
# Computational Mathematics for Learning and Data Analysis 2021/22

*Antonio Di Mauro - 599785 - a.dimauro3@studenti.unipi.it - MSc in Artificial Intelligence*
*Fabio Murgese - 583714 - f.murgese@studenti.unipi.it - MSc in Artificial Intelligence*

**ML Project 19**

June 30, 2022

**Abstract**

**(M)** is a so-called extreme learning, i.e., a neural network with one hidden layer, $y = W2(W1x)$, where the weight matrix for the hidden layer W1 is a fixed random matrix, $\sigma(\cdot)$ is an element wise activation function of your choice, and the output weight matrix W2 is chosen by solving a linear least-squares problem (with L2 regularization).

**(A1)** is an algorithm of the class of accelerated gradient methods [1], cf. also here [2], applied to (M1).

**(A2)** is a closed-form solution with the normal equations and your own implementation of Cholesky (or LDL) factorization.

# Contents

# 1 Introduction

The aim of this project is to develop an Artificial Neural Network (ANN) implementing different optimization techniques:

- regularized linear least-squares problem;
- closed-form solution with LDL factorization.

## 1.1 Artificial Neural Network

An ANN, or a Multi-layer Perceptron (MLP), is a computational model able to produce a desired output of a given specific input. The model is able to learn the mapping between input and output in the learning phase. In this phase the model's parameters are tuned using a learning algorithm.
More in details, the ANN is a fully-connected set of layers, each one having a set of weights ($\mathbf{w}$) according to the size of neurons plus a bias (b). A layer takes an input ($\mathbf{x}$), that is a data sample if it is the first layer or the output of the previous layer, and returns in output the combination between the input and the weights (**net**). The so called *net* is passed through an *activation function* ($f_\sigma$) with the aim of introducing a non-linearity. Formally, for a single layer with $\mathbf{x}$, $\mathbf{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$:

$$net = w^T x + b \tag{1}$$

$$o = f_\sigma(net) \tag{2}$$

The output of an hidden layer $l$ (Eq. 2) is the input of the successive layer's *net* (Eq. 1).

In the extreme learning setting, the ANN is composed by an hidden layer with an activation function ($f_h$), whose weights ($\mathbf{w_h}$) are randomly initialized and remained untrained, and an output layer, whose parameters ($\mathbf{w_o}$) are trained to solve the linear least-squared problem. Formally, the full equation of the "extreme-ANN":

$$\mathbf{y}(\mathbf{x}) = \mathbf{w_o^T} * (f_h(\mathbf{w_h^T x} + b_h)) + b_o \tag{3}$$

where $\mathbf{y}(\mathbf{x})$ are the predictions of the inputs $\mathbf{x}$, $\mathbf{w_h}$ and $b_h$ are the hidden fixed randomly initialized parameters, $f_h$ is the hidden activation function, $\mathbf{w_o}$ and $b_o$ are the output parameters adjusted with a learning algorithm.

## 1.2 Activation Functions

The activation function is a component of an ANN responsible for the kind of output emitted by a specific layer. According to the specific architecture of the ANN we would consider different types of activation functions for the hidden layer. Specifically for this project, we will consider the following:

- Linear: $f(x) = x$;
- Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$;
- ReLu: $f(x) = max(0, x)$.

It is not important that an activation function (e.g. ReLu) is not derivable because it is applied to a combination ($net$) of the fixed and non-trained parameters of the hidden layer. Furthermore, non-linearities are needed for the Universal Approximation Theorem to be valid [3].

## 1.3 Learning methods

We consider two learning methods in order to solve the linear least-squares problem:

- the iterative method with backpropagation;
- the closed-form/direct method.

### 1.3.1 Iterative method

Since we are in a supervised setting, our model will learn exploiting a dataset composed by a number of samples having a set of features. Each sample has a specific label or target that is the expected outcome. More formally:

$$\{(\mathbf{x_i}, \mathbf{d_i})\}_{i=1}^{n} \tag{4}$$

where $n$ is the number of samples, $d_i$ is the label associated to the $i$-th sample and it has specific values according to the task (e.g. {0,1} for binary classification).
In order to perform *learning* we must access to the error/performance of the model in a specific moment and the loss function is responsible of this. It describes the distance of the model from the objective function that we want to learn. The loss function considered for this project is the *Sum of Squared Error* (SSE) with L2 regularization (Ridge Regression) in order to solve the *linear least-squares problem*:

$$SSE = \sum_{i=1}^{n} (y(x_i) - d_i)^2 = \|y(x) - d\|_2^2 = \xi \tag{5}$$

where $y(x_i)$ is the prediction of the model for the input $x_i$ (recalling Eq. 3) and $d_i$ is the desired output of the input $x_i$.

In order to learn the model's parameters $\theta = \{\mathbf{w_o}, b_o\}$, we have to address the problem as a *minimization problem* in which we have to find the minimum of the loss function. The more the loss function become small, the better our model performs.

The *backpropagation* [4] is the technique used to minimize the loss function. From this point on when we refer to the model's parameter we consider only the weights $\mathbf{w_o}$ that includes also the bias $b_o$, so $\theta = \{\mathbf{w_o}\}$. Formally, the *backpropagation* algorithm applied to the output layer of our ANN (as defined in Eq. 3):

**Algorithm 1** Backpropagation

1: $w_o \leftarrow Initialize()$
2: compute output $y(x)$ and loss $\xi$
3: **while** not converged **do**
4: $\quad \Delta w_o = -\frac{\delta \xi}{\delta w_o}$ ▷ compute the anti-gradient
5: $\quad w_o = w_o + \alpha \Delta w_o + ...$ ▷ weights update rule
6: $\quad$ compute output $y(x)$ and loss $\xi$
7: **end while**

In the lines 2 and 6 of the Alg. 1, first it's computed the output of the model by propagating the input through the layers in a forward step and then it's evaluated the loss function (the error). Then the error is propagated back to the output layer and the weights are updated according to the correction term $\Delta w_o$. The parameter $\alpha$ (Alg. 1 line 5) is a scalar that defines the magnitude of parameter updates during gradient descent, it's commonly defined as *learning rate*, but it could also be addressed as a *step length*.

The correction applied to the weights is proportional to the partial derivative $\frac{\delta \xi}{\delta w_o}$ and is expressed by the chain rule:

$$\frac{\delta \xi}{\delta w_{ji}} = \frac{\delta \xi}{\delta o_j} * \frac{\delta o_j}{\delta net_j} * \frac{\delta net_j}{\delta w_{ji}} \tag{6}$$

where $w_{ji}$ is the j-th output neuron connected to the i-th input, $o_j$ is the output/prediction of the j-th neuron as in Eq. 2 and $net_j$ is the linear combination according to Eq. 1.

Differentiating each component of Eq. 6 we obtain:

$$\frac{\delta \xi}{\delta w_{ji}} = 2 * (o_j - d_j) * x_i \tag{7}$$

For this project we use FISTA (2.1) to compute $\Delta w_o$ (Alg. 1 line 4) and update the weights $w_o$.

### 1.3.2 Closed-form/direct method

In the language of linear algebra, the problem here is the solution of a system of equations $Ax = b$, where $A$ is the output of the hidden layer of Eq. 3 (e.g. $A = f_h(\mathbf{w_h^T x} + b_h)$). The least squares idea is to "solve" such a system by minimizing the 2-norm of the residual $b - Ax$ [5].

Consider a linear system of equations having $n$ unknows but $m > n$ equations. Symbolically, we wish to find a vector $x \in \mathbb{R}^n$ that satisfies $Ax = b$, where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. In general, such a problem has no solution. A suitable vector $x$ exists only if $b$ lies in *range(A)*. The vector known as the *residual*,

$$r = b - Ax \in \mathbb{R}^m, \tag{8}$$

can be made quite small by a suitable choice of $x$, but in general it cannot be equal to zero. Solving a problem that has no solution means make the residual $r$ as small as possible. Measuring the smallness $r$ entails choosing a norm. If we choose the 2-norm, given $A \in \mathbb{R}^{m \times n}, m \geq n, b \in \mathbb{R}^m$, the problem takes the following form:

$$\min_{x \in \mathbb{R}^n} \|b - Ax\|_2^2 \tag{9}$$

The formulation in Eq. 9 is the (linear) least squares problem. This is always solvable and it's a convex problem. If $A$ is square and is nonsingular then there is a unique solution. If $A$ is rectangular (tall thin) there is a unique solution if and only if $A$ has full column rank (or $A^T A$ is positive definite).

A geometric interpretation of the problem in Eq. 9 is finding the closest vector to $b$ inside the hyperplane *Im(A)* by orthogonal projection.

Suppose $A$ has full column rank, then the problem written in Eq. 9 can also be written as:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \|b - Ax\|_2^2 &= \min_{x \in \mathbb{R}^n} (Ax - b)^T (Ax - b) \\ &= \min_{x \in \mathbb{R}^n} x^T A^T Ax - b^T Ax - x^T A^T b + b^T b \\ &= \min_{x \in \mathbb{R}^n} x^T A^T Ax - 2b^T Ax + b^T b \end{aligned} \tag{10}$$

The Gradient of Eq. 10 is the following:

$$\nabla \|b - Ax\|_2^2 = 2A^T Ax - 2A^T b \tag{11}$$

The Hessian of Eq. 10 is the following:

$$\nabla^2 \|b - Ax\|_2^2 = 2A^T A \tag{12}$$

The minimum of the gradient exists and is unique and can be found with:

$$2A^T Ax - 2A^T b = 0 \textbf{ or } A^T Ax = A^T b \tag{13}$$

The linear system written in Eq. 13 has a unique solution since $A^T A$ is square invertible (because it's positive definite).

The linear system $A^T Ax = A^T b$ can be solved with many methods: Gaussian elimination, LU factorization, QR factorization, Cholesky factorization, LDL factorization etc. For this project we use LDL factorization (2.2) for solving the linear system in Eq. 13.

### 1.4 Regularization

In machine learning, the regularization is used to ensure a trade-off between accuracy in the training set and complexity of the model [4]. In this project we implemented L2 regularization adding a penalty term multiplied by a lambda parameter to the loss function:

$$\mathbf{W} \in \mathbb{R}^n,\ L(\mathbf{W}) + \lambda\Omega(\mathbf{W}) \tag{14}$$

where $\lambda$ is an hyperparameter to be tuned, $\Omega(\mathbf{W})$ is the L2 regularization usually named "Ridge regression" and defined as

$$\Omega(\mathbf{W}) = \sum_{i=1}^{k} w_i^2 = \|\mathbf{W}\|_2^2 \tag{15}$$

From now on, when we referring to the *loss function*, we will refer to the objective plus the penalty/regularization term.

The regularized objective of Eq. 5 is reformulated as follow:

$$SSE = \|y(x) - d\|_2^2 + \lambda \|w_o\|_2^2 = \xi \tag{16}$$

The problem in Eq. 9 is reformulated as follow:

$$\min_{x \in \mathbb{R}^n} \|b - Ax\|_2^2 + \lambda \|x\|_2^2 \tag{17}$$

## 2 Methods

In this section we will talk about iterative and direct methods in order to find the best parameter $\theta$ that minimizes/solves the linear least-squares problem.

### 2.1 FISTA

*Fast Iterative Shrinkage-Thresholding Algorithm* (`FISTA`) is an iterative algorithm belonging to the family of Accelerated Gradient algorithms [6]. It is used for solving general smooth convex problems [7].

The unconstrained smooth convex optimization problem we want to solve is:

$$\min_{x \in \mathbb{R}^n} f(x) = \min_{x \in \mathbb{R}^n} \|Ax - b\|_2^2 + \lambda \|x\|_2^2 \tag{18}$$

where $f$ is $L$-smooth and convex.

**Definition 2.1** ($L$-smooth)**.** A function $f$ is $L$-smooth if it is continously differentiable and its gradient is Lipschitz continuous with Lipschitz constant L:

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L \|x - y\|_2, \forall x, y \in \mathbb{R}^n \tag{19}$$

**Definition 2.2** (convexity)**.** A function $f : \mathbb{R}^n \to \mathbb{R}$ is called convex if

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y) \tag{20}$$

holds for all $x, y \in \mathbb{R}^n$ and $\alpha \in [0, 1]$.

**Theorem 2.1.** (First and second order characterizations of convex functions [8])
Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is twice differentiable over an open domain. Then, the following are equivalent:

(i) $f$ is convex.

(ii) $f(y) \geq f(x) + \nabla f(x)^T (y - x), \forall x, y \in dom(f)$.

(iii) $\nabla^2 f(x) \succeq 0, \forall x \in dom(f)$.

**Theorem 2.2.** (L-smoothness [9]) Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is two times continuously differentiable function. Then, the function is $L$-smooth if and only if:

$$0 \preceq \nabla^2 f(x) \preceq LI, \forall x \in \mathbb{R}^n \tag{21}$$

where $L$ is the Lipschitz constant and $I$ is the identity matrix.

In order to solve the problem in Eq. 18 we need to recall the *Gradient Descent* algorithm [1].

**Algorithm 2** Gradient Descent

choose $x_0 \in \mathbb{R}^n$
choose $T \in \mathbb{N}$ ▷ number of steps
choose positive $L \in \mathbb{R}$ ▷ Lipschitz constant
**for** `t from 1 to T` **do**
    $x_t = x_{t-1} - \frac{1}{L} \nabla f(x_{t-1})$ ▷ update rule
**end for**

**Theorem 2.3.** (Convergence of `GD` [10]) Let $f$ be $L$-smooth and convex, then the solution $x_t$ from the Alg. 2 satisfies:

$$f(x_t) - f(x_*) \leq \frac{2L \|x_0 - x_*\|}{t}.$$

As stated in Theorem 2.3 the Alg. 2 has rate $\mathcal{O}(1/t)$ if $f$ is convex and the $\nabla f$ is Lipschitz continuous. If we apply acceleration to Alg. 2 we can achieve the optimal rate $\mathcal{O}(1/t^2)$.

Applying acceleration to Alg. 2 resulting in the general definition of the `FISTA` algorithm.

**Algorithm 3** FISTA (general formulation)

choose $x_0 \in \mathbb{R}^n$
choose $T \in \mathbb{N}$ ▷ number of steps
set $\beta_0 = 0$ and $y_0 = x_0$
choose positive $L \in \mathbb{R}$ ▷ Lipschitz constant
**for** $t$ `from 1 to` $T$ **do**
    $\beta_t = \frac{1+\sqrt{1+4\beta_{t-1}^2}}{2}$ ▷ momentum parameter
    $x_t = y_{t-1} - \frac{1}{L} \nabla f(y_{t-1})$ ▷ update rule
    $y_t = x_t + \frac{\beta_{t-1}-1}{\beta_t}(x_t - x_{t-1})$
**end for**

**Theorem 2.4.** (Convergence of `FISTA` [7]) Let $f$ be $L$-smooth and convex, then the solution $x_t$ from the Alg. 3 satisfies:

$$f(x_t) - f(x_*) \leq \frac{2L \|x_0 - x_*\|}{t^2}$$

*Proof.* We start with the following fact:

$$f(x_{t+1}) - f(x) = f(x_{t+1}) - f(y_t) + f(y_t) - f(x)$$

Using the facts that $f(x_{t+1}) - f(y_t) \leq -\frac{1}{2L} \|\nabla f(y_t)\|_2^2$ and $f(y_t) - f(x) \geq \nabla f(y_t)^T (y_t - x)$, we have

$$f(x_{t+1}) - f(x) \leq -\frac{1}{2L} \|\nabla f(y_t)\|_2^2 + \nabla f(y_t)^T (y_t - x)$$

Hence, by definition of $x_{t+1} = y_t - \frac{1}{L} \nabla f(y_t)$, we have

$$f(x_{t+1}) - f(x) \leq -\frac{L}{2} \|x_{t+1} - y_t\|_2^2 - L(x_{t+1} - y_t)^T (y_t - x)$$

Let $x = x_t$ and $x = x_*$, respectively, and multiply the resulting equation with two separate factors:

$$[f(x_{t+1}) - f(x_t)](\beta_t - 1) \leq \left[ -\frac{L}{2} \|x_{t+1} - y_t\|_2^2 - L(x_{t+1} - y_t)^T (y_t - x_t) \right] (\beta_t - 1)$$

$$[f(x_{t+1}) - f(x_*)] \leq \left[ -\frac{L}{2} \|x_{t+1} - y_t\|_2^2 - L(x_{t+1} - y_t)^T (y_t - x_*) \right]$$

Note that $\beta_t \geq 1$, $\forall t \geq 1$. Adding these two equations together, we arrive at

$$\beta_t f(x_{t+1}) - (\beta_t - 1) f(x_t) - f(x_*) \leq -\frac{\beta_t L}{2} \|x_{t+1} - y_t\|_2^2 - L(x_{t+1} - y_t)^T (\beta_t y_t + (\beta_t - 1) x_t - x_*)$$

Let $\epsilon_t = f(x_t) - f(x_*)$, this leads to:

$$\beta_t \epsilon_{t+1} - (\beta_t - 1)\beta_t \leq -\frac{\beta_t L}{2} \|x_{t+1} - y_t\|_2^2 - L(x_{t+1} - y_t)^T (\beta_t y_t + (\beta_t - 1) x_t - x_*)$$

Multiplying both sides by $\beta_t$:

$$\beta_t^2 \epsilon_{t+1} - \beta_t (\beta_t - 1)\epsilon_t \leq -\frac{L}{2} [\beta_t^2 \|x_{t+1} - y_t\|_2^2 + 2\beta_t L (x_{t+1} - y_t)^T (\beta_t y_t + (\beta_t - 1) x_t - x_*)]$$

By definition of $\beta_t$, we know the following is true,

$$\beta_{t+1}^2 - \beta_{t+1} = \beta_t^2$$

Rearranging terms, we have

$$\beta_t^2 \epsilon_{t+1} - \beta_{t-1}^2 \epsilon_t \leq -\frac{L}{2} (\|\beta_t x_{t+1} - (\beta_t - 1) x_t - x_*\|_2^2 - \|\beta_t y_t - (\beta_t - 1) x_{t+1} - x_*\|_2^2)$$

Invoking the definition of $y_{t+1}$, one can show that

$$\beta_t x_{t+1} - (\beta_t - 1) x_t - x_* = \beta_{t+1} y_{t+1} - (\beta_{t+1} - 1) x_{t+1} - x_*$$

Hence, defining $u_0 = x_0$, $u_t = \beta_t y_t - (\beta_t - 1) x_t - x_*, \forall t \geq 1$, the above equation simplifies to

$$\beta_t^2 \epsilon_{t+1} - \beta_{t-1}^2 \epsilon_t \leq -\frac{L}{2} (\|u_{t+1}\|_2^2 - \|u_t\|_2^2)$$

Therefore, by induction, we have

$$\beta_{t-1}^2 \leq \frac{L}{2} \|u_0\|_2^2$$

From this, we realize the following:

$$\epsilon_t \leq \frac{L \|x_0\|_2^2}{2\beta_{t-1}^2}$$

By simple induction, we can show that

$$\beta_{t-1} \geq \frac{t}{2}, \forall t \geq 1$$

Therefore:

$$\epsilon_t \leq \frac{2L \|x_0\|_2^2}{t^2}$$

■

In the Alg. 4 we have the specific implementation of `FISTA` considering the structure of the ANN and the objective.

**Algorithm 4** FISTA

1: choose $x_0 \in \mathbb{R}^n$, $T \in \mathbb{N}$ ▷ model parameters and time steps
2: choose positive $\lambda \in \mathbb{R}$ ▷ regularization parameter
3: set $\beta_0 = 0$ and $y_0 = x_0$
4: given $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ ▷ data samples and targets
5: set $L = \rho_{max}(2A^TA)$ ▷ maximum eigenvalue (Lipschitz constant)
6: $f(x) = \|Ax - b\|_2^2 + \lambda \|x\|_2^2 = x^TA^TAx - 2b^TAx + b^Tb + \lambda x^Tx$ ▷ objective function
7: $g(y) = \nabla f(y) = 2A^T(Ay - b) + 2\lambda y$ ▷ objective gradient
8: $h(z) = \nabla^2 f(z) = 2A^TA$ ▷ objective hessian
9: **for** $t$ `from 1 to T` **do**
10: $\beta_t = \frac{1+\sqrt{1+4\beta_{t-1}^2}}{2}$ ▷ momentum parameter
11: $x_t = y_{t-1} - \frac{1}{L} g(y_{t-1})$ ▷ update rule
12: $y_t = x_t + \frac{\beta_{t-1}-1}{\beta_t}(x_t - x_{t-1})$
13: **end for**

The function $f$ is convex (Def. 2.2) because, as stated in Theorem 2.1, the Hessian (Eq. 12) is positive semidefinite ($2A^TA \succeq 0$).

The function $f$ is $L$-smooth (Def. 2.1) because, recalling the Theorem 2.2 that states that $\nabla^2 f(x) \preceq LI, \forall x$, this means that the eigenvalues of the Hessian are bounded above by $L$. Then, the minimum $L$ is the maximum eigenvalue of $2A^TA$. This means that $v^T\nabla^2 f(u)v \leq v^T(LI)v, \forall u, v$, or that $v^T\nabla^2 f(u)v \leq L\|v\|^2$ [11].

Considering the above sentences in the Alg. 4 holds the Theorem 2.4 because the function $f$ (Alg. 4 line 6) is $L$-smooth and convex. So the Alg. 4 achieves the rate of convergence of $\mathcal{O}(1/t^2)$ and reaches an error $\epsilon$ after $\mathcal{O}(1/\sqrt{\epsilon})$ iterations. Following the previous reasoning we can say that, as stated in Theorem 2.3, `GD` achieves the optimal rate of convergence of $\mathcal{O}(1/t)$.

## 2.2 LDL

*LDL Factorization* is a Gaussian elimination algorithm on symmetric matrices. Any symmetric matrix $M = M^T \in \mathbb{R}^{m \times m}$ (for which we do not encounter zero pivots in the algorithm) admits a factorization $M = LDL^T$, where $L$ is lower triangular with ones on its diagonal, and $D$ is diagonal [5].

In order to solve the problem of $\min_{x \in \mathbb{R}^n} \|b - Ax\|_2^2$, we need to compute its gradient (Eq. 11) that corresponds in solving the linear system $A^TAx = A^Tb$. Since $A^TA$ is symmetric, we can factorize $M = A^TA = LDL^T$ and because of $M$ is square invertible (because it's positive definite) the linear system as a unique solution.

**Definition 2.3** (Backward stability)**.** An algorithm to compute $\mathbf{y} = f(\mathbf{x})$ is called *backward stable* if the computed output $\tilde{\mathbf{y}}$ can be written as $\tilde{\mathbf{y}} = f(\tilde{\mathbf{x}})$, where

$$\frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} = \mathcal{O}(u). \tag{22}$$

where $u$ is the range of representability of the numbers and $\mathcal{O}(u)$ notation often hides polynomial factors in the dimension n. Backward stable algorithms are as accurate as theoretically possible (given the condition number of a problem), up to a polynomial factor in n.

**Theorem 2.5.** (Residual and error) Let $A \in \mathbb{R}^{m\times m}, \mathbf{b} \in \mathbb{R}^m$ and $x$ be the solution of $A\mathbf{x} = \mathbf{b}$. Then,

$$\frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \kappa(A)\frac{\|\mathbf{r}\|}{\|\mathbf{b}\|} \tag{23}$$

where $\kappa$ is the condition number and $\mathbf{r} = A\tilde{x}-b$ ($\tilde{x}$ is the solution of the linear system) is the residual.

Since $A \in \mathbb{R}^{m\times m}$ is symmetric, $LDL^T factorization$ is a more efficient method than LU/QR. The idea is to choose $L_1$ in order to add multiples of row 1 to rows 2 ... n to kill $A_{2:end,1}$ and then compute $L_1AL_1^T$. This matrix is symmetric and block upper triangular (because both $L_1A$ and $L_1^T$ are so). In the end we have

$$L_{m-1}L_{m-2}\dots L_1AL_1^T\dots L_{m-2}^TL_{m-1}^T = D \tag{24}$$

where $D$ is diagonal, or

$$A = L_1L_2\dots L_{m-1}DL_{m-1}^T\dots L_2^TL_1^T = LDL^T \tag{25}$$

The cost of $LDL^T factorization$ is $\frac{1}{3}m^3 + \mathcal{O}(m^2)$.

| Method | Cost |
|---|---|
| $LDL^T$ | $\frac{1}{3}m^3 + \mathcal{O}(m^2)$ |
| $LU$ | $\frac{2}{3}m^3 + \mathcal{O}(m^2)$ |
| QR | $\frac{4}{3}m^3 + \mathcal{O}(m^2)$ |

Table 1: Methods costs.

After solving a linear system, to check if the computed solution $\tilde{x}$ is accurate, one can take the computed $\tilde{x}$ and check if the $residual$ $r = A\tilde{x} - b$ is small. But it does not imply that $\tilde{x}$ is close to the true solution $x$, so the residual is computed as

$$r = \frac{\|b - A\tilde{x}\|}{\|b\|} \tag{26}$$

The $LDL^T factorization$ (Alg. 5) is not backward stable (Def. 2.3), unless we do some form of symmetric pivoting (e.g. $P^TAP$) (Alg. 6).

**Algorithm 5** LDL

given $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $\lambda \in \mathbb{R}$
let $I \in \mathbb{R}^{n \times n}$ be the identity matrix
let $L = I$
let $D \in \mathbb{R}^{n \times n}$ be full of zeros
let $M = A^T A + \lambda I \in \mathbb{R}^{n \times n}$ ▷ Regularized matrix
let $y = A^T b \in \mathbb{R}^n$
**for** $k$ `from 1 to` $n-1$ **do**
    $D(k,k) = M(k,k)$
    $L(k+1:end, k) = M(k+1:end, k)/M(k,k)$
    $M(k+1:end, k+1:end) = M(k+1:end, k+1:end) - L(k+1:end, k) * M(k, k+1:end)$
**end for**
$D(n,n) = M(n,n)$
$x = L^T \backslash ((L \backslash y)./diag(D))$ ▷ Solution of the linear system

**Algorithm 6** LDL (with pivoting)

given $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $\lambda \in \mathbb{R}$
let $I \in \mathbb{R}^{n \times n}$ be the identity matrix
let $L = I$
let $P = I$ ▷ Permutation matrix
let $D \in \mathbb{R}^{n \times n}$ be full of zeros
let $M = A^T A + \lambda I \in \mathbb{R}^{n \times n}$ ▷ Regularized matrix
let $y = A^T b \in \mathbb{R}^n$

$M\_diag = diag(M)$ ▷ Diagonal of M
**for** $k$ `from 1 to` $n-1$ **do**
    $p = max(abs(M\_diag(k:n)))$ ▷ Pivot position
    $p = p + k - 1$ ▷ Adjust from k:end to 1:end
    **if** $p$ `is different from` $k$ **then**
        $P(:, [k,p]) = P(:, [p,k])$ ▷ Update permutation matrix
        $M([k,p], :) = M([p,k], :)$ ▷ Swap rows
        $M(:, [k,p]) = M(:, [p,k])$ ▷ Swap columns
    **end if**
**end for**

**for** $k$ `from 1 to` $n-1$ **do**
    $D(k,k) = M(k,k)$
    $L(k+1:end, k) = M(k+1:end, k)/M(k,k)$
    $M(k+1:end, k+1:end) = M(k+1:end, k+1:end) - L(k+1:end, k) * M(k, k+1:end)$
**end for**
$D(n,n) = M(n,n)$

$L = P * L;$ ▷ Apply permutation
$x = L^T \backslash ((L \backslash y)./diag(D))$ ▷ Solve the linear system

The pivoting strategy used in Alg. 6 is a complete diagonal pivoting.

# 3 Experiments

In this chapter we will describe the data used for our experiments, the experimental setup, the hyper-parameters configuration and, in the end, the results obtained.

### 3.1 Experimental setup

For what concerns the optimizer, we chose the topology of the model a-priori with a manual exploration (see Tab. 2); after that, we proceed with an exploration of the possible hyper-parameters. Also, the starting point with reference to the weights initialization of the model is an hyper-parameter which allows to find a good starting point for the optimization process: we sampled a random vector from uniform distribution in range $(0, 1)$ for all the experiments.
The experiments has been carried out on a PC Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz with 6-cores and 16 GB RAM.
Aim of the experiments is to compare the performances of our approaches in terms of speed and convergence rate. For the purposes of our comparisons, we will consider the following parameters:

- $f^*$: the optimal value found by a closed-form/direct method;
- $f(x_t)$: the loss value at step $t$ of the optimization process;
- *number of iterations* $t$ until convergence;
- $\epsilon$: absolute error threshold.

To compare the different approaches, we chose to involve the *CUP* dataset used in the internal competition of the Machine Learning course. The *CUP* dataset consists of 1765 examples and a set of 20 numerical features. The representative task for this dataset is the regression problem with a bidimensional output.

The input matrix used as input for the optimization algorithms is $A = f_h(\mathbf{w_h^T x} + b_h)$ as stated in Eq. 3, where $x$ is the dataset of the problem considered.

### 3.2 Stopping criteria

For our experiments, we chose the following two criteria to detect when to stop the optimization process:

- *absolute error* ($\epsilon_{abs}$): if $\epsilon_{abs} = f(x_t) - f^* \leq \epsilon$ the optimization process will stop;
- *max number of iterations*: as condition to stop trying to reach a minimun infinitely.

In this way, we can reach a minimum (in terms of distance from the optimal) or stop the optimization process after a certain number of iterations.

### 3.3 Convergence rate

We are interested in analizing the algorithm in terms of the *convergence rate*. To compute it we consider the following equation:

$$\lim_{k\to\infty} \frac{|f(x_{k+1}) - f^*|}{|f(x_k) - f^*|^p} = R \tag{27}$$

where $f^*$ is, as stated before, the optimal value found by the optimization process and $f(x_k)$ is the optimal value at the iteration $k$.

We can say that the algorithm has an order of convergence $p$ if there exists $R > 0$. The convergence rate $p$ is said to be linear if $p = 1$, superlinear if $p > 1$ and quadratic if $p = 2$. To estimate $p$, we use the following equation

$$p = \lim_{k\to\infty} \frac{\log|f(x_{k+1}) - f^*| - \log R}{\log|f(x_k) - f^*|} \approx \lim_{k\to\infty} \frac{\log|f(x_{k+1}) - f^*|}{\log|f(x_k) - f^*|} \tag{28}$$

We will show the convergence rate of our optimizer by using meaningful plots.

### 3.4 Results

For the CUP dataset a model has been created exploiting the configuration of hyper-parameters described in Tab. 2.

| Parameter | Values |
|---|---|
| Layers | 1 |
| Activation function | Sigmoid |
| Hidden size | 100 |
| Lambda | 0.1 |
| $\epsilon$ | $1e-4$ |

Table 2: Model hyper-parameters.

We explored the hyper-parameter space with a random-search to choose the best algorithm parameters, based on the lowest loss value. In Tab. 3 there is the evidence of the parameters that result in the lowest loss value, at the end of the optimization process. These values have been chosen with respect to theoretical expectations, as stated in Section 2.1. In addition to `FISTA`, we measured the performance of `GD` algorithm, in order to have a criterion for comparison of the rate of convergence of our algorithm implementation. Also for `GD`, we run a random-search in order to take the best parameters, using the same model configuration previously used, stated in Tab. 2; as for `FISTA`, in Tab. 3 we have the best parameters for this algorithm.

| Algorithm | Max. num. of iterations | Learning rate |
|---|---|---|
| FISTA | 500 | $1/L$ |
| GD | 10000 | $1/L$ |

Table 3: `FISTA` and `GD` best parameters.

Consider the fact that, as anticipated in Section 3.1, the $f^*$ is obtained using the optimal parameter $x^*$ found by the execution of matlab implementation of `LDL` because it returns the lowest residual value of the bunch. An interesting fact is that the matlab implementation of `LDL` and our implementation of `LDL` with pivoting reach the same $f^*$ differently from our implementation of `LDL` without pivoting.

In Tab. 4 are shown the results of the best model of `FISTA` and `GD` and the different versions of `LDL` performing on the CUP dataset. We can notice that the residuals are much lower for `LDL` with respect to the residuals obtained with `FISTA` and, also, the execution times are much lower with `LDL` approach. Hence, `LDL` results emphasizes the better capabilities of this approach with respect to `FISTA` to solve convex problems, like the one at hand.

Another thing we can appreciate from these results is that the `LDL` implementation with pivoting performs slightly better than the one without pivoting - but worse than the standard Matlab implementation, with respect to the relative residual. This reflects our expectations because of the improved stability that the pivoting should bring to the algorithm.

| Algorithm | Residual (Eq. 26) | Exec. time (sec) | $f(x_t)$ | $f^*$ | $t$ | $t^*$ | Stop reason |
|---|---|---|---|---|---|---|---|
| `FISTA` | 3.4311e-01 | 0.07737 | 2.8802e+01 | 2.8762e+01 | 250 | 100 | thresh. reached |
| `GD` | 3.4534e-01 | 1.55349 | 2.8763e+01 | 2.8762e+01 | 8058 | 10000 | thresh. reached |
| `LDL` (matlab) | 1.2917e-15 | 0.00019 | - | 2.8762e+01 | - | - | - |
| `LDL` (no pivoting) | 1.4761e-15 | 0.00222 | - | 4.1257e+08 | - | - | - |
| `LDL` (with pivoting) | 1.4474e-15 | 0.00448 | - | 2.8762e+01 | - | - | - |

Table 4: Results on CUP dataset.

From the results and the plot of the convergence rate (Fig. 1), we notice that `FISTA`, in the first iterations results in being sublinear, as expected; then, when close to approaching $f^*$ it has a superlinear convergence peak, after which it stops because of reaching the threshold; in the same plot we can appreciate also the rate of convergence of `GD`, that's still in a sublinear fashion for the majority of the iterations until, around iteration 6700, has a superlinear convergence peak; after that it keeps being almost sublinear and it stops after some more iterations.

As we can see in Fig. 4, this let us achieve the stop condition below the threshold $\epsilon$ at 250 iterations of the optimization process for `FISTA` and at 8058 iterations for `GD`. Observing the convergence rate of `GD` reflects our expectations in a speed of convergence faster for `FISTA` (optimal rate $\mathcal{O}(1/t^2)$) with respect to `GD` (optimal rate $\mathcal{O}(1/t)$), as highlighted in Table 4 where $t^*$ and $t$ represent, respectively, the theoretical expected and the practical obtained iterations necessary for convergence.

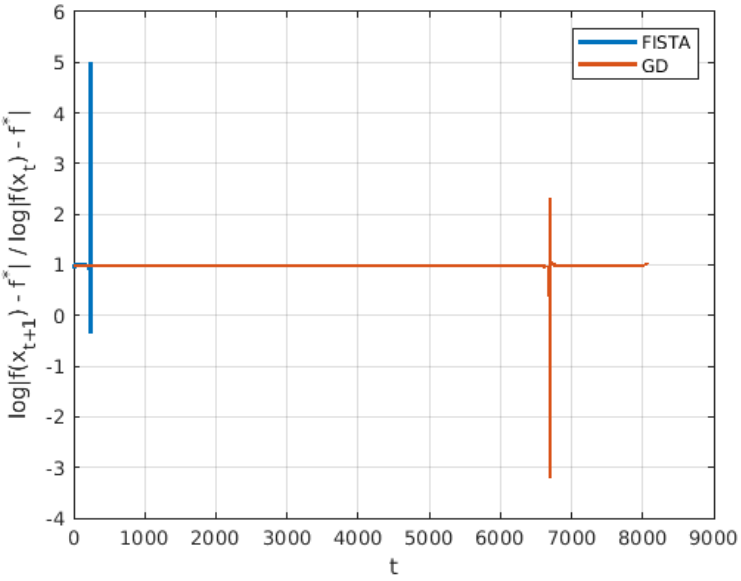


Figure 1: Convergence rates.

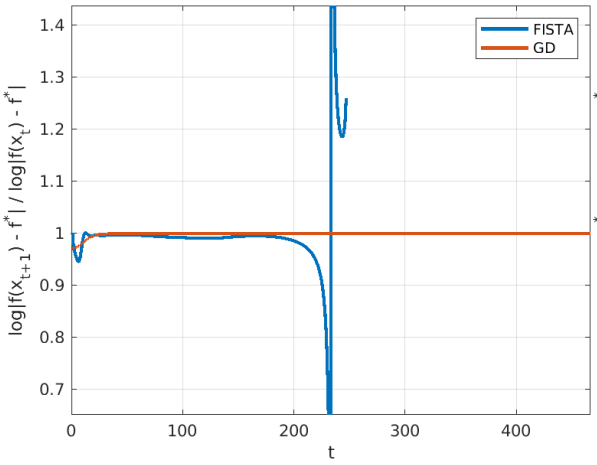


Figure 2: Zoom on FISTA convergence rates.

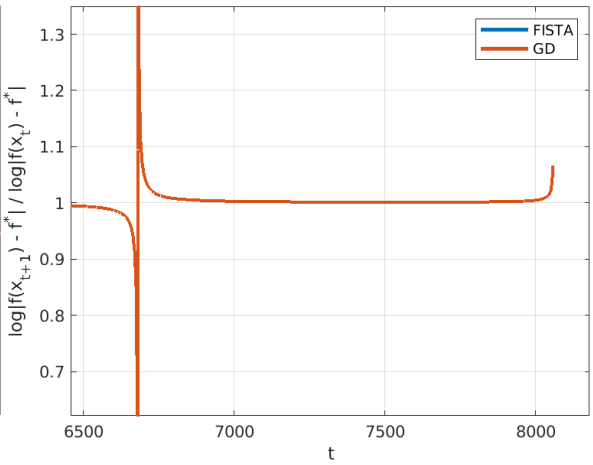


Figure 3: Zoom on GD convergence rates.

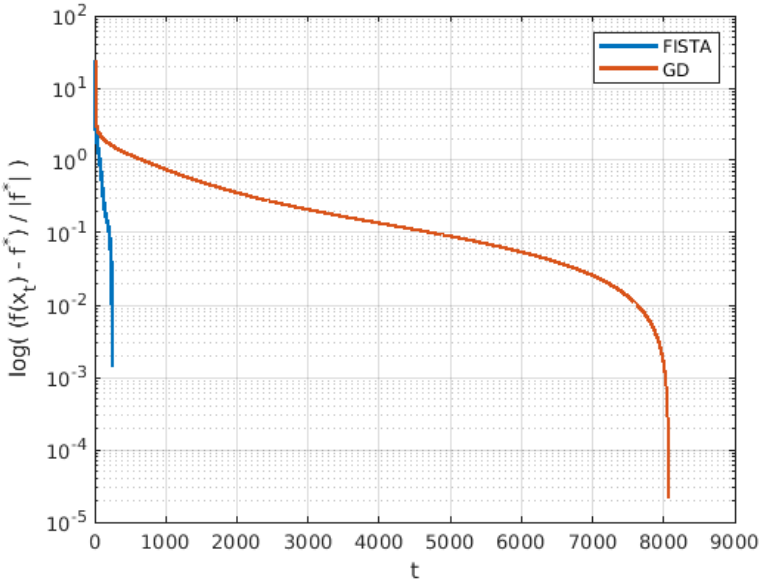


Figure 4: Logarithmic distances from optimal value during time.

# 4 Conclusions

This work let us investigate on two different approaches to solve linear least-squares problems involving the `FISTA` optimization algorithm, of the accelerated gradient methods, and the `LDL` factorization, for a closed-form solution of the problem. We analyzed theoretical aspects of both, applying these methods to a real-scenario dataset to understand their convergence behavior. As a result, we observed that `LDL` performs much better than `FISTA` in solving convex problems, at least concerning the task taken into account, having much lower execution times and residuals of many order of magnitude smaller. Moreover, we could observe the different behaviors of two similar optimization algorithms, comparing `FISTA` with `GD` and analyzing their different rates of convergence, which results, as expected, with a faster convergence for `FISTA` and a slower one for `GD`.

# References


[1] Ryan Tibshirani Geoff Gordon. Accelerated first-order methods. *https://www.cs.cmu.edu/ ggordon/10725-F12/slides/09-acceleration.pdf*.

[2] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *https://arxiv.org/pdf/1412.6980.pdf*, 2014.

[3] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[4] Simon Haykin. *Neural networks and learning machines, 3/E*. Pearson Education India, 2010.

[5] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.

[6] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[7] Niao He. Lecture 9: Gradient descent and acceleration. *https://github.com/niaohe/Big-Data-Optimization-Course/blob/main/lecture$_s$cribe/IE598 − lecture9 − gradient − descent − and − acceleration.pdf*.

[8] A.A. Ahmadi. Lecture 7: Theory of convex functions. *http://www.princeton.edu/ aaa/Public/Teaching/ORF523/S16/ORF523$_S$16$_L$ec7$_g$h.pdf*.

[9] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2003.

[10] Ryan Tibshirani Geoff Gordon. Lecture 5: Gradient desent revisited. *https://www.cs.cmu.edu/ ggordon/10725-F12/scribes/10725$_L$ecture5.pdf*.

[11] Mark Schmidt. Lipschitz continuity of the gradient. *https://www.cs.ubc.ca/ schmidtm/Courses/540-W18/L4.pdf*.