UNIVERSITÀ DI PISA

Department of Computer Science

Master programme in Data Science and Business Informatics

Optimization for Data Science
Project report
Extreme Learning Optimization

Montinaro Aldo
Poiani Marco

A.A. 2023/2024

Contents

Chapter 1

Introduction

1.1 Background

The breakthrough in the field of artificial intelligence research came around 2010 with the extensive use of Deep Neural Networks (DNNs). The structure of DNNs is exactly the one based on the idea introduced by McCulloch and Pitts [5]: it is structured with layers of nodes, commonly referred to as *artificial neurons*, which include an input layer, one or more hidden layers, and an output layer. Each node is interconnected with others and possesses its own unique weight and threshold. When a node is active can transmit data to the subsequent layer of the network. Conversely, if the output does not surpass the threshold, no data is sent to the next layer. Neural networks rely on training datasets to enhance their learning and accuracy over time. Once optimized, they can be used on tasks such as speech and image recognition, which can be completed in minutes rather than hours, especially when compared to the manual efforts of human specialists. Generally speaking, a neural network learns the value of the parameters θ that yield the best approximation and defines a mapping function $y = f(x;\theta)$. A Feedforward Neural Network (FNN) is a type of neural network in which information flows only in one direction, from the input layer through hidden layers, reaching the output layer, so it is free of loops and cycles. In this study, we focused on optimization algorithms applied to an Extreme Learning Machine, which is a simple learning method for single-hidden layer feedforward neural networks (SLFN). The algorithm can be summarized as follows ([3]): Given a training set $\mathbb{N} = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbb{R}^n, \mathbf{t}_i \in \mathbb{R}^m, i = 1, \ldots, N\}$, activation function $g(x)$, and hidden node number $\tilde{N}$,

- **Step 1:** Randomly assign input weight $\mathbf{w}_i$ and bias b_i, $i = 1, \ldots, \tilde{N}$.
- **Step 2:** Calculate the hidden layer output matrix $\mathbf{H}$.
- **Step 3:** Calculate the output weight $\boldsymbol{\beta}$ with

$$\boldsymbol{\beta} = \mathbf{H}^{\dagger}\mathbf{T},$$

where

$$H = \begin{bmatrix} g(w_1 * x_1 + b_1) & \cdots & g(w_L * x_1 + b_L) \\ \vdots & \ddots & \vdots \\ g(w_1 * x_N + b_1) & \cdots & g(w_L * x_N + b_L) \end{bmatrix}_{N \times L}$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_L^T \end{bmatrix}_{L \times m} \qquad T = \begin{bmatrix} t_1^T \\ \vdots \\ t_N^T \end{bmatrix}_{N \times m}$$

The algorithm described can work with any infinitely differentiable activation function, including commonly used ones like sigmoidal, radial basis, and others. The number of hidden nodes required is bounded by the number of distinct training samples.

1.2 Problem

In our project, we focus on optimizing an Extreme Learning Machine involving two sets of weights: W_1 which represents the input-to-hidden layer weights, and W_2, which concerns the hidden-to-output layer weights. W_1 is a fixed random matrix, its values are initialized randomly and remain constants during the entire training process, while W_2 contains the values that must be optimized throughout the process. These values in W_2 are adjusted to minimize a loss function. Our project exploits the Mean Squared Error (MSE) between the network's predicted output and the target output as a loss function. The MSE is a smooth and differentiable function of the following form:

$$\text{MSE}(W_2) = \|W_2\sigma(W_1x) - y\|^2 \tag{1.1}$$

The function itself shouldn't create issues when gradient descent techniques are applied. However, the challenge arises when the regularization term is introduced to the model, in particular, the LASSO regularization (L1) term:

$$\|W_2\|_1 = \sum_{i,j} |(W_2)_{i,j}| \tag{1.2}$$

The reason why we introduce the L1 norm to the function is to improve the learning process by encouraging sparsity, nullifying some variables by assigning zeros and improving the overall generalization ability of the model. By putting together the previous two equations we obtain the following one.

$$f(W_2) = \|W_2\sigma(W_1x) - y\|^2 + \lambda\|W_2\|_1 \tag{1.3}$$

where, λ is the regularization parameter, σ is the activation function, and x and y are the input and output data, respectively. The current formula is non-smooth and non-differentiable anymore, so we cannot still use the traditional gradient-based optimization methods but some alternatives. Our aim is to apply an efficient solution based on those proposed in the project.

1.3 Proposed algorithms

In the following chapters, we will try to understand how to apply the following optimization algorithms, comparing them in terms of performance and results:

- A standard momentum descent approach: the method involves a gradient descent step followed by a momentum-like stage, so the parameters are updated twice. The momentum stage uses a weighted difference between the new and previous parameter states. Finding the correct learning rate and momentum values is critical, but often requires manual tuning as per Sutskever's recommendations [9].
- A smoothed gradient algorithm is used for transforming a non-smooth function into a smooth one, enabling the application of efficient gradient-based optimization methods. Part of the implementation will be based on the content of paper [7].

Chapter 2

Algorithms

2.1 Momentum Descent

Training deep and recurrent neural networks (DNNs and RNNs) has historically been difficult, even if they are powerful models vastly used in areas like vision and speech. DNNs gained renewed interest after Hinton et al. (2006)[2] introduced greedy layerwise pre-training, followed by methods that train layers sequentially before fine-tuning the whole network with optimization methods like stochastic gradient descent (SGD). Martens (2010)[4] later showed that Hessian-free optimization (HF) could train DNNs from random initializations without pre-training, achieving lower errors on auto-encoding tasks compared to prior methods. While HF outperforms plain SGD in some cases, using momentum methods, particularly Nesterov's accelerated gradient, can close or even surpass this performance gap. Despite the extensive study of momentum in neural network optimization, its importance in deep learning was previously underestimated: past studies focused too much on local convergence and poorly chosen initializations, leading to the underestimation of momentum's effectiveness in deep learning.

The momentum method, introduced by Polyak (1964) [8] and referred to as classical momentum (CM), accelerates gradient descent by accumulating a velocity vector in directions where reduction in the objective function persists across iterations. Given an objective function $f(\theta)$ to be minimized, the update rules for CM are as follows:

$$v_{t+1} = \eta v_t - \varepsilon \nabla f(\theta_t) \tag{2.1}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{2.2}$$

where ε is the learning rate, η is the momentum coefficient, and $\nabla f(\theta_t)$ is the gradient at θ_t. The variable v, representing velocity, denotes the direction and speed at which the parameters move through the parameter space. In low-curvature directions, where changes in the gradient are slow, CM amplifies these steps, similarly to how second-order methods reweight updates based on curvature.
Nesterov's Accelerated Gradient (NAG), introduced by Nesterov (1983)[6], is another first-order optimization method with better convergence rates than gradient descent for smooth, non-strongly convex functions. The update rules for NAG are:

$$v_{t+1} = \eta v_t - \varepsilon \nabla f(\theta_t + \eta v_t) \tag{2.3}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{2.4}$$

While NAG is not typically viewed as a momentum method, it is closely related to CM. The key difference is that NAG performs a partial update to θ_t before computing the gradient, which allows for more responsive adjustments to the velocity vector. This subtle difference enables NAG to be more stable, especially when η is large, making it more effective than CM in avoiding oscillations in high-curvature directions. Imagine a scenario where adding ηv_t leads to an immediate increase in the objective function f: the gradient correction to the velocity v_t is calculated at the point $\theta_t + \eta v_t$. If ηv_t is a poor update, the gradient will more strongly pull back toward θ_t, offering a faster and more substantial correction than the classical momentum method. Although each iteration of NAG may offer only a slightly better correction than CM, this difference can accumulate over multiple iterations. While CM and NAG behave similarly in low-curvature directions, NAG's partial update strategy allows it to perform better in high-curvature scenarios, especially when the learning rate and momentum

coefficient are large. This difference makes NAG more suitable for faster convergence in deep learning and other complex optimization problems. To better compare the classical moment with the NAG, the authors of [9] use Theorem 2.1, which we report below:

Theorem. *Let $p(y) = \sum_{i=1}^{n}[p]_i([y]_i)$ such that $[p]_i(t) = \lambda_i t^2/2 + c_i t$.. Let ε be arbitrary and fixed. Denote by $CM_x(\eta, p, y, v)$ and $CM_v(\eta, p, y, v)$ the parameter vector and the velocity vector, respectively, obtained by applying one step of CM (i.e., Eq. (2.1) and then Eq. (2.2)) to the function p at point y, with velocity v, momentum coefficient η, and learning rate ε. Define NAG_x and NAG_v analogously. Then the following holds for $z \in \{x, v\}$:*

$$CM_z(\eta, p, y, v) = \begin{bmatrix} CM_z(\eta, [p]_1, [y]_1, [v]_1) \\ \vdots \\ CM_z(\eta, [p]_n, [y]_n, [v]_n) \end{bmatrix}$$

$$NAG_z(\eta, p, y, v) = \begin{bmatrix} CM_z(\eta(1-\lambda_1\varepsilon), [p]_1, [y]_1, [v]_1) \\ \vdots \\ CM_z(\eta(1-\lambda_n\varepsilon), [p]_n, [y]_n, [v]_n) \end{bmatrix}$$

The theorem implies that Classical Momentum (CM) and Nesterov's Accelerated Gradient (NAG) are equivalent when the learning rate ε is small, specifically when $\varepsilon\lambda \ll 1$ for each eigenvalue λ of matrix A. NAG and CM only differ when ε is large. In this case, NAG applies smaller effective momentum in high-curvature directions, preventing oscillations or divergence, allowing a larger momentum coefficient η compared to CM for the same ε.

2.1.1 NAG as a Momentum Method

In order to derive Nesterov Accelerated Gradient as a momentum method, we can break the process down step by step as described in [6].
Assume we have the parameter θ_t at time t and velocity v_t. The velocity term contains information about the past gradients. Instead of computing the gradient at the current position θ_t, we compute it at the anticipated future position $\theta_t + \eta v_t$. This is the key difference between standard momentum and Nesterov's method:

$$\text{Look-ahead point: } \tilde{\theta}_t = \theta_t + \eta v_t \tag{2.5}$$

Then we use the gradient at the look-ahead point to update the velocity:

$$v_{t+1} = \eta v_t - \epsilon \nabla f(\tilde{\theta}_t)$$

After computing the new velocity, we update the parameters using the same velocity:

$$\theta_{t+1} = \theta_t + v_{t+1}$$

Nesterov showed that introducing a specific sequence a_t improves the convergence rate from $O(1/t)$ (standard gradient descent) to $O(1/t^2)$. To solve a problem $\{f(x)|, x \in E\}$ (where E is a Hilbert space[1]) we can use the following method: select a point $y_0 \in E$ and set $k = 0, a_0 = 1, x_{-1} = y_0, \alpha_{-1} = \|y_0 - z\|/\|f'(y_0) - f'(z)\|$, where $z \neq y_0$ is an arbitrary point in E and $f'(z) \neq f'(y_0)$. At k-th iteration we can compute the smallest index $i \geq 0$ for which

$$f(y_k) - f(y_k - 2^{-i}\alpha_{k-1}f'(y_k)) \geq 2^{-i-1}\alpha_{k-1}\|f'(y_k)\|^2$$

Then, we can develop the calculus as it follows:

$$\alpha_k = 2^{-k}\alpha_{k-1}, \quad x_k = y_k - \alpha_k f'(y_k), \tag{2.6}$$

$$\alpha_{k+1} = \frac{1 + \sqrt{4\alpha_k^2 + 1}}{2}, \tag{2.7}$$

$$y_{k+1} = x_k + \frac{(\alpha_k - 1)(x_k - x_{k-1})}{\alpha_{k+1}}. \tag{2.8}$$

[1] A Hilbert space is a complete vector space with an inner product, allowing for notions of distance and angles. In mathematical optimization, it's used when optimizing over infinite-dimensional spaces (like function spaces), enabling techniques such as gradient-based methods to generalize beyond finite dimensions while ensuring convergence and stability.

The variable a_t, helps describe how the momentum coefficient η_t evolves over time. As t increases, the momentum term should gradually increase as well to ensure faster convergence. For large t, we approximate $a_t \approx \frac{t+4}{2}$, and thus:

$$\eta_t \approx 1 - \frac{3}{t+5}$$

This shows that the momentum grows stronger over time, and η_t converges to 1 as $t \to \infty$.
For convex functions with an L-Lipschitz continuous gradient, Nesterov's method guarantees (from Theorem 1 of [6], page 373):

$$f(\theta_t) - f(\theta^*) \leq \frac{4L\|\theta_0 - \theta^*\|^2}{(t+2)^2} \tag{2.9}$$

This shows that the function value decreases at a rate proportional to $1/t^2$, compared to $1/t$ in standard gradient descent. In practice, NAG is implemented with a fixed learning rate ϵ and a momentum coefficient η, typically between 0.8 and 0.9. The pseudocode for the method is: (1) Initialize θ_0 and $v_0 = 0$; (2) For each iteration t: compute the look-ahead point $\tilde{\theta}_t = \theta_t + \eta v_t$, compute the gradient at the look-ahead point $g_t = \nabla f(\tilde{\theta}_t)$, update the velocity $v_{t+1} = \eta v_t - \epsilon g_t$, update the parameters $\theta_{t+1} = \theta_t + v_{t+1}$.

2.1.2 NAG in a nonsmooth context

We have seen that the goal is to minimize the objective function 1.3, which includes both the mean squared error (MSE) and the L1 regularization term (to induce sparsity):

$$f(W_2) = \underbrace{\|W_2\sigma(W_1x) - y\|^2}_{\text{smooth part: } g(W_2)} + \underbrace{\lambda\|W_2\|_1}_{\text{non-smooth part: } h(W_2)}$$

where $g(W_2) = \|W_2\sigma(W_1x) - y\|^2$ is smooth and differentiable and $h(W_2) = \lambda\|W_2\|_1$ is convex but non-smooth. NAG guarantees Eq. 2.9 for convex objective functions with an L-Lipschitz continuous gradient, let's analyze our loss function:

- Convexity: MSE function is convex as it is a sum of squared terms, the L1 norm is also a convex function. Therefore, the sum of MSE and L1 term results in an overall convex objective function (under the assumption that the activation function σ is linear or a convex function itself).
- Lipschitz-continuity[2] of the gradient: gradient of MSE is smooth while gradient of L1 norm is not due to its absolute value nature (in particular where any component of W_2 is zero).

Theorem 2.1 not directly apply to ELMs because the theorem is specific to quadratic optimization problems and first-order gradient-based methods, while ELMs typically use a non-iterative method for learning the output weights (solving a linear system using least squares). To ensure that NAG is applicable to the ELM with an MSE loss regularized by L1, we need to account for the fact that the L1 regularization introduces non-smoothness. First, we can compute the gradient of the MSE term:

$$\nabla\text{MSE}(W_2) = 2(W_2\sigma(W_1x) - y)\sigma(W_1x)^\top$$

For the L1 regularization, the gradient is non-differentiable at zero, so a possible solution is to replace the gradient with a *subgradient*, which generalizes the concept of a gradient for convex, non-smooth functions.
A subgradient s of a convex function f at a point x satisfies the condition:

$$f(z) \geq f(x) + \langle s, z - x\rangle \quad \text{for all } z \in \mathbb{R}^n$$

This inequality generalizes the notion of a tangent hyperplane to convex functions, even when they are not differentiable. The set of all subgradients at a point x is called the *subdifferential* and is denoted $\partial f(x)$. For differentiable functions, the subdifferential contains only the gradient, but for non-differentiable points, it can contain many vectors. For this reaseon, a key issue with non-smooth functions is that fixed step sizes don't work, because in certain cases, the algorithm may get stuck in a cycle, oscillating without making progress. The fixed

[2] A function has an L-Lipschitz continuous gradient if $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$ for all points x and y. Functions with this property are also called L-smooth

step leads to repeated values making the algorithm ineffective.
We can then calculate the subgradient of the nonsmooth part as

$$\partial h(W_2) = \lambda \cdot \text{sign}(W_2)$$

where the *sign function* is applied element-wise to W_2:

$$\text{sign}(w_j) = \begin{cases} 1, & w_j > 0 \\ -1, & w_j < 0 \\ 0, & w_j = 0 \end{cases}$$

The subgradient method updates the parameter W_2 iteratively using:

$$W_2^{(k+1)} = W_2^{(k)} - \alpha_k \cdot g(W_2^{(k)}) - \alpha_k \cdot \partial h(W_2^{(k)})$$

where α_k is the step size at iteration k, $g(W_2^{(k)})$ is the gradient of the smooth part, $\partial h(W_2^{(k)})$ is the subgradient of the non-smooth part. Combining the subgradients, the update step becomes:

$$W_2^{(k+1)} = W_2^{(k)} - \alpha_k \left(2(W_2^{(k)}\sigma(W_1 x) - y)\sigma(W_1 x)^T + \lambda \cdot \text{sign}(W_2^{(k)})\right)$$

A diminishing step size is commonly used to ensure convergence, such as $\alpha_k = \frac{1}{k}$. Alternatively, we could use an adaptive step size like Polyak's step size, which is based on the current subgradient magnitude.
We have seen that using the subgradient could be a solution, but the computational cost is commonly $O(1/\sqrt{k})$, which is typically slower than gradient-based methods for smooth functions. We can therefore also introduce the concept of smoothed gradient, with which we test empirically whether the convergence time is less for smoothed functions.

2.2 Smoothed gradient method

The majority of the information presented in this section is derived from the paper "Smooth minimization of non-smooth functions", by Nesterov [7], which offers a systematic approach to approximate an initial non-smooth function with a smooth one whose gradient is Lipschitz-continuous and can be minimized by an efficient gradient method.

2.2.1 Theoretical Framework

The main problem of interest in this work is as follows (as described by Eq. 2.1 in [7]):

$$f^* = \min_x \{f(x) : x \in Q_1\} \tag{2.10}$$

where $f(x)$ is a continuous convex function on Q_1, which is a bounded closed convex set[3]. The convex property of both Q_1 and $f(x)$ ensures that the optimization problem has a global minimum. According to the paper, we cannot assume the differentiability of f, so we cannot solely rely on a gradient-based method, we should consider methodologies that can handle non-differentiability. This idea perfectly reflects our problem, indeed not all the terms of our function are differentiable by definition (see also 2.1.2). We can assume the following general model to represent our $f(x)$, as equation 2.2 of the paper[7] shows:

$$f(x) = \hat{f}(x) + \max_u \left\{\langle Ax, u\rangle - \hat{\phi}(u) : u \in Q_2\right\} \tag{2.11}$$

where $\hat{f}(x)$ is a known function that captures the differentiable part of the problem. In the context of our problem $\hat{f}(x)$ is the MSE $\|W_2\sigma(W_1 x) - y\|^2$. The term $\max_u \left\{\langle Ax, u\rangle - \hat{\phi}(u) : u \in Q_2\right\}$ introduces complexity to the objective function by defining a maximization problem over u, where A is the identity matrix and $\phi(u)$ corresponds to the indicator function of the dual norm of $\|x\|_1$. Once we know the structure of the function

[3]In Nesterov's method, Q_1 is the convex set for the primal problem, where the optimization takes place, and Q_2 is a convex set in the dual space used to smooth non-smooth terms.

2.11 we can exploit it to construct a smooth approximation of the objective function 2.10. For performing the smoothing, Nesterov introduces a prox-function $d_2(u)$ of the set $Q_2 = \{u : \|u\|_\infty \leq 1\}$. This means that $d_2(u)$ is continuous and strongly convex on Q_2 with some convexity parameter $\sigma_2 > 0$. The prox-centre of $d_2(u_0) = 0$, thus, according to the 2.4 in [7] for any $u \in Q_2$ we have

$$d_2(u) \geq \frac{1}{2}\sigma_2\|u - u_0\|_2^2 \tag{2.12}$$

According to the 2.5 in [7] we let μ be positive smoothing parameter, which allows us to compute the smoothed version of the 2.11

$$f_\mu(x) = \max_u \left\{ \langle Ax, u\rangle_2 - \hat{\phi}(u) - \mu d_2(u) : u \in Q_2 \right\} \tag{2.13}$$

Since the function $d_2(u)$ is strongly convex, the optimal solution for this problem is unique and it is denoted by $u_\mu(x)$. If the function wasn't strongly convex and then, the solution for $u_\mu(x)$ wasn't unique, the gradient $\nabla f_\mu(x)$ could be ill-defined or discontinuous, violating the condition required for Lipschitz continuity. Since the function $f_\mu(x)$ is well-defined, the paper [7] states in the Theorem 2.6:

Theorem. *The function $f_\mu(x)$ is well defined and continuously differentiable at any $x \in E_1$. Moreover, this function is convex, and its gradient*

$$\nabla f_\mu(x) = A^* u_\mu(x) \tag{2.14}$$

is Lipschitz continuous with constant

$$L_\mu = \frac{1}{\mu\sigma_2}\|A\|_{1,2}^2. \tag{2.15}$$

To conclude the smoothing process of the objective function Nesterov introduces

$$D_2 = \max_u\{d_2(u) : u \in Q_2\}, \qquad f_0(x) = \max_u\{\langle Ax, u\rangle_2 - \hat{\phi}(u) : u \in Q_2\}.$$

leading to the following inequality for any $x \in E_1$:

$$f_\mu(x) \leq f_0(x) \leq f_\mu(x) + \mu D_2 \tag{2.16}$$

The inequality helps determine the optimal μ. According to the inequality 2.16, the smoothed function lies below the original function but remains close to it. Specifically,

$$f_\mu(x) \to f(x) \text{ as } \mu \to 0, \tag{2.17}$$

with the original function bounded between the smoothed function and the smoothed function plus a constant D_2, which depends on the function to minimize and is not always easy to compute. As μ approaches zero, the smoothed function converges to the original uniformly for all x. By estimating D_2, μ can be chosen to balance the smoothness. The larger μ the faster the convergence rate, as the function will be smoother. As μ decreases, the smoothed function better approximates the original function, but the Lipschitz constant L increases because of the error ϵ decrease, because the Lipschitz constant L depends on the ratio $1/\mu$. Finally, the trade-off is between the approximation accuracy and the converge rate, which increases as $\mu \to 0$ and since we aim for a solution where the result of the optimization process applied to the smooth function is close to the result of the original function, the convergence rate will be $O(1/\epsilon)$.
From the formula 2.16 we can see that μ directly impacts how closely the smoothed function approximatex the original non-smooth one. According to Theorem 3 of [7], the smoothing parameter should be chosen as

$$\mu = \frac{2\|A\|_{1,2}}{N+1}\sqrt{\frac{D_1}{\sigma_1\sigma_2 D_2}}$$

where N is the number of iterations, D_1 is the maximum value of a prox-function $d_1(x)$ over the set Q_1 (see 2.10), σ_1 and σ_2 are the convexity parameters of the prox-functions for the sets Q_1 and Q_2, respectively, D_2 is the diameter of the set Q_2. This shows that μ is inversely proportional to the number of iterations, meaning that as you increase the number of iterations, a smaller is required for the smoothing approximation. After N iterations, the method provides approximate solutions x_N and u_N such that the following inequality holds:

$$0 \leq f(x_N) - \phi(u_N) \leq \frac{4\|A\|_{1,2}}{N+1} \cdot \sqrt{\frac{D_1 D_2}{\sigma_1\sigma_2}} + \frac{4MD_1}{\sigma_1(N+1)^2}.$$

This gives an error bound on the solution, which decreases as N increases. The optimal value for the smoothing parameter, according to the theorem, is

$$\mu = \frac{\epsilon}{2D_2} \tag{2.18}$$

In this context, D_2 is defined as the maximum value of the prox-function $d_2(u)$ over the set Q_2:

$$D_2 = \max_{u \in Q_2} d_2(u)$$

Given that the prox-function is continuous and strongly convex with convexity parameter σ_2, we have the inequality 2.12. The key point is that D_2 provides an upper bound for the value of the prox-function over the entire domain Q_2: this means that, larger D_2 leads to more space to smooth the non-smooth function, while increasing the error bound of the approximation.

In section 4 of the paper [7], the Nesterov addresses the situation in which the objective function 2.11 is made up of a differentiable term, $\hat{f}(x)$ whose gradient is Lipschitz-continuous with some constant $M \geq 0$ and a smoothed, originally non-differentiable, term $f_\mu(x)$. Then, the result will be the following objective function:

$$\overline{f}_\mu(x) = \hat{f}(x) + f_\mu(x) \tag{2.19}$$

Under this assumption, the equation 2.15 presented in the theorem 2.2.1 becomes:

$$L_\mu = M + \frac{1}{\mu\sigma_2}\|A\|_{1,2}^2. \tag{2.20}$$

As the gradient of the function 2.19 is Lipschitz continuous with the constant L_μ.
The reason why the author addresses this situation derives from the fact that the theory behind the choice of μ works if the entire function has been smoothed, otherwise if one term has been smoothed while the other has not, the selected μ may not be the optimal one anymore.

2.2.2 Application of the smoothing method

Having defined the framework, we want to use it to approximate the non-smooth term $h(W_2) = \lambda\|W_2\|_1$. We can represent the L_1 norm $\|W_2\|_1$ through the following maximization form:

$$\lambda\|W_2\|_1 = \lambda \max_{\|u\|_\infty \leq 1} \langle W_2, u \rangle \tag{2.21}$$

which introduces the dual variable $u \in Q_2$ and the closed convex set $Q_2 = \{u : \|u\|_\infty \leq 1\}$. To smooth $\lambda\|W_2\|_1$, according to the framework, we exploit a prox-function $d_2(u)$ for the set Q_2 (with convexity parameter $\sigma_2 > 0$), which satisfies 2.12, where u_0 is the prox-center of Q_2. Now, the smooth approximation of $h(W_2)$ is given by the introduction of the parameter μ:

$$h_\mu(W_2) = \lambda \max_{u \in Q_2} \{\langle W_2, u \rangle - \mu d_2(u)\} \tag{2.22}$$

where $\mu > 0$ is the smoothness parameter. By choosing $d_2(u) = \frac{u^2}{2}$, since it is a simple quadratic function we can compute D_2 analytically: we need to find its maximum assuming that the value of u is bounded by the set Q_2, then $u \in Q_2 = \{u : \|u\|_\infty \leq 1\}$. Since $u \in [-1, 1]$, the maximum value occurs at $u = 1$ (or $u = -1$), leading to

$$D_2 = \frac{1}{2}$$

. This means that the value of μ will be based on the chosen value of the boundary of the set Q_2, thus $\mu = \frac{\epsilon}{2D_2} = \epsilon$. Since $\|W_2\|_1 = \sum_i |W_2^{(i)}|$ we can rewrite the approximation for each individual component as

$$h_\mu(W_2) = \lambda \max\{W_2^{(i)} u_i - \mu\frac{u_i^2}{2} : u_i \in [-1, 1]\} \tag{2.23}$$

where quadratic penalty $d_2 = \frac{u_i^2}{2}$ smooths the sharp transition of the maximum. Given that the prox-function $d_2(u_i)$ has the maximum value of $\frac{1}{2}$ for each component of u, the final total value of D_2 for the entire W_2 will be

$$D_2 = \frac{n}{2} \tag{2.24}$$

where n is the number of components in W_2. By differentiating $W_2^{(i)}u_i - \mu\frac{u_i^2}{2}$. At this point, for each W_2, the optimal solution $u_\mu(W_2)$ is the maximizer in the expression for $h_\mu(W_2)$, and since $d_2(u)$ is strongly convex, $u_\mu(W_2)$ is unique and can be formulated as

$$u_\mu(W_2^{(i)}) = \frac{W_2^{(i)}}{\mu}$$

which can be projected onto the interval $[-1, 1]$ by

$$u_\mu(W_2^{(i)}) = \max\left\{-1, \min\left\{1, \frac{W_2^{(i)}}{\mu}\right\}\right\}$$

By inserting back this term in the formula 2.23 we obtain:

$$h_\mu(W_2^{(i)}) = \lambda\left(W_2^{(i)}u_\mu(W_2^{(i)}) - \frac{\mu}{2}\left(u_\mu(W_2^{(i)})\right)^2\right) \tag{2.25}$$

which results in

$$h_\mu(W_2^{(i)}) = \begin{cases} \frac{\lambda}{2\mu}\left(W_2^{(i)}\right)^2, & \text{if } |W_2^{(i)}| \le \mu, \\ \lambda\left(|W_2^{(i)}| - \frac{\mu}{2}\right), & \text{if } |W_2^{(i)}| > \mu. \end{cases}$$

The smoothed approximation $h_\mu(W_2)$ now is continuously differentiable and the gradient is given by:

$$\nabla h_\mu(W_2^{(i)}) = \lambda u_\mu(W_2^{(i)}) = \begin{cases} \frac{\lambda}{\mu}W_2^{(i)}, & \text{if } |W_2^{(i)}| \le \mu, \\ \lambda\ \text{sign}(W_2^{(i)}), & \text{if } |W_2^{(i)}| > \mu. \end{cases} \tag{2.26}$$

The gradient of the smoothed objective function $f_\mu(W_2) = g(W_2) + h_\mu(W_2)$ is Lipschitz continuous, according to Theorem 2.2.1:

$$\nabla f_\mu(W_2) = \nabla g(W_2) + \lambda u_\mu(W_2) \tag{2.27}$$

We have now derived the smoothed function $h_\mu(W_2^{(i)})$ for each individual component $W_2^{(i)}$ in the piecewise form. This function approximates the non-smooth L_1-norm term and is differentiable. As explained in the previous paragraph after the introduction of the inequality 2.16, the parameter μ controls the trade-off between the smoothness and the accuracy of the approximation, allows control over the approximation quality, and as $\mu \to 0$, the smoothed objective function $f_\mu(W_2)$ converges to the original non-smooth function $f(W_2)$. In this case, as described by 2.18 and 2.24 we can derive the value of μ as

$$\mu = \frac{\epsilon}{n} \tag{2.28}$$

We can use all the information above to apply a fast gradient algorithm in order to solve our problem. Below the schema of the steps involved.
First we initialize weights W_2, smoothing parameter $\mu = \epsilon/n$, learning rate $\nu = 1/L_\mu$ (see 2.2.1), momentum η, and convergence tolerance ω. The main steps are:

- Calculate a look-ahead point y_t using the current and previous weights:

  $$y_t = W_2^t + \eta(W_2^t - W_2^{t-1}),$$

  where η is the momentum factor
- Compute the gradient of the smoothed objective function $\nabla f_\mu(W_2)$ at the look-ahead point, including gradients from both the MSE and smoothed L_1-norm terms.
- Update momentum factor η dynamically for faster convergence using the gradient

  $$W_2^{t+1} = y_t - \nu\nabla f_\mu(y_t)$$

The gradient norm must be below the tolerance ω; if so, stop the iteration (or we can set a max number of iterations). The output of the algorithm contains the final weight matrix W_2, which is the optimal solution to the smoothed problem.

Summary

The chapter introduced two approaches of solving the problem presented in section 1.2: we described the Nesterov Accelerated Gradient (NAG) as a momentum-based method (in which we introduced the subgradient as a smoothing method) and then the use of the Nesterov [7] smoothing technique for solving the problem with a fast gradient-based algorithm.
In order to apply this type of algorithm to the problem of this study, it is necessary to adapt the Extreme Learning Machine so that it contains an iterative mechanism for updating the W_2 weights. The ELM algorithm is still based on a single layer network structure but in the optimization phase we will introduce an iterative update, which is also described in the paper [1].

Chapter 3

Experiments

Chapter 4

Results

Chapter 5

Conclusions

Bibliography

[1] Wanyu Deng, Qinghua Zheng, and Lin Chen. "Regularized Extreme Learning Machine". In: *2009 IEEE Symposium on Computational Intelligence and Data Mining*. 2009, pp. 389–395. DOI: 10.1109/CIDM.2009.4938676.

[2] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural Computation* 18.7 (July 2006), pp. 1527–1554. ISSN: 1530-888X. DOI: 10.1162/neco.2006.18.7.1527. URL: http://dx.doi.org/10.1162/neco.2006.18.7.1527.

[3] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. "Extreme learning machine: Theory and applications". In: *Neurocomputing* 70.1–3 (Dec. 2006), pp. 489–501. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2005.12.126. URL: http://dx.doi.org/10.1016/j.neucom.2005.12.126.

[4] James Martens. "Deep learning via Hessian-free optimization". In: ICML'10 (2010), pp. 735–742. URL: https://www.cs.toronto.edu/~jmartens/docs/Deep_HessianFree.pdf.

[5] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* 5.4 (Dec. 1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/bf02478259. URL: http://dx.doi.org/10.1007/BF02478259.

[6] Yurii Nesterov. "A method for solving the convex programming problem with convergence rate $O(1/k^2)$". In: *Proceedings of the USSR Academy of Sciences* 269 (1983), pp. 543–547. URL: https://jucaleb4.github.io/files/nesterov_83.pdf.

[7] Yurii Nesterov. "Smooth minimization of non-smooth functions". In: *Mathematical Programming* 103.1 (2005), pp. 127–152. DOI: 10.1007/s10107-004-0552-5.

[8] B.T. Polyak. "Some methods of speeding up the convergence of iteration methods". In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (Jan. 1964), pp. 1–17. ISSN: 0041-5553. DOI: 10.1016/0041-5553(64)90137-5. URL: http://dx.doi.org/10.1016/0041-5553(64)90137-5.

[9] Ilya Sutskever et al. "On the importance of initialization and momentum in deep learning". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147. URL: https://proceedings.mlr.press/v28/sutskever13.html.

Appendix A

Appendix