $\{\delta^{(i)}\}$, the max number of epochs `MAX_EPOCHS`.

```
$\theta \leftarrow \theta_0$
$J_{\text{best}} \leftarrow J(\theta)$
for $i = 1, \dots,$ MAX_EPOCHS do
    $\mathbf{g} \leftarrow \nabla_\theta J(\theta)$
    if $i = 1$ then
        $\mathbf{d} \leftarrow \mathbf{g}$
    else
        $\beta \leftarrow \max\left(0, -\gamma \frac{\mathbf{d}^T\mathbf{g}}{\|\mathbf{d}\|_2^2}\right)$
        $\mathbf{d} \leftarrow \mathbf{g} + \beta\mathbf{d}$
    end if
    $\alpha \leftarrow \frac{J(\theta) - J_{\text{best}} + \delta^{(i)}}{\|\mathbf{d}\|_2^2}$
    $\theta \leftarrow \theta - \alpha\mathbf{d}$
    $J_{\text{best}} \leftarrow \min(J_{\text{best}}, J(\theta))$
end for
```

For these values of the hyperparameters, the algorithm generates a sequence of points $\{\theta^{(i)}\}$ which converges linearly to the optimal θ^*:

$$\left\|\theta^{(i+1)} - \theta^*\right\| \leq r \left\|\theta^{(i)} - \theta^*\right\|, \qquad r = \frac{\sqrt{L} - \sqrt{\tau}}{\sqrt{L} + \sqrt{\tau}}.$$

Thus, the number of steps required to reach relative ϵ-accuracy, i.e. $\frac{\|\theta^{(i)}\|}{\|\theta^{(0)}\|} < \epsilon$, is $\mathcal{O}\left(\sqrt{\frac{L}{\tau}} \log \frac{1}{\epsilon}\right)$. Note that L and τ are usually unknown at optimization time, so grid search may be required.

For non-convex functions the convergence results are much weaker: [7] proves that HGB convergence is ensured for L-smooth functions as long as $\beta \in [0, 1)$ and $\alpha \in \left(0, 2\frac{1-\beta}{L}\right)$, as a special case of the proof for the more general iPiano optimization algorithm.

Unfortunately, our loss function is not L-smooth (indeed, it is not even differentiable). Theoretical convergence is then not ensured a priori.

3.2 CFM

In [1] many convergence results are given for subgradient algorithms in case of no deflection ($\gamma = 0$). In particular, suppose that J is convex and that at each epoch i the norm of its subrgadient is bounded by $G \in \mathbb{R}$:

$$\|\mathbf{g}^{(i)}\| = \|\nabla_\theta J(\theta^{(i)})\| \leq G.$$

Then, it can be proven that the the subgradient method with Polyak step size converges ($J_{\text{best}}^{(i)} \to J^*$), and the number of steps needed before we can guarantee relative ϵ-accuracy is proven to be optimal, and equal to $\mathcal{O}\left(\left(\frac{GR}{\epsilon}\right)^2\right)$, where $R \in \mathbb{R} : R \geq \|\theta^{(1)} - \theta^*\|$ is an upper bound on the distance between the starting point and the optimal solution.

Moreover, it can be proven that this convergence result also holds for the subgradient method with Polyak step size, standing the following assumptions on the threshold of the target level: $\delta^{(i)} > 0$, $\delta^{(i)} \to 0$ and $\sum_{i=1}^{\infty} \delta^{(i)} = \infty$.

[3] generalizes these results to CFM deflected subgradient, given the same assumptions. Specifically, it shows that CFM is theoretically at least as good as the non-deflected subgradient algorithm. Formally, it is proven that CFM deflected direction has a smaller angle towards the optimal value than the non deflected subgradient. It follows that the convergence proofs for the non deflected subgradient algorithm work for CFM as well, and indeed that CFM can be expected to perform better computationally. Further theoretical results can be proven: [2] proves the optimality of the parameters choice of CFM, while [6] proves stronger convergence results than those originally derived by the authors.

However, our loss function is not convex, so these results do not apply to our problem, and theoretical convergence is not ensured a priori.

4 Experiments

In this section we describe our experiments and comment the results we got. For brevity's sake, if not otherwise specified, we will refer to CFM with target level (algorithm 4]) simply as CFM.

4.1 Methodological note

The objective function defined in 1.3 is not theoretically ensured to converge, as we extensively explained in section 3. Still, this is not a big issue when training a MLP model. In fact, the global convergence of the training loss is not even desired at all, as it may imply the overfitting of the training data, worsening the generalization capabilities of the model. Thus, it is very common in machine learning to apply an optimization algorithm even without theoretical guarantees, and settle for non optimal solutions.

4.2 Dataset and general settings

The dataset we used for our experiments is the ML-CUP21 dataset, from the *Machine Learning CUP 2021-2022* student competition of the Machine Learning course at University of Pisa. This dataset consists of 1477 samples, each of which is defined by 10 numerical input features and 2 numerical targets.

Unless otherwise specified, whenever we trained a model, we set a maximum number of epochs of 10000. As a further stopping criterion, we allowed for early stopping, with a tolerance of 10^{-4} and a patience of 150 epochs, to prevent trainings to zigzag too much between suboptimal solutions.

We also made large use of grid search, and tested the hyperparameters combinations with different random initializations of the weights of the models, to get some understanding of the variances of the optimization algorithms presented. In both these situations, the random seeds for the weights inizialization were the same. This was done to guarantee the same starting points when training a model with different algorithms, thus avoiding introducing biases based on more or less lucky initializations.

When training a model with CFM, we limited ourselves to consider the thresholds $\delta^{(i)} = \frac{a}{a+i}$, with $a \in \mathbb{R}$. Note that this set of sequences verifies the hypothesis outlined in 3.2. Of course, this does not imply theoretical convergence by itself; still, we wanted our hyperparameter choice to be as close as possible to the theoretical requirements for convergence.

Lastly, table 1 shows the average time per epoch for each of the experiments done. As we can see, there is no relevant difference between the time required by HBG and CFM to complete an epoch of training. For this reason, for the sake of simplicity, we limited ourselves to the number of epochs as the only metric of comparison between the speed of the two algorithms.

Experiment 0.2		HBG: 2.719 ± 0.068
		CFM with target level: 2.845 ± 0.022
		CFM with exact Polyak step size: 2.796 ± 0.073
Experiment 1	Sigmoid	HBG: 8.221 ± 0.123
		CFM: 8.975 ± 0.187
	ReLU	HBG: 3.476 ± 0.153
		CFM: 3.408 ± 0.094
Experiment 2	Sigmoid	HBG: 10.864 ± 0.554
		CFM: 10.553 ± 0.106
	ReLU	HBG: 4.853 ± 0.066
		CFM with target level: 5.131 ± 0.070
		CFM with exact Polyak step size: 4.991 ± 0.084
Experiment 3		HBG: 7.781 ± 0.283
		CFM: 7.990 ± 0.079

Table 1: Average time per epoch (ms)

4.3 Experiment 0.1 - Threshold influence in CFM

In the first preliminary experiment we wanted to test the influence of the target level threshold on CFM speed of convergence and accuracy. To do so, we set up a network with a 30 neurons hidden layer, ReLU activation function and L1 regularization with $\lambda = 0.1$, and we trained 5 different models with CFM, with $a \in \{50, 75, 100, 150, 200\}$ and $\gamma = 0.5$. The results are shown in figure 1.

As we can see, larger values of a lead to a faster convergence, but to a worse result, thus defining a very sharp trade-off between the speed of convergence of the algorithm and the suboptimality of the solution found (cfr. [1]). In light of this result, when performing grid search for CFM in the following experiments, we kept fixed the threshold value to guarantee a reasonable trade-off between speed and accuracy of convergence, and we fine tuned only the momentum hyperparameter.

4.4 Experiment 0.2 - Convex loss function

In the second preliminary experiment we wanted to test the presented algorithms on a convex function. Thus, to enforce the convexity of the loss, we built a MLP with identity as activation function. Of course, such a network serves no practical purpose, but lets us show the results presented in 3.

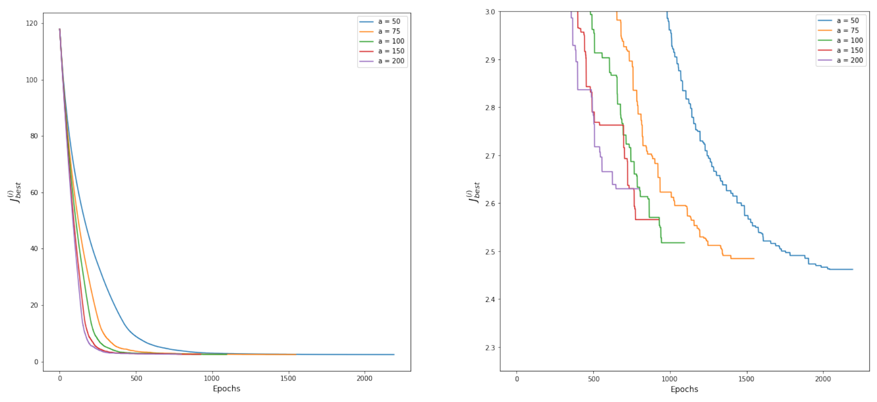


Figure 1: Threshold influence in CFM

Specifically, we built a MLP with a single hidden layer composed by 60 neurons. The Lasso regularization coefficient was set to $\lambda = 0.1$. We fine tuned the hyperparameters via a grid search, as shown in table 2.

HBG	$\alpha \in 10^{-3} \times \{0.5, 0.75, 1, 2.5, 5, 7.75, 10, 25, 50\}$ $\beta \in \{0.1, 0.25, 0.5, 0.75, 0.9\}$
CFM	$\gamma \in \{0.1, 0.15, 0.2, 0.3, 0.4, 0.5, 0.6, 0.75, 1, 1.25, 1.5\}$ $a = 150$

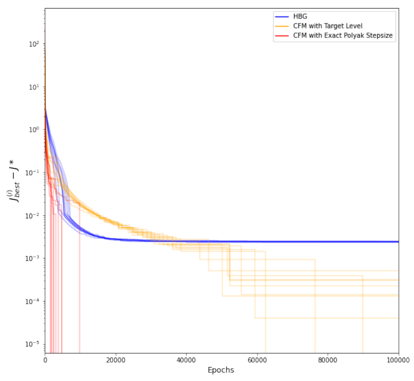
Table 2: Experiment 0.2 - Grid search set up

The results are shown in table 3. We note that HBG seemed to prefer low step sizes and a very high momentum, while CFM would rather have a low deflection. This result also suggests that HBG outperformed CFM, but this is not necessarily true: for a fair comparison of the two algorithms we have to increase the maximum number of epochs and inhibit early stopping.

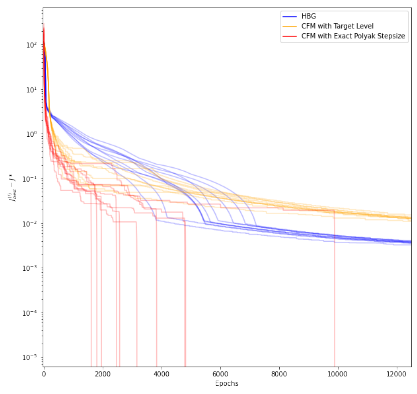
HBG			CFM	
α	β	loss	γ	loss
0.75e-3	**0.9**	**2.188**	**0.2**	**2.238**
0.5e-3	0.9	2.188	0.1	2.251
1e-3	0.9	2.189	0.15	2.270

Table 3: Experiment 0.2 - Grid search results

Thus, we used the best parameters combinations found by grid search to train the model for 100.000 epochs, without early stopping, using both CFM and HBG. We repeated the training for 10 times, each time with a different random weights initialization. At last, the loss obtained using HBG was 2.182, while that achieved by CFM was 2.180; in both cases the variance was negligible. As a next step we used this last value of the loss to approximate J^*, and trained the model using CFM with exact Polyak step size, for the same number of random runs, using the same value of $\gamma = 0.2$ we got from grid search. The convergence curves are shown in figure 2.



First 100.000 epochs



First 12.500 epochs

Figure 2: HBG, CFM with target level and CFM with exact Polyak step size convergence for a convex loss function

As we can see, CFM with exact Polyak step size quickly outperforms the other two optimization algorithms, converging fast to our approximation of J^* for every random run. CFM with target level, instead, takes much longer to converge; still, the convergence is steady, and we can appreciate the theoretical $\mathcal{O}(\epsilon^{-2})$ complexity. The same does not hold for HBG: even with a convex function, theoretical convergence is not guaranteed, as L-smoothness is further needed. As a matter of fact, HBG gets stuck in a suboptimal solution and gets slowly outperformed by CFM with target level. Still, it is noteworthy that HBG prevails over CFM from epoch ~ 5.000 to epoch ~ 35.000.

4.5 Experiment 1 - MLP with one hidden layer

In this experiment we considered two MLPs with one hidden layer consisting of 75 neurons, the first one with Sigmoid activation function and the second one with ReLU activation function. The Lasso regularization coeffiecnt was set to $\lambda = 0.01$ in both cases.

For both models we performed grid search as shown in table 4. As we observed some variance in training a model with CFM, we repeated each run of grid search for CFM for 5 times, each time with a different random weights initialization, and took their average as the result of the run.

HBG	$\alpha \in 10^{-3} \times \{0.5, 0.75, 1, 2.5, 5, 7.75, 10, 25, 50\}$ $\beta \in \{0.1, 0.25, 0.5, 0.75, 0.9\}$
CFM	$\gamma \in \{0.5, 0.75, 1, 1.25, 1.5, 1.75\}$ $a = 100$

Table 4: Experiment 1 - Grid search set up

The results are shown in table 5 (CFM losses are the average ones, and we omit the variance as it always turned out to be negligible). We note that the best parameters combinations are the same both for Sigmoid and ReLU; still, ReLU allows a much better fitting of the training data, and achieves a much smaller loss. Once again, HBG seems to prefer high momentum, while the preferred step sizes are the medium ones. As for CFM, the best values of γ oscillates much and the algorithm does not seem to have a particular tendency to favor low values of the hyperparameter over high values or vice versa.

Sigmoid					ReLU				
HBG			CFM		HBG			CFM	
α	β	loss	γ	loss	α	β	loss	γ	loss
0.025	**0.9**	**1.188**	**0.75**	**1.200**	**0.025**	**0.9**	**0.864**	**0.75**	**0.889**
0.05	0.9	1.190	1.25	1.233	0.05	0.5	0.866	1	0.894
0.05	0.75	1.236	1.75	1.245	0.05	0.75	0.867	0.5	0.898

Table 5: Experiment 1 - Grid search results

The best parameters combinations were then used to train the two models. The training was repeated for 25 times in both cases, each time with a different random weights initialization. The results are shown in table 6; the convergence curves are shown in figure 3.

Sigmoid	**HBG: 1.210 $\pm$ 0.020** CFM: 1.407 $\pm$ 0.661
ReLU	**HBG: 0.878 $\pm$ 0.016** CFM: 0.895 $\pm$ 0.020

Table 6: Experiment 1 - Resulting losses

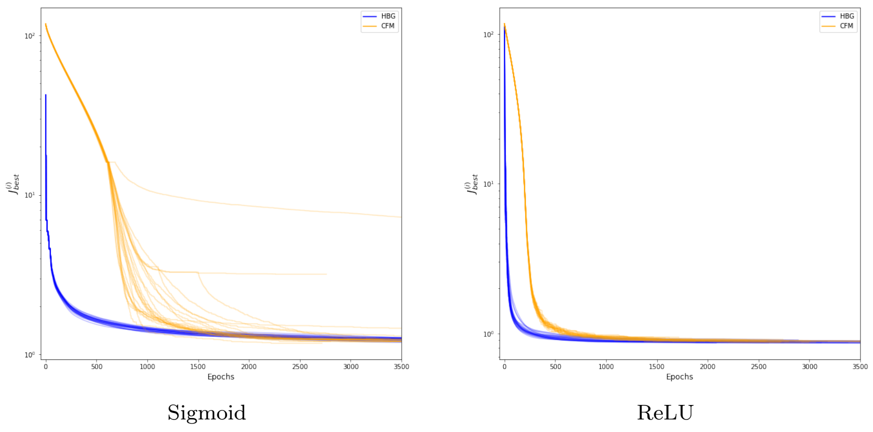


Figure 3: HBG and CFM convergence for a one hidden layer MLP

These results prove HBG superiority in both cases, in terms of accuracy and variance of the solution found. This is particularly evident in the case of Sigmoid activation function: CFM has a very high variance, and seems to easily get stuck into suboptimal solutions, while HBG rapidly and steadily converges. In the case of ReLU activation function, HBG superiority is much less evident, but still persistent.

4.6 Experiment 2 - MLP with two hidden layers

In this second experiment we increased the complexity of our models and built two MLPs, each one with two hidden layers, the first consisting of 60 neurons and the second of 30. We kept using Sigmoid activation function for the first MLP and ReLU for the second. Lasso regularization hyperparameter was set to $\lambda = 0.01$.

Sigmoid	HBG	$\alpha \in 10^{-3} \times \{0.5, 0.75, 1, 2.5, 5, 7.75, 10, 25, 50\}$ $\beta \in \{0.1, 0.25, 0.5, 0.75, 0.9\}$
	CFM	$\gamma \in \{0.5, 0.75, 1, 1.25, 1.5, 1.75, 1.9\}$ $a = 150$
ReLU	HBG	$\alpha \in 10^{-3} \times \{0.5, 0.75, 1, 2.5, 5, 7.75, 10, 25, 50\}$ $\beta \in \{0.1, 0.25, 0.5, 0.75, 0.9\}$
	CFM	$\gamma \in \{0.1, 0.15, 0.2, 0.3, 0.4, 0.5, 0.6, 0.75, 1, 1.25, 1.5\}$ $a = 100$

Table 7: Experiment 2 - Grid search set up

We fine tuned the hyperparameter by grid search, as shown in table 7. When performing grid search for CFM, we repeated each run for 5 times, and took their average as the result of the run.

The results are shown in table 8. CFM losses in the case of ReLU are the average ones, and we omit the variance as it always turned out to be negligible. ReLU seems to allow a much better fitting of the training data, achieving a much smaller loss. HBG seems to prefer high momentum terms, with low step sizes for ReLU and medium step sizes for Sigmoid, while CFM seems not to have a particular tendency to favor low momentum over high momentum or vice versa. HBG seems superior in both cases, in particular with Sigmoid, where CFM loss is just too high and hugely variant.

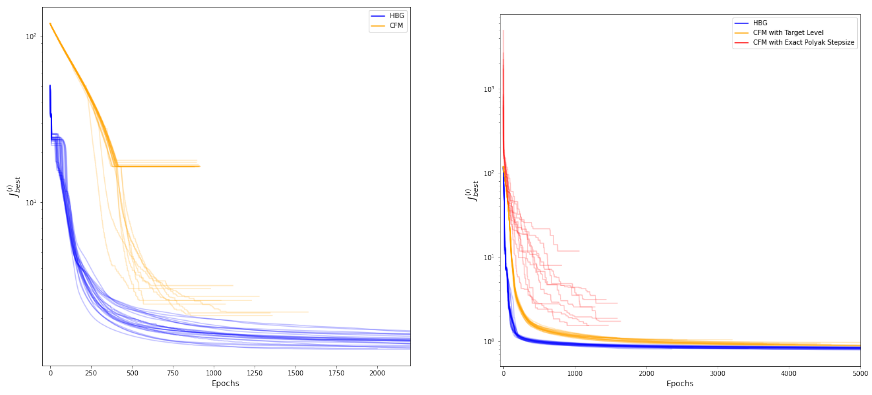
Sigmoid					ReLU				
HBG			CFM		HBG			CFM	
α	β	loss	γ	loss	α	β	loss	γ	loss
0.05	**0.9**	**1.401**	**0.75**	$\mathbf{5.267 \pm 5.540}$	**0.00775**	**0.9**	**0.816**	**0.5**	**0.915**
0.05	0.75	1.407	1.9	5.817 ± 3.622	0.01	0.9	0.818	0.6	1.005
0.025	0.9	1.444	1.5	7.084 ± 4.911	0.005	0.9	0.830	0.15	1.075

Table 8: Experiment 2 - Grid search results

The best parameters combination were used to train the two models. We repeated the training for 25 times, each time with a different random weights initialization. The results are shown in table 9; the convergence curves are shown in figure 4.

As we can see, HBG outperforms CFM in both cases, both in terms of optimality of the solution and in terms of variance, with a steady and faster convergence. CFM convergence compares decently to that of HBG in the case of ReLU, while in the case of Sigmoid is very chaotic, and gets stuck into multiple suboptimal solutions.

The results we got for CFM in the case of Sigmoid were so bad that we repeated the whole process with different values of a. We omit the intermediate



Sigmoid ReLU

Figure 4: HBG and CFM convergence for a two hidden layers MLP

Sigmoid	**HBG: $\mathbf{1.353 \pm 0.045}$** CFM: 11.502 ± 6.734
ReLU	**HBG: $\mathbf{0.816 \pm 0.017}$** CFM: 0.942 ± 0.068

Table 9: Experiment 2 - Resulting Losses

results and just show the convergence curves in figure 5. We note that lower values of a allow for some of the runs to achieve better losses, while larger values speed up the convergence, as we could have expected (cfr. 4.3). Still, the overall quality of the convergence does not seem to improve, and the variance remains very high and independent of the value of a.

As a further step, we wanted to test CFM with exact Polyak step size in the case of ReLU. Thus, we used the value 0.816 from table 9 to approximate J^*, and fine tuned the momentum hyperparameter by grid search, choosing the best $\gamma \in \{0.1, 0.15, 0.2, 0.3, 0.4, 0.5, 0.6, 0.75, 1, 1.25, 1.5\}$. We kept the usage of 5 random initializations for each grid search run, but we increased the early stopping patience to 500 epochs. The results are shown in table 10. The algorithm prefers low momentum, but does not seem capable of getting close to the results achieved by CFM with target level and HBG.

γ	loss
0.2	$\mathbf{3.486 \pm 2.572}$
0.15	4.284 ± 2.987
0.1	4.523 ± 2.207

Table 10: Experiment 2 - Grid search results for CFM with exact Polyak step size.

The best value of γ was then used to train the model for 10 times, each time with a different random weights initialization, achieving a final loss of 3.762 ± 3.207 (the convergence curves are also shown in figure 4), a result not even remotely comparable to those obtained by CFM with target level and HBG.

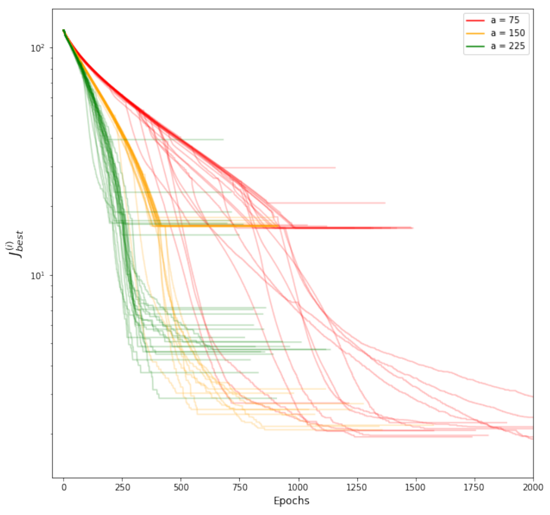


Figure 5: CFM convergence for a two hidden layers MLP for $a = 75, 150, 225$

4.7 Experiment 3 - MLP with three hidden layers

In this last experiment we increased even more the complexity of our model and built a MLP with three hidden layers, with respectively 60, 45 and 30 neurons, but we limited ourselves to consider only the ReLU as activation function. We also increased the value of Lasso regularization hyperparameter to $\lambda = 0.1$. First, we fine tuned CFM and HBG hyperparameters by grid search, as shown in table 11. As usual, we repeated each run of grid search for CFM 5 times, and took their average as the result of the run, to better deal with the high variance of the algorithm.

HBG	$\alpha \in 10^{-3} \times \{0.5, 0.75, 1, 2.5, 5, 7.75, 10, 25, 50\}$ $\beta \in \{0.1, 0.25, 0.5, 0.75, 0.9\}$
CFM	$\gamma \in \{0.3, 0.4, 0.5, 0.6, 0.75, 1, 1.25\}$ $a = 100$

Table 11: Experiment 3 - Grid search set up

The results are shown in table 12. HBG seems to prefer medium-low step sizes and very high momentum, while CFM tends to favor medium-low momentum terms. CFM also seems to have a quite high variance, but is able to achieve far better results.

HBG			CFM	
α	β	loss	γ	loss
0.001	**0.9**	**15.867**	**0.6**	$\mathbf{4.815 \pm 0.739}$
0.0025	0.9	15.882	0.4	4.959 ± 2.666
0.00075	0.9	15.887	0.3	6.139 ± 5.424

Table 12: Experiment 3 - Grid search results

We used the best combinations of parameters found by grid search to train the model. We repeated the training for 25 times, each time with random initialization. At the end, CFM was able to achieve a loss of 5.549 ± 2.414, while HBG could not improve over 15.881 ± 0.010. The convergence curves are shown in figure 6.

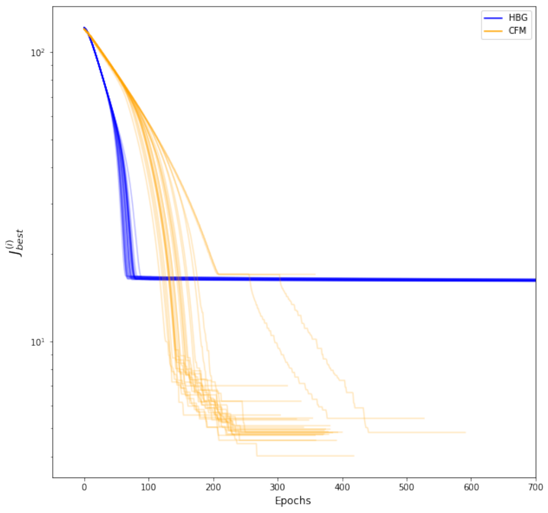


Figure 6: HBG and CFM convergence for a three hidden layers MLP

As we can see, even if with a much higher variance, CFM is able to achieve far better solutions, while HBG can't avoid getting stuck into a local minimum, and is easily outperformed by CFM.

5 Conclusions

In this chapter we explored the use of two different optimization algorithms for training MLPs using L1 regularization, the first one being Heavy Ball Gradient (HBG), and the second being Camerini-Fratta-Maffioli deflected subgradient (CFM).

After a brief but consistent summary of the theoretical properties of the two, we run many experiments to compare them.

In 4.3 we showed an interesting property of the target level in CFM, which also came up in 4.6: the faster the convergence of the target level, the faster the convergence of training, but the larger the final loss.

In 4.4 we experimentally confirmed the convergence properties of CFM for convex objective functions, forcing the convexity of our loss by using the identity as activation function of the MLP. The results we got also show CFM superiority with respect to HBG when dealing with convex functions. This is especially true when an approximation of the best value of the loss is known, otherwise CFM might took longer than HBG, but in the long run would still achieve better results.

In 4.5 and 4.6 we tested CFM and HBG on respectively one hidden layer and two hidden layers MLPs, using ReLU and Sigmoid activation functions. HBG always outperformed CFM, having a faster and steadier conergence, especially in the case of Sigmoid. With Sigmoid, in fact, CFM was always found to have a very high variance, and could not achieve training losses as low as those achieved by HBG. In 4.6 we also tried to use CFM with exact Polyak stepsize to train the MLP with ReLU activation function, but the results we got were not even remotely comparable to those achieved by HBG and CFM with target level, proving the need of convexity of the objective function for this algorithm to work effectively.

Eventually, in 4.7 we tested HBG and CFM on a three hidden layers MLP. It turned out that in this case it would be CFM to outperform HBG, which did not seem capable of overcoming a local minimum. CFM, instead, even if with a higher variance of speed and optimality of the solution found, would almost always achieve a far better result.

Summing up, our experiments show that HBG is superior to CFM by far in training MLPs of low and medium sizes, in terms of variance, speed of convergence and flexibility with respect to the activation function. Still, when training bigger MLPs with ReLU activation function, CFM might be worthy of consideration, and would plausibly achieve much better results than HBG, whose effectiveness does not scale as well.

References

[1] Boyd, S., *Subgradient Methods*, May 2014, available at: `https://web.stanford.edu/class/ee364b/lectures/subgrad_method_notes.pdf`.

[2] Brännlund, U, *A generalized subgradient method with relaxation step*, in *Mathematical Programming*, 71, 207-219, 1995, available at: `https://link.springer.com/content/pdf/10.1007/BF01585999.pdf`.

[3] Camerini, P. M., Fratta, L., Maffioli, F., *On improving relaxation methods by modifying gradient techniques*, in *Math. Programming Study*, August 1975.

[4] D'Antonio, G., Frangioni, A., *Convergence Analysis of Deflected Conditional Approximate Subgradient Methods*, in *SIAM Journal of Optimization*, 357-386, 2009, available at: `http://pages.di.unipi.it/frangio/papers/DeflProjSubG.pdf`

[5] Goodfellow, I., Bengio, Y., Courville, A., *Deep Learning*, The MIT Press, 2016, available at: `https://www.deeplearningbook.org/`.

[6] Kim, S., Ahn, H., *Convergence Properties of the Modified Subgradient Method of Camerini et al.*, in *Naval Research Logistics*, 37, 961-966, 1990.

[7] Ochs, P., *Local Convergence of the Heavy-Ball Method and iPiano for Non-convex Optimization*, in *Journal of Optimization Theory and Applications 177*, 153-180, 2018, available at: `https://link.springer.com/content/pdf/10.1007/s10957-018-1272-y.pdf`.

[8] Polyak, B. T., *Some methods of speeding up the convergence of iteration methods*, in *USSR Computational Mathematics and Mathematical Physics*, 4(5), 1-17, 1964.