TRAINING A NEURAL NETWORK WITH NONLINEAR CONJUGATE GRADIENT AND LIMITED-MEMORY BFGS METHODS

Alessandro Cudazzo
alessandro@cudazzo.com

Giulia Volpi
giuliavolpi25.93@gmail.com

Department of Computer Science, University of Pisa, Pisa, Italy

Computational Mathematics for Learning and Data Analysis Project
Academic Year: 2019/2020
October 12, 2020

ABSTRACT

Neural Networks are highly expressive models that have achieved the state of the art performance in many tasks as pattern recognition or natural language processing. Usually, stochastic momentum methods coupled with the classical Backpropagation algorithm for the gradient computation is used in training a neural network. In recent years several methods have been developed to accelerate the learning convergence of first-order methods such as *Classic Momentum*, also known as *Polyak's heavy ball method* [1], or the *Nesterov* momentum [2]. This work aims to go beyond the first-order methods and analyse some variants of the nonlinear conjugate gradient (NCG) and a specific case of limited-memory quasi-Newton class called L-BFGS as optimization methods. Them are combined with the use of a line search that respects the strong Wolfe conditions to accelerate the learning processes of a feedforward neural network.

1. Introduction

Over the years neural networks (NN) have been widely used in many areas, such as pattern recognition or image segmentation where they have overcome classic computer vision algorithms with the use of Convolution Neural Networks (CNN) [3] and very quickly became the state of the art in many other areas (i.e. speech recognition, natural language processing, audio recognition all well illustrated in [4]). As defined in [5], a NN, as any other machine learning (ML) models, is said to learn from experience to some class of tasks if the performance, measured by a metric, improves with that experience. Usually, the learning process that allows the improvement of the performance is related to an optimization algorithm where a quantity, called Error, is minimized/maximized: this is why the field of optimization is so central in machine learning. The optimization process can concern both the area of constrained optimization (i.e. in Support Vector Machine) or unconstrained (i.e in NN) and we will focus on the last one. With respect to neural networks and unconstrained optimization, the stochastic gradient descent (SGD) coupled with the back-propagation (BP) algorithms, for the gradient computation, remains a popular supervised learning algorithm to train a NN but is not the fastest in terms of convergence. The SGD is a first-order method and the convergence can be improved through the use of acceleration methods that are already well known in the classical literature of unconstrained optimization: *Classic Momentum* also known as *Polyak's heavy ball method* [1] or the *Nesterov*

momentum [2] developed respectively in 1960 e 1983. In [6] has been proved that the last two methods are very efficient in performing the training of a neural network without the use of second-order optimization methods. With this work, we want to go further beyond the classical first-order methods and investigate more complex methods that use high-order information during the training phase such as Nonlinear Conjugate Gradient and the L-BFGS from the class of Limited-Memory Quasi-Newton. Those methods lead to a more accurate choice of the search direction and of the step size, by using information from the second-order approximation. Even better, L-BFGS achieves super-linear convergence.

The remaining text is structured as follows. In Section 2, we have summarised all the assumptions for neural networks seen as an unconstrained optimization problem. In section 3 and 4 the conjugate gradient and the limited-memory BFGS methods will be illustrated. The experiments are shown in Section 5 and contains the results for three objective function derived from the MONKs dataset, while Section 6 is devoted to conclusions.

2. The neural network

The design of a Neural Networks model is biologically inspired by the brain and it's offer properties and capabilities [5, 4, 7]. This machine learning model has an elastic and adaptive ability, capable of learning from experience (the data provided) through a supervised process. It can be considered as a network of interconnected units (artificial neurons) that adapt to perform a given task with the ability also to discover new knowledge. It can solve either a classification (binary/multi-class) or regression tasks. In this work, we will analyze a neural network from a mathematical point of view and a NN can be seen as a flexible function $h(\mathbf{x})$ that is a nested non-linear function:

$$h(\mathbf{x}) = f_k \left(\sum_j^J w_{kj} f_j \left(\sum_i^I w_{ji} x_i \right) \right) \tag{1}$$

where $h(\mathbf{x})$ is relative to a NN with one output neuron on the output layer, one hidden layer with J hidden neurons and input $\mathbf{x} \in \mathbb{R}^I$. Where f_k is the activation function output neuron (linear or nonlinear function), w_{kj} is a specific wights of the output neuron connected to the j-th hidden output neuron f_j (a nonlinear activation function) that takes as input the linear combination of the products between $w_{ji} * x_i$.

So, the most important components of a NN are the nodes and architecture. Each node in the network will have associated a number of weights equal to the input of that node, a bias (which in our work will be included in the weights vector) and an activation function. The network can have different architectures with one or more hidden layers but the important thing is that each node in the hidden layers has a non-linear activation function (e.g. sigmoid). This allows the neural network to implement a linear basic expansion that is flexible and adaptive during the learning process thought the free parameters ($\mathbf{w}$) inside the non-linear activation function. Moreover, different architectures can lead to different hypothesis spaces that make the network more or less expressive for the task we are solving. This characterizes the neural networks by an important result which is the universal approximation theorem: an MLP network can approximate (arbitrarily well) every input-output mapping (provided enough units in the hidden layers). As concerns, the activation functions of neurons in the output layer can be either linear in case of a regression task or non-linear (e.g. sigmoid, softmax) in case of binary or multiclass classification tasks.

As in any other ML framework, we need a *learning algorithm* that allows adapting the free-parameters of the model based on data provided in order to obtain the best approximation of the target function. This is often realized in terms of minimization of an error function on the training data set:

$$\mathbf{w}^* = \min_{\mathbf{w} \in \mathbb{R}^n} E(\mathbf{w}) \tag{2}$$

given a set of l training examples $(\mathbf{x}_p, \mathbf{d}_p)$, where $\mathbf{x}_p \in \mathbb{R}^I$ and $\mathbf{d}_p \in \mathbb{R}^R$ we want to find the weights vector $\mathbf{w}^* = (w_1^*, w_2^*, ..., w_n^*) \in \mathbb{R}^n$ that minimize a Error fuction E that measure the performace of a NN model on the training data.

$$E(\mathbf{w}) = \sum_{p=1}^{l} J(h(\mathbf{x}_p), \mathbf{d}_p) \tag{3}$$

We can have different error function and the J function defines the quantity to measure for each pattern. It is well known that in order to find a good model it is not only essential to minimize the error on the training set but that the model has to be able to generalize correctly concerning data not yet seen (e.g. on a validation set). We will analyze the optimization algorithms and their properties from a mathematical point of view, so it will be considered only a training set on how to perform the minimization.

The optimization process is carried out using a classic iterative algorithm, chosen a starting point $\mathbf{w}_0$ up to a stop condition these steps are performed in an iterative manner:

1. the function $h(\mathbf{x})$ is evaluated on the training set and then the error function is computed.
2. the gradient is computed: $\nabla E(\mathbf{w})$.
3. the weights are updated thought an update rule.

For the discussion of the two main algorithms, the conjugate gradient and limited-memory BFGS, we need to define some information, on neural networks, that will be used later.

2.1 Weights' initialization

The weights' initialization defines the starting point of the iterative algorithm. It is a very important phase and can lead to a better starting point and faster convergence. Many deep learning applications make use of a random weights initialization where each weight is drawn from a zero-mean Gaussian distribution $N(0, v^2)$ with a fixed standard deviation v (e.g 0.01 in [8]) or from an uniform distribution $U(-a, a)$ in the interval $(-a, a)$. Another technique, proposed by Glorot and Bengio in [9], advice a properly scaled uniform distribution for initialization called **normalized initialization**:

$$\mathbf{w} \sim U\left[-\frac{\sqrt{6}}{\sqrt{n+m}}, \frac{\sqrt{6}}{\sqrt{n+m}}\right]$$

where, n and m represent, respectively, the input and output size of a given layer. This initialization scheme is commonly referred to as the *Xavier initialization*.

2.2 Error function

To minimize an error function, we need to be able to compute the gradient so we must have a differentiable error function, differentiable activation functions and a network to follow the information flow. There are various types of error functions and for the analysis, we have decided to use the mean squared error (MSE) that measures the average of the squares of the errors overall examples, for both the classification and the regression tasks, namely:

$$E(\mathbf{w}) = \frac{1}{l} \sum_{p}^{l} \sum_{r}^{R} (\mathbf{d}_{pr} - h(\mathbf{x}_{pr}))^2 = \sum_{p}^{l} E_p \tag{4}$$

where $\mathbf{d}_{pr}$ is the actual output of a patter p of the r-th output neuron, R is the number of output neurons of the output layer and l the total number of patterns in the training set.

The error function has the following properties:

- It is bounded below in $\mathbb{R}^n$ by zero.
- By assuming to use only sigmoid activation functions on the hidden layer and linear or sigmoid activation functions on the output layer, we can say that since both are continuous and differentiable, E is a differentiable function. As sigmoid function we will use the logistic function defined as follow:

$$f(x) = \frac{1}{1+e^{-x}} \ , \ f^{'}(x) = f(x)(1-f(x))$$

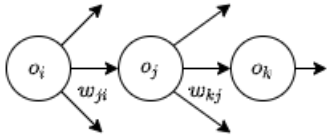
- The MSE is nonconvex function since is a composition of nonlinear activation function. It is characterized, by its nature, to be a nonconvex multimodal objective function that posses multi-local minima.

2.3 The Backpropagation method for the gradient computation

Now that we have a differentiable error function we can compute the gradient with the chain rule and the Backpropagation method.

$$\nabla E = -\frac{\partial E}{\partial \mathbf{w}} = -\frac{1}{l}\sum_{p}^{l}\frac{\partial E_p}{\partial \mathbf{w}} = \frac{1}{l}\sum_{p}^{l}\nabla_p \mathbf{w}$$

Where $E = E(\mathbf{w})$ and p refers to the p-th pattern in input the neural network. For the computation, let's consider for now only a network with 1 hidden and 1 output layers.



.

Figure 2.1: Network graph computation: o_i, o_j, o_k are respectively the output of the i-th input unit, j-th hidden unit and k-th output unit. w_{ji} is the weight associated with node j connected to input i (same for w_{kj})

The signal flow through the network in a forward direction as shown in Fig. 2.1 and a generic unit t will compute the output o_t with its activation function f_t after computing the net_t:

$$\begin{cases} net_t = \sum_q w_{tq} o_q \\ o_t = f_t(net_t) \end{cases} \tag{5}$$

With Eq. 1, 4 and 5 we can write the gradient respect a generic weights d of unit t assuming a pattern p:

$$\nabla_p w_{td} = -\frac{\partial E_p}{\partial w_{td}} = -\frac{\partial E_p}{\partial net_t}\cdot\frac{\partial net_t}{\partial w_{td}} = \delta_t \cdot o_d$$

Where, δ_t is the delta of unit t and o_d is the input from a generic unit d to the unit t, since

$$\frac{\partial net_t}{\partial w_{td}} = \frac{\partial \sum_q w_{tq} o_q}{\partial w_{td}} = o_d$$

Now let's see the development of δ_t:

$$\delta_t = -\frac{\partial E_p}{\partial net_t} = -\frac{\partial E_p}{\partial o_t}\cdot\frac{\partial o_t}{\partial net_t} = -\frac{\partial E_p}{\partial o_t}\cdot f_t'(net_t)$$

Where $-\frac{\partial E_p}{\partial o_t}$ changes according to the type of unit:

- **Out unit K**

$$-\frac{\partial E_p}{\partial o_k} = -\frac{\partial \sum_r (d_r - o_r)^2}{\partial o_k} = 2(d_k - o_k)$$

- **Hidden unit J**

$$-\frac{\partial E_p}{\partial o_j} = \sum_k -\frac{\partial E_p}{\partial net_k} \cdot \frac{\partial net_k}{\partial o_j} = \sum_k \delta_k \cdot w_{kj}$$

$$\text{Since} \begin{cases} -\frac{\partial E_p}{\partial net_k} = \delta_k \quad \text{(already computed)} \\ \frac{\partial net_k}{\partial o_j} = \frac{\partial \sum_z w_{kz} o_z}{\partial o_j} = w_{kj} \end{cases}$$

It expresses the variation of E with respect to all the output unit o_k. Each o_k (and net_k) depends on o_j, hence we introduce a sum over k.

Summary: We derived the gradient for a generic unit t with input from o_d

$$\nabla_p w_{ti} = \delta_t o_d$$

- IF $t = k$ (output unit)

$$\delta_k = 2(d_k - o_k) f_k^{'}(net_k) \tag{6}$$

- IF $t = j$ (hidden unit)

$$\delta_j = \left(\sum_k \delta_k w_{kj} \right) f_j^{'}(net_j) \tag{7}$$

This process can be generalised to m hidden layers. Deltas come not only from the output layer but also from any up-layer to the considered layer in the computational graph. In other words, deltas flow from any units to which the considered unit is connected to and this is the reason why this process is called **Backpropagation**.

2.4 Loss function: enforce Lipschitz continuity on the gradient

From the analysis made it is clear that the neural network lives in a nonconvex domain and we have to solve a nonlinear unconstrained optimization problem. Some of the methods, that will be discussed in the next sections, require that the objective function gradient is global Lipschitz continuous for the convergence analysis. Looking at the result found in the backpropagation it is immediately clear that it is not possible to take this property into account. To better understand why this is not true, let's understand which functions are Lipschitz continuous if we consider the product and the linear combination of two Lipschitz continuous functions.

Lipschitz continuity is designed to measure change of function values versus change in the independent variable for a general function[1] $f : I \subseteq \mathbb{R}^n \to \mathbb{R}^m$:

Theorem 2.1. *Suppose that $f_1, ..., f_n$ are Lipschitz continuous on I with Lipschitz constants $L_1, ..., L_n$ respectively. Then the linear combination $c_1 f_1 + ... + c_n f_n$ is Lipschitz continuous on I with Lipschitz constant $c_1 L_1 + ... + c_n L_n$.*

If we consider the product of two Lipschitz continuous functions on a bounded set I:

Theorem 2.2. *If $f_1(x)$ and $f_2(x)$ are Lipschitz continuous on a bounded set I then $f_1(x) f_2(x)$ is Lipschitz continuous on I.*

1. In practical implementations, with the typical floating point numbers, instead: $f : I \to Q$ where I is a set of rational numbers.

By analyzing the individual components of the gradient, we can see that it does not always respect the Th. 2.1 and 2.2. This can be seen immediately by looking at Eq. 7, where the partial derivative is a composition of products of bounded Lipschitz continuous functions and the weights that are not restricted within a bounded set. In fact, in a iterative algortithm the gradient will be computed k times until a stop condition is reached, the gradient Lipschitz continuity can be considered true only on the first iteration since the the weights are generated by a bounded and closed set, as seen in 2.1, but not in any further iterations.

For the purpose of the analysis, instead of considering the error function E, we are going to consider as objective function, from now on, the loss function that is generally characterized by a further term,

$$L(\mathbf{w}) = E(\mathbf{w}) + \lambda \left\|\mathbf{w}\right\|^2. \tag{8}$$

Where $\lambda \left\|\mathbf{w}\right\|^2$ is the regularization term and $\|\cdot\|$ denotes the Euclidean norm. This new function is still bounded below in $\mathbf{R}^n$ by zero, differentiable and preserves all properties of E. Now, if we consider an iterative optimization method that generate a monotonous decreasing sequence of values of the objective function, as some of the methods that we will see (FR, PR+ and HS+ conjugate methods, and the L-BFGS), and $0 < \lambda \leq 1$, we can derive :

$$L(\mathbf{w}_i) \leq L(\mathbf{w}_0)\ ,\ \forall i \in (1, 2, 3, ...k)$$
$$E(\mathbf{w}_i) + \lambda \left\|\mathbf{w}_i\right\|^2 \leq L(\mathbf{w}_0)$$

Since, $E(\mathbf{w})$ is bounded below to zero, $E(\mathbf{w}_i)$ is a constant and at least it will be zero, we can write that:

$$\lambda \left\|\mathbf{w}_i\right\|^2 \leq L(\mathbf{w}_0)$$
$$\left\|\mathbf{w}_i\right\|^2 \leq \frac{L(\mathbf{w}_0)}{\lambda}$$
$$\left\|\mathbf{w}_i\right\| \leq \sqrt{\frac{L(\mathbf{w}_0)}{\lambda}}$$

we have thus proved that, in this case, the weights are bounded and the gradient is Lipschitz continuous. Moreover, weights live in a compact set because the set is closded too.

It is interesting to note that the regularization term is quite often used in the machine learning world and it allows to control the complexity of the model. Moreover, if the λ constant is very low the weights will be subject to a very low or no regularization effect and they will be able to change arbitrarily (e.g. where a flexible model is needed it for a function approximation task). In this case, the error term will be the main part to be minimized to decrease the overall value of the loss, while still maintaining Lipschitz's continuous property on the gradient. Otherwise, with high λ value the regularization term will be more important to minimise and it will lead to low weight and very rigid models.

3. Nonlinear conjugate gradient method

To improve the performances of a Neural Network, instead of using the simple steepest-descent algorithm, an interesting second-order optimizer can be used: the *Conjugate Gradient* method. It was proposed by Hestenes and Stiefel [10] in 1952 and extended to nonlinear functions by Fletcher and Reeves [11] in 1964. This optimizer is faster than the steepest-descent and ensures low-memory requirements. The methods generate a sequence of weights $\{\mathbf{w}_k\}$, for every iteration of training k, using the iterative formula

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k \, \mathbf{d}_k, \ \ k = 0, 1, ..., \tag{9}$$

where $\alpha_k > 0$ is the step-length and $\mathbf{d}_k$ is a descent search direction. Hence the new weight update rule will be

$$\Delta \mathbf{w}_k = \alpha_k \mathbf{d}_k. \tag{10}$$

The search direction, also called the *conjugate direction*, at the k-th iteration is defined by

$$\mathbf{d_k} = \begin{cases} -\mathbf{g}_k, & \text{if } k = 0 \\ -\mathbf{g}_k + \beta_k \, \mathbf{d}_{k-1}, & \text{if } k > 0, \end{cases} \tag{11}$$

where the *conjugate gradient parameter* β_k is a scalar that quantifies how much of the previous direction should be added to the new one, and $\mathbf{g}_k = \nabla L(\mathbf{w}_k)$. Each new direction it's a linear combination of the steepest descent and the previous direction, hence the algorithm can compute a new vector $\mathbf{d}_k$ by only using the previous stored vector $\mathbf{d}_{k-1}$ and the current gradient $\mathbf{g}_k$. In fact, no operation with matrices are required compared to classical second-order methods and this is one of the main reasons why the algorithm is appealing for large optimization problem, as a neural networks optimization process, because allows to maintain the trade-off between rate of convergence, computational and storage cost.

With regard to the conjugate gradient paramenter there are different formulas and each of them leads to different conjugate gradient methods with its computational efficiency and properties. The best-known formulas are called the Fletcher-Reeves (FR), Polak-Ribière (PR), and Hestenes-Stiefel (HS), and are given by

$$\beta_k^{FR} = \frac{\mathbf{g}_k^\mathsf{T} \, \mathbf{g}_k}{\mathbf{g}_{k-1}^\mathsf{T} \, \mathbf{g}_{k-1}}, \tag{12}$$

$$\beta_k^{PR} = \frac{\mathbf{g}_k^\mathsf{T} \, (\mathbf{g}_k - \mathbf{g}_{k-1})}{\|\mathbf{g}_{k-1}\|^2}, \tag{13}$$

$$\beta_k^{HS} = \frac{\mathbf{g}_k^\mathsf{T} \, (\mathbf{g}_k - \mathbf{g}_{k-1})}{(\mathbf{g}_k - \mathbf{g}_{k-1})^\mathsf{T} \, \mathbf{d}_{k-1}}. \tag{14}$$

Now, in order to make complete the whole algorithm, after computing $\mathbf{d}_k$, the step-length $\alpha_k > 0$ has to be found. In order to do that we need to pay some attention about the line search used to choice the alpha. When dealing with strongly convex quadratic functions and *exact line search*, namely,

$$\alpha_k = \arg\min_{\alpha} \{L(\mathbf{w}_k + \alpha \mathbf{d}_k); \ \alpha > 0\}, \tag{15}$$

that compute the exact minimizer, the algorithm reduces to the linear conjugate gradient method and this ensures that each direction is a descent direction [12] for all the methods. But we are dealing with a non-linear case and the direction $\mathbf{d}_k$ not always hold the condition to be a descent direction unless certain conditions are satisfied, so we have to ensure that $\mathbf{d}_k$ has a descent property, namely,

$$\mathbf{g}_k^T \mathbf{d}_k \ < \ 0. \tag{16}$$

Let's consider the Fletcher-Reeves method for now, the inner product between Eq. 11 and the gradient $\mathbf{g}_k$ will tells us that at the first iteration we will have a descent direction by construction and for others we have to analyse

$$\mathbf{g}_k^T \mathbf{d}_k = - \|\mathbf{g}_k\|^2 + \beta_k^{FR} \mathbf{g}_k^T \mathbf{d}_{k-1}. \tag{17}$$

Thus if the the line search is exact, we have $\mathbf{g}_k^T \mathbf{d}_{k-1} = 0$ becouse the α_{k-1} is a local minimizer of our objective function along the direction $\mathbf{d}_{k-1}$. Consequently, this case led to a descent direction. Since the exact line search is usually expensive and impractical, a *inexact line search* is often consider in the implementation of non-linear conjugate gradient methods. However, when the line search is not exact, the second term in Eq.

17 may dominate the first term and the descent property may not be true. A simple restart with $\mathbf{d}_k = -\mathbf{g}_k$ might help to avoid this situation, but will probably degrade the numerical performance since the previous direction $\mathbf{d}_k$ is discarded [13]. However if we consider an inexact line-search with strong Wolfe conditions, which are,

$$L(\mathbf{w}_k + \alpha\, \mathbf{d}_k) \;\leq\; L(\mathbf{w}_k) \;+\; c_1\, \alpha\, \mathbf{g}_k^{\mathsf{T}}\, \mathbf{d}_k, \tag{18}$$

$$|\mathbf{g}(\mathbf{w}_k + \alpha\, \mathbf{d}_k)^{\mathsf{T}} \mathbf{d}_k| \;\leq\; -c_2\, \mathbf{g}_k^{\mathsf{T}}\, \mathbf{d}_k, \tag{19}$$

with $0 < c_1 < c_2 < \frac{1}{2}$ (in place of the classical condition $c_2 < 1$ [12] as shown in Sec. 3.1) condition 19 implies that Eq. 17 is negative (see Sec 3.2). With regard to other methods (PR, HS) strong Wolfe conditions do not guarantee that $\mathbf{d}_k$ is always a descent direction. Nevertheless, Powell [14] showed that the PR method cycles without obtaining the solution, with an exact line search, same can be shown for HS since they are identical with an exact line search $\mathbf{g}_k^T \mathbf{d}_{k-1} = 0$. To overcome this problem, as suggested by Powell [14], those methods must be modified in

$$\beta^+ = max\{\beta,\, 0\}. \tag{20}$$

Then a simple adaptation of the strong Wolfe conditions ensures that the descent property holds and, from now on, we will refer to them as PR+ and HS+. This modification introduces a sort of restart of the algorithm, in case the β found is negative. Indeed the algorithm puts aside the last search direction and restarts with the steepest descent.

We can finally put everything together and define the whole algorithm in Algo 1.

Algorithm 1: Nonlinear Conjugate Gradient

init: starting point $\mathbf{w}_0$, max_iter, $\varepsilon = 10^{-4}$, $k = 0$;
while *k < max_iter* **do**
  Evaluate the Loss function L_k and its gradient g_k;
  if $\|g_k\| < \varepsilon$ **then**
    return Loss L_k
  end
  Compute β_k with one of the methods PR, FR, HS;
  Compute the descent direction $\mathbf{d}_k$ by *conjugate direction*;
  Compute the step-legnth α_k with a Line Search satisfying the Strong Wolfe conditions;
  Update the weights $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \alpha_k\, \mathbf{d}_k$;
  Set $k \leftarrow k + 1$;
end

Several beta formulas have been presented and each of them requires a different theoretical analysis. As discussed in [15] the Fletcher-Reeves [11] method can be as efficient as the Polak-Ribire and Hestenes Stiefel methods, but it is often much slower and erratic. Moreover, as shown by Powell [13], the Fletcher-Reeves, under some circumstances, can produce very small displacements and it needs some restart along the gradient direction in order to recover. Despite these drawbacks, Zoutendijk [16] has shown that the method cannot fail. He proved that the Fletcher-Reeves method with exact line searches is globally convergent on general functions and Al-Baali [17] extended this result to inexact line searches. The Polak-Ribire and Hestenes-Stiefel methods tend to be more robust and efficient. They are to be preferred over the Fletcher-Reeves method. They are very similar in term of theoretical convergence properties too, in fact, as we've already said, they need to restart, since the direction can fail to be a descent direction [12, 13]. So Polak-Ribire and Hestenes-Stiefel are not featured by having a global convergence result without restart. All

this makes those methods attractive to be studied in our case. Thus, we will try to observe how these behaviours can affect our neural network objective function optimization process used in the experimental phase.

A strong Wolfe inexact line search was implemented for our experiments, this will be explained extensively in the next subsection. Having defined this we will focus on the convergence analysis of the algorithm in our specific case and finally conclude with the convergence rate analysis.

3.1 Strong Wolfe inexact line search

For our implementation, we decided for a line search that satisfies the strong Wolfe conditions. This line search is done in two stages: a *bracketing phase*, that finds an interval containing step lengths that minimize the Loss and an *interpolation phase* that, given the interval, finds an acceptable step length. Strong Wolfe condition have already been introduced in Eq. 18 and 19. Here, we will discuss their behaviour in detail.

- *Armijo* condition Eq. 18 is the *sufficient decrease* condition where $c_1 \in (0, 1)$ is a constant and $\mathbf{g}_k^\mathsf{T}\mathbf{d}_k$ is the directional derivative. It ensures that the step-length α is not too large and forces a sufficient decrease in the function. However, not sufficient to guarantee convergence, because it is satisfied for all sufficiently small values of $\alpha > 0$. The constant c_1 has to be quite small in general.
- *Strong Wolfe* condition Eq. 19 where $c_2 \in (c_1, 1)$ is a constant. This condition ensures that the slope of the function after moved by a step α (that is not too small because of this condition) along the direction $\mathbf{d}$ is greater than c_2 times the initial slope. Moreover, the strong condition doesn't allow the derivative $\mathbf{g}(\mathbf{w}_k + \alpha\,\mathbf{d}_k)^\mathsf{T}\mathbf{d}_k$ to be too positive and ensures that α is close to a minimizer of the function.

In our experimental setting, as suggested in [12], we used $c_1 = 10^{-4}$ and $c_2 = 0.1$ and our strong Wolfe line search follows the one described in [12] and is composed by two functions: `line_search` and `zoom`. They are both illustrated in Algo. 2 and 3 where it is assumed that: $\phi(\alpha) = L(\mathbf{w}_k + \alpha\mathbf{d}_k)$ and $\mathbf{d} = \mathbf{d}_k$.

3.2 Convergence analysis

Now we are going to understand which method is globally convergent and what is needed to ensure this property for each nonlinear conjugate gradient method using a practical line search. Finally, we will conclude with the same analysis but by considering our loss function defined in Eq. 8.

Let's consider, once again, first the analysis for FR method. First, we have to consolidate what has been said about the descent direction when an inexact line-search is used to compute the step-length.

For the purposes of this section, we need the following assumptions on the objective function.

Assumption 3.1. *The level set $\mathcal{L} = \{\mathbf{w} \mid L(\mathbf{w}) \leq L(\mathbf{w}_0)\}$ is bounded, namely, there exists a constant $B > 0$ such that:*

$$\|\mathbf{w}\| \leq B,\ \forall i \in \mathcal{L}$$

Assumption 3.2. *In some neighborhood $\mathcal{N}$ of $\mathcal{L}$, the objective function L is continuously differentiable, and its gradient is Lipschitz continuous, there exists a constant $C > 0$ such that*

$$\|\mathbf{g}(\boldsymbol{w}) - \mathbf{g}(\tilde{\boldsymbol{w}})\| \ \leq\ C\|\mathbf{w} - \tilde{\mathbf{w}}\|,\ \forall \mathbf{w}, \tilde{\mathbf{w}} \in \mathcal{N}$$

It follows directly from A. 3.1 and A. 3.2 that there exists a positive constant $\gamma > 0$ such that:

$$\|\mathbf{g}(\boldsymbol{w})\| \ \leq\ \gamma,\ \ \forall \mathbf{w} \in \mathcal{L}$$

Under assumptions (3.1,3.2) we can state the following Lemma.

Algorithm 2: Line Search

init: $\alpha_0 \leftarrow 0$, choose $\alpha_{max} > 0$ and $\alpha_1 \in (0,\, \alpha_{max})$;
$i \leftarrow 1$;
while *i < max_iter* **do**
  Evaluate $\phi(\alpha_i)$;
  if $[\phi(\alpha_i) > \phi(0) + c_1\, \alpha_i\, \nabla\phi(0)^{\mathsf{T}} d]$ *or* $[\phi(\alpha_i) \leq \phi(\alpha_{i-1})$ *and* $i > 1]$ **then**
    $\alpha_* \leftarrow \mathbf{zoom}(\alpha_{i-1},\, \alpha_i)$;
    return α_*;
  end
  Evaluate $\nabla\phi(\alpha_i)^{\mathsf{T}} d$;
  if $|\nabla\phi(\alpha_i)^{\mathsf{T}} d| \leq -c_2\, \nabla\phi(0)^{\mathsf{T}} d$ **then**
    $\alpha_* \leftarrow \alpha_i$;
    return α_*;
  end
  if $\nabla\phi(\alpha_i)^{\mathsf{T}} d \geq 0$ **then**
    $\alpha_* \leftarrow \mathbf{zoom}(\alpha_i,\, \alpha_{i-1})$;
    return α_*;
  end
  Choose $\alpha_{i+1} \in (\alpha_i,\, \alpha_{max})$;
  $i \leftarrow i + 1$;
end

Algorithm 3: zoom

while *i < max_iter* **do**
  $\alpha_j \leftarrow quadratic\_cubic\_interpolation(\alpha_{lo},\, \alpha_{hi})$;
  Evaluate $\phi(\alpha_j)$;
  if $[\phi(\alpha_j) > \phi(0) + c_1\, \alpha_j\, \nabla\phi(0)^{\mathsf{T}}\, d_0]$ *or* $[\phi(\alpha_j) \leq \phi(\alpha_{lo})]$ **then**
    $\alpha_* \leftarrow \alpha_j$;
    return α_*;
  else
    Evaluate $\nabla\phi(\alpha_j)^{\mathsf{T}} d$;
    if $|\nabla\phi(\alpha_j)^{\mathsf{T}} d| \leq -c_2\, \nabla\, \phi(0)^{\mathsf{T}}\, d$ **then**
      $\alpha_* \leftarrow \alpha_j$;
      return α_*;
    end
    if $\nabla\phi(\alpha_j)^{\mathsf{T}} d\, (\alpha_{hi} - \alpha_{lo}) \geq 0$ **then**
      $\alpha_{hi} \leftarrow \alpha_{lo}$;
    end
  end
  Set $\alpha_{lo} \leftarrow \alpha_j$;
  $i \leftarrow i + 1$;
end

Lemma 3.3. *Suppose that Algorithm 1 is implemented with a step length α_k that satisfies the strong Wolfe conditions (18,19) with $0 < c_1 < c_2 < \frac{1}{2}$. Then the method generates descent directions $\mathbf{d}_k$ that satisfy the following inequalities:*

$$-\frac{1}{1-c_2} \leq \frac{\mathbf{g}_k^T \mathbf{d}_k}{\|\mathbf{g}_k\|^2} \leq \frac{2c_2 - 1}{1-c_2}, \ \textit{for all } k = 0, 1, ... \tag{21}$$

This Lemma, well proved in [12], tells us that we have a descent direction

Then, the Zoutendijk's theorem will be our main tool and is defined as follows.

Theorem 3.4 (Zoutendijk's theorem)**.** *Consider any iteration of the form (9), where $\mathbf{d}_k$ is a descent direction and α_k satisfies the Wolfe conditions (18,19). Suppose that Assumptions 3.1 and 3.2 hold, then*

$$\sum_{k>0} \cos^2 \theta_k \, \|\mathbf{g}_k\|^2 \ < \ 0 \tag{22}$$

This theorem is proved and well explained in [12, 15]. By combining Zoutendijk's result and Lemma 3.3, Al-Baali [17] has proved that the FR method is globally convergent with the following theorem.

Theorem 3.5 (Al-Baali [17])**.** *If Assumptions 3.1 and 3.2 hold, and that Algorithm 1 is implemented with a line search that satisfies the strong Wolfe conditions, with $0 < c_1 < c_2 < \frac{1}{2}$ in order to have a descent direction. Then*

$$\lim_{k \to \infty} \inf \|\mathbf{g}_k\| = 0. \tag{23}$$

Instead, PR and HS methods do not always guarantee a descent direction so, the same global convergence result, as for FR, cannot be achieved. Gilbert and Nocedal have studied the global convergence for nonnegative β_k (see [15]) and the following theorem proves that those method are globally convergent.

Theorem 3.6. *If Assumptions 3.1 and 3.2 hold, the Algorithm 1 is implemented with a line search that satisfies the strong Wolfe conditions in order to have a descent direction and $\beta_k \geq 0$ for all k. Then*

$$\liminf_{k \to \infty} \ \|\mathbf{g}_k\| = 0. \tag{24}$$

Therefore, the global convergence property can be proved too for the modified β_k in Eq. 20 (PR+ and HS+) suggested by Powell [14].

It seems that FR is characterised by some very fascinating theoretical convergence properties when compared to methods such as PR and HS, unless restart is used. However, from a practical point of view, there are some observations to be taken into account. We have already mentioned that FR can be subject to produce very small displacements, in fact, this is can be consider the main weakness of this method. We will argue that if the method generate a very small step or a bad direction, then this behaviour can continue for a very large number of iteration unless a restart is made. Initially observed by Powell [14] with an exact line search, this can be proved too for an inexact line searches as showed by Nocedal [15] using Lemma 3.3. We will state here what Nocedal has proven and in order to show this behaviour, let's define θ_k as the angle between $\mathbf{d}_k$ and the steepest descent direction $-\mathbf{g}_k$, namely,

$$\cos \theta_k = \frac{-\mathbf{g}_k^T \mathbf{d}_k}{\|\mathbf{g}_k\| \, \|\mathbf{d}_k\|}, \tag{25}$$

and suppose that at iteration k a bad search direction is generated, such that $\cos\theta_k \approx 0$ given by an angle of nearly 90°of $\mathbf{d}_k$ with $\mathbf{g}_k$. By using Eq. 25 and multiplying both side of Eq. 17 by $\|\mathbf{g}_k\| \,/\, \|\mathbf{d}_k\|$, we get as a result

$$-\frac{1-2c_2}{1-c_2}\frac{\|\mathbf{g}_k\|}{\|\mathbf{d}_k\|} \leq \cos\theta_k \leq \frac{1}{1-c_2}\frac{\|\mathbf{g}_k\|}{\|\mathbf{d}_k\|}, \quad \text{for all } k = 0, 1, ... \tag{26}$$

then, $\cos\theta_k \approx 0$ if and only if

$$\|\mathbf{g}_k\| \ll \|\mathbf{d}_k\|.$$

This means that $\mathbf{w}_{k+1} \approx \mathbf{w}_k$, therefore $\mathbf{g}_{k+1} \approx \mathbf{g}_k$ and

$$\beta_{k+1}^{FR} \approx 1.$$

By using all this approximation we will have that $\|\mathbf{g}_{k+1}\| \approx \|\mathbf{g}_k\| \ll \mathbf{d}_k$ and we can conclude that

$$\mathbf{d}_{k+1} \approx \mathbf{d}_k.$$

It's clear that a bad move led to a long sequence of bad moves and the only way to avoid this is to restart with steepest descent direction at some point. Nocedal [15] showed a practical example where, this undesirable behavior, can be observed in practice, moreover in that example the FR method performs much better if it is periodically restarted along the steepest descent direction in order to avoid the cycle of bad moves. This is why the FR method is highly recommended to implement it with a kind of restart strategy. On the other hand, the other methods (PR, HS, PR+, HS+) would behave quite differently in this situation. As we have already said, all of them behave identically to FR method in the quadratic case but the definition differs in nonlinear case and, in particular, when there is stagnation if $\mathbf{g}_{k+1} \approx \mathbf{g}_k$ then those methods will have $\beta^+ \approx 0$ given an automatic restart and that why they tend to be more robust and efficient. Clearly restart is very important in the implementation of the conjugate algorithm for different purposes, avoid weakness (in the FR) or avoid negative directions of PR and HS in order to reach global convergent properties (PR+ and HS+). Anyway, the methods' behaviour could lead to different efficiency results depending on the problem being solved. So, given the global convergence results, the pros and cons found for each method, we will be compare them in our specific case in the experimental section to understand which one works best in the case of a nonlinear function derived from a neural network.

To conclude, let's analyze each method in our specific case. First, we focus in the case where the Loss function (see Eq. 8) has $0 < \lambda \leq 1$. We are able to prove that the FR method will converge to a stationary point with Th. 3.5 because a descent direction is always taken if a strong Wolfe line search with $0 < c_1 < c_2 < \frac{1}{2}$ is used in the Algo. 1, assumptions 3.1 and 3.2 always hold. Those assumptions are satisfied because, as it has been proved in Sec. 2.4, due to the monotonic property of the FR method and the regularization term, the level set is bounded and, as a direct consequence, given the composition of the gradient, the gradient of the loss function is Lipschitz continuous. Anyway, attention must be paid to the main weakness of the FR method indicated in the analysis. Instead, we are not able to prove that PR and HS method will converge to a stationary point because a descent direction is not always guarantee and, because of this, Th. 3.5 does not hold and the assumptions required (A. 3.1, 3.2) cannot be proved with the analysis in Sec. 2.4 due to the absence of the monotonic properties in these methods. As regards the PR+ and HS+, Th. 3.6 proves that we have the convergence to a stationary point because a descent direction is always ensured through the restart of the modified method (see Eq. 20) and all the remaining assumptions of the theorem can be proved as for the FR method. On the other hand, if we consider the case where the L function has $\lambda = 0$ we cannot prove the convergence property to any of the described methods because the analysis made in Sec. 2.4 can't be applied and assumptions 3.1 and 3.2 will not hold.

3.3 Convergence rate analysis

For what concerns the convergence rate analysis, most of the theory of conjugate gradient methods assume that the line search is exact. It has been hypothesised that the conjugate gradient method for minimizing functions of several variables has a superlinear rate of convergence, but it was later proved that this hypothesis was false. In fact, the first result on this subject has been reported by Crowder and Wolfe [18] where they showed that the rate of convergence is linear and, by constructing an example, that Q-superlinear convergence is not achievable. Then, Powell [19] has taken into account the analysis of a quadratic function with n variables. The quadratic case is very interesting since an iterative algorithm should be able to manage it efficiently. Moreover, if we consider a general differentiable function and if the minimization algorithm is taking small steps, then we can say that it enters a region where the objective function is quadratic. The follow stronger result has been proved by Powell.

Theorem 3.7. *If the conjugate gradient algorithm is applied to a quadratic function or enters a region where the objective function is quadratic, then either termination is obtained within* $(n+1)$ *iterations or convergence to the solution occurs at a linear rate.*

But conjugate gradient methods are used to minimize general function too (nonlinear, nonquadratic, functions on $\mathbb{R}^n$). Cohen in [20] has proven that restarting leads to n-step quadratic convergence of the methods for general objective functions. We will state here the main result of its work.

Theorem 3.8. *Suppose the conjugate gradient algorithm (PR) or (FR) is used to solve the Minimum Problem (Eq. 2) where we reinitialize the conjugate variable* β *every* r *steps (where* $r \geq n$*). Let* $\mathbf{w}_0$, $\mathbf{w}_1$, $\mathbf{w}_2$, ... *be the sequence formed; then there exists a constant* C *that*

$$\limsup_{k \to \infty} \frac{|\mathbf{w}_{kr+n} - \mathbf{w}^*|}{|\mathbf{w}_{kr} - \mathbf{w}^*|^2} \leq C \leq \infty. \tag{27}$$

Where n is the number of variables and r is some constant bigger or equal to n that is the iteration in which the descent direction is reset to the steepest descent direction. It is true for the PR or FR methods but can be extended to the other methods in order to ensure that the error, when starting from a point of reinitialization, will be decreased by order 2 after n steps in a neighbourhood of the minimum $\mathbf{w}^*$.

The proof will not be included but, to understand Th. 3.8, we will state here some observations related to the conjugate methods implementation and an analysis analogous to the one made by Nocedal in [12] . As said before, in a way or in another restart is needed and a common practice is to implement the conjugate gradient procedures with a *restart* at every n steps by setting $\beta_k = 0$ (or with other criteria). This restart allows the algorithm to be periodically refreshed, erasing old information that may not be beneficial. Then we have to focus on the implementation of the line search. Cohen has used an exact line search in the analysis. However, it's a common use to keep the nonlinear conjugate gradient methods implementations in a close connection with the linear conjugate gradient method. Usually, we want to preserve the fact that if we consider a strictly convex quadratic function the nonlinear conjugate gradient method has to reduce to the linear method. This is granted by the fact that a quadratic (or cubic) interpolation, along the search direction $\mathbf{d}_k$, is used into the line search procedure and the step length α_k is chosen to be the exact one-dimensional minimizer; see Algo 3. Using those observations is not surprising that the result given by Cohen can be proved. Assume a strongly convex quadratic function in a neighbourhood of the solution (nonquadratic everywhere else) and that, in some point, the NCG eventually enter in a quadratic region while is converging to the solution in question. At some point n it will restart and its behavior will be equal to the linear conjugate gradient method. This is true even if the function is not exactly quadratic in the region of a solution. Taylor's theorem tells us that if a function is twice continuously differentiable we can still approximate it closely to a quadratic.

We have a nice result from a theoretical point of view, but in practice, things can be quite different. The nonlinear conjugate gradient is usually used to solve problems with large n variables and the restart could never occur in n because the problem could be solved in few or before reaching n steps. So, usually, restart is implemented with some criteria or with a cycle reset that is less than n. For example, a criteria could be the one suggested by Powell [13], where a restart is performed if the following inequality is violated

$$\left|\mathbf{g}_k^T \mathbf{g}_{k-1}\right| \leq \nu \left\|\mathbf{g}_k\right\|^2. \tag{28}$$

Moreover, the FR method requires a restart when small displacement are met but this may not be enough as a restart to reach the n-step quadratic rate, because small displacement may never be reached and this restart may not be fired at any point. So another hyperparameter must be added in order to end up into the possibility to restart when a quadratic region is reached. If restarts are chosen correctly we expect to reach n-quadratic rate otherwise only linear rate is expected. PR and HS methods perform an automatic restart and another restart hyperparameter is needed to have the same possibility as FR to end up into n-step quadratic rate. Modified β in Eq. 20 can be consider as an early restarting policy but these restarts are rather infrequent because PR or HS are positive most of the time. Having said that, we can only make those conjectures and analyze the behavior in the experimental section.

We can conclude that, since our objective function L is twice continuously differentiable and Taylor's theorem says that can be approximate closely to a quadratic in a neighbourhood of the solution, an n-step quadratic convergence rate can be proved with Th 3.8, for all the conjugate methods presented in this section, if the conjugate method is re-initialised periodically. But we cannot say, with certainty, that this convergence rate will always occur because of the practical analysis described above. Therefore, we expect an n-quadratic rate by wisely choosing the restart otherwise only linear.

4. Limited-Memory BFGS Method

The conjugate gradient has led us to nice properties and faster convergence than the classic gradient descent method. Another class of optimization algorithms is the *quasi-Netwon methods*, which are characterized by further interesting convergence properties such as *superlinear convergence* rate.

The first quasi-Newton method was developed by W. C. Davidon in 1959 [21], and, like steepest descent, only the gradient must be provided at each iteration [12] and allows to avoid the direct computation of the Hessian, usually not feasible to compute and is essential for Newton method. By measuring the changes in gradients, it constructs a model of the objective function that is good enough to produce superlinear convergence. In 1963 Fletcher and Powell [22] have demonstrated theoretically and with numerical examples that it was considered superior to other previously methods available at that time - It transformed the nonlinear optimization. In fact, the improvement over steepest descent was dramatic and this is the reason why, over the years, several variants of this method were proposed to achieve even better results in the nonlinear optimization. The most popular quasi-Newton algorithm is the *Broyden–Fletcher–Goldfarb–Shanno (BFGS)* method that constructs positive definite Hessian inverse approximations $\mathbf{H}_k$ at each iteration k. However, when large problems are taken into account, computing the Hessian approximations can be unfeasible in terms of computation and memory cost, in fact, $\mathbf{H}_k$ could be sparse and requires $O(n^2)$ in storage. Many applications raise unconstrained optimization problems with thousands or millions of variables, as in a neural network model, and those approaches become unfeasible to use.

Since our objective function lies in the case of large scale optimization problem, here we will take in analysis the study of a limited-memory quasi-Newton method, useful when Hessian matrices cannot be computed at a reasonable cost or are not sparse (respectively problems occurred in Newton and quasi-Netwon methods), known as the *limited-memory BFGS* (L-BFGS). This method keeps in memory only a few

vectors of length n that allow storing the $\mathbf{H}_k$ in an implicitly way, instead of storing the full inverse Hessian approximation. In order to describe how the L-BFGS works, we have to start with the BFGS method first.

4.1 Road to L-BFGS: the BFGS update formula

In order to derive the BFGS update formula for the Hessian approximation, we need to state here the following quadratic model that approximates our objective function at the current $\mathbf{w}_k$, namely,

$$m_k(\mathbf{d}) = L(\mathbf{w}_k) + \mathbf{g}_k^T \mathbf{d} + \frac{1}{2}\mathbf{d}^T \mathbf{B}_k \mathbf{d}. \tag{29}$$

The symmetric positive definite matrix $\mathbf{B}_k \in \mathbb{R}^{n \times n}$ is the Hessian approximation used instead of the true Hessian $\nabla^2 L_k$. The next step update will be

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k \, \mathbf{d}_k, \tag{30}$$

where α_k is the step length chosen to satisfy strong Wolfe conditions (see Sec. 3.1), and the new search direction $\mathbf{d}_k$ is the minimizer of the convex quadratic model (29) and is defined as follow

$$\mathbf{d}_k = -\mathbf{B}_k^{-1} \mathbf{g}_k = -\mathbf{H}_k \, \mathbf{g}_k. \tag{31}$$

The BFGS method proposes an updating formula that allows us to compute the inverse approximation of the Hessian $\mathbf{H}_k$ at each iteration in an updated manner instead of computing it anew. To derive it, we need to do some observation. If we consider the quadratic model of Eq. 29 but this time for $\mathbf{w}_{k+1}$ and $\mathbf{B}_{k+1}$, then the gradient of m_{k+1} needs those requirements,

$$\nabla m_{k+1}(0) = \nabla L_{k+1}, \tag{32a}$$

$$\nabla m_{k+1}(-\alpha_k \, \mathbf{d}_k) = \nabla L_{k+1} - \alpha_k \mathbf{B}_{k+1} \mathbf{d}_k = \nabla L_k. \tag{32b}$$

The first condition (32a) is satisfied automatically by the model, but the second one (32b) needs to be enforced. We can rewrite Eq. 32b and obtain the *secant equation*

$$\mathbf{B}_{k+1} \mathbf{s}_k = \mathbf{y}_k, \tag{33}$$

by define those vectors

$$\mathbf{s}_k = \mathbf{w}_{k+1} - \mathbf{w}_k, \quad \mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k. \tag{34}$$

Since we need the inverse approximation of the Hessian $\mathbf{H}_{k+1}$, we can rewrite the secant equation in this way

$$\mathbf{H}_{k+1} \mathbf{y}_k = \mathbf{s}_k. \tag{35}$$

The updated approximation $\mathbf{H}_{k+1}$ must be symmetric and positive definite, and must satisfy the *secant equation* (35). This is possible only if $\mathbf{s}_k$ and $\mathbf{y}_k$ satisfy the *curvature condition*

$$\mathbf{s}_k^\mathsf{T} \mathbf{y}_k > 0. \tag{36}$$

For non-convex functions, like in our case, the inequality (36) is guaranteed to hold if we impose the strong Wolfe conditions on the line search that chooses α_k.

When the curvature condition is verified, the secant equation (35) admits an infinite number of solutions. Therefore, to determine $\mathbf{H}_{k+1}$ uniquely, we have to solve the problem

$$\min_{\mathbf{H}} \|\mathbf{H} - \mathbf{H}_k\| \tag{37a}$$

$$\text{s.t.} \quad \mathbf{H} = \mathbf{H}^\mathsf{T}, \quad \mathbf{H}\mathbf{y}_k = \mathbf{s}_k, \tag{37b}$$

where $\mathbf{s}_k$ and $\mathbf{y}_k$ satisfy (36) and $\mathbf{H}_k$ is *symmetric* and *positive definite*. Solving (37a) is equal to impose the condition that *among all symmetric matrices satisfying the secant equation, $\mathbf{H}_{k+1}$ is closest to the current matrix $\mathbf{H}_k$*. This minimization problem is solved using the *weighted Frobenius norm*, which allows us to have the scale-invariant optimization method (38) (different norms leads to different quasi-Newton methods).

We can finally state here the BFGS formula update,

$$\mathbf{H}_{k+1} = \mathbf{V}_k^{\mathsf{T}}\, \mathbf{H}_k\, \mathbf{V}_k + \rho_k\, \mathbf{s}_k\, \mathbf{s}_k^{\mathsf{T}}, \tag{38}$$

where

$$\rho_k = \frac{1}{\mathbf{y}_k^{\mathsf{T}}\, \mathbf{s}_k}, \qquad \mathbf{V}_k = \mathbf{I} \,-\, \rho_k\, \mathbf{y}_k\, \mathbf{s}_k^{\mathsf{T}}, \tag{39}$$

and an initial fixed positive definite $\mathbf{H}_0$, chosen with some heuristics (see [12]), must be provided. This formula keeps $\mathbf{H}_k$ positive definite at each iteration.

4.2 L-BFGS formula and algorithm

This limited-memory method has the same update rule of the BFGS for the next iterate (Eq. 30), but it differs on the $\mathbf{H}_k$ computation in Eq. 31 and in the choice of the initial matrix $\mathbf{H}_0$ that is allowed to vary from iteration to iteration (in contrast with what we have said for the classic BFGS formula).

The product $\mathbf{H}_k\mathbf{g}_k$ in Eq.31 can be obtained by a recursive formula involving a sequence of inner products and vector summations of the gradient $\mathbf{g}_k$, an initial inverse Hessian approximation $\mathbf{H}_k^0$ and the set of vector pairs $\{\mathbf{s}_i,\, \mathbf{y}_i\}$ that includes curvature information from the m most recent iterations that reduce the memory complexity. Indeed, after each iteration, the oldest vector pair $\{\mathbf{s}_i,\, \mathbf{y}_i\}$ is replaced by the new one obtained at the current iteration $\{\mathbf{s}_k,\, \mathbf{y}_k\}$. By combining everything, we can repeat the Eq. 38 and derive the following L-BFGS approximation $\mathbf{H}_k$ update formula:

$$\begin{aligned}
\mathbf{H}_k = &(\mathbf{V}_{k-1}^{\mathsf{T}} \dots \mathbf{V}_{k-m}^{\mathsf{T}})\, \mathbf{H}_k^0\, (\mathbf{V}_{k-m} \dots \mathbf{V}_{k-1}) \\
&+ \rho_{k-m}\, (\mathbf{V}_{k-1}^{\mathsf{T}} \dots \mathbf{V}_{k-m+1}^{\mathsf{T}})\, \mathbf{s}_{k-m}\mathbf{s}_{k-m}^{\mathsf{T}}\, (\mathbf{V}_{k-m+1}^{\mathsf{T}} \dots \mathbf{V}_{k-1}^{\mathsf{T}}) \\
&+ \rho_{k-m+1}\, (\mathbf{V}_{k-1}^{\mathsf{T}} \dots \mathbf{V}_{k-m+2}^{\mathsf{T}})\, \mathbf{s}_{k-m}\mathbf{s}_{k-m+1}^{\mathsf{T}}\, (\mathbf{V}_{k-m+2}^{\mathsf{T}} \dots \mathbf{V}_{k-1}^{\mathsf{T}}) \\
&+ \dots \\
&+ \rho_{k-1}\mathbf{s}_{k-1}\mathbf{s}_{k-1}^{\mathsf{T}}.
\end{aligned} \tag{40}$$

This method stores a *modified* version of $\mathbf{H}_k$ *implicitly* in a compact way by using just a finite number (m) of vector pairs $\{\mathbf{s}_i,\, \mathbf{y}_i\}$. The m value can be controlled by the user to manage the trade-off between the storage complexity and better approximation of the Hessian. Eq. 40 leads to Algo. 4 that allows recursively computing product $\mathbf{H}_k\, \mathbf{g}_k$ and has a $O(mn)$ computational complexity, where m is the number of vector pair stored in memory.

Regarding the choice of the initial Hessian, the BFGS method usually use a fixed identity matrix (note that $\mathbf{I}$ is positive definite), but here we can start from the identity matrix $\mathbf{I}$ and after one iteration change it in an iterative manner as follow

$$\mathbf{H}_k^0 = \gamma_k\, \mathbf{I} = \frac{\mathbf{s}_{k-1}^{\mathsf{T}}\mathbf{y}_{k-1}}{\mathbf{y}_{k-1}^{\mathsf{T}}\mathbf{y}_{k-1}}\, \mathbf{I}. \tag{41}$$

The scaling factor γ_k attempts to estimate the size of the true Hessian matrix along the most recent search direction (allowing $\mathbf{H}_k^0$ to vary between iterations). This choice helps to ensure that the search direction $\mathbf{d}_k$ is well scaled, and as a result, the step length $\alpha_k = 1$ is accepted in most iterations.

We can finally state the whole algorithm in Algo. 5. Note that is very important to use a line search that satisfies the strong Wolfe conditions in order to satisfy the curvature condition in Eq. 36 and keep the

Algorithm 4: L-BFGS two-loop recursion

$\mathbf{q} \leftarrow \mathbf{g}_k$;
for $i = k-1,\ k-2,\ \dots,\ k-m$ **do**
  $\alpha_i \leftarrow \rho_i\, \mathbf{s}_i^{\mathsf{T}}\, \mathbf{q}$;
  $\mathbf{q} \leftarrow \mathbf{q} - \alpha_i\, \mathbf{y}_i$;
end
$\mathbf{r} \leftarrow \mathbf{H}_k^0\, \mathbf{q}$;
for $i = k-m,\ k-m+1,\ \dots,\ k-1$ **do**
  $\beta \leftarrow \rho_i\, \mathbf{y}_i^{\mathsf{T}} \mathbf{r}$;
  $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{s}_i(\alpha_i - \beta)$;
end
return $\mathbf{H}_k\, \mathbf{g}_k = \mathbf{r}$

BFGS stable in the updating process. The procedure described in Sec. 3.1 can be used here too. However, as suggested by Nocedal [12], in our experiment we will use $c_1 = 10^{-4}$, $c_2 = 0.9$ and step $\alpha_0 = 1$ will be always tried as initial step length.

Algorithm 5: L-BFGS

init: starting point $\mathbf{w}_0$, max_iter, $\varepsilon = 10^{-4}$, $m > 0$;
$k \leftarrow 0$;
while *k < max_iter* **do**
  Evaluate the Loss function L_k and its gradient g_k;
  if $\|g_k\| < \varepsilon$ **then**
    return Loss L_k
  end
  Choose the initial matrix $\mathbf{H}_k^0$;
  Compute the search direction $\mathbf{d}_k \leftarrow -\mathbf{H}_k\, \mathbf{g}_k$ with the L-BFGS formula;
  Compute the step length α_k with a Line Search satisfying the Strong Wolfe conditions;
  Update the weights $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \alpha_k \mathbf{d}_k$;
  if $k > m$ **then**
    Discard the oldest vector pair $\{\mathbf{s}_{k-m},\ \mathbf{y}_{k-m}\}$ from storage;
  end
  Compute and save $\mathbf{s}_k \leftarrow \mathbf{w}_{k+1} - \mathbf{w}_k$, $\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$;
  Set $k \leftarrow k+1$;
end

It is interesting to note that $\mathbf{d}_k$ will be a *descent direction* if $\mathbf{H}_{k+1}$ is *positive-definite*. As shown by Nocedal in [12], this holds whenever $\mathbf{H}_k$ (previous approximation) is positive definite, and the curvature condition is verified. Indeed, if $\mathbf{s}_k^{\mathsf{T}}\mathbf{y}_k$ is positive, for any $\mathbf{z} \neq 0$ we have

$$\mathbf{z}^{\mathsf{T}}\mathbf{H}_{k+1}\mathbf{z} = \mathbf{w}^{\mathsf{T}}\mathbf{H}_k\mathbf{w} + \rho_k(\mathbf{z}^{\mathsf{T}}\mathbf{s}_k)^2 \geq 0, \tag{42}$$

where $\mathbf{w} = \mathbf{z} - \rho_k \mathbf{y}_k(\mathbf{s}_k^{\mathsf{T}}\mathbf{z})$. In order to have $\mathbf{z}^{\mathsf{T}}\mathbf{H}_{k+1}\mathbf{z} = 0$ we must have $\mathbf{w} = 0$ and $\mathbf{z}^{\mathsf{T}}\mathbf{s}_k = 0$. But $\mathbf{z}^{\mathsf{T}}\mathbf{s}_k = 0$ implies $\mathbf{w} = \mathbf{z} = 0$ that is in contrast with the previous assumption $\mathbf{z} \neq 0$. Thus, $\mathbf{z}^{\mathsf{T}}\mathbf{H}_{k+1}\mathbf{z}$ and $\mathbf{w}^{\mathsf{T}}\mathbf{H}_k\mathbf{w}$ are greater than zero, and $\mathbf{H}_{k+1}$ and $\mathbf{H}_k$ are positive definite. The L-BFGS method has this property since the initial $\mathbf{H}_0$ in Eq. 41 is positive definite and the curvature condition in Eq. 36 is satisfied by the strong Wolfe line search.

4.3 Convergence analysis

In this section, we introduce the global convergence properties of the BFGS method, which are also inherited by its limited-memory version. First, we will consider the general analysis of a nonlinear function with an exact line search case, then we will go to analyze the convex and non-convex case with a practical line search. First, the analysis will consider a general objective function f, then, our objective function L.

Zangwill's Global Convergence Theorem

The first analysis for the nonlinear case is the one shown by Bazaraa in [23], which uses an exact line search, and the general result given by the Zangwill's Global Convergence Theorem [24], which considers a general iterative algorithm **A** as a composition map **A = CB** (where **B**, only for this paragraph, is not meant to be the Hessian approximation $\mathbf{B}_k$). To understand this general result, we need to state the following two theorems.

Theorem 4.1. *Let X, Y and Z be nonempty closed sets in $\mathbb{R}^n$, $\mathbb{R}^p$, $\mathbb{R}^q$, respectively. Let $\mathbf{B} : X \to Y$ and $\mathbf{C} : Y \to Z$, and consider the composite map **A = CB**. Suppose that **B** is closed at **x** and that **C** is closed on **B(x)**. Furthermore, suppose that if $\mathbf{x}_k \to \mathbf{x}$ and $\mathbf{y}_k \in \mathbf{B}(\mathbf{x}_k)$, then there is a convergent subsequence of $\{\mathbf{y}_k\}$. Then A is closed at x.*

If we consider **B** as the step of a convergent algorithm (spacer step) that satisfies the assumptions of Th. 4.1, and **C** as the set of intermediate steps of the complex algorithm; then, we can state by Th. 4.2 the global convergence for a general algorithm **A = CB** in the nonlinear case.

Theorem 4.2. *Let X be a nonempty closed set in $\mathbb{R}^n$, and let $\Omega \subseteq X$ be a nonempty solution set. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuous function, and consider the point-to-set map $C : X \to X$ satisfying the following property: given $\mathbf{x} \in X$, then $f(\mathbf{y}) \leq f(\mathbf{x})$ for $\mathbf{y} \in \mathbf{C}(\mathbf{x})$. Let $\mathbf{B} : X \to X$ be a point-to-set map that is closed over the complement of Ω and that satisfies a $f(\mathbf{y}) < f(\mathbf{x})$ for each $\mathbf{y} \in \mathbf{B}(\mathbf{x})$ if $\mathbf{x} \notin \Omega$.*

*Now consider the algorithm defined by the composite map **A = CB**. Given $\mathbf{x}_0 \in X$, suppose that the sequence $\{\mathbf{x}_k\}$ is generated as follows: If $\{\mathbf{x}_k\} \in \Omega$, then stop; otherwise, let $\mathbf{x}_{k+1} \in \mathbf{A}(\mathbf{x}_k)$, replace k by $k + 1$, and repeat. Suppose that the set $\mathcal{L} = \{\mathbf{x} : f(\mathbf{x}) \leq f(\mathbf{x}_0)\}$ is compact. Then either the algorithm stops in a finite number of steps with a point in Ω or all accumulation points of $\{\mathbf{x}_k\}$ belong to Ω.*

By putting together Th. 4.1 and Th. 4.2, we can prove that the BFGS algorithm, if we consider it as a composite map **A=CB**, converges to a point in the solution set Ω if the following assumptions hold:

1. **B** is closed at points not in Ω;
2. if $\mathbf{y} \in \mathbf{B}(\mathbf{x})$, then $f(\mathbf{y}) < f(\mathbf{x})$ for $\mathbf{x} \notin \Omega$;
3. if $\mathbf{z} \in \mathbf{C}(\mathbf{y})$, then $f(\mathbf{z}) \leq f(\mathbf{y})$;
4. $\mathcal{L} = \{\mathbf{x} : f(\mathbf{x}) \leq (f\mathbf{x}_0)\}$ is compact and $\mathbf{x}_0$ is the starting point solution.

For the BFGS the map $\mathbf{y} \in \mathbf{B}(\mathbf{x})$ corresponds to minimizing f starting from x along a direction $\mathbf{d} = -\mathbf{D}\nabla f(\mathbf{f})$, where $\mathbf{D}_k = \mathbf{H}_k$ at each iteration k. As we have already said in Sec. 4.1 and 4.2, $\mathbf{H}_k$ is definite positive by definition and $\mathbf{d}$ is a descent direction, since $\mathbf{d}^T \nabla f(\mathbf{x}) = -\nabla f(\mathbf{x})^T D \nabla f(\mathbf{x}) < 0$. Furthermore, starting from a $y \in \mathbf{B}(\mathbf{x})$, the map **C** is defined by minimizing the function f along the directions specified by the particular algorithms (the direction is the same used in the **B** map). Therefore, assumptions 2 and 3 are satisfied. Moreover, Bazaraa shows that, with an exact line search, **B** is closed. So, if we consider true all these assumptions, the BFGS will converge to a stationary point with zero gradient in the set Ω.

This general analysis cannot be achieved in our specific case because of the exact line search. Anyway, there are other analyses to do when a practical line search, like the one that satisfied the strong Wolfe conditions, is taken into consideration.

Practical line search and convex case

The global convergence analysis of the BFGS for solving convex optimization problems, with a practical line search, under some assumptions has been well established [25, 26, 27]. In order to prove the global convergent property for the convex case we need to recall the result of the Zoutendijk's condtion 3.4:

$$\sum_{k>0} \cos^2 \theta_k \left\|\mathbf{g}_k\right\|^2 \; < \; 0 \tag{43}$$

This result implies that

$$\cos^2 \theta_k \left\|\mathbf{g}_k\right\|^2 \to 0. \tag{44}$$

This limit can be used to derive the global converge results for the algorithm that make use of a practical line search. In fact, to prove it, it is sufficient to show that the angle θ_k between the vectors $\mathbf{d}_k$ and $\mathbf{g}_k$ does not approach $\frac{\pi}{2}$ too rapidly or that θ_k is bounded away from $90°$, so there is a positive constant δ such that

$$\cos \theta_k \geq \delta > 0, \quad \text{for all } k \tag{45}$$

To ensure Eq. 45 when a quasi-Newton method is taken into consideration, we need that matrices $\mathbf{B}_k$ (or it's inverse $\mathbf{H}_k$) are positive definite with a uniformly bounded condition number such that

$$\left\|\mathbf{B}_k\right\| \left\|\mathbf{B}_k^T\right\| \leq M, \quad \text{for all } k.$$

From Eq. 25 can be shown that

$$\cos \theta_k \geq 1/M.$$

By combining this result with Eq. 44 we can find that

$$\lim_{k\to\infty} \left\|\mathbf{g}_k\right\| = 0.$$

or the weaker result

$$\liminf_{k\to\infty} \left\|\mathbf{g}_k\right\| = 0.$$

So, let's assuming the following assumptions on a general objective function f,

Assumption 4.3. *The objective function f is twice continuously differentiable.*

Assumption 4.4. *The level set $\mathcal{L} = \{\mathbf{x} \mid f(\mathbf{x}) \leq f(\mathbf{x}_0)\}$ is convex, and there exist positive constants m and M such that*

$$m \left\|\mathbf{z}\right\|^2 \leq \mathbf{z}^\mathsf{T} \nabla^2 f(\mathbf{x})\, \mathbf{z} \leq M \left\|\mathbf{z}\right\|^2$$

for all $\mathbf{z} \in \mathcal{R}^n$ and $x \in \mathcal{L}$.

The following theorem states the global convergence of the BFGS for smooth strongly convex functions.

Theorem 4.5. *Let $\mathbf{B}_0$ be any symmetric positive definite initial matrix, and let x_0 be a starting point for which Assumption 4.3 and Assumption 4.4 are satisfied. Then the sequence x_k generated by BFGS method converges to the minimizer x^* of f.*

So, the same global convergence result can be extended for the L-BFGS.

Practical line search and nonlinear nonconvex case

In the case where f is not smooth strongly convex, the BFGS still generate descent directions if Wolfe line search is used. However, in this case, an example given by Dai [28] shows that the BFGS method is not necessary globally convergent. Someone, like Li and Fukushima [29], were able to prove the global convergence only with some modifications of the BFGS formula but we will not treat this case. The difficulty of determining if the BFGS is globally convergent, in the nonlinear case, is given by the fact that it is not easy to determine the bound of eigenvalues and therefore compute the condition number.

To conclude, if we consider now our objective function $L(\mathbf{w})$, Th. 4.2 cannot be applied because we are assuming a strong Wolfe line search and the analysis is made with an exact line search. Th. 4.5 don't hold since A. 4.4 requires convexity on the level set and we are considering a nonlinear objective function, even more, it requires that $\nabla^2 L(\mathbf{w})$ is positive definite and bounded on the level set, and this cannot be said in advance. From a theoretical point of view, we cannot say that the BFGS will certainly converge to a stationary point from a starting point. This undesirable result is true for any $L(\mathbf{w})$ with $0 \leq \lambda \leq 1$.

But, if the algorithm at some point enters and stays in a compact region where the function is smooth strongly convex, then it converges because A. 4.3 and A. 4.4 hold, then the Th. 4.5 can be applied. In fact, if the function is smooth and its gradient L-continuous, in that compact region, we have that the largest eigenvalue of the Hessian is upper bounded and, moreover, if it is strongly convex the smallest eigenvalue of the Hessian is bounded below, so we will have a uniformly bounded condition number which guarantees convergence through Th. 3.4 and Th. 4.5 will also hold. Furthermore, if the algorithm approaches a strinct local minimizer $\mathbf{w}^*$, where the Hessian is strictly positive definite then it will converge at this point.

Note that Th. 4.5 also requires $\mathbf{B}_0$ to be symmetric and positive definite in order to have a descent direction at every iteration with the BFGS or L-BFGS formula, which is satisfied because we have chosen its inverse $\mathbf{H}_0$ as Eq. 41 that is positive definite (so $\mathbf{B}_0 = \mathbf{H}_0^{-1}$ will be too), this allows keeping $\mathbf{H}_k$ positive definite at every iteration when the BFGS or L-BFGS formula are used as we have seen at the end of the Sec. 4.2.

4.4 Convergence rate analysis

After the convergence analysis, we see the convergence rate of the BFGS method with a practical line search. The results that will be shown for the BFGS method also apply for its limited-memory version.

From Th. 4.5 we have seen that, if the function is convex and the Hessian matrix is positive definite with bound eigenvalues, the BFGS is globally convergent. Moreover, an extension analysis of the Th. 4.5 tell us that the convergence of the iterates is linear. In particular, the sequence $\|\mathbf{x}_k - \mathbf{x}^*\|$ converges to zero rapidly enough that

$$\sum_{k=1}^{\infty} \|\mathbf{x}_k - \mathbf{x}^*\| < \infty, \tag{46}$$

where $\mathbf{w}^*$ is a minimizer and if (46) holds, can be proved that the rate of convergence is actually superlinear.

To extend the convergence rate analysis also to general nonlinear functions, we need to make the following assumption.

Assumption 4.6. *The Hessian matrix $\mathbf{G}$ is Lipschitz continuous at $\mathbf{x}^*$, that is,*

$$\|\mathbf{G}(\mathbf{x}) - \mathbf{G}(\mathbf{x}^*)\| \leq C \|\mathbf{x} - \mathbf{x}^*\|, \tag{47}$$

for all $\mathbf{x}$ *near* $\mathbf{x}^*$*, where* C *is a positive constant.*

Moreover, the following theorem by Dennis and More [30] will be an important tool for this section.

Theorem 4.7. *Suppose* f *is twice continuously differentiable. Consider the iteration* $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$*, where* $\mathbf{d}_k$ *is a descent direction and* α_k *satisfies the strong Wolfe conditions with* $c_1 \leq \frac{1}{2}$*. If the sequence* $\{\mathbf{x}_k\}$ *converges to a point* $\mathbf{x}^*$ *such that* $\nabla f(\mathbf{x}^*) = 0$ *and* $\nabla^2 f(\mathbf{x}^*)$ *is positive definite, and i the search direction satisfies*

$$\lim_{k \to \infty} \frac{\|\nabla f_k - \nabla^2 f_k \, \mathbf{d}_k\|}{\|\mathbf{d}_k\|} = 0, \tag{48}$$

then

- *the step length* $\alpha_k = 1$ *is admissible for all* $k > k_0$*; and*
- *if* $\alpha_k = 1$ *for all* $k > k_0$*,* $\{\mathbf{x}_k\}$ *converges to* $\mathbf{x}^*$ *superlinearly.*

Since $\mathbf{d}_k$ is a quasi-Newton search direction in the form of $\mathbf{d}_k = -\mathbf{B}_k^{-1} \, \mathbf{g}_k$, then (48) is equivalent to

$$\lim_{k \to \infty} \frac{\|(\mathbf{B}_k - \nabla^2 f(\mathbf{x}^*)) \, \mathbf{d}_k\|}{\|\mathbf{d}_k\|} = 0. \tag{49}$$

Condition (49), which ensures that a superlinear convergence rate can be obtained if $\mathbf{B}_k$ becomes increasingly accurate approximations to $\nabla^2 f(\mathbf{x}^*)$ *along the search direction* $\mathbf{d}_k$, is *both necessary and sufficient* for the superlinear convergence of quasi-Newton methods in the convex case.

Finally, by the following theorem, the superlinear convergence for a general nonlinear function is proven.

Theorem 4.8. *Suppose that* f *is twice continuously differentiable and that the iterates generated by BFGS algorithm converge to a minimizer* $\mathbf{x}^*$ *at which Assumption 4.6 holds. Suppose also that (46) holds. Then* $\mathbf{x}_k$ *converges to* $\mathbf{x}^*$ *at a superlinear rate.*

Regarding our loss function $L(\mathbf{w})$, if we consider the case where the objective function has a regularization term $0 < \lambda \leq 1$ can easily be shown that by computing the Hessian, deriving the gradient, we can state that A. 4.6 hold (not true for $\lambda = 0$). However, as well explained in the conclusion of the global convergence analysis, only in the case where the algorithm enters and stays in a compact region where the function is smooth strongly convex Th. 4.5 hold and as a result Eq. 46 will be true. Only under those considerations, we can say that the algorithm, for our loss function, is superlinear convergent. Note that the compact region ensures that A. 4.6 hold even if $\lambda = 0$ allowing to reach superlinear convergent in this case too.

It would be interesting, during the experimental phase, to compute the Hessian. In this way, we will have the confirmation that the superlinear convergence holds only when the matrix is positive definite and with a uniformly bounded condition number, but this analysis will not be discussed.

5. Experiments

Now, we can go further and see some experiments in detail. We decided to derive three objective functions from the three MONK problems [31] and study the convergence at their local minimum with the methods described above. Let's see in details our experimental setup.

Objective functions:
Built through the following procedure:

- A MONK's training set (1,2,3) is taken, since the features are categorical, the *one-hot encoding* technique has been applied on them, obtaining 17 binary features.
- A fixed network topology: 17 (input units) - 4 (hidden units) - 1 (output unit). The sigmoid activation function has been used on both hidden and output layers.
- The objective function is in the form of the loss function seen in Eq.8 which consists of the MSE plus a L2 regularization term. The L2 term allows to keep the objective function continuous and differentiable (property is already given by the sigmoid activation used in the neural network). Moreover, it makes the level set compact, the gradient and the hessian L-continuous (see Sec. 2.4). Here the objective function for each Monk dataset i composed by the error and the regularization term with $\lambda = 10^{-4}$,

$$L_i(w) = E_i(\mathbf{w}) + 10^{-4} \|\mathbf{w}\|^2, \quad i \in (1, 2, 3). \tag{50}$$

We will refer to them as $Monk1$, $Monk2$ and $Monk3$. We have considered only objective functions with a regularization term to take advantage of the global and local convergence properties given for the NCG methods. These three objective functions may not have a global minimum equal to 0 due to the regularization term. Moreover, they are not convex, and each method could stop in different local minima.

Starting point and $\mathbf{w}_0$ policy selection:

MONK problems need very small weights to converge correctly, so we decided to use random weights extracted from an *uniform distribution* in the interval $(-a, +a)$ on each layer. The parameter a is chosen according to Tab. 5.1.

Weight Initialization		
f	hidden layer a	output layer a
$Monk1$	0.003	0.003
$Monk2$	$1 \,/\, \sqrt{17}$	$1 \,/\, \sqrt{4}$
$Monk2$	0.003	0.003

Table 5.1: Weight initialization for each objective function by considering individual layers of the the NN.

In order to generate the same initialization point $\mathbf{w}_0$, a *seed* has been plugged in the NumPy function used to sample weights from a specific *uniform distribution*. So a seed combined with a given distribution generates a precise starting point.

Optimization method details:

All the optimization method described in theory have been taken in analysis. Then, we will study all objective functions using the NCG (FR, PR, HS and beta+ variation) and L-BFGS methods. In order to compare them, on the same objective function, we decided to use those thresholds as stop criteria: $\|\mathbf{g}\|_k < ng\_eps$, $L(\mathbf{w}_k) < l\_eps$ and *iteration* $<$ *max_iter*. All the method use the same line search illustrated in Algo 2.

This section will be divided into two parts. First, in Sec 5.1, we will try to observe the methods' behaviors in order to check the correctness of our implementation and verify that the results described in the theoretical part, in terms of global convergence and local convergence, can be observed from a practical point of view. Instead, in Sec 5.2 we will compare the efficiency of all the methods from the same starting point. All our experiments were performed on a Intel CPU with 2 cores and 4 thread at 2.6GHz and with OpenBLAS as optimized math routine for Numpy.

5.1 Methods behaviors and validation

To inspect the methods' behaviors and check the correctness of our implementation, we decided to study each algorithm on different configurations. We check if some of the behaviors proved in Sec. 3 and Sec. 4 are observable. For the purposes of this section only a few interesting cases have been reported and all the experiments, that can be found here, use the stop criteria shown in Tab. 5.2. Each results table specify the seed for that particular experiment, which is related to a specif starting point $\mathbf{w}_0$, moreover, this allows to obtain reproducible results. Regarding the line search, a percentage value has been defined, called ls_hit_rate. This parameter indicates the percentage of success of the line search during the whole optimization process, where, as success, we mean that the line search was able to return an alpha value that satisfies the strong Wolfe conditions. Instead, the rate of convergence will be studied by plotting the $(f_k - f^*)$ on a logarithm scale. Where f_k is the loss computed as Eq. 50 (MSE plus the regularization term), and the f^* is the optimal value founded by a specific method.

Stop Criteria			
Optimizer	**ng_eps**	**l_eps**	**max_iter**
NCG - FR	1e-5	1e-5	20000
NCG - PR/PR+	1e-5	1e-5	1000
NCG - HS/HS+	1e-7	1e-7	1000
L-BFGS	3e-5	3e-5	1000

Table 5.2: Stop criteria for the various methods, where ng_eps, l_eps, and max_iter are respectively the thresholds for the gradient norm, the loss function, and the number of iterations.

To simplify the discussion we insert here the table with the statistics of the line search relative to all the experiments effectuated in this section. As can be see from Tab. 5.3, the *Ls_Max_Iter* parameter is always equal to 100. We have chosen this value during the experimental phase because it allowed us to obtain a line search hit rate almost always equal to 100%, as it happens in all experiments in this section reported in Tab. 5.3.

Line search statistics				
#	**Ls Max Iter.**	**Ls Iter.**	**Ls Hit Rate**	**Ls Time (s)**
1	100	108181	99%	76.69
2	100	728	100%	0.92
3	100	995	100%	1.31
4	100	747	100%	1.11
5	100	595	100%	0.67
6	100	1591	100%	2.35
7	100	950	100%	1.23
8	100	1520	100%	2.27
9	100	1785	100%	2.97
10	100	1	100%	0.06
11	100	0	100%	0.1

Table 5.3: Experiments Line search details. *Ls Iter.* are the iterations of the line search, *Ls Hit Rate* is the success percentage of the line search, and *Ls Time* is the time spent in the line search in second.

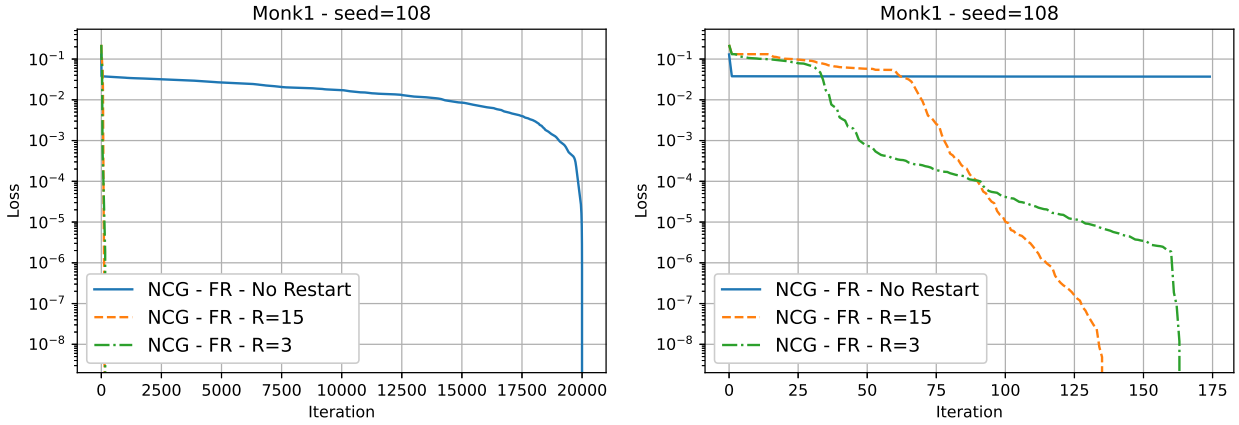
5.1.1 FR METHOD

First, we have considered the NCG FR method and, as said from the theory, to ensure that a descent direction is always taken if a strong Wolfe line search is used, the c_1 and c_2 must be set as follow: $0 < c_1 < c_2 < \frac{1}{2}$.

FR analysis results										
#	f	**Seed**	$\mathbf{c_1}$	$\mathbf{c_2}$	**restart**	**Ls Max Iter.**	f^*	$\|\mathbf{g_k}\|$	**Conv. Iter.**	**Time (s)**
1	$Monk1$	108	1e-4	0.1	-	100	1.20e-1	3.93e-2	20000	143.97
2	$Monk1$	108	1e-4	0.1	15	100	2.63e-2	9.16e-6	137	1.42
3	$Monk1$	108	1e-4	0.1	3	100	2.63e-2	8.07e-6	165	2.10

Table 5.4: Results of the experiments with the NCG - FR.

By considering the *Monk1* objective function and the starting point generated by seed = 108, we immediately observed that, if the FR method is used without the restart parameter it may produce very small displacements. This behavior is observed in experiment 1 in Tab. 5.4, where the optimization process stops only after 20000 iterations. On the contrary, experiments 2 and 3, those with restart, converge in less than 200 iterations. Also, it is interesting to observe that by using the restart parameter, with the same starting point, the FR has been able to converge to lower values (2.63e-2 instead of 1.20e-1) faster. Regarding the convergence rate, all the methods seem to have a linear trend, with a few superlinear peaks (see Fig. 5.1. These results fully reflect the theoretical analysis of the FR method made in Sec 3. The last thing to highlight is that the two experiments with the restart (experiments 2 and 3 in Tab 5.4) reach a local minima (that could be different) of the same quality ($f^* = 2.63\text{e}{-2}$), and the method with restart equal to 15 is more efficient than the one with restart equal to 3 (137 conv. iter. istead of 165).



(a) FR with no restart, FR with R=15, and FR with R=3.

(b) Zoomed version of Fig.5.1a.

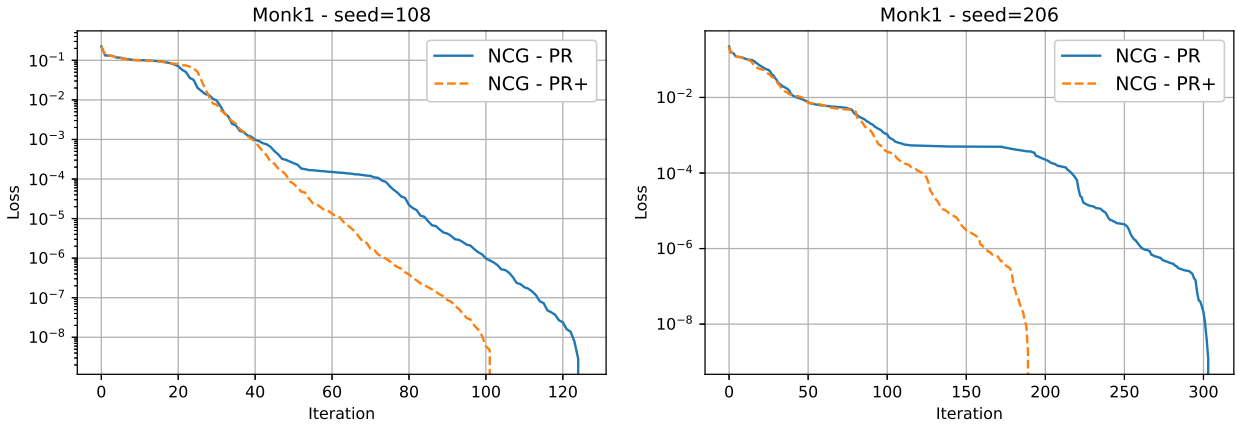
Figure 5.1: Loss with respect to the minima. NCG - FR applied to $Monk1$, with three different restarts. The details of these experiments (No restart - #1, R=15 - #2, R=3 - #3) are in Tab.5.4 and in Tab.5.3.

5.1.2 PR AND PR+ METHODS

PR and PR+ analysis results										
#	f	**Seed**	**Method**	$\mathbf{c_1}$	$\mathbf{c_2}$	**Ls Max Iter.**	f^*	$\|\mathbf{g_k}\|$	**Conv. Iter.**	**Time (s)**
4	$Monk1$	108	NCG PR	1e-4	0.1	100	2.63e-2	6.98e-6	126	1.83
5	$Monk1$	108	NCG PR+	1e-4	0.1	100	2.63e-2	9.19e-6	103	1.09
6	$Monk1$	206	NCG PR	1e-4	0.3	100	2.64e-2	5.39e-6	305	3.93
7	$Monk1$	206	NCG PR+	1e-4	0.3	100	2.64e-2	8.40e-6	191	2.08

Table 5.5: Results of the experiments with the NCG - PR/PR+.

Even if this section it's not dedicated to the study of methods efficiency from the same starting point, for the PR and PR+ we decided to analyze them with the same seed used for the FR method in order to check if these two methods behave better than FR. In fact, experiments 4 and 5, shown in Tab. 5.5, reveal the efficiency of the PR/PR+ formulas, which reach a minimum of the same quality of the one found by the FR with restart, in fewer iterations. Moreover, from both figures Fig. 5.2a and Fig. 5.2b, we can see that the PR+ approaches the same minimum of PR faster. This is very interesting because it points out that the presence of beta negative in some iterations may affect the convergence efficiency of the method.



(a) PR and PR+, respectively experiments #4 and #5.

(b) PR and PR+, respectively experiments #6 and #7.

Figure 5.2: Loss with respect to the minima. NCG - PR and NCG - PR+ applied to $Monk1$. Details are in Tab.5.5 and in Tab.5.3.

Regarding the convergencerate, all the methods seem to have a linear trend, with a few superlinear peaks (see Fig. 5.2).

In addition, we have verified that the use of the restart parameter was not very useful, thanks to the nature of the beta formula that performs an automatic restart when bad directions are reached, as said from the theory. For this reason no analysis has been reported when this parameter changes. Therefore, also the PR/PR+ methods behave as expected from the theoretical analysis made.

5.1.3 HS AND HS+ METHODS

HS and HS+ analysis results										
#	f	**Seed**	**Method**	$\mathbf{c_1}$	$\mathbf{c_2}$	**Ls Max Iter.**	f^*	$\|\mathbf{g_k}\|$	**Conv. Iter.**	**Time (s)**
8	$Monk3$	353	NCG HS	1e-4	0.1	100	3.53e-2	9.31e-8	281	3.58
9	$Monk3$	353	NCG HS+	1e-4	0.1	100	3.53e-2	9.06e-8	331	4.62

Table 5.6: Results of the experiments with the NCG - HS/HS+.

General tests have shown that the behavior of HS/HS+ behave quite similar to the PR/PR+. For this reason we are not going to examine it in details and we leave the comparison analysis from the same starting point in the next section. We decided to present here an experiment on the $Monk3$ objective function, with the $\mathbf{w}_0$ generated by seed 353 and compare the HS and the HS+ methods. We can observe that the methods reach a minimum of the same quality ($f^* = 3.53\text{e}{-2}$), but HS, in this case, converges in fewer iterations (see Fig. 5.3). As expected, the convergence rate is linear, superlinear in very few iterations. This concludes the analysis and validation of NCG methods and shows that beta negative is not always a bad thing in terms of

efficiency. Attention must be paid to the choice of method according to the objective function that is taken in the analysis.

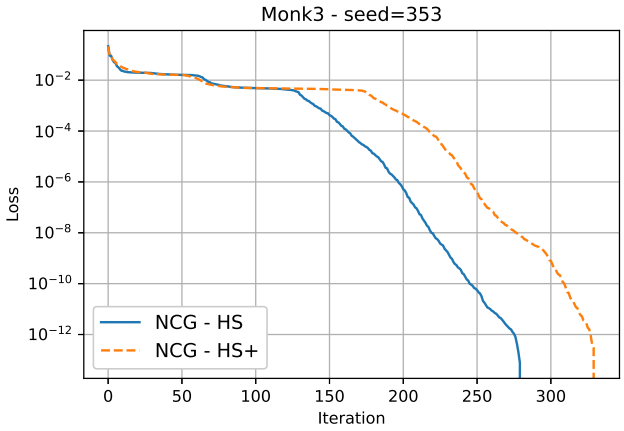


Figure 5.3: Loss with respect to the minima. NCG - PR and NCG - PR+, respectively experiments #8 and #9, applied to $Monk3$. Details are in Tab.5.6 and in Tab.5.3.

5.1.4 L-BFGS METHOD

L-BFGS analysis results										
#	f	**Seed**	**m**	$\mathbf{c_1}$	$\mathbf{c_2}$	**Ls Max Iter.**	f^*	$\|\mathbf{g_k}\|$	**Conv. Iter.**	**Time (s)**
10	$Monk2$	987	3	1e-4	0.9	100	2.82e-2	2.80e-5	123	0.32
11	$Monk2$	987	30	1e-4	0.9	100	2.82e-2	2.91e-5	74	0.58

Table 5.7: Results of the experiments with the L-BFGS.

From the theory in Sec. 4.3, we saw that the L-BFGS does not converge with certainty to a stationary point in our case. Indeed, this property strongly depends on the starting point. However, if the algorithm at some point enters and stays in a compact region where the function is smooth strongly convex, then it converges. Moreover, during the optimization process, we have checked that the curvature condition in Eq. 36 is epsilon respected at each iteration: $\mathbf{s}_k^T \mathbf{y}_k > 1\text{e}{-8}$. Indeed, a bad starting point and the wrong parameters would bring to a negative curvature condition, and at the end of the process. This behavior was widely seen during our experiments. We have also noticed that the method is more unstable numerically in the convergence process then the NCG. In fact, when it reach the gradient norm of the order of $1\text{e}{-6}$ the curvature condition becomes negative.

We reported in Tab. 5.7 only two experiment (10 and 11) with different value of m parameter, and from Fig. 5.4 we can see that the algorithm is able to converges in these configurations. The convergence rate for both the experiments is only linear, with some superlinear peaks. Regarding the line search, as we expect from the theory, an $\alpha = 1$ is almost always accepted. Indeed, how we can see from Tab. 5.3, the L-BFGS performs very few line search iterations compared to other methods, and this is a confirmation of the above. Finally, from Fig. 5.4 we can see how the convergence speed of the algorithm varies as m varies. Indeed, the experiment wit $m = 30$ converges faster than the one with $m = 3$, highlighting that, if more curvature information is stored, fewer iterations are needed for the algorithm to reach a minimum of the same quality ($f^* = 2.82\text{e}{-2}$).

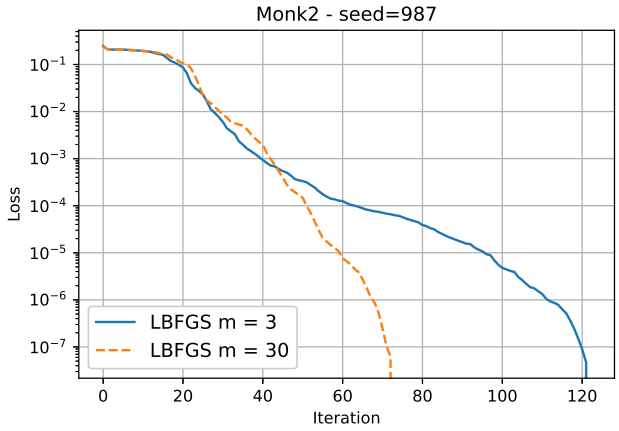


Figure 5.4: Loss with respect to the minima. L-BFGS experiments #10 (=3) and #11 (m=30), on $Monk2$. Details in Tab.5.7 and in Tab.5.3.

5.2 Methods comparisons

From the analysis carried out in the previous section, we decided to exclude the PR and HS methods from the comparison, because with our objective functions and the starting points chosen in this comparisons, we never achieved better results than their modified counterparts. In addition, this allows us to better visualize and compare methods, being less. To study the efficiency of the various methods we have selected a starting point for each objective function, through a seed. Then we have chosen the tolerances for all methods, so that we can compare the behavior towards a certain loss value. After that we compared the result of the optimization process for the following methods: FR with restart, PR+, HS+ and L-BFGS. The following table contains the seed and the tolerance used for each objective function. $ls\_\max\_iter = 100$ has been used for line search given the excellent results of the previous studies.

Stop criteria and seed				
f	**seed**	**ng_eps**	**l_eps**	**max_iter**
$Monk1$	6	3e−5	1e−6	1000
$Monk2$	189	3e−5	1e−6	1000
$Monk3$	783	10e−6	1e−6	1000

Table 5.8: Stop criteria and seeds used for the methods comparison.

As regards the analysis of the convergence rate, we used both the logarithm scale plots, as explained in the previous section, and also those related to the rate of convergence p estimation. A sequence $\{f_k\}$ that converges to f^* is said to have rate of convergence p if

$$\lim_{k\to\text{inf}} \frac{|f_{k+1} - f^*|}{|f_k - f^*|^p} = R > 0,$$

and from that p can be estimate with the following formula,

$$p = \lim_{k\to\text{inf}} \frac{\log|f_{k+1} - f^*| - \log R}{\log|f_k - f^*|^p} \approx \lim_{k\to\text{inf}} \frac{\log|f_{k+1} - f^*|}{\log|f_k - f^*|^p}$$

The convergence rate is linear if $p = 1$, $p > 1$ is superlinear and $p = 2$ for a quadratic rate.

5.2.1 MONK1

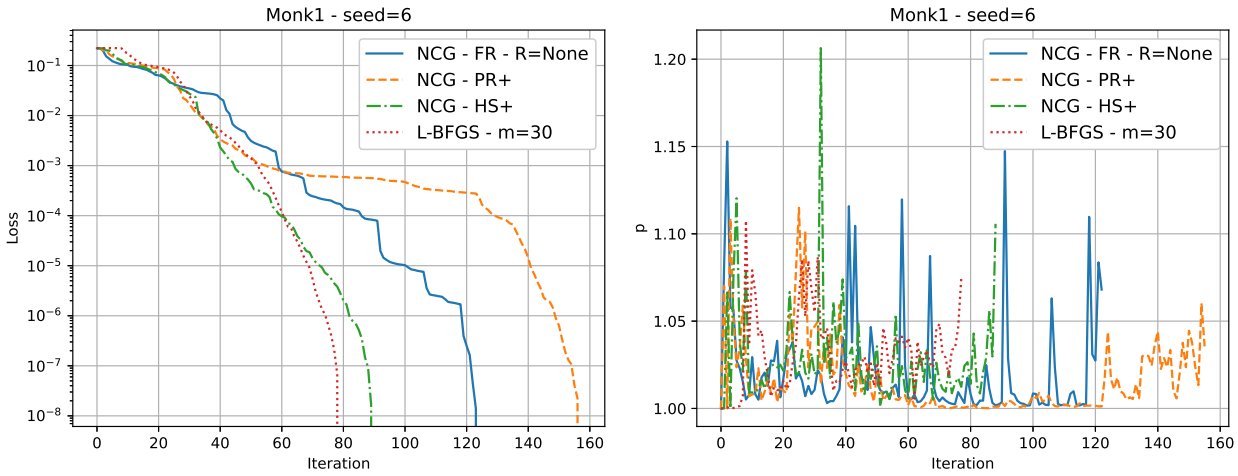
All the methods stop because they have reached epsilon optimality on the gradient norm. Tab. 5.9 shows the result of the methods for the $Monk1$ objective function. All the algorithms converge to a minimum of the same quality (the values differ at most of 0.22e-2), but the L-BFGS reaches the optimal value in fewer iterations than the others. In this case, the L-BFGS is the most efficient because it has the lowest f^*, and it is also the fastest (needs only 0.26s to reach the stop condition). The reason is related to the number of iterations carried out in the line search method. Indeed, as can be seen from Tab 5.10, the L-BFGS needs only 20 iterations, because $\alpha = 1$ is almost always accepted (fewer iterations than those of other methods, for example, HS+ requires 507). The convergence rate of all the NCG methods is linear, with some superlinear peaks, instead the L-BFGS is almost always superlinear (see Fig. 5.5b).

Best results on f: $Monk1$									
f	**Optimizer**	**c1**	**c2**	**restart**	**m**	f^*	$\|g_k\|$	**Conv. Iter.**	**Time (s)**
$Monk1$	NCG FR	1e-4	0.3	3	-	2.92e-2	2.24e-5	125	0.94
$Monk1$	NCG PR+	1e-4	0.4	-	-	2.84e-2	2.80e-5	158	0.96
$Monk1$	NCG HS+	1e-4	0.6	-	-	2.92e-2	2.71e-5	91	0.75
$Monk1$	L-BFGS	1e-4	0.9	-	30	2.70e-2	1.98e-5	80	0.26

Table 5.9: Methods comparisons results with the $Monk1$ objective function and the $\mathbf{w}_0$ generated by seed 6.

Line search statistics of Tab. 5.9					
f	**Optimizer**	**Ls Max Iter.**	**Ls Iter.**	**Ls Hit Rate**	**Ls Time (s)**
$Monk1$	NCG FR	100	678	100%	0.73
$Monk1$	NCG PR+	100	886	100%	0.74
$Monk1$	NCG HS+	100	507	100%	0.56
$Monk1$	L-BFGS	100	20	100%	0.05

Table 5.10: Results of the line search statistics related to Tab. 5.9



(a) $Monk1$ **(b)** $Monk1$ p estimation.

Figure 5.5: Comparisons between methods considering the $Monk1$ objective function and the same starting point generated by seed 6.

5.2.2 MONK2

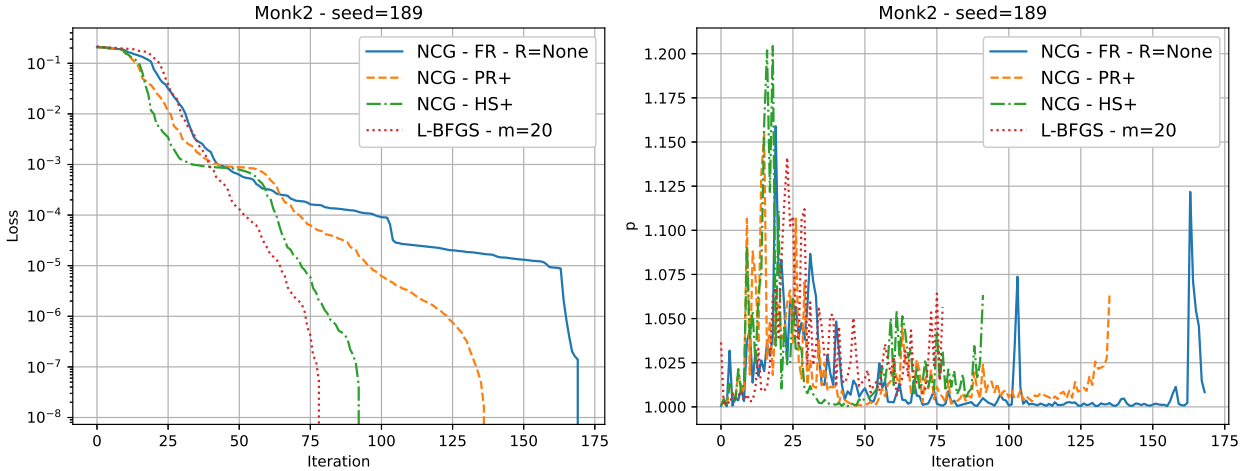
As for $Monk1$, here the same occurs, epsilon optimality is reached on the gradient norm and all the methods converge to a minimum of the same quality (the values differ at most of 0.02e-2), but the L-BFGS reaches the optimal value in fewer iterations than the others (see Tab. 5.11). Moreover, the computational time of the L-BFGS is the smallest. The convergence rate of all the NCG methods is linear, with some superlinear peaks, instead the L-BFGS is almost always superlinear (see Fig. 5.6b)

Best results on f: $Monk2$									
f	**Optimizer**	**c1**	**c2**	**restart**	**m**	f^*	$\|\mathbf{g}_k\|$	**Conv. Iter.**	**Time (s)**
$Monk2$	NCG FR	1e-4	0.3	3	-	2.82e-2	2.89e-5	171	1.04
$Monk2$	NCG PR+	1e-4	0.1	-	-	2.82e-2	2.60e-5	138	0.91
$Monk2$	NCG HS+	1e-4	0.1	-	-	2.82e-2	2.45e-5	94	0.6
$Monk2$	L-BFGS	1e-4	0.9	-	20	2.84e-2	1.62e-5	75	0.3

Table 5.11: Method comparison results with the $Monk2$ obj. function and the $\mathbf{w}_0$ generated by seed 189.

Line search statistics of Tab. 5.11					
f	**Optimizer**	**Ls Max Iter.**	**Ls Iter.**	**Ls Hit Rate**	**Ls Time (s)**
$Monk2$	NCG FR	100	889	100%	0.8
$Monk2$	NCG PR+	100	834	100%	0.73
$Monk2$	NCG HS+	100	555	100%	0.48
$Monk2$	L-BFGS	100	1	100%	0.05

Table 5.12: Results of the line search statistics related to Tab. 5.11



(a) $Monk2$ **(b)** $Monk2$ p estimation.

Figure 5.6: Comparisons between methods considering the $Monk2$ objective function and the same starting point generated by seed 189.

5.2.3 MONK3

Finally, for $Monk3$ objective function, the optimal minimum reached by the methods is of the same quality (the values differ at most of 0.83e-2, see Tab. 5.13). Moreover, L-BFGS reaches the lowest f^* value in fewer

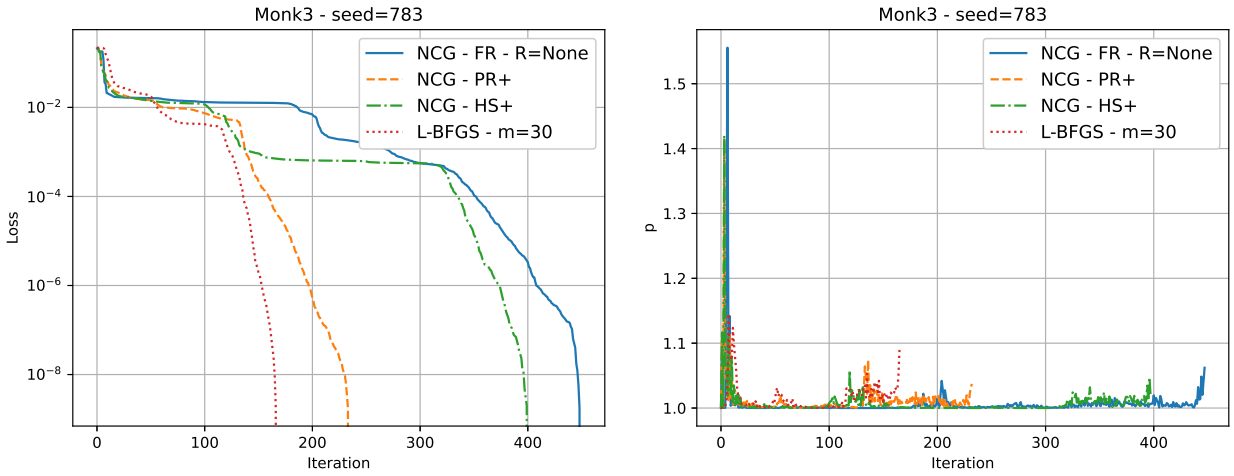
iterations than the others methods. In addition, the computational time of the L-BFGS is the smallest. The convergence rate of all the methods is linear, with some superlinear peaks (see Fig. 5.7b).

Best results on f: $Monk3$									
f	**Optimizer**	**c1**	**c2**	**restart**	**m**	f^*	$\|g_k\|$	**Conv. Iter.**	**Time (s)**
$Monk3$	NCG FR	1e-4	0.1	6	-	3.84e-2	6.51e-6	450	2.22
$Monk3$	NCG PR+	1e-4	0.1	-	-	3.58e-2	9.43e-6	235	1.25
$Monk3$	NCG HS+	1e-4	0.3	-	-	3.84e-2	7.16e-6	401	1.94
$Monk3$	L-BFGS	1e-4	0.9	-	30	3.01e-2	5.32e-6	115	0.49

Table 5.13: Method comparison results with the $Monk3$ obj. function and the $\mathbf{w}_0$ generated by seed 783.

Line search statistics of Tab. 5.13					
f	**Optimizer**	**Ls Max Iter.**	**Ls Iter.**	**Ls Hit Rate**	**Ls Time (s)**
$Monk3$	NCG FR	100	2490	100%	1.74
$Monk3$	NCG PR+	100	1249	100%	0.98
$Monk3$	NCG HS+	100	2004	100%	1.47
$Monk3$	L-BFGS	100	12	100%	0.07

Table 5.14: Results of the line search statistics related to Tab. 5.13



(a) $Monk3$ **(b)** $Monk3$ p estimation.

Figure 5.7: Comparisons between methods considering the $Monk3$ objective function and the same starting point generated by seed 783.

6. Conclusions

In this work, we have presented some methods that make use of more information than classic algorithms such as the nonlinear conjugate gradient and the L-BFGS. Indeed, as a general result, we have noticed that L-BFGS can obtain equal or better results than all the other methods presented here, thanks to the fact that more information is added to generate a descent direction.

References

[1] Boris Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr Computational Mathematics and Mathematical Physics*, 4:1–17, 12 1964.

[2] Y. E. NESTEROV. A method for solving the convex programming problem with convergence rate o($1/k^2$). *Dokl. Akad. Nauk SSSR*, 269:543–547, 1983.

[3] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, abs/1511.00561, 2015.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[5] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[6] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. *30th International Conference on Machine Learning, ICML 2013*, pages 1139–1147, 01 2013.

[7] Simon S. Haykin. *Neural networks and learning machines*. Pearson Education, Upper Saddle River, NJ, third edition, 2009.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, page 2012.

[9] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

[10] Stiefel Hestenes. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 12 1952.

[11] Reeves Fletcher. Function minimization by conjugate gradients. *Computer Journal*, pages 149–154, 7 1964.

[12] Stephen J. Wright Jorge Nocedal. *Numerical Optimization*. Springer Series in Operations Research. Springer, 1999.

[13] M. J. D. Powell. Restart procedures for the conjugate gradient method. *Mathematical Programming*, 12(1):241–254, Dec 1977.

[14] M. J. D. Powell. Nonconvex minimization calculations and the conjugate gradient method. In David F. Griffiths, editor, *Numerical Analysis*, pages 122–141, Berlin, Heidelberg, 1984. Springer Berlin Heidelberg.

[15] Jean Charles Gilbert and Jorge Nocedal. Global convergence properties of conjugate gradient methods for optimization. *SIAM Journal on Optimization*, 2:21–42, 02 1992.

[16] G. Zoutendijk, Nonlinear Programming, Computational Methods, In: J. Abadie Ed., Integer, and Nonlinear Programming. *North-Holland, Amsterdam*. 1970.

[17] Mehiddin Al-Baali. Descent property and global convergence of the fletcher–reeves method with inexact line search. *IMA Journal of Numerical Analysis*, 5, 01 1985.

[18] H. Crowder and P. Wolfe. Linear convergence of the conjugate gradient method. *IBM Journal of Research and Development*, 16(4):431–433, 1972.

[19] M. J. Powell. Some convergence properties of the conjugate gradient method. *Math. Program.*, 11(1):42–49, December 1976.

[20] Arthur I. Cohen. Rate of convergence of several conjugate gradient algorithms. *SIAM Journal on Numerical Analysis*, 9(2):248–259, 1972.

[21] W C Davidon. Variable metric method for minimization. 5 1959.

[22] R. Fletcher and M. J. D. Powell. A Rapidly Convergent Descent Method for Minimization. *The Computer Journal*, 6(2):163–168, 08 1963.

[23] Mokhtar S. Bazaraa. *Nonlinear Programming: Theory and Algorithms*. Wiley Publishing, 3rd edition, 2013.

[24] Willard I. Zangwill. Convergence conditions for nonlinear programming algorithms. *Management Science*, 16(1):1–13, 1969.

[25] Richard H. Byrd and Jorge Nocedal. A tool for the analysis of quasi-newton methods with application to unconstrained minimization. *SIAM Journal on Numerical Analysis*, 26(3):727–739, 1989.

[26] Global convergence of a class of quasi-newton methods on convex problems. *SIAM Journal on Numerical Analysis*, 24(5):1171–1190, 1987.

[27] M. Powell. Some global convergence properties of a variable metric algorithm for minimization without exact lin. 1976.

[28] Yu-Hong Dai. Convergence properties of the bfgs algoritm. *SIAM Journal on Optimization*, 13:693–701, 01 2002.

[29] Dong-Hui Li and Masao Fukushima. A modified bfgs method and its global convergence in nonconvex minimization. *Journal of Computational and Applied Mathematics*, 129(1):15 – 35, 2001. Nonlinear Programming and Variational Inequalities.

[30] John E. Dennis and Jorge J. More. Quasi-newton methods, motivation and theory. Technical report, USA, 1974.

[31] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.