Project of Optimization and Algorithms

João Xavier
Instituto Superior Técnico
September 2024

Contents

1 Building a classifier robust against adversarial attacks

A binary classifier is a system that sorts data into one of two categories. For example, it might be used to identify whether a given image of an handwritten digit corresponds to the digit 0 or the digit 1.

If we represent the two categories as the numbers -1 and 1, we can view a binary classifier as a map $C : \mathbf{R}^D \to \{\pm 1\}$ that sorts an input data vector $x \in \mathbf{R}^D$ into an output category label $y \in \{\pm 1\}$, where $y = C(x)$.

Classifiers. In this section, we focus on classifiers of the form

$$C_{w_0,w}(x) = \mathbf{sign}(w_0 + x^T w), \tag{1}$$

where

$$\mathbf{sign}(u) = \begin{cases} 1 & , \text{ if } u \geq 0 \\ -1 & , \text{ if } u < 0. \end{cases}$$

The number $w_0 \in \mathbf{R}$ together with the vector

$$w = \begin{bmatrix} w_1 \\ \vdots \\ w_D \end{bmatrix} \in \mathbf{R}^D,$$

determine the classifier: different choices of (w_0, w) determine different classifiers.

How to find the ideal classifier for a given task? Commonly, the parameters (w_0, w) are learned from a given dataset $\mathcal{D}$ of examples, say,

$$\mathcal{D} = \{(x_n, y_n) : \ 1 \leq n \leq N\},$$

an example being a pair $(x_n, y_n) \in \mathbf{R}^D \times \{\pm 1\}$ that tells us that the feature vector x_n should be preferably sorted into the category y_n.

1.1 The ideal classifier

Learning the ideal classifier from the dataset means finding the classifier that makes the fewest number of mistakes on the dataset, a mistake occurring whenever the classifier outputs a label that disagrees with the indicated one, that is, when $C_{w_0,w}(x_n) \neq y_n$, or, equivalently, when

$$y_n C_{w_0,w}(x_n) < 0.$$

Finding the best classifier thus corresponds to solving

$$\underset{w_0,w}{\text{minimize}} \quad \underbrace{\frac{1}{N}\sum_{n=1}^{N} \mathbf{1}_{\mathbf{R}_-}\left(y_n C_{w_0,w}(x_n)\right)}_{f_{\mathcal{D}}(w_0,w)}, \tag{2}$$

where $\mathbf{1}_{\mathbf{R}_-} : \ \mathbf{R} \to \mathbf{R}$ is the function

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

Note that the function $f_{\mathcal{D}} : \ \mathbf{R} \times \mathbf{R}^D \to \mathbf{R}$ gives the average number of mistakes that the classifier determined by $(w_0, w) \in \mathbf{R} \times \mathbf{R}^D$ makes on the dataset $\mathcal{D}$.

Task 1. [Theoretical task] Show that the function $f_{\mathcal{D}}$ defined in (2) is not convex. It is sufficient that you show this for the simple case $N = 1$ and $D = 1$.

1.2 The hinge-loss classifier

Given that the optimization problem (2) is non-convex, solving it is challenging. A typical approach in such scenarios is to substitute it with a convex optimization problem that approximates the original problem.

To develop such approximate convex problem, we carry out three steps:

- We start by noting that

$$\mathbf{1}_{\mathbf{R}_-}\left(y_n C_{w_0,w}(x_n)\right) = \mathbf{1}_{\mathbf{R}_-}\left(y_n(w_0 + x_n^T w)\right); \tag{3}$$

  (You should take some minutes to check that this equality is indeed true.)

- Next, we approximate the function $\mathbf{1}_{\mathbf{R}_-}$ by the function h, where $h : \mathbf{R} \to \mathbf{R}$ is given by

  $$h(u) = (1-u)_+.$$

  The function h is often called the hinge loss.

Task 2. [Theoretical task] Show that the function $\mathbf{1}_{\mathbf{R}_-}$ is majorized by h, that is, show that $\mathbf{1}_{\mathbf{R}_-}(u) \le h(u)$ holds for all $u \in \mathbf{R}$. Furthermore, show that h is a convex function.

Such replacement yields the optimization problem

$$\underset{w_0,w}{\text{minimize}} \quad \underbrace{\frac{1}{N}\sum_{n=1}^{N} h\left(y_n(w_0 + x_n^T w)\right)}_{g_{\mathcal{D}}(w_0,w)}. \tag{4}$$

The function $g_{\mathcal{D}} : \mathbf{R} \times \mathbf{R}^D \to \mathbf{R}$ thus gives an upper-bound on the average number of mistakes that the classifier determined by $(w_0, w) \in \mathbf{R} \times \mathbf{R}^D$ makes on the dataset $\mathcal{D}$.

Task 3. [Theoretical task] Show that the function $g_{\mathcal{D}}$ defined in (4) is convex for any N and D (and not only for the special case $N = 1$ and $D = 1$).

- Finally, although (4) is already a convex problem, we add an extra term that penalizes large values in w, thus arriving at our final formulation:

  $$\underset{w_0,w}{\text{minimize}} \quad \underbrace{\underbrace{\frac{1}{N}\sum_{n=1}^{N} h\left(y_n(w_0 + x_n^T w)\right)}_{g_{\mathcal{D}}(w_0,w)} + \rho \underbrace{\|w\|_2^2}_{r(w_0,w)}}_{g(w_0,w)}, \tag{5}$$

  where $\rho > 0$ is a positive constant and $\|\cdot\|_2$ is the Euclidean norm ($\|u\|_2 = \sqrt{u^T u}$). The extra term r is known as a regularization term. (This term is primarily added due to techniques in machine learning, where ρ acts as a hyper-parameter to prevent the classifier from overfitting—though this concern is not relevant here.)

Task 4. [Theoretical task] Show that the function g is convex for any N and D (and not only for the special case $N = 1$ and $D = 1$).

Task 5. [Theoretical task] Is the function g strongly convex? Hint: for fixed w, investigate the function $w_0 \mapsto g(w_0, w)$.

Your next task is to solve numerically problem (5). For this, start by installing the numerical solver CVX from `http://cvxr.com/cvx`.

Next, download the Matlab file `classifier_dataset.mat` from the course webpage. This file contains the following four variables:

- `traindataset`: This is a matrix of size $N \times D$, where $N = 400$ and $D = 784$. The nth row of this matrix corresponds corresponds to x_n^T in (5).

  We obtained each such x_n by flattening a 28×28[1] image from the MNIST dataset, which contains images of handwritten digits. We consider only images of the digits 0 and 1.

  As illustrative examples, Figure 1 displays an image of the MNIST dataset that corresponds to the digit 0, whereas Figure 2 displays an image of the digit 1. You can

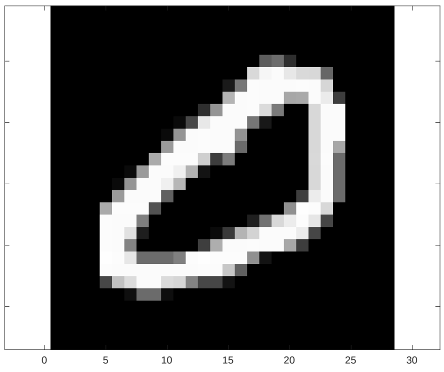

Figure 1: Image of an handwritten 0

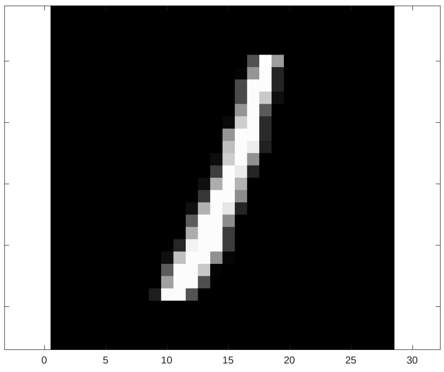

Figure 2: Image of an handwritten 1

[1] To flatten a matrix means to reshape it as a vector by stacking its columns from left to right. For example, flattening the matrix $X = \begin{bmatrix} 4.5 & -0.7 \\ 3.2 & 9.1 \end{bmatrix}$ gives the vector $\begin{bmatrix} 4.5 \\ 3.2 \\ -0.7 \\ 9.1 \end{bmatrix}$.

similarly visualize any row of the matrix traindataset by passing it to the following function:

```matlab
function show_im(x)
    image(rescale(reshape(x,28,28),0,255));
    axis square equal;
    colormap(gray)
end
```

- trainlabels: This is a vector of length N. The nth component of this vector corresponds to y_n in (5) and indicates the category of the nth row of the matrix traindataset. (We chose to represent the category of digit 0 as $y_n = -1$ and the category of digit 1 as $y_n = 1$.)

- testdataset: This is a matrix of size 1600×784, each row containing a flattened image of an handwritten digit 0 or 1. The images in this matrix are different from those included in traindataset. This matrix will be used to validate the extrapolation capacity of our classifiers.

- testlabels: This is a vector of length 1600. The nth component of this vector indicates the category of the nth row of the matrix testdataset.

Task 6. [Numerical task] Use CVX to solve problem (5) where x_n and y_n denote the nth row of the matrix traindataset and the nth component of the vector trainlabels, respectively. Use $\rho = 0.1$.

With the parameters (w_0, w) thus obtained, assess the performance of the corresponding classifier $C_{w_0,w}$ defined in (1) on the training dataset and the test dataset: specifically, evaluate the function $f_{\mathcal{D}}$ defined in (2) by first considering that (x_n, y_n) come from traindataset and trainlabels, which gives the classifier error rate on the training dataset; next, consider that (x_n, y_n) come from testdataset and testlabels, which gives the classifier error rate on the test dataset.

So that you can check your code, we now give such values of $f_{\mathcal{D}}$ for the case $\rho = 0.5$: for the training dataset, $f_{\mathcal{D}}$ evaluates to approximately 0.25%, and, for the test dataset, $f_{\mathcal{D}}$ evaluates to 0.25%.

Hint: when implementing (5) in CVX, you should try to avoid building the objective function with a for loop and use instead a compact vector notation. For example, suppose you were interested in solving a problem of the form

$$\underset{u}{\text{minimize}} \quad \underbrace{\sum_{k=1}^{K} (a_k^T u - b_k)_+}_{f(u)}, \tag{6}$$

where $a_k \in \mathbf{R}^K$ and $b_k \in \mathbf{R}$ are given. You could build the objective function in CVX with a `for` loop as follows:

```
% constructing f with a for loop
f = 0
for k = 1:K
        f = f + pos( A(k,:)*x - b(k) );
end
```

Here, we assume that the matrix A contains a_k^T as its kth row, and the vector b contains b_k as its kth component. However, a better approach is to note that the objective function f in this example can be written compactly as

$$f(u) = \mathbf{1}_K^T (Au - b)_+,$$

where $\mathbf{1}_K$ is a vector of length K with all components equal to 1, and the operator $(\cdot)_+$ acts component-wise on vectors (that is, if $u = (u_1, u_2, u_3)$, then $u_+ = ((u_1)_+, (u_2)_+, (u_3)_+)$. This compact version can be coded directly in CVX as follows:

```
% a better way: constructing f without a for loop
f = sum(pos(A*x -b));
```

1.3 Designing the worst attack to a feature vector

Consider now an attacker that knows the classifier parameters (w_0, w), knows the category y of a feature vector x, and has the power to change the feature vector x *before* it is presented to the classifier.

We assume the attacker can change every component of x, but only up to a maximal deviation of $P \geq 0$ in each component. To illustrate, consider $D = 3$, $P = 0.5$, and let

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

be the original feature vector. In this case, the attacker can transform the feature vector into the attacked feature vector

$$\widetilde{x} = \begin{bmatrix} \widetilde{x}_1 \\ \widetilde{x}_2 \\ \widetilde{x}_3 \end{bmatrix},$$

where $\widetilde{x}_d$ can be any number in the interval $[x_d - 0.5; x_d + 0.5]$, for $d = 1, 2, 3$. It is this transformed vector $\widetilde{x}$ that is seen by the classifier.

The goal of the attacker is to design $\widetilde{x}$ so as to try to induce a mistake in the classifier $C_{w_0,w}$. Because the classifier makes a mistake whenever the quantity

$$y(w_0 + \widetilde{x}^T w) \tag{7}$$

is negative (recall (3)), the goal of the attacker is thus to solve

$$\begin{array}{ll} \underset{\widetilde{x}}{\text{minimize}} & y(w_0 + \widetilde{x}^T w) \\ \text{subject to} & |\widetilde{x}_d - x_d| \leq P, \quad \text{for } 1 \leq d \leq D. \end{array} \tag{8}$$

Task 7. [Theoretical task] (a) Show that $\widetilde{x} = x - P\mathbf{sgn}(yw)$ solves (8), where the function $\mathbf{sgn}(\cdot)$ acts component-wise on vectors (that is, if $u = (u_1, u_2, u_3)$, then $\mathbf{sgn}(u) = (\mathbf{sgn}(u_1), \mathbf{sgn}(u_2), \mathbf{sgn}(u_3))$. (b) For such $\widetilde{x}$, show that the cost function in (8) evaluates to $y(w_0 + x^T w) - P\|yw\|_1$, where $\|\cdot\|_1$ denotes the ℓ_1-norm (for a vector $u = (u_1, \ldots, u_D)$, we have $\|u\|_1 = |u_1| + \cdots + |u_D|$.)

Task 8. [Numerical task] From now on, take $P = 0.18$. Using the result of Task 7, attack the test dataset by replacing each example (x_n, y_n) in that dataset with the corresponding attacked example $(\widetilde{x}_n, y_n)$. Then, assess the performance of the classifier from Task 6 in this attacked dataset: specifically, evaluate the function $f_{\mathcal{D}}$ defined in (2) but with (x_n, y_n) replaced by $(\widetilde{x}_n, y_n)$, which gives the classifier error rate on the attacked test dataset.

So that you can check your code, we now give such value of $f_{\mathcal{D}}$ for the case $\rho = 0.5$: for the attacked test dataset, $f_{\mathcal{D}}$ evaluates to 21.9%.

1.4 Developing a classifier that reduces the impact of attacks

As Task 8 shows, the classifier built in Task 6 is very vulnerable to adversarial attacks. How can we develop a classifier that is more robust to attacks?

We can reason as follows. For an unattacked feature vector x, the classifier $C_{w_0,w}$ in (1) makes a mistake whenever

$$y(w_0 + x^T w) < 0,$$

where y denotes the category of x. This is why $g_{\mathcal{D}}$ in (5) is trying to prevent the quantity $y_n(w_0 + x_n^T w)$ from becoming negative, across the training dataset.

Now, in our setup the attacker accesses x before the classifier, replacing it by $\widetilde{x}$ given in part (a) of Task 7. The classifier acts thus only on $\widetilde{x}$ and makes a mistake whenever

$$y(w_0 + \widetilde{x}^T w) < 0.$$

This means that the classifier now wants to prevent $y_n(w_0 + \widetilde{x}_n^T w)$ from becoming negative, across the training dataset.

Recalling that part (b) of Task 7 tells us that $y_n(w_0 + \widetilde{x}_n^T w) = y_n(w_0 + x_n^T w) - P \left\| y_n w \right\|_1$, we are led to the optimization problem

$$\underset{w_0, w}{\text{minimize}} \quad \frac{1}{N} \sum_{n=1}^{N} h \left(y_n(w_0 + x_n^T w) - P \left\| y_n w \right\|_1 \right) + \rho \left\| w \right\|_2^2 . \tag{9}$$

Task 9. [Numerical task] Use CVX to solve problem (9) where x_n and y_n denote the nth row of the matrix `traindataset` and the nth component of the vector `trainlabels`, respectively. Use $\rho = 0.1$.

With the parameters (w_0, w) thus obtained, assess the performance of the corresponding classifier $C_{w_0,w}$ defined in (1) on the training dataset and original test dataset: specifically, evaluate the function $f_{\mathcal{D}}$ defined in (2) by first considering that (x_n, y_n) come from `traindataset` and `trainlabels`, which gives the classifier error rate on the training dataset; next, consider that (x_n, y_n) come from `testdataset` and `testlabels`, which gives the classifier error rate on the test dataset. Finally, and more importantly, assess the performance of this new classifier in the attacked dataset that you generated in Task 8.

Comment these results by comparing them with the results you obtained in Tasks 6 and 8.

So that you can check your code, we now give such values of $f_{\mathcal{D}}$ for the case $\rho = 0.5$: for the training dataset, $f_{\mathcal{D}}$ evaluates to approximately 0.25%, for the test dataset, $f_{\mathcal{D}}$ evaluates to 0.19%, and for the attacked test dataset, $f_{\mathcal{D}}$ evaluates to 3.69%.

2 Fitting a piecewise-linear signal to measurements

Let $\mathcal{D} = \{(x_n, y_n) : \ 1 \leq n \leq N\}$ be noisy measurements of a signal, where $x_n \in \mathbf{R}$ represents a time instant and $y_n \in \mathbf{R}$ the corresponding measurement of the signal.

Fitting a linear signal. If the signal is nearly linear (that is, it looks as a straight line when plotted), then fitting a linear model to it may be a good way to denoise the signal. To fit such linear model, one would solve

$$\underset{s, r}{\text{minimize}} \quad \sum_{n=1}^{N} \left(s x_n + r - y_n \right)^2 ,$$

where the variables to optimize are the slope $s \in \mathbf{R}$ and the intercept at the origin $r \in \mathbf{R}$ of the linear model.

In terms of notation, it is advantageous to re-write this problem as

$$\underset{s, r}{\text{minimize}} \quad \sum_{n=1}^{N} \left(\widehat{y}(x_n) - y_n \right)^2 , \tag{10}$$

where, for a generic x, we have

$$\widehat{y}(x) = sx + r \tag{11}$$

being the output of the linear model at x. Note that the variables to optimize in problem (10) are still s and r; they are simply not directly visible, being embedded within the notation $\widehat{y}$.

Fitting a piecewise-linear signal. Some signals, however, are poorly explained by a single line, being instead composed by multiple linear pieces. An example of such a signal is given in Figure 3.

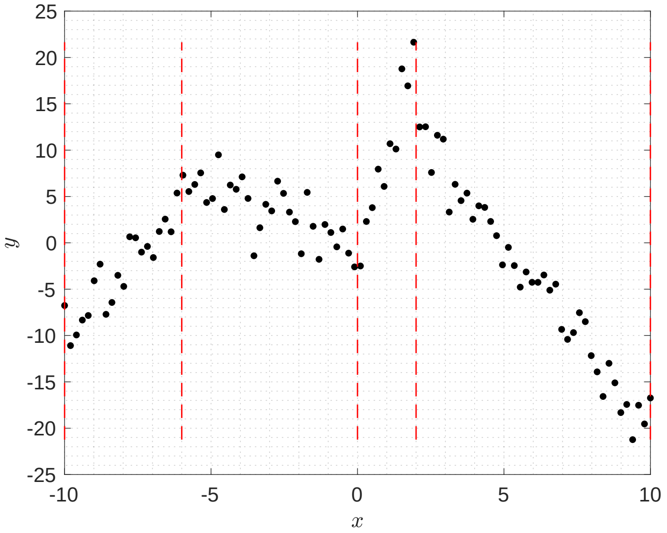

Figure 3: A black dot in the graph represents a measurement (x_n, y_n) of a signal composed of four linear segments. For clarity, the start and end of each segment are marked by a dashed red line. While the signal behaves locally in an almost linear manner within each segment, it does not exhibit linear behavior globally.

Measurements such as those in Figure 3 can be effectively denoised by fitting a piecewise-linear signal to the given data, which is the objective we will now address.

2.1 Mixture of linear models

For simplicity, we assume from now on that we know the number of linear pieces in the underlying signal, which we denote by K. For example, for the signal in Figure 3, we have $K = 4$.

Even with the knowledge of K, fitting the linear segments is challenging because the switching points—where one segment ends and another begins—are not known and can be difficult to determine due to the noise in the measurements. If these switching points were provided, fitting the segments would just reduce to K individual tasks of fitting a single line to disjoint intervals of the time axis.

To address all these issues, we pass from fitting a single linear model, as in (11), to fitting a mixture of linear models, as follows:

$$\widehat{y}(x) = \sum_{k=1}^{K} w_k(x)\widehat{y}_k(x). \tag{12}$$

Here,

$$\widehat{y}_k(x) = s_k x + r_k$$

represents the output of the individual kth linear model at x, and

$$w_k(x) = \frac{e^{u_k x + v_k}}{e^{u_1 x + v_1} + \cdots + e^{u_K x + v_K}} \tag{13}$$

represents the weight given to the kth linear model in forming the combined output $\widehat{y}(x)$. Thus, each linear model $\widehat{y}_k$ is parameterized by two variables, s_k and r_k, and the set of weights $w_1, \ldots, w_k$ are parameterized by $2K$ variables, u_1 to u_K and v_1 to v_K.

Weights. Note that the weights $w_1, \ldots, w_K$ are positive and sum to one; hence, the output of the new model $\widehat{y}$ is a convex mixture of the outputs of the basic linear models $\widehat{y}_1, \ldots, \widehat{y}_K$. More importantly, the weights are themselves a function of x. This is a key property. Indeed, if the weights were constant, then (12) would collapse to a single, global linear model of the form (11).

To understand the role played by the weights, consider an x such that the quantity $u_k x + v_k$ is much larger than $u_l x + v_l$ for $l \neq k$. This implies that the single term $e^{u_k x + v_k}$ dominates the denominator in (13), leading to $w_k(x) \simeq 1$ and $w_l(x) \simeq 0$ for $l \neq k$. That is, $\widehat{y}(x) \simeq \widehat{y}_k(x)$, which means that for *that* x the linear model $\widehat{y}_k$ is "switched on", while the remaining models $\widehat{y}_l$ are "switched off". The variables $u_1, v_1, \ldots, u_K, v_K$ that parameterize the weights $w_1, \ldots, w_K$ thus serve to determine how the basic linear models $\widehat{y}_1, \ldots, \widehat{y}_K$ are to be switched along the x axis.

Let δ_u and δ_v be arbitrary real numbers. It is easy to check that, if each u_k is replaced by $u_k + \delta_u$ and v_k by $v_k + \delta_v$, the weight functions $w_1, \ldots, w_K$ remain the same. To get rid of this redundancy in the parameterization of the weights, we consider from now on that $u_K = v_K = 0$. This reduces the parameters of the weights to the $2(K-1)$ variables $u_1, v_1, \ldots, u_{K-1}, v_{K-1}$.

2.2 Fitting the mixture to data

We thus arrive at the optimization problem

$$\underset{\substack{s_1, r_1, \ldots, s_K, r_K \\ u_1, v_1, \ldots, u_{K-1}, v_{K-1}}}{\text{minimize}} \quad \textstyle\sum_{n=1}^{N} \left(\widehat{y}(x_n) - y_n\right)^2, \tag{14}$$

where $\widehat{y}$ is given by (12).

Task 10. [Numerical task] Solve the optimization problem (14) by coding yourself the Levenberg–Marquardt (LM) method described in the slides of the course. In particular, you need to find the gradient of the objsective function analytically. For LM, start with $\lambda_0 = 1$ and stop the iterations when the norm of the gradient of f falls below $\epsilon = 10^{-4}$ or a maximum of 5000 iterations is reached.

This task uses $K = 4$ and $N = 100$ an the measurements can be found in the file `lm_dataset_task.mat`, which contains the following column vectors:

- `x` of length N with the measurements $\{x_n : 1 \leq n \leq N\}$;
- `y` of length N with the measurements $\{y_n : 1 \leq n \leq N\}$;
- `u` of length $K-1$ that contains values of $\{u_k : 1 \leq k \leq K\}$;
- `v` of length $K-1$ that contains values of $\{v_k : 1 \leq k \leq K\}$.
- `s` of length K that contains values of $\{s_k : 1 \leq k \leq K\}$;
- `r` of length K that contains values of $\{r_k : 1 \leq k \leq K\}$;

Use the vectors `u`, `v`, `s`, and `r` above as the initialization of your LM iterates.

Once your LM algorithm has converged, report the values of the vectors u, v, s, and r, to which LM converged. Also, plot the obtained linear mixture (12) and the weights as a function of the time instants x_n in the dataset. Finally, plot both the objective function of (14) and its gradient across the LM iterates. Comment the results you observe.

An important numerical technique. Suppose you are given real numbers $\alpha_1, \ldots, \alpha_K$ and you want to compute

$$\frac{e^{\alpha_k}}{e^{\alpha_1} + \cdots + e^{\alpha_K}},$$

for some $1 \leq k \leq K$. Such kind of computation appear, for example, in (13). If some α_i is large enough, the exponential of such α_i may overflow. A more robust technique is as follows: start by computing the maximum value of the α's, say, $\overline{\alpha} = \max\{\alpha_1, \ldots, \alpha_K\}$; then, compute

$$\frac{e^{\alpha_k - \overline{\alpha}}}{e^{\alpha_1 - \overline{\alpha}} + \cdots + e^{\alpha_K - \overline{\alpha}}}.$$

A solution for checking your code. So you can check your code, we give the solution for the dataset `lm_dataset_check.mat`. For the initialization in that file, LM converged to

$$u = \begin{bmatrix} -161.4 \\ 82.9 \\ -97.5 \end{bmatrix}, \quad v = \begin{bmatrix} -170.1 \\ -425.9 \\ 107.1 \end{bmatrix}, \quad s = \begin{bmatrix} -3.6 \\ -0.57 \\ 0 \\ -10.1 \end{bmatrix}, \quad r = \begin{bmatrix} 23.5 \\ 14.0 \\ 34 \\ 51.2 \end{bmatrix}.$$

Figure 4 gives the fitted signal, while Figure 5 gives the weights.

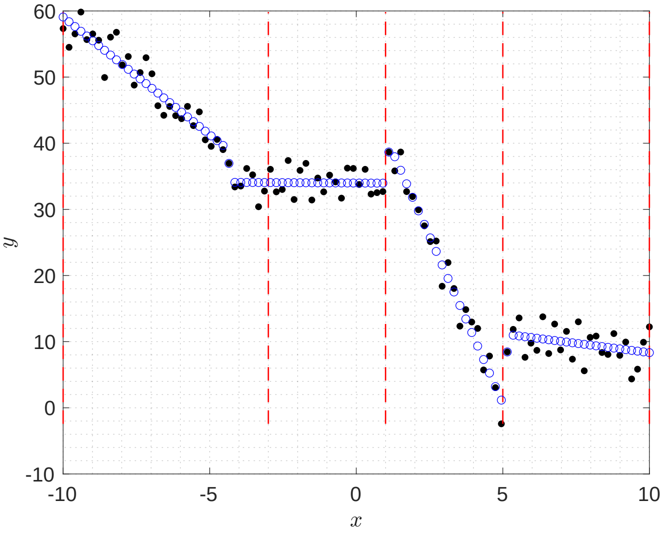

Figure 4: A black dot in the graph represents a measurement (x_n, y_n) in `lm_dataset_check.mat`. The blue circles represent the combination of linear models found by LM.

Figures 6 and 7 give the values and the norm of the gradient of the objective function of (14), across the LM iterates.

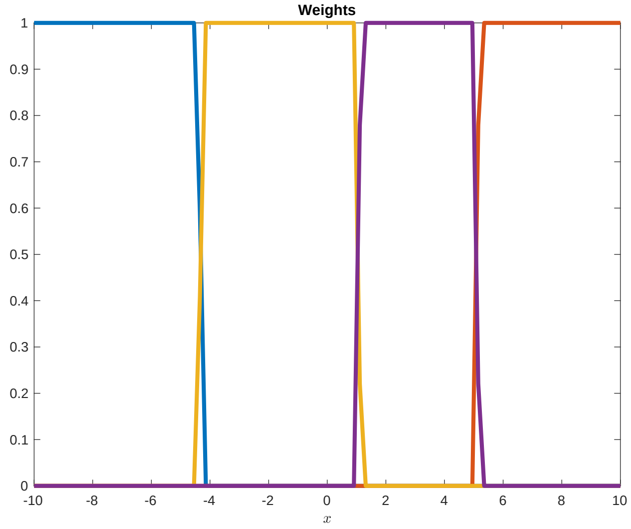

Figure 5: The $K = 4$ weight functions found by LM for the data in `lm_dataset_check.mat`. Each weight is plotted in a different color.

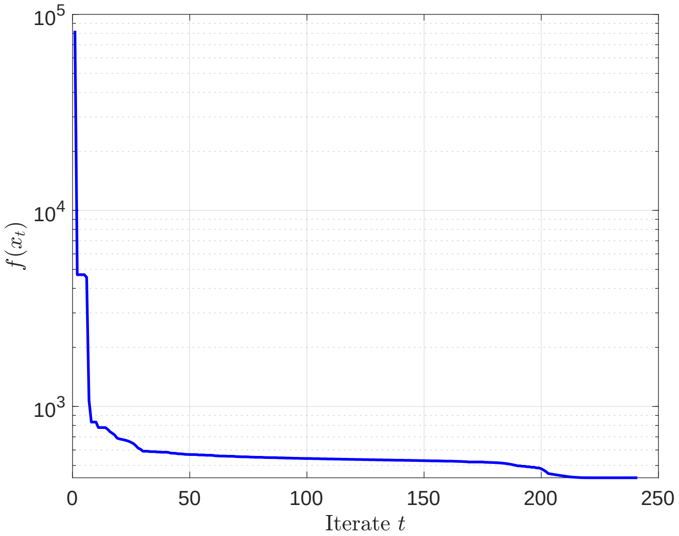

Figure 6: Values of the objective function of (14) across the LM iterates, for the data in `lm_dataset_check.mat`.

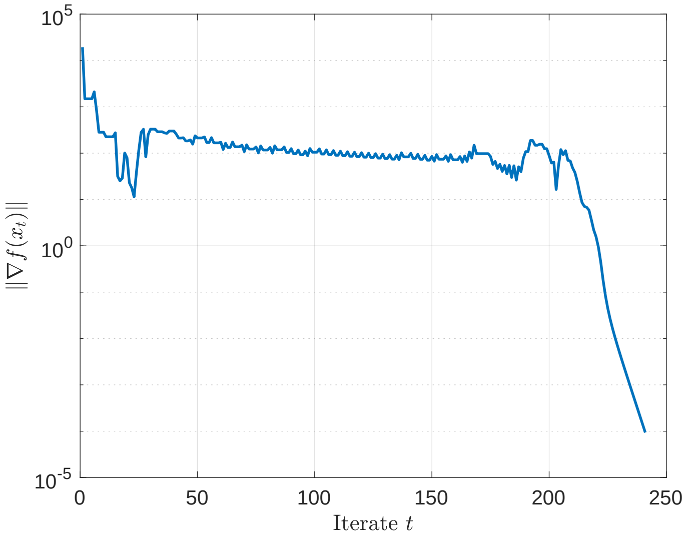

Figure 7: The norm of the gradient of the objective function of (14) across the LM iterates, for the data in `lm_dataset_check.mat`.