Instituto Superior Técnico, Universidade de Lisboa

Master of Science in Electrical and Computer Engineering

A.Y. 2024/2025

Optimization and Algorithms Project

Professores: Joao Xavier, Joao Sequeira, Hugo Pereira

Group 1:

- Lino Di Lucia (ist1112859) - lino.dilucia@studentmail.unicas.it
- Pascal Seitter (ist1112432) - pascal.seitter@tum.de
- João Costa (ist199088) - joaoarcosta@tecnico.ulisboa.pt
- Matteo Talè (ist1113101) - matteo.tale@tecnico.ulisboa.pt

Index

Chapter 1

Aim of the project

This project aims to solve some tasks assigned in class. For the analysis of the respective tasks, it was necessary to use MATLAB and the CVX tool (important in solving problems of a convex nature). The tasks are divided into theoretical tasks for which a graphical and/or analytical solution was presented and numerical tasks, for which the code will be shown in detail and the final result.

1.1 Task 1: Examination of non convexity

The goal of this task is to demonstrate that the function f_D defined as

$$\frac{1}{N}\sum_{n=1}^{N} 1_{R_-}(y_n C_{w_0,w}(x_n)) \tag{1.1}$$

is not convex. We will focus on a simple case where $N = 1$ and $D = 1$.

For the case where $N = 1$ and $D = 1$, this simplifies to:

$$f_D = 1_{R_-}(y_1 C_{w_0,w}(x_1)) \tag{1.2}$$

To illustrate the non-convexity of f_D, we can analyze the graph of the function for specific values of y_1, C, and *w(*x_1*)*. The non-convexity becomes evident when the graph exhibits a shape that fails to meet the definition of convexity — specifically, when the line segment connecting any two points on the graph does not lie entirely above the graph.

Rather than substituting all the values of x_1, y_1, w_0, and w_1 into the definition of a convex function, as discussed in class, it is far more practical to perform a straightforward plot of the function for the case under consideration. This visual approach allows for a clear identification of the non-convex behavior.

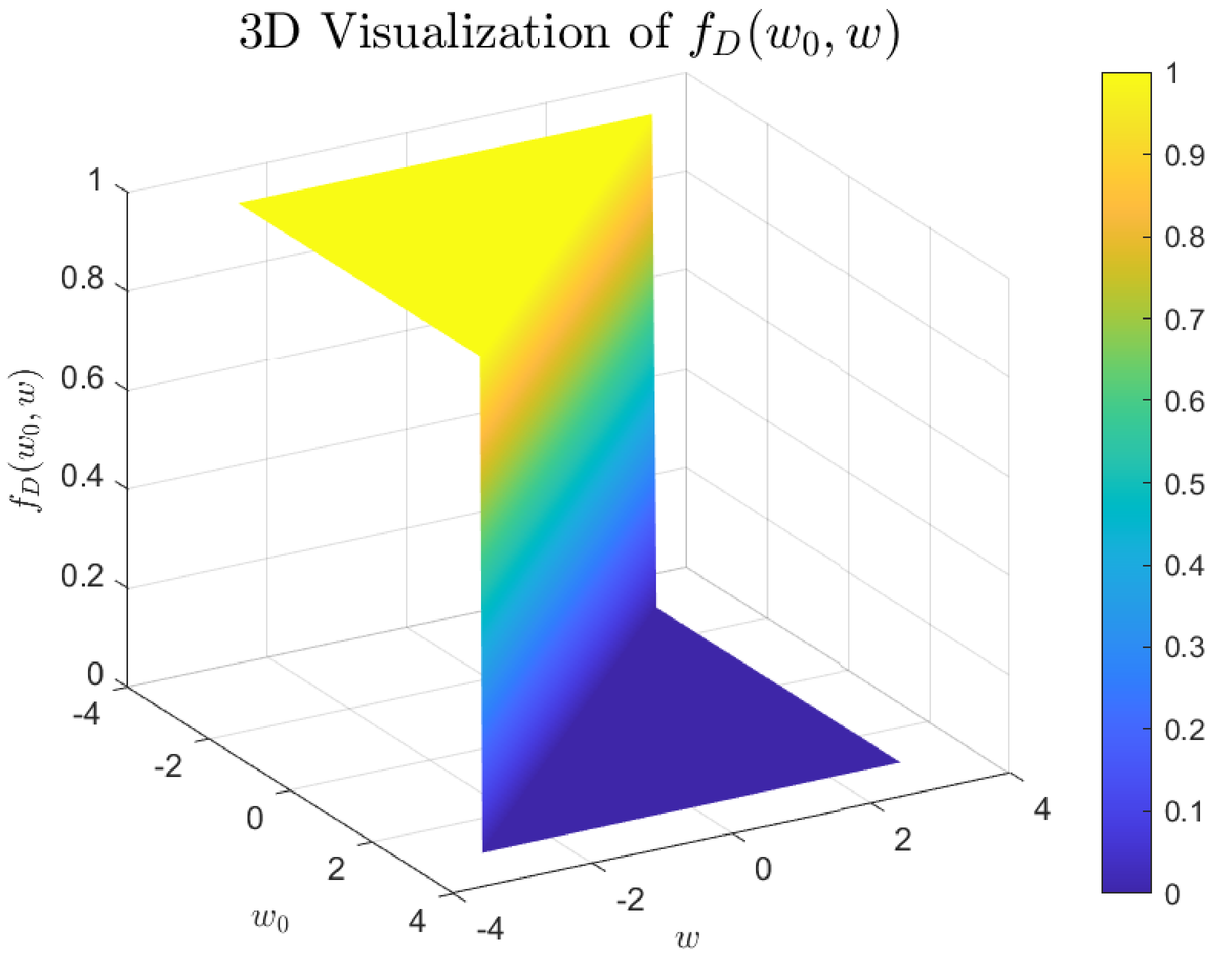


Figure 1.1: Graph illustrating the non-convexity of the function f_D for the case $N = 1$ and $D = 1$.

Geometrical interpretation: The graph above demonstrates that the function f_D is not convex, as evidenced by the fact that the line segment connecting two points on the curve does not lie entirely above the curve itself.In other words, if two points that are chosen on the graph, where one point has the function value 1 and the other point has the function value 0, and both points are connected with a straight line, it can be clearly seen that the graph does not lie completely below this straight line between those two points and thus violates the condition of convexity that from a geometrical point of view requires that the chord is above the graph .This confirms our assertion regarding the non-convexity of the function for the specified case.

1.2 Task 2: Comparison of $\mathbf{1}_{\mathbb{R}_-}$ and Hinge Loss

Our goal is to show that the function $\mathbf{1}_{\mathbb{R}_-}$, defined as:

$$\mathbf{1}_{\mathbb{R}_-}(u) = \begin{cases} 1 & \text{if } u < 0, \\ 0 & \text{if } u \geq 0, \end{cases} \tag{1.3}$$

is majorized by the hinge loss function $h(u) = (1 - u)_+$, where the hinge loss is defined as:

$$h(u) = \max(0, 1 - u). \tag{1.4}$$

In other words, we need to prove that $\mathbf{1}_{\mathbb{R}_-}(u) \leq h(u)$ for all $u \in \mathbb{R}$.

1.2.1 Analytical Proof

We define the following auxiliary function for positive values of u, denoted by $\mathbf{1}_{\mathbb{R}_+}(u)$:

$$\mathbf{1}_{\mathbb{R}_+}(u) = \begin{cases} 1 & \text{if } u > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{1.5}$$

We have the relation:

$$\mathbf{1}_{\mathbb{R}_-}(u) = \mathbf{1}_{\mathbb{R}_+}(-u). \tag{1.6}$$

Analyzing the cases:

- For $u < 0$: $\mathbf{1}_{\mathbb{R}_+}(-u) = 1$. Therefore, $h(u) = \max(0, 1-u) = 1-u \geq 1$ (since $u < 0$). Thus, $\mathbf{1}_{\mathbb{R}_-}(u) = 1 \leq h(u)$.
- For $u \geq 0$: $\mathbf{1}_{\mathbb{R}_+}(-u) = 0$, and $h(u) = \max(0, 1-u) = 0$. Therefore, $\mathbf{1}_{\mathbb{R}_-}(u) = 0 \leq h(u)$.
- we can conclude:

$$\mathbf{1}_{\mathbb{R}_-}(u) = \mathbf{1}_{\mathbb{R}_+}(-u) \leq \max(0, 1-u) = h(u) = (1-u)_+$$

This completes the analytical proof that $\mathbf{1}_{\mathbb{R}_-}(u) \leq h(u)$ for all $u \in \mathbb{R}$.

1.2.2 Graphical Proof

In addition to the analytical proof, a graphical comparison is provided between the indicator function $\mathbf{1}_{\mathbb{R}_-}(u)$ and the hinge loss function $h(u)$ in fig. 1.2.

Note: A code that allows to obtain the previous plot is present in the folder documentation.

Discussion of the Graph:

As observed in the plot(fig 3.2):

- The red dashed line represents the indicator function $\mathbf{1}_{\mathbb{R}_-}(u)$, which takes the value 1 for $u < 0$ and 0 for $u \geq 0$.
- The blue solid line represents the hinge loss function $h(u) = \max(0, 1-u)$, which decreases linearly for $u \leq 1$ and is equal to zero for $u \geq 1$.

Clearly, the hinge loss function $h(u)$ is always greater than or equal to the indicator function $\mathbf{1}_{\mathbb{R}_-}(u)$, thus confirming the inequality $\mathbf{1}_{\mathbb{R}_-}(u) \leq h(u)$ holds true for all $u \in \mathbb{R}$.

1.2.3 Convexity of the Hinge Loss Function

Finally, we show that $h(u)$ is a convex function. The hinge loss function is piecewise linear, consisting of two linear segments: one with a negative slope for $u < 1$ and another constant at 0 for $u \geq 1$. Since linear functions are convex, and the pointwise maximum of convex functions is also convex, it follows that $h(u)$ is convex. (ex. $h(u)$ is obviously convex. In fact, for $u \geq 1$, $h(u) = 0$, which is a constant and therefore convex. For $u < 1$, $h(u) = 1 - u$, which is linear and thus convex)

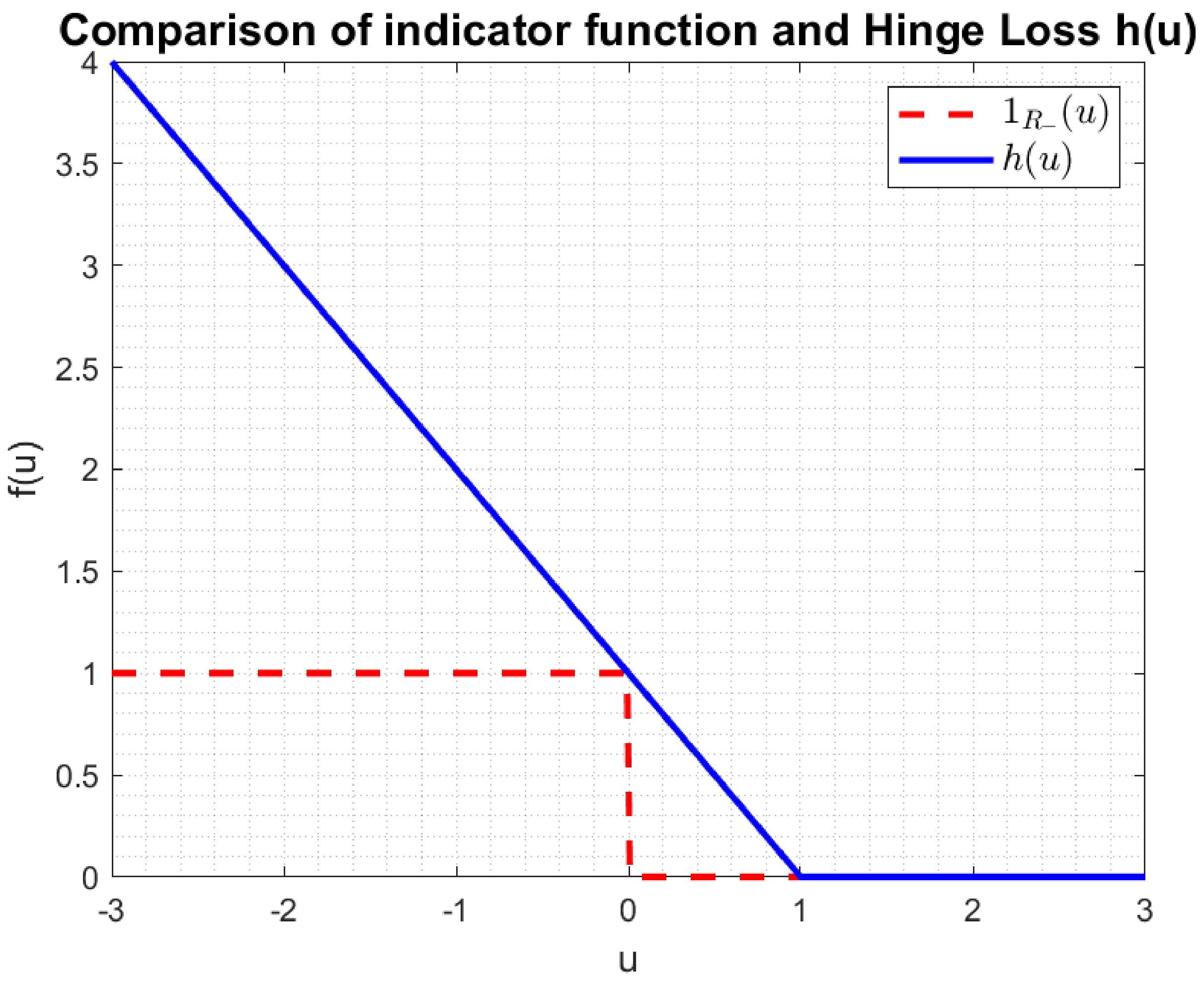


Figure 1.2: Comparison of the indicator function $\mathbf{1}_{\mathbb{R}_-}(u)$ (red dashed) and the hinge loss function $h(u)$ (blue solid).

1.3 Task 3: Convexity of the Function $g_D(w_0, w)$

We are tasked with showing that the function:

$$g_D(w_0, w) = \frac{1}{N}\sum_{n=1}^{N} h\left(y_n\left(w_0 + x_n^T w\right)\right) \tag{1.7}$$

is convex for any values of N and D.

1.3.1 Analytical Proof

Let us begin by decomposing the (1.7) function and considering the following properties:

- **Convexity of $h(u)$:** The function $h(u) = (1-u)_+$, is convex because it is piecewise linear. This has already been demonstrated both analytically and graphically in Task 2 (see Task 2 for reference). The hinge loss function, which $h(u)$ represents, is well-known to be convex
- **Affine Transformation of Variables:** Define $u_n = y_n(w_0 + x_n^T w)$. Since u_n is a linear function of w_0 and w, it is an affine transformation. The composition of a convex function with an affine transformation preserves convexity.

We start with the function:

$$g_D = \frac{1}{N}\sum_{n=1}^{N} h(u_n) \tag{1.8}$$

where:

$$u_n = y_n(w_0 + x_n^T w) = y_n\left(\begin{bmatrix} x_n^T & 1\end{bmatrix}\begin{bmatrix} w \\ w_0\end{bmatrix}\right) = y_n \cdot \tilde{x}_n^T \cdot \tilde{w}. \tag{1.9}$$

where $\tilde{x}_n$ and $\tilde{w}$ are the extended vectors, and it reppresents the linear change of variable u. But, in general just let focus on $h(y_n(w_0 + x_n^T w))$, since the summation requires another property specification. The function g_D is convex in $\tilde{w}$ because $h(u)$ is convex and u represents the result of a linear transformation of the extended variables $\tilde{x}_n$ and $\tilde{w}$.

where $(w_0 + x_n^T w)$ is an affine function of (w_0, w), being a linear combination of w_0 and w.

It is important to notice that aside from the affine function $w_0 + x_n^T w$, u_n also contains a multiplication of the function by the scalar y_n. This could also be represented as

$$u_n = \begin{cases} w_0 + x_n^T w & \text{if } y_n = 1, \\ -w_0 + x_n^T(-w) & \text{if } y_n = -1. \end{cases} \tag{1.10}$$

which does not influence the affinity of the function.

Since $h(u)$ is convex and u_n is the result of an affine transformation, we conclude that $h(y_n(w_0 + x_n^T w))$ is convex for each n. (Theorem 7.17 states the preservation of convexity under linear change of variables, as detailed in Chapter 7.4 of Amir Beck's book. Example 7.3 illustrates the convexity of affine functions, and can be found in Chapter 7.1)

Note: All the theorems and examples cited are present in the Amir Beck's book

- **Summation of Convex Functions:** The sum of convex functions remains convex. Since each term in the summation is convex, the entire function $g_D(w_0, w)$, which is an average of convex functions, remains convex. Let $f_1, f_2, \ldots, f_p$ be convex functions defined over a convex set $C \subseteq \mathbb{R}^n$. Then, the sum function

$$f = f_1 + f_2 + \ldots + f_p \tag{1.11}$$

  is also convex over C. (This property holds true as stated in Theorem 7.16, Chapter 7, Section 7.4, titled "Operations Preserving Convexity").

- **Averaging convex functions**: Finally, multiplying by $\frac{1}{N}$, which is a positive scalar, preserves convexity. Thus, the average of convex functions is still convex.

Thus, we have shown that $g_D(w_0, w)$ is convex.

1.4 Task 4: Convexity of the Second Function $g(w_0, w)$

We are tasked to prove that the function:

$$g(w_0, w) = \underbrace{\frac{1}{N}\sum_{n=1}^{N} h\left(y_n\left(w_0 + x_n^T w\right)\right)}_{g_D(w_0,w)} + \underbrace{\rho\|w\|_2^2}_{r(w_0,w)} \tag{1.12}$$

is convex for any values of N and D

1.4.1 Analytical Proof

Beginning with mentally separating the equation (1.12), into the sum of the previously proven convex function ($g_D(w_0, w)$), and the rest of the equation ($\rho\|w\|_2^2$); consider the following:

- **Convexity of $g_D(w_0, w)$:** The function $g_D(w_0, w) = \frac{1}{N}\sum_{n=1}^{N} h(y_n(w_0 + x_n^T w)))$ can be proven that it is in fact convex, when giving use of the theorems stated in Amir Beck's book:
  - **Theorem 7.16 (b)** (preservation of convexity under summation and multiplication by non-negative scalars): for the summation of multiple convex functions, represented by the summatory $\sum_{n=1}^{N}$.
  - **Theorem 7.17** (preservation of convexity under linear change of variables): for the composition of a convex function over an affine function.

  In order to conclude if $g(w_0, w)$ is itself convex (following the **Theorem 7.16 (b)**), we are left to validate if $\rho\|w\|_2^2$ is a convex function.

  Let us consider a bottom-up approach:

  - Starting from the basics, it is easily deducible that $\|w\|_2$ will always represent non-negative value.
  - Applying the square function, a known convex function, to $\|w\|_2$ will always end up in the positive side of the parabola. This familiar situation has been represented in class as $(\cdot)_+^2$. Thus, secured by the **Theorem 7.22**, we can state that the composition $\|w\|_2^2$ represent a non-decreasing convex function.
  - Following the **Theorem 7.16 (a)** we can conclude that the multiplication of the convex function $\|w\|_2^2$ by the non-negative scalar ρ, the convexity of the outcome is not affected.

  By following the **Theorem 7.16 (b)** once another time, we can say that the summation of the convex function $g_D(w_0, w) = \frac{1}{N}\sum_{n=1}^{N} h(y_n(w_0 + x_n^T w)))$ by the convex function $\rho\|w\|_2^2$ ends up also as convex function.

1.5 Task 5: Strong Convexity

For this task, we should investigate the function in (1.12) to see whether the function is strongly convex or not.

1.5.1 Analytical Proof

Strong Convexity (S-CVX):

In general a function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *strongly convex* if there exists a constant $m > 0$ such that for all $x, y \in \mathbb{R}^n$ and $0 \leq \alpha \leq 1$, the following inequality holds:

$$f\left((1-\alpha)x + \alpha y\right) \leq (1-\alpha)f(x) + \alpha f(y) - m\frac{\alpha(1-\alpha)}{2}\|x-y\|^2 \tag{1.13}$$

In our case, we need to analyze the function f to see if it is convex or not to determine if the function g is strongly convex:

$$f(w, w_0) = g(w, w_0) - \frac{\beta}{2}\|w\|^2 - \frac{m}{2}w_0^2 \tag{1.14}$$

The convexity analysis will be carried out below using two different approaches.

- When restricting to $w = 0$ and leaving w_0 as a free variable, the regular expression of $f(w, w_0)$:

$$f(w, w_0) = \frac{1}{N}\sum_{n=1}^{N} h\left(y_n\left(w_0 + \mathbf{x}_n^T w\right)\right) + \rho\|w\|_2^2 - \frac{\beta}{2}\|w\|_2^2 - \frac{m}{2}w_0^2, \tag{1.15}$$

  which reduced to

$$f(w_0) = \frac{1}{N}\sum_{n=1}^{N} h\left(y_n w_0\right) - \frac{m}{2}w_0^2, \tag{1.16}$$

  takes the values:

$$f(0) = 0, \tag{1.17}$$
$$\lim_{w_0 \to +\infty} f(w_0) = \lim_{w_0 \to -\infty} f(w_0) = -\infty \tag{1.18}$$

  The limit value analysis show that the function (1.15) is not convex, and subsequently the original equation at (1.12) is not strongly convex. The function f for setting w equal to 0 is also shown in Fig. 1.3, where you can clearly see that it is not convex.

- When w take values such that: $h(y_n(w_0 + x_n^T w)) > 0$

  That is

$$y_n \tilde{x}^T \tilde{\omega} > 0$$

  Rewriting the function $f(\omega, \omega_0)$:

$$f(\omega, \omega_0) = \frac{1}{N}\sum_{n=1}^{N} y_n(\omega_0 + x_n^T \omega) + \frac{\rho}{2}\|\omega\|_2^2 - \frac{m}{2}\|\omega\|_2^2 - \frac{m}{2}\omega_0^2 \tag{1.19}$$

  Calculating the gradient $\nabla f(\omega, \omega_0)$:

$$\nabla f(\omega, \omega_0) = \begin{bmatrix} \frac{1}{N}\sum_{n=1}^{N} y_n x_n + \rho\omega - m\omega \\ \frac{1}{N}\sum_{n=1}^{N} y_n - m\omega_0 \end{bmatrix} \tag{1.20}$$

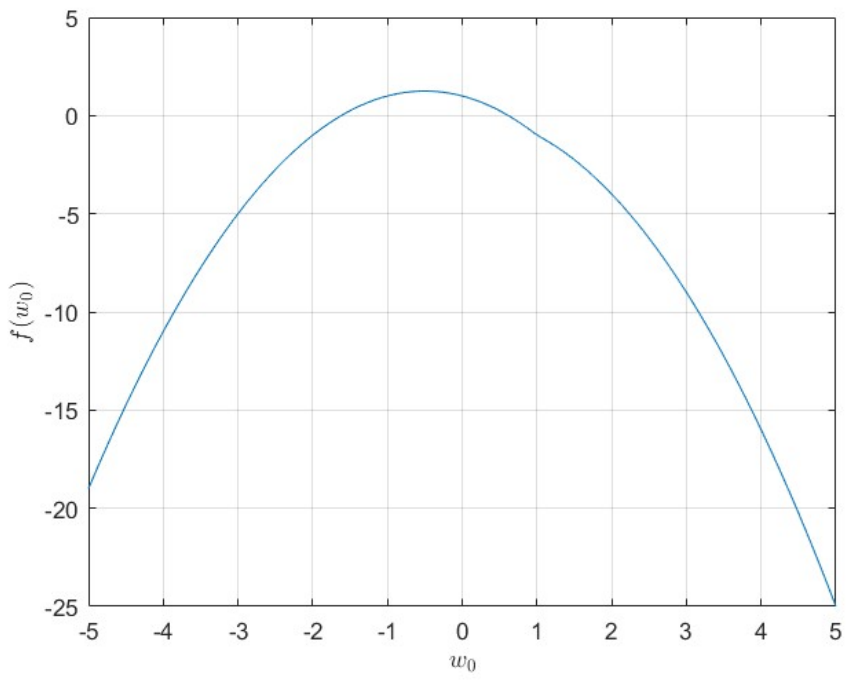


Figure 1.3: Plot of the function g for setting $\omega = \mathbf{0}$ and ω_0 variable.

Computing the Hessian matrix $\nabla^2 f(\omega, \omega_0)$:

$$\begin{pmatrix} \rho & 0 & \cdots & 0 & 0 \\ 0 & \rho & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \rho & 0 \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix} \tag{1.21}$$

Observation: The Hessian matrix is not positive definite.

Conclusion on convexity:

$$\nabla^2 f(\omega, \omega_0) \not\succeq 0 \quad \Rightarrow \quad \text{the function is not convex!} \tag{1.22}$$

Conclusion: This proofs the intuition from plot that the function in (1.12) is not strongly convex.

1.6 Task 6: Numerical Solution Using CVX

In this task, we aim to solve the following problem using the CVX toolbox.

$$\min_{w_0, w} \quad \frac{1}{N} \sum_{n=1}^{N} h\left(y_n \left(w_0 + \mathbf{x}_n^T w\right)\right) + \rho \|w\|_2^2, \tag{1.23}$$

$$\underbrace{\left(\frac{1}{N} \sum_{n=1}^{N} h\left(y_n \left(w_0 + \mathbf{x}_n^T w\right)\right)\right)}_{g_D(w_0, w)} + \underbrace{\rho \|w\|_2^2}_{r(w_0, w)}$$

Using the parameters (w_0, w) obtained from the solution of the previous problem , we evaluate the performance of the classifier $C_{w_0,w}(x)$, defined as:

$$C_{w_0,w}(x) = \text{sign}(w_0 + x^T w),$$

In the appendix, we present the code used to solve this problem, along with comments for clarity. After running the code, it is possible to obtain the following results for the case $\rho = 0.1$ and $\rho = 0.5$ respectably:

Results with $\rho = 0.1$
Status: Solved
Optimal value (cvx_optval): +0.0342602
Training error rate: 0.00%
Test error rate: 0.12%

Results with $\rho = 0.5$
Status: Solved
Optimal value (cvx_optval): +0.103117
Training error rate: 0.25%
Test error rate: 0.25%

Therefore, we can conclude that our analysis for task 6 is complete.

Note: You can achieve the same result using a for loop. Of course, using "tic-toc" command, you can see that this is not an optimal choice, since it requires more calculation time.

1.7 Task 7: Designing the worst attack to a feature vector

In this task you will be dealing with a problem of worst attack to a feature vector. Given the feature vector x as:

$$x = x_d = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad d = 1, \cdots, 3 \tag{1.24}$$

category y of a feature vector x, classifier parameters (ω_0, ω) and known that attacker can change every components of x in:

$$\tilde{x} = \tilde{x}_d = \begin{bmatrix} \tilde{x_1} \\ \tilde{x2} \\ \tilde{x_3} \end{bmatrix} \qquad d = 1, \cdots, 3 \tag{1.25}$$

$$\tilde{x} = [x_d - P; x_d + P] \quad \text{in this case} \qquad P = 0,5$$

the goal of the attacker is solve:

$$\begin{aligned} \min_{\tilde{x}} \quad & y(w_0 + \tilde{x}^T w) \\ \text{subject to} \quad & |\tilde{x}_d - x_d| \leq P, \quad \text{for} \quad 1 \leq d \leq D \end{aligned} \tag{1.26}$$

where P is the maximal deviation the attacker can introduce

The aims of this task were:
a) show that $\tilde{x} = x - P\,\text{sgn}(yw)$ solves the cost function above (1.26)
b) show that, for such $\tilde{x}$, (1.26) evaluates to $y(w_0 + x^T w) - P\|y\omega\|_1$

1.7.1 Analytical Proof

a) First of all, we can note that the function $y(w_0 + \tilde{x}^T w)$ can be written as:

$$y(w_0 + \tilde{x}^T w) = yw_0 + y\tilde{x}^T w = yw_0 + (y\tilde{x}_1^T w + \cdots + y\tilde{x}_d^T w) \tag{1.27}$$

Attacker goal is to minimize $y(w_0 + \tilde{x}^T w)$; we must therefore modify $\tilde{x}$ as to try to force this quantity to be negative. The optimal strategy to minimize (1.26) is to modify each component $\tilde{x}$ in such a way that the value of $x^T w$ is reduced as much as possible i.e. $\tilde{x}$ need to be chosen:

- as large as possible if $\operatorname{sgn}(yw) < 0$.
- as small as possible if $\operatorname{sgn}(yw) > 0$.

$\tilde{x}$ is in the interval $[x_d - P; x_d + P]$ and should be set to the marginal values of the interval depending on the sign of of $y\omega$. The value $\tilde{x}^T w$ is minimized when x takes the value farthest in the opposite direction of ω_0 i.e. when:

$$\tilde{x} = x - P\operatorname{sgn}(yw) = \tilde{x}\begin{bmatrix} x_1 - P\operatorname{sgn}(yw_1) \\ \cdots \\ x_d - P\operatorname{sgn}(yw_d) \end{bmatrix} \tag{1.28}$$

b) The cost function we have to minimize is (1.26). Knowing that:

$$\tilde{x} = x - P\operatorname{sgn}(yw) = \tilde{x} \tag{1.29}$$

we can we can rewrite (1.26) as follows;

$$\begin{aligned} &y(w_0 + \tilde{x}^T w) = \\ &y(w_0 + (x - P\operatorname{sgn}(yw))^T \omega) = \\ &y(w_0 + x^T w) - yP\operatorname{sgn}(yw)^T \omega = \\ &y(w_0 + x^T w) - P(\operatorname{sgn}(yw_1) \cdot yw_1 + \cdots + \operatorname{sgn}(yw_d) \cdot yw_d) \end{aligned} \tag{1.30}$$

now knowing the **sign property** according to which:

$$\operatorname{sgn}(yw) \cdot yw = |yw| \tag{1.31}$$

and **l_1−norm property**:

$$|yw_1| + \cdots + |yw_d| = \|yw\|_1 \tag{1.32}$$

(1.30) thus becomes:

$$y(w_0 + x^T w) - P\|yw\|_1 \tag{1.33}$$

1.8 Task 8

In this task, the data set was attacked by modifying the feature vector accordingly:

$$\tilde{x} = x - P\operatorname{sgn}(yw) \tag{1.34}$$

After the attack on the data set, you can see a strong increase in the error rate of the classifier ($\rho = 0.1$) :

attacked test dataset: 43.56%

1.9 Task 9

This task is about designing a classifier that is robust against attacks on the dataset. Therefore, the classifier needs to be retrained based on a different cost function that takes into account that the feature vector is manipulated.

For a hyperparameter configuration of $\rho = 0.1$, the robust classifier achieves the following error-rate:

train dataset: 0.75%

test dataset: 0.44%

attacked test dataset: 2.19%

These results indicate that the robust classifier is highly effective. The low error rates on both the training and test datasets, demonstrate good performance.Although the error rate increases to 2.19% on the attacked test dataset, it's still relatively low, suggesting that the classifier maintains resilience against adversarial manipulations. The result is really incredible if we compare it with the result obtained in task 8.

1.10 Task 10

In this task the goal is to fit a mixture of linear models to a nonlinear function. The mixture of linear models

$$\hat{y}(x) = \sum_{k=1}^{K} w_k(x)\hat{y}_k(x) \tag{1.35}$$

is a composition of linear functions

$$\hat{y}_k(x) = s_k x + r_k \tag{1.36}$$

, where each linear function is weighted by a weighting parameter:

$$w_k(x) = \frac{e^{u_k x + v_k}}{e^{u_1 x + v_1} + \cdots + e^{u_K x + v_K}}. \tag{1.37}$$

The mixture of linear mixture models is parametrized by the parameter vector

$$\theta = [s_1, r_1, \ldots, s_K, r_K, u_1, v_1, \ldots, u_{K-1}, v_{K-1}]^T \tag{1.38}$$

, which should be determine such that the objective function $f(\theta)$ is minimized:

$$\min f(\theta) = \sum_{n=1}^{N} \underbrace{(\hat{y}_n(\theta) - y_n)^2}_{=:f_n(\theta)}. \tag{1.39}$$

The loss function can be minimized using the Levenberg-Marquardt (LM) algorithm. To apply the algorithm, the gradient of the functions $f(\theta)$ and $f_n(\theta)$ need to be determine. The gradient of f can be simply calculated by applying the chain rule:

$$\nabla f(\theta) = 2\sum_{n=1}^{N} f_n(s,r,u,v)\nabla f_n(s,r,u,v). \tag{1.40}$$

To calculate the gradient of f_n, this can be done by looking at the partial derivatives

$$\frac{\partial f_n}{\partial s_i} = w_i x_n, \tag{1.41}$$

$$\frac{\partial f_n}{\partial r_i} = w_i, \tag{1.42}$$

$$\frac{\partial f_n}{\partial u_i} = \left(\sum_{k=1}^{K} -w_k w_i \hat{y}_k + w_i \hat{y}_i \right) x_n, \tag{1.43}$$

$$\frac{\partial f_n}{\partial v_i} = \sum_{k=1}^{K} -w_k w_i \hat{y}_k + w_i \hat{y}_i. \tag{1.44}$$

, where $i = 1, \dots, K$. The LM-algorithm was implemented in Matlab (see 1.5.1) and the algorithm stops when the gradient of f drops under $\epsilon = 10^{-4}$ or a maximum number of 5000 iterations is reached. In our case, the algorithm stopped after 78 iterations, where the following values for our optimization variables are obtained:

$$u = \begin{bmatrix} -13.0771 \\ -80.1045 \\ 88.2025 \end{bmatrix}, \quad v = \begin{bmatrix} -2.3460 \\ -415.5622 \\ -184.8455 \end{bmatrix}, \quad s = \begin{bmatrix} -1.2015 \\ 3.2655 \\ -3.9997 \\ 12.1257 \end{bmatrix}, \quad r = \begin{bmatrix} -0.0769 \\ 23.0588 \\ 19.8766 \\ -2.6533 \end{bmatrix}.$$

This results in the weighting parameters shown in the fig. 1.4, which clearly shows the sections in which the non-linear function is approximated by linear functions. The

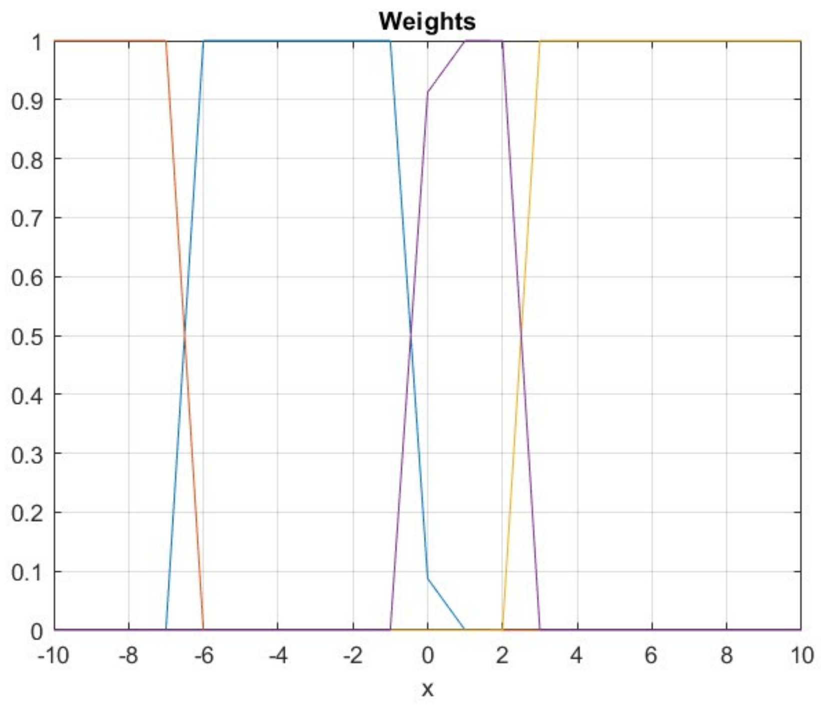
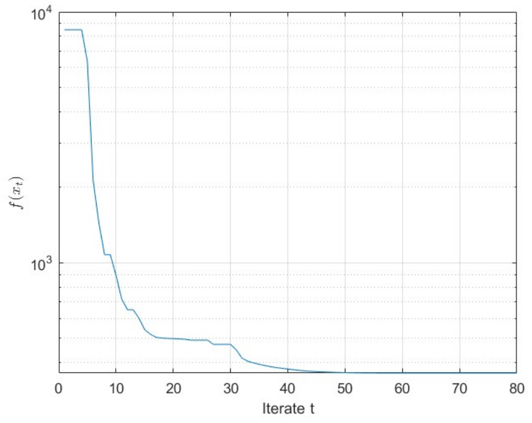
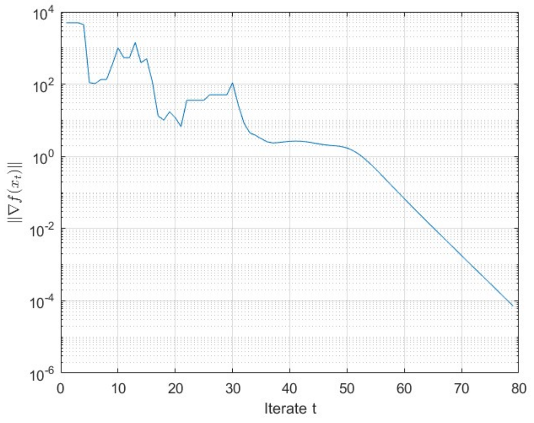


Figure 1.4: The $K = 4$ weight functions, each weight is plotted in a different color.

transient behavior of the loss function and the gradient of the the loss function are given in fig. 1.5. Finally, the result of the approximation of the nonlinear function y by the mixture of the linear function $\hat{y}$ is shown in fig. 1.6. It can be clearly seen that the approximation corresponds well to the underlying function.



(a)



(b)

Figure 1.5: The value (a) and the norm of the gradient (b) of the objective function of across the LM iterates.

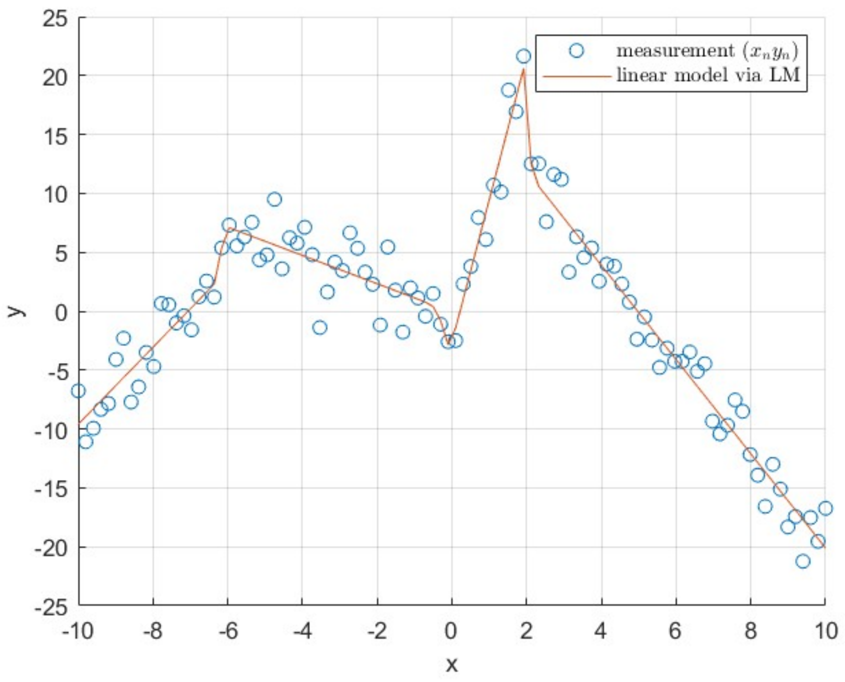


Figure 1.6: Approximation of the respective function using a mixture of linear models.

Chapter 2

Appendix: Matlab Code Task

2.0.1 Task 6 Code

Below, we present the code used to solve this problem, along with comments for clarity.

```matlab
%% setup the data
% load data from workspace
load("dataset/classifier_dataset.mat");

% define variables
Ntrain = size(traindataset, 1); D = 784;
Ntest = size(testdataset, 1);
n = D + 1;  % Corrected to D + 1
ytrain = trainlabels;
Xtrain = [traindataset ones(Ntrain, 1)];  % Adding bias term
ytest = testlabels;
Xtest = [testdataset ones(Ntest, 1)];  % Adding bias term

% configuration of the hyperparameters
rho = 0.1;

%% solving the convex optimization problem
cvx_begin quiet
    variable w(n)
    minimize(1/Ntrain * sum(max(0, 1 - ytrain .* (Xtrain * w)
       )) + rho * square_pos(norm(w(1:D))));
cvx_end

%% evaluation of the performance
% testing on trainset
eps = ytrain .* (Xtrain * w);
eps(eps>0) = 0;
eps(eps<0) = 1;
errtrain = 1/Ntrain * sum(eps);
% testing on testset
eps = ytest .* (Xtest * w);
eps(eps>0) = 0;
eps(eps<0) = 1;
errtest = 1/Ntest * sum(eps);

% return error
fprintf('Train error [%%]: %.2f\n', errtrain * 100);
fprintf('Test error [%%]: %.2f\n', errtest * 100);
```

2.0.2 Task 8 Code

Below, the code for resolution of task 8:

```
%% setup the data
% load data from workspace
load("dataset/classifier_dataset.mat");

% configuration of the hyperparameters
rho = 0.1;
P = 0.18;

% load w
load(['w_' num2str(rho)]);

% define variables
Ntrain = size(traindataset, 1); D = 784;
Ntest = size(testdataset, 1);
n = D + 1;  % Corrected to D + 1
ytrain = trainlabels;
ytest = testlabels;
Xtrain = traindataset;
Xtest_attack = testdataset;

% attack the dataset
Xtest_attack = Xtest_attack - P*sign(ytest*w(1:D)');

% adding bias term
Xtrain = [Xtrain ones(Ntrain, 1)];
Xtest_attack = [Xtest_attack ones(Ntest, 1)];

%% evaluation of the performance
% testing on trainset
eps = ytrain .* (Xtrain * w);
eps(eps>0) = 0;
eps(eps<0) = 1;
errtrain = 1/Ntrain * sum(eps);
% testing on testset
eps = ytest .* (Xtest_attack * w);
eps(eps>0) = 0;
eps(eps<0) = 1;
```

```
errtest = 1/Ntest * sum(eps);

% return error
fprintf('Train error [%%]: %.2f\n', errtrain * 100);
fprintf('Test error [%%]: %.2f\n', errtest * 100);
```

2.0.3 Task 9 Code

The code for Task 9 for training the model and performance evaluation is given below.

```
%% setup the data
% load data from workspace
load("dataset/classifier_dataset.mat");

% configuration of the hyperparameters
rho = 0.1;
P = 0.18;

% define variables
Ntrain = size(traindataset, 1);
D = 784;
Ntest = size(testdataset, 1);
ytrain = trainlabels;
ytest = testlabels;
Xtrain = traindataset;
Xtest = testdataset;

% Adding bias term
Xtrain = [Xtrain ones(Ntrain, 1)];
Xtest = [Xtest ones(Ntest, 1)];

%% solving the convex optimization problem
cvx_begin quiet
    variable w(D+1)
     minimize(1/Ntrain * sum(max(0, 1 - (ytrain .* (Xtrain *
        w) - P * abs(ytrain)*abs(w(1:D)')*ones(D,1)))) + rho
        * square_pos(norm(w(1:D))));
cvx_end

%% Attacking
%attack the dataset
Xtest_attack = testdataset;
Xtest_attack = Xtest_attack - P*sign(ytest*w(1:D)');
Xtest_attack = [Xtest_attack ones(Ntest, 1)];

%% evaluation of the performance
% testing on trainset
eps = ytrain .* (Xtrain * w);
```

```
eps(eps>0) = 0;
eps(eps<0) = 1;
errtrain = 1/Ntrain * sum(eps);
% testing on testset
eps = ytest .* (Xtest * w);
eps(eps>0) = 0;
eps(eps<0) = 1;
errtest = 1/Ntest * sum(eps);
% testing on attacked data
eps = ytest .* (Xtest_attack * w);
eps(eps>0) = 0;
eps(eps<0) = 1;
errtest_attack = 1/Ntest * sum(eps);

% return error
fprintf('Train error [%%]: %.2f\n', errtrain * 100);
fprintf('Test error [%%]: %.2f\n', errtest * 100);
fprintf('Test error attack [%%]: %.2f\n', errtest_attack *
    100);
```

2.0.4 Task 10 Code

```
%% setup the data
% load data from workspace
load("dataset/lm_dataset_task.mat");

% initilize parameters
K = size(s,1);
N = size(y,1);
n = 14;
lambda = 1;
eps = 1e-4;
iter = 5000; %5000
cnt = 0;

% Calculate initial values
f = 0;
gradf = 0;
for j = 1:N
    [fp, gradfp] = func_eval(x(j),y(j), r, s, v, u);
    f = f + fp^2;
    gradf = gradf + 2*fp * gradfp;
end

% log data
history.f = f;
history.lambda = lambda;
history.gradf = norm(gradf);
history.f(end+1) = f;

%% LM algorithm
while norm(gradf) > eps & cnt < iter
    % constructing Least-Square problem
    A = zeros(N+n,n);
    b = zeros(N+n,1);
    for i = 1:N
        [fp, gradfp] = func_eval(x(i),y(i), r, s, v, u);
        A(i,:) = gradfp;
        b(i) = gradfp'* [r;  s;  v;  u] - fp ;
    end
    A(N+1:end,:) = lambda^0.5 * eye(n);
    b(N+1:end) = lambda^0.5 * [r;  s;  v;  u];

    % solving Least-Square problem
    theta = A\b;
    x_r = theta(1:4); x_s = theta(5:8); x_v = theta(9:11);
       x_u = theta(12:14);

    % evaluation function at new value
    fnew = 0;
    gradfnew = 0;
    for j = 1:N
            %fnew = fnew + func(x(j),y(j), x_r, x_s, x_v, x_u
               )^2;
            [fp, gradfp] = func_eval(x(j),y(j), x_r, x_s, x_v
               , x_u);
            fnew = fnew + fp^2;
            gradfnew = gradfnew + 2*fp * gradfp;
    end

    % check update parameters
    if fnew < f
        % update function value
        f = fnew;
        % update parameters
        r = x_r; s = x_s; v = x_v; u = x_u;
        % update new gradient
        gradf = gradfnew;
        % decrease step size
        lambda = 0.7 * lambda;
    else
        lambda = 2 * lambda;
    end
     % log data
     history.f(end+1) = f;
     history.gradf(end+1) = norm(gradf);
     history.lambda(end+1) = lambda;
     cnt = cnt + 1;
     disp(['iteration: ', num2str(cnt), ', loss: ', num2str(
        history.f(end)), ', norm gradient: ', num2str(history
        .gradf(end))]);
end
% log final parameters
history.r = r;
history.s = s;
history.v = v;
history.u = u;

function [f, gradf] = func_eval(x, y, r, s, v, u)
    u = [u ; 0];
    v = [v; 0];

    % Calculate alpha
    alpha = u * x + v;
    alpha_bar = max(alpha);

    % Calculate w_k(x_n)
    wx_numer = exp(alpha - alpha_bar);
    wx_denom = sum(wx_numer);
    w_k = wx_numer / wx_denom;

    % Calculate y_hat_k
    y_hat_k = s * x + r;

     % Calculate the overall y_hat(x_n)
     y_hat = sum(w_k .* y_hat_k);

     % Compute the error term (y_hat - y_n)
     f = y_hat - y;

        % Comupte the gradients
    grad_r = w_k;
    grad_s = grad_r * x;
    grad_v = sum(-w_k .* y_hat_k * w_k', 1)' + w_k .* y_hat_k
       ;
    grad_u = grad_v * x;

    grad_u(end) = [];
    grad_v(end) = [];
    gradf = [grad_r; grad_s; grad_v; grad_u];

end
```

References

1. `https://github.com/Linux-99/Optimization-and-Algorithms-Group-IST-` (Group's Github Project page)

2. "Slides Optimization and Algorithms," 2024, Team: Joao Xavier, Joao Sequeira, Hugo Pereira

3. `https://cvxr.com/cvx/` (CVX: MATLAB Software for Disciplined Convex Programming) (Needed to solve the tasks)

4. "Optimization and Algorithms: Proofs of the Theorems in the Slides," 2024, João Xavier.

5. "Introduction to NonLinear Optimization," Amir Beck, (Chapter 7 "Convex Functions"; 7.4 (Operations Preserving Convexity))