DATA-SELECTION IN LEARNING ALGORITHMS

Jonathas de Oliveira Ferreira

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Paulo Sergio Ramirez Diniz

Rio de Janeiro
Março de 2020

DATA-SELECTION IN LEARNING ALGORITHMS

Jonathas de Oliveira Ferreira

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Orientador: Paulo Sergio Ramirez Diniz

Aprovada por: Prof. Paulo Sergio Ramirez Diniz
Prof. João Baptista de Oliveira e Souza Filho
Prof. Gabriel Matos Araújo

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2020

Ferreira, Jonathas de Oliveira

Data-Selection in Learning Algorithms/Jonathas de Oliveira Ferreira. – Rio de Janeiro: UFRJ/COPPE, 2020.

XIX, 95 p.: il.; 29,7cm.

Orientador: Paulo Sergio Ramirez Diniz

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2020.

Referências Bibliográficas: p. 84 – 92.

1. Data Selection. 2. Filter Adaptive. 3. Kernel Method. 4. Neural Network. I. Diniz, Paulo Sergio Ramirez. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

To my Family and Friends

Agradecimentos

Agradeço à minha família, principalmente aos meus pais Francisco Ferreira e Irlanda de Oliveira e ao meu irmão João Pedro, que sempre estiveram ao meu lado desde o meu nascimento. Boa parte do que eu sou hoje como pessoa devo a eles. Agradeço também à minha vó Joana e as minhas tias Hildene, Filoca e Tunica, que me ajudaram bastante em vários momentos e fizeram os meus dias mais felizes. Quero agradecer também a minha madrinha e meu padrinho, Ilma e Paulinho, por serem pessoas tão legais e boas comigo.

Quero fazer um agradecimento aos meus amigos: Aloizio Macedo, Baby, Barata, Benice, Daphne Poll, Diogo Lemos, Fael, Guilherme, Ian Martins, Ivani Ivanova, Jéssica Richards, Juliana Pessin, Karina Livramento, Ligeiro, Leo Vasconcelos, Leo Gama, Mangelli, Pedro Gil, Pacheco, Presida, Produção, Raphael 04, Rodrigo Lima, Turano. Essas pessoas me ajudaram de alguma forma no mestrado e/ou graduação e também me fizeram companhia nesse período de aprendizado da faculdade.

Agradeço também ao pessoal da Atlética, principalmente ao Alexandre e a Krissy, por terem começado essa atlética tão maravilhosa onde eu pude passar alguns momentos da minha vida. Isso também inclui o time de handebol, onde mesmo não sabendo jogar nada por lá (risos), eles sempre me acolheram.

Um salve especial ao pessoal que foi representante comigo, Ian, Baby, Karina, Barata, Produção, Mesquita por esses momentos que a gente passou junto tentando melhorar a ABC, saiba que aprendi bastante durante esse período e que levarei para o resto da minha vida esses dias de governança.

Quero agradecer a todo pessoal da ABC 116, até as pessoas que eu não conheço muito bem, por vivenciar seus dias nesse local tão bom que é a ABC. Espero que as pessoas do presente e do futuro de nosso laboratório cuidem dele, assim como eu tive o cuidado de gerenciá-lo para vocês que são a geração atual da ABC.

Quero agradecer a dois professores que foram importantes na minha jornada durante o mestrado: Paulo Diniz e Fábio Ramos. Fábio Ramos, obrigado por ter me orientado durante o tempo que fiquei na Matemática Aplicada no mestrado, por ter me dado forças para seguir adiante, mesmo depois daquele acontecimento, mostrando que ali não era o fim. Paulo Diniz, obrigado por me orientar durante esses anos, eu sei que você sentirá bastante saudades de mim depois que eu sair da

faculdade. Em muitos momentos você foi como um segundo pai, sempre me dando forças e mostrando qual caminho eu deveria seguir.

Quero agradecer a minha irmã de orientador, Marcele. Por ter sido a minha dupla em vários momentos nessa faculdade, seja nos artigos ou nos trabalhos que o Diniz passava para a gente fazer. E também por me ajudar a finalizar essa minha dissertação.

Quero agradecer ao pessoal do SMT: Vinicius, Wesley, Domenica, Rafael Padilla, Cinelli, Matheus, Gabriel, Markus, Tadeu, pois mesmo não conhecendo vocês a fundo, sempre gostei de conversar com vocês.

E por último, quero agradecer ao pessoal da Elogroup, principalmente ao pessoal de Analytics, por terem me acolhido tão bem nesses últimos 6 meses, saibam que com vocês eu obtive bastante aprendizado.

Enfim, obrigado a todos por deixarem eu fazer parte de duas vidas, pois se hoje estou escrevendo essa dissertação foi graças a vocês.

Agradeço também ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo suporte financeiro durante o mestrado.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

SELEÇÃO DE DADOS EM ALGORITMOS DE APRENDIZAGEM

Jonathas de Oliveira Ferreira

Março/2020

Orientador: Paulo Sergio Ramirez Diniz

Programa: Engenharia Elétrica

Nos últimos anos, a quantidade de informação armazenada em dispositivos de aquisição de dados tem aumentado exponencialmente, em virtude das antenas *massive multiple-input multiple-output* (MIMO), redes sociais e redes distribuídas. Nesta era do *Big Data*, enfrentamos o desafio de utilizar com eficiência uma grande quantidade de dados para extrair informações importantes. Portanto, é essencial definir critérios para decidir se o dado é relevante ou não durante o processo de aprendizagem. Este trabalho propõe uma estratégia de seleção de dados para duas áreas: filtragem adaptativa e redes neurais. Em ambas as situações, os dados podem ser descartados, reduzindo o custo computacional e, em alguns casos, a acurácia da estimativa. As aplicações analisadas neste trabalho incluem dados sintéticos e reais, estas verificam a eficácia dos algoritmos propostos que podem obter reduções significativas nos custos computacionais sem sacrificar a acurácia da estimativa devido à seleção dos dados.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

DATA-SELECTION IN LEARNING ALGORITHMS

Jonathas de Oliveira Ferreira

March/2020

Advisor: Paulo Sergio Ramirez Diniz

Department: Electrical Engineering

In recent years, the amount of information stored in data acquisition devices has increased exponentially, due to the massive multiple-input multiple-output (MIMO) antennas, social networks, and distributed networks. In this era of Big Data, we face the challenge of efficiently utilizing a large amount of data to extract valuable information. Therefore, it is essential to define criteria to decide if the data is relevant or not during the learning process. This work proposes a data selection strategy for two areas: adaptive filtering and neural networks. In both situations, data could be discarded, reducing the computational cost and, in some cases, the accuracy of the estimate. The applications analyzed in this work include synthetic and real data, these verify the effectiveness of the proposed algorithms that may achieve significant reductions in computational costs without sacrificing estimation accuracy due to the selection of the data.

Contents

List of Figures

List of Tables

List of Symbols

ξ_{exc} excess MSE (chapters 2 and 3)

(ω_1, ω_2) linearly separable two-classes (chapter 4)

α step size in the coefficient filter (chapters 2 and 3)

β step size in the conjugate direction (chapters 2 and 3)

$\boldsymbol{\Delta}^l$ Matrix representing the sensitivity vector of layer l (chapter 4)

$\boldsymbol{\delta}^l$ sensitivity vector for the layer l (chapter 4)

$\boldsymbol{\Delta}^l_{\mathcal{C}}$ Matrix representing the sensitivity vector of layer l computed from data selection for classification problem (chapter 4)

$\boldsymbol{\Delta}^l_{\mathcal{R}}$ Matrix representing the sensitivity vector of layer l computed from data selection for regression problem (chapter 4)

$\boldsymbol{\epsilon}^l$ multiplication between the weigth matrix and sensitivity vector (chapter 4)

$\boldsymbol{\phi}$ kernel function vector (chapter 3)

$\boldsymbol{\Upsilon}$ mapping of the input matrix into the Hilbert Space (chapter 3)

$\boldsymbol{\upsilon}$ feature function (chapter 3)

$\Delta\mathbf{w}$ error in the coefficients (chapters 2 and 3)

η constant in step size (chapters 2 and 3)

γ small constant for regularization (chapters 2 and 3)

$\hat{\mathbf{Y}}$ Matrix of estimated outputs (chapter 4)

$\hat{\mathbf{y}}$ estimated output (chapter 4)

$\hat{\mathbf{Y}}_{\mathcal{C}}$ Matrix of estimated outputs selected from data selection for classification problem (chapter 4)

$\hat{\mathbf{Y}}_{\mathcal{R}}$ Matrix of estimated outputs selected from data selection for regression problem (chapter 4)

$\hat{d}$ estimated desired signal (chapters 2 and 3)

$\hat{P}_{\text{up}}$ estimated probability of update (chapters 2 and 3)

λ forgetting factor (chapters 2 and 3)

$\mathbf{c}$ conjugate direction (chapters 2 and 3)

$\mathbf{d}$ desired vector (chapter 3)

$\mathbf{g}$ negative gradient of the objective function (chapters 2 and 3)

$\mathbf{H}$ finite impulse response (FIR) channel convolution matrix (chapters 2 and 3)

$\mathbf{h}$ impulse response (chapters 2 and 3)

$\mathbf{K}$ Gram matrix (chapter 3)

$\mathbf{p}$ cross-correlation between input filter and desired signal (chapters 2 and 3)

$\mathbf{R}$ autocorrelation matrix of the input filter (chapters 2 and 3)

$\mathbf{s}$ input signal (chapters 2 and 3)

$\mathbf{v}$ Multiplication between input transpose matrix and conugate direction vector (chapter 3)

$\mathbf{W}$ all weight matrices in the neural networks framework (chapter 4)

$\mathbf{w}$ adaptive coefficient update (chapter 2 and 3)

$\mathbf{W}^l$ weight matrix between layers $l-1$ and l (chapter 4)

$\mathbf{X}$ input data matrix (chapter 3)

$\mathbf{x}$ filter input (chapters 2 and 3)

$\mathbf{x}$ input in Neural Network (chapter 4)

$\mathbf{x}^l$ input vector of layer l (chapter 4)

$\mathbf{X}_I$ input matrix with I input vectors added in the data dictionary (chapter 3)

$\mathbf{x}_i$ the i-th input included in the data dictionary (chapter 3)

$\mathbf{y}$ output in Neural Network (chapter 4)

$\mathbf{y}^l$ output vector of layer l (chapter 4)

$\mathbf{Z}$ Residual gradient matrix (chapter 3)

$\mathbf{z}$ residual gradient for CGLS algorithm (chapter 3)

$\mathcal{C}$ set related to the indexes of E values greater or equal to the t_{bin}-th largest value (chapter 4)

$\mathcal{E}$ vector error in regression Neural Network for data selection (chapter 4)

$\mathcal{E}^k$ vector error in classification Neural Network for the data selection in the n-th class (chapter 4)

$\mathcal{H}$ Hilbert Space (chapter 3)

$\mathcal{H}^*$ dual space of $\mathcal{H}$ (chapter 3)

$\mathcal{R}$ set selected to update the weights at iteration (chapter 4)

$\mathcal{X}$ subset of Hilbert Space (chapter 3)

μ step size in Neural Network (chapter 4)

π inner product between of $\mathbf{g}(k)$ and $\mathbf{g}(k)$ (chapter 3)

ρ constant for misadjustment (chapters 2 and 3)

σ_e error variance (chapters 2 and 3)

σ_y output variance (chapters 2 and 3)

σ_e^t error variance in Neural Network (chapter 4)

σ_n noise variance (chapters 2 and 3)

τ threshold for the data selection (chapters 2, 3 and 4)

τ_{max} threshold for outliers (chapters 2 and 3)

ξ steady state MSE (chapters 2 and 3)

ξ_{min} Minimum MSE (chapters 2 and 3)

ζ_i kernel coefficient (chapter 3)

a_i eigenvalue of the reproducing kernel $\kappa(\cdot,\cdot)$ (chapter 3)

b mini-batch size (chapter 4)

d desired signal (chapters 2 and 3)

E sum total of error vector in Neural Network (chapter 4)

e error signal (chapters 2 and 3)

f activation function in Neural Network

f_L output activation function in Neural Network

$h_{\mathbf{W}}$ parametric nonlinear function in Neural Network (chapter 4)

H_{new} new hyperplane (chapter 4)

H_{old} old hyperplane (chapter 4)

I number of input vectors included in the data dictionary (chapters 2 and 3)

$iter$ number of iteration in Neural Network (chapter 4)

J objective function (chapters 2, 3 and 4)

J_{test} objective function in test dataset (chapter 4)

J_{train} objective function in training dataset (chapter 4)

J_m^k objective function in the output node k for the m-th input signal (chapter 4)

k_{up} number of updates at each iteration to the KCG algorithm (chapter 3)

$L+1$ total number of layers in Neural Network (chapter 4)

M Total number of examples in a offline dataset (chapter 3)

n noise signal (chapters 2 and 3)

n_{ep} number of epochs in Neural Network (chapter 4)

o^l total number of nodes in the layer l without the bias node (chapter 4)

P_{up} prescribed probability of update (chapters 2, 3 and 4)

$q_i(\cdot)$ eigenfunction of the reproducing kernel $\kappa(\cdot,\cdot)$ (chapter 3)

r autocorrelation between input (chapters 2 and 3)

t_{bin} value obtained from binomial distribution with $n = b$ and $p = P_{\text{up}}$ (chapter 4)

w_{ij}^{l} weight communication between the i-th node of a layer $l-1$ to the j-th node of the next layer l (chapter 4)

y filter output (chapters 2 and 3)

$\mathbf{w_o}$ optimal coefficient (chapters 2, 3 and 4)

e error vector in Neural Network for data selection (chapter 4)

X^{l} Input matrix of layer l in Neural Network (chapter 4)

$X_{(t,i)}$ Inputs signal selected in training dataset in iteration i and epoch t (chapter 4)

$X_{\mathcal{C}}^{l}$ Input matrix of layer l in classification Neural Network selected from data selection (chapter 4)

$X_{\mathcal{R}}^{l}$ Input matrix of layer l in regression Neural Network selected data selection (chapter 4)

Y^{l} Output matrix of layer l in Neural Network (chapter 4)

$Y_{(t,i)}$ Outputs signal selected in training dataset in iteration i and epoch t (chapter 4)

$Y_{\mathcal{C}}$ Outputs signal in training dataset selected from data selection for classification problem (chapter 4)

$Y_{\mathcal{C}}^{l}$ Output matrix of layer l in classification Neural Network selected from data selection (chapter 4)

$Y_{\mathcal{R}}$ Outputs signal in training dataset selected from data selection for regression problem (chapter 4)

$Y_{\mathcal{R}}^{l}$ Output matrix of layer l in regression Neural Network selected from data selection (chapter 4)

Chapter 1

Introduction

Learning algorithms are tools to acquire knowledge from data using statistical methods and optimization. These algorithms can solve specific tasks based on patterns and inferences obtained from a dataset, known as training dataset [1–3]. Examples of such tasks include image recognition [4], spam email identification [5], diagnoses in medicine [6] and other pattern recognition tasks in biology, economics, astronomy, etc [7, 8].

This scientific area started to flourish around the 90's, when the field of artificial intelligence started to handle problems using statistics and probability theory. This growth occurred mainly due to the increase in computer capacity to process information and the discovery of significant mathematical results [9–11].

Learning algorithms are intimately linked with optimization: most problems in our main field contain a loss function (that captures the adequacy of the current model) that should be minimized utilizing the training dataset [1–3]. The minimization problem requires optimization methods.

1.1 Why Do We Select Data?

Most subareas of learning algorithms require enormous computational costs due to the complexity of the related problems [12, 13]. As a consequence, a huge amount of resources, such as power consumption, might be needed in order to produce important results in these scientific areas. If we were able to reduce, even if by a small amount, data volume, we would also reduce the computational time to solve the problems. Therefore, our main goal is to save energy by properly managing technological issues.

The growth in the amount of available information in data acquisition devices can result in faulty systems, malicious agents, or input data that are irrelevant or/and

redundant. Among these, the outliers[1] that impair the performance of inferences stand out and are not related to the postulated problem. Therefore, it is important to rigorously mine the data before considering the full training dataset as the final dataset.

One of the objectives of this study is to identify whether the subset generated by some data selection method [14–23], called the data dictionary, leads to approximately the same result as using the full training dataset. In this case, the reduction of the amount of data can be beneficial.

Besides helping improving algorithm's speed, the use of some data selection method incurs in other advantages. In some cases, if we have a significant amount of irrelevant information and eliminate it, we would obtain better inference results. Also, reducing the amount of data might decrease the possibility of overfitting.

1.2 General Considerations

Roughly, the learning process mainly consists of two parts: training (learning) and test (inference), taken in a set of disjoint data, as illustrated in Figure 1.1. The data selection method mentioned in the previous subsection is included in the training phase, where an optimization algorithm acts as feedback in the process. In the test phase, we do not use the data selection, because in this case we merely apply the inferred model obtained from the training phase. This small amount of data can reflect on all the possible data, i.e., it provides an overview of the entire sample space. In both phases, Data Preprocessing is the step in which data is transformed, to bring it to a state where the interpretability of the data becomes more accessible to the algorithm with the new coding. As an example, we can mention handling null values Min-Max Normalization[2] [2, 24].

In the applications presented in this dissertation, we only consider examples with an extensive amount of data when compared to the number of variables, justifying the use of the data selection. Another relevant factor when dealing with learning algorithms is the choice of parameters, which in this text are made according to usual settings recommended for each area leading to a better performance of the model.

[1] An outlier is a data point that differs significantly from other values in a random sample from a population.

[2] Min-Max Normalization consists of normalizing the input variables in the range 0 to 1.

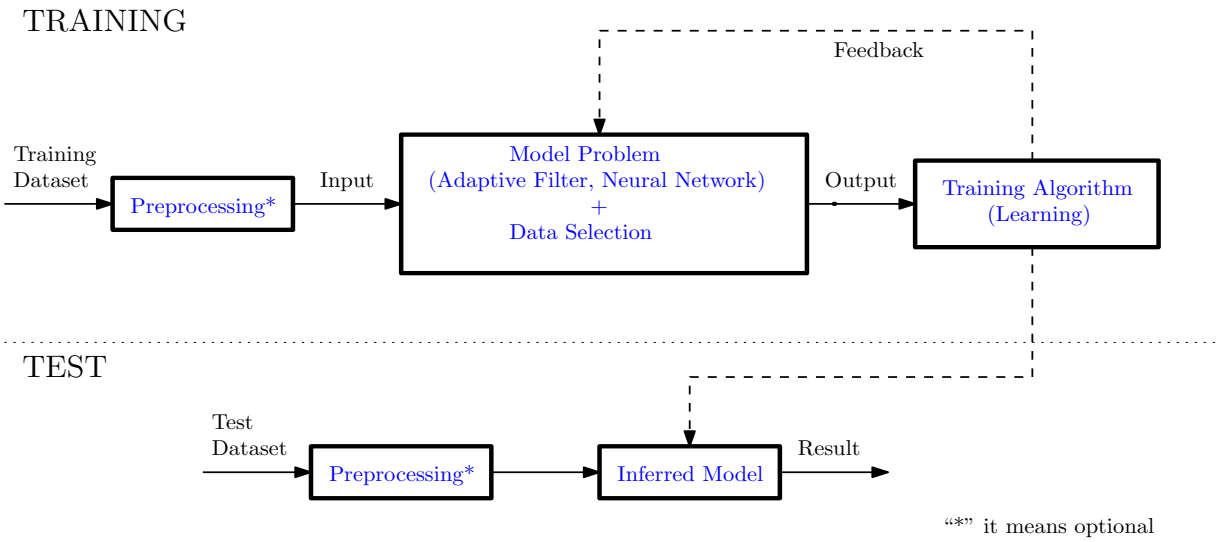


Figure 1.1: Scheme of a learning algorithm.

1.3 Dissertation Goals

The main goal of this work is to propose a procedure for managing and saving the available computing resources. The strategy proposed is a new data selection method constructed from a statistical point of view. This procedure is explored in threes areas of this dissertation that are connected to each other in several ways:

- Adaptive Filtering (linear systems) [1, 25, 26];
- Kernel Adaptive Filtering (nonlinear systems) [27, 28];
- Neural Networks [3, 24, 28].

Several application areas retain a considerable amount of data and one way of managing the excess of redundant information is to apply a selection method. This procedure is important due to the increasing amount of irrelevant or redundant information available for processing.

In linear and nonlinear adaptive filtering, there are many important applications in which data arrive in real-time, one sample at a time. At each iteration, the parameters for the model established in the learning algorithm is updated. This procedure is considered expensive because it is an online learning and also depends on the optimizer algorithm chosen. Furthermore, some data may be regarded as unnecessary by not bringing new information to the algorithm. The data selection method proposed in this work aims to detect this redundant data so that the algorithm is not updated when this sort of information arrives. The notion of kernel in adaptive filtering was explored to deal with applications where an unknown nonlinear regression modeling is required.

The Neural Networks (NNs) is a model used mainly in datasets with a large amount of information available, considering an offline learning. In this case, the issue we intend to address is the same as in the adaptive filtering, that is, which data are relevant to the learning process at each iteration of the parameter update phase. With some modifications, we adapted the previously data selection proposals for adaptive filtering (linear and nonlinear) to obtain a new approach applicable to NN.

1.4 Organization

This dissertation is organized in five chapters. Chapter 1 provides an introduction to the subjects presented. In addition, it highlights and motivates the main points of each unit studied subsequently.

The data selection in linear adaptive filters is introduced in Chapter 2. The concepts of adaptive filtering are detailed in the first part of the chapter. The applications considered are prediction, system identification, and equalization. The learning algorithm chosen is the online conjugate gradient (CG) with exponentially decaying window. The simulations for the proposed Data Selection CG (DS-CG) algorithm are performed in different scenarios.

In the Chapter 3, we introduce the kernel method with the goal of proposing the kernel conjugate gradient (KCG) and its joint use with the data selection method. The conjugate gradient least-squares (CGLS) is presented in Section 2.2, showing the differences from the method concerning the conjugate gradient presented in Chapter 2. The data selection kernel conjugate gradient (DS-KCG) is introduced and then different applications are performed to show the benefits of applying the data selection method during the learning process.

In Chapter 4, we define some concepts of NNs. Then, a concept similar to data selection is applied in NN aiming to reduce the computational cost while achieving results as good as to the standard NN, but with the benefit of eliminating irrelevant or redundant data. The simulations are performed on classification and regression problems with different datasets. In addition, we quantify the computational complexity by counting the number of flops showing the advantage on performing data selection in NNs.

Finally, some conclusions are included in Chapter 5 alongside possible future work.

1.5 Notation

Vectors are denoted by characters in bold type with lower-case letters, whereas non-bold lower-case letters correspond to scalar variables. The kth component of vector $\mathbf{b}$ is represented as b_k. In the same way, the notation employed in matrices is a bold type with upper-case letters. The entries of a matrix $\mathbf{A}$ are of the form a_{mn} in which m represents the row and n the column of $\mathbf{A}$. We represent a column of the matrix $\mathbf{A}$ as $\mathbf{a}_n$ where n is the nth column of $\mathbf{A}$. Similarly, the notation $\mathbf{a}_m^{\mathrm{T}}$ is used to represent the mth row of $\mathbf{A}$. Therefore, the representation of a matrix and a vector can be written as

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M1} & a_{M2} & \dots & a_{MN} \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \dots & \mathbf{a}_N \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_K \end{bmatrix} \tag{1.1}$$

When the elements of a vector (or matrix) represent random variables, the representation is a character in bold italic type, i.e., $\boldsymbol{b}$ ($\boldsymbol{A}$). The distribution $\mathcal{N}(\mathbf{0}, \mathbf{R})$ is a real multivariate Gaussian distribution with zero mean and covariance matrix $\mathbf{R}$. The expected value and variance are represented respectively as $\mathbb{E}[\cdot]$ and $\mathrm{Var}[\cdot]$.

In most cases, we use subscripts in vectors and matrices just to represent the variable name. However, subscripts in parentheses refer to the size of a square matrix. For example, $\mathbf{I}_{(N)}$ is the identity matrix with size $N \times N$.

The real, complex and natural sets are represented by the following symbols $\mathbb{R}$, $\mathbb{C}$ and $\mathbb{N}$. For example, we can state that matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$ and $\mathbf{b} \in \mathbb{R}^K$ in equation (1.1).

In the List of Symbols, we indicate in which chapters these nomenclatures will be used. In some cases, we have the same symbol for two different notations, but with similar meaning. In these circumstances, we chose to use the nomenclature according to the area, causing this duality in the symbol.

The operators used throughout the text are organized in Table 1.1.

Table 1.1: Operators used throughout this work

Operator	**Argument**	**Output**
$(\cdot)^{\mathrm{T}}$	vector or matrix	input vector or matrix with transposed elements
$(\cdot)^{\mathrm{H}}$	vector or matrix	input vector or matrix with transposed and conjugated elements
$(\cdot)^{-1}$	matrix	inverse of input matrix
$\left\|\mathbf{x}\right\|_p$	vector	p-norm, $\left(\sum_{k=1}^{K} \left\|x_k\right\|^p\right)^{1/p}$
$\mathbf{b}_1^H \mathbf{b}_2$	two vectors	inner product in the Euclidean space
$\langle \cdot, \cdot \rangle_{\mathcal{H}}$	two vectors	inner product in the feature space
$\left\| \cdot \right\|$	scalar	absolute value
$Q(\cdot)$	scalar	complementary Gaussian cumulative distribution
$\kappa(\cdot,\cdot)$	two vectors	kernel function
$\otimes$	two matrices	matrix multiplication element by element
$ones(\cdot,\cdot)$	two scalars	matrix composed only of values ones

Chapter 2

Data Selection in Conjugate Gradient Algorithm

The first part of this chapter introduces the concepts of adaptive filtering. Also, it explains the scheme of some classical applications, including input and output settings. Then, properties and definitions of Conjugate Gradient (CG) algorithm are presented with the purpose of investigating the advantages of using data selection in adaptive algorithms.

2.1 Introduction to Adaptive Filtering

Over the last decades, adaptive filtering has achieved considerable success, being incorporated into many technologies, such as mobile phones, radio communication, and medical equipment[1, 25].

Adaptive filtering is a system that models in real time the relationship between an input signal and the desired signal by changing the filter coefficients through an optimization algorithm, such as Conjugate Gradient [29–33], Least Mean Squares (LMS) [1, 34–36] or Recursive Least Squares (RLS) [1, 37, 38]. Basically, the adaptive algorithm updates the filter coefficients over time, using the current error signal. The adaptive filter performs a data-driven approximation step.

In the loop process, an objective function quantifies the performance of the filter in each iteration. The derivative of this objective function shows how much the filter coefficients should be changed according to the current information, aiming at achieving the convergence to an optimal solution. We consider the mean squared error (MSE) as the objective function throughout this chapter.

The input applied to the adaptive filter is denoted by $x(k)$, where k is the iteration index, and $y(k)$ is the filter output. The desired signal is defined by $d(k)$ and the error is calculated as $e(k) = d(k) - y(k)$. The error signal plays an important

role in the adaptive algorithm, determining the update of the filter coefficients. If the adaptive filter achieves the convergence in the learning process, it means that the output signal is matching the desired signal according to some measure, depending on the objective function assumed. The general configuration of an adaptive filter is illustrated in Figure 2.1.

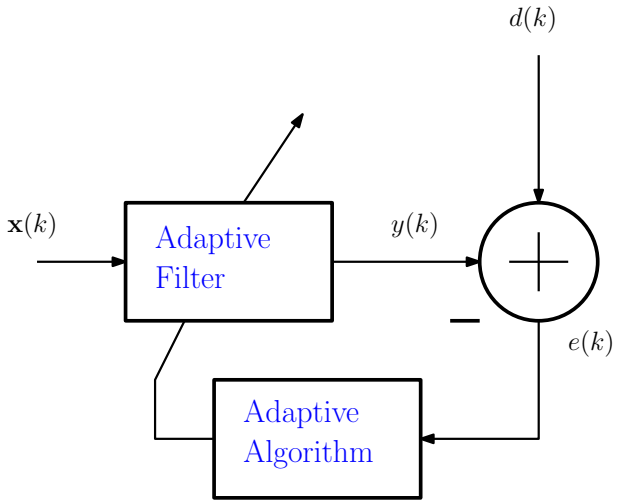


Figure 2.1: The general configuration of an adaptive filter

The input and output signals depend on the application task. The number of different applications in which the adaptive filters are employed has increased during the last decades. Some examples of applications are:

- **System identification**: here, the desired signal consists of the output of an unknown system when submitted to a known signal. The known signal is also used as input in the adaptive filter.

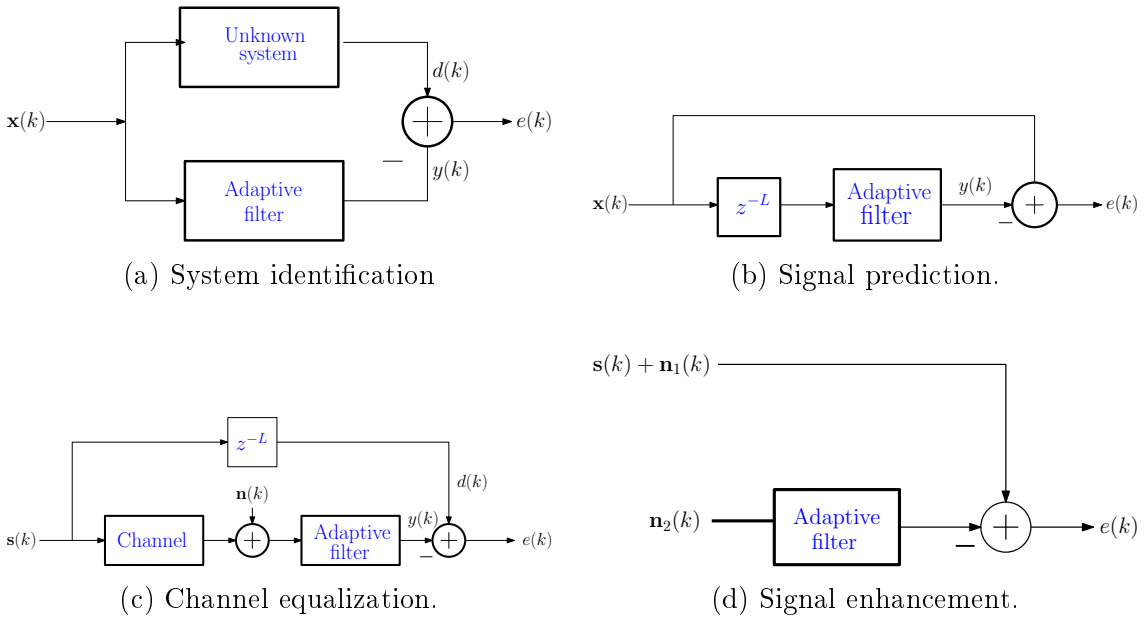
- **Prediction**: in this case, the desired signal is defined as a forward version of the adaptive filter inputs. Its output will correspond to a forecast of the input signal. Mainly, after convergence, the adaptive filter can be considered as a predictor model for the input signal.

- **Channel equalization**: now, the desired signal is produced from a delayed version of an input signal. The known signal is transmitted through a channel and then corrupted by an environment noise to define the inputs of the adaptive filter. In the noiseless case, the final adaptive filter represents an inverse model of the channel.

- **Signal enhancement**: a signal $\mathbf{s}(k)$ corrupted by a noise $\mathbf{n}_1(k)$ is considered as the desired signal. Another signal $\mathbf{x} = \mathbf{n}_2(k)$ correlated with the previous noise is used in the adaptive filter. As result, after convergence, the error signal will represent an enhanced version of the signal $\mathbf{s}(k)$.

In the prediction and system identification frameworks, the filter input $\mathbf{x}(k)$ is equal to the input signal $\mathbf{s}(k)$. In our other applications, these values can be different.

The schemes for each one of the pointed cases are shown in Figure 2.2.



(a) System identification

(b) Signal prediction.

(c) Channel equalization.

(d) Signal enhancement.

Figure 2.2: Common applications in adaptive filtering.

2.2 Conjugate Gradient

The history of the conjugate gradient began in the 1950s, as an iterative method proposed by Hestenes and Stiefel to solve linear systems with positive-definite matrices. Around 1960, the first nonlinear conjugate gradient method was introduced by Fletcher and Reeves. Over the years, many variants of the original algorithm have been proposed and some are widely used.

A popular algorithm in adaptive filtering is the RLS, which has a faster convergence than the LMS algorithm, but presents a higher computational cost. One possible solution is to replace the RLS algorithm by the CG algorithm [39, 40]. The CG algorithm has remarkable characteristics that determine its choice as optimizer in this text. Among these characteristics, we can mention fast convergence, low computational cost, small misadjustment and non-requirement of Hessian matrix inversion. One of the main points of CG optimization is the ability to minimize quadratic functions by reducing the cost function through line searches in conjugated directions.

In this section, we propose the Data Selection Conjugate Gradient (DS-CG) algorithm, which accepts only innovative data in the iteration process, avoiding irrelevant and redundant information. This criterion is established from the

distribution of the error signal, such that we define a threshold level to determine if the data quality is sufficient to modify the coefficient of the adaptive filter. Further, an additional threshold level can be utilized to verify if the data represents an outlier. The data selection method has already been proposed in several previous works incorporated in other algorithms (LMS, RLS), showing its efficiency in applications of adaptive filters [19–21, 41, 42].

2.2.1 The Online Conjugate Gradient

In CG method, a natural starting point is to derive the algorithm by looking at the minimization of the objective function

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^H(k)\mathbf{R}\mathbf{w}(k) - \mathbf{p}^H\mathbf{w}(k), \tag{2.1}$$

where $\mathbf{R} = \mathbb{E}[\mathbf{x}(k)\mathbf{x}^H(k)]$ is the autocorrelation matrix of the filter input, $\mathbf{p} = \mathbb{E}[d(k)\mathbf{x}(k)]$ is the cross-correlation between filter input and desired signal, $\mathbf{w}(k)$ is the adaptive coefficient update. The minimization problem (2.1) is equivalent to solve the following system of linear equations

$$\mathbf{R}\mathbf{w}(k) = \mathbf{p}. \tag{2.2}$$

In this method, one distinguishing characteristics is the conjugate property, that is, a set of nonzero vectors $\{\mathbf{u}_0, \cdots, \mathbf{u}_{M-1}\}$ is said to be conjugate with a symmetric positive definite (SPD) matrix $\mathbf{A}$ if

$$\mathbf{u}_i^H\mathbf{A}\mathbf{u}_j = 0, \qquad \text{for all } i \neq j. \tag{2.3}$$

Given an initial point $\mathbf{w}(0)$ and a set of conjugate directions $\{\mathbf{c}(0),...,\mathbf{c}(k)\}$ obtained at each iteration, we can express the coefficient update as

$$\mathbf{w}(k) = \mathbf{w}(k-1) + \alpha(k)\mathbf{c}(k), \tag{2.4}$$

where $\alpha(k)$ is the minimizer of the objective function along $\mathbf{w}(k-1) + \alpha(k)\mathbf{c}(k)$. As a result, we obtain

$$\alpha(k) = \frac{\mathbf{c}^H(k)\mathbf{g}(k-1)}{\mathbf{c}^H(k)\mathbf{R}\mathbf{c}(k)}, \tag{2.5}$$

where $\mathbf{g}(k) = \mathbf{p} - \mathbf{R}\mathbf{w}(k)$ is the negative gradient of the objective function (residual). The iterative equation for $\mathbf{g}(k)$ can be obtained by algebraic manipulations using

the equation (2.4) and the residual equation

$$\mathbf{g}(k) = \mathbf{g}(k-1) - \alpha(k)\mathbf{R}\mathbf{c}(k). \tag{2.6}$$

Another important characteristic that we should mention is related to the set of conjugate vectors: each new conjugate vector $\mathbf{c}(i)$ is generated only using the previous vector $\mathbf{c}(i-1)$ without the need of knowing the previous elements $\{\mathbf{c}(1), \cdots, \mathbf{c}(i-2)\}$. The iterative equation for $\mathbf{c}(k+1)$ is composed by the current negative gradient $\mathbf{g}(k)$ plus the current conjugate vector $\mathbf{c}(k)$ corrected as:

$$\mathbf{c}(k+1) = \mathbf{g}(k) + \beta(k)\mathbf{c}(k), \tag{2.7}$$

where $\beta(k)$ is used as in equation (2.8) to ensure the conjugate property and the convergence of the algorithm. One way of computing $\beta(k)$ is called the Polak-Ribiere method:

$$\beta(k) = \frac{(\mathbf{g}(k) - \mathbf{g}(k-1))^H \mathbf{g}(k)}{\mathbf{g}^H(k-1)\mathbf{g}(k-1)}. \tag{2.8}$$

As can be seen in [40], the correlation matrix $\mathbf{R}$ and the cross-correlation $\mathbf{p}$ can be estimated using an exponentially decaying window. Since the RLS algorithm also uses both estimates, we expect that the performance of the CG and RLS algorithms to be similar.

The correlation and cross-correlation estimation functions are given by

$$\mathbf{R}(k) = \lambda\mathbf{R}(k-1) + \mathbf{x}(k)\mathbf{x}^H(k), \tag{2.9}$$

$$\mathbf{p}(k) = \lambda\mathbf{p}(k-1) + d(k)\mathbf{x}(k), \tag{2.10}$$

wherein λ is the forgetting factor.

From these changes in equations, we can obtain another expression for the negative gradient $\mathbf{g}(k)$, using only equations (2.4), (2.9) and (2.10):

$$\begin{aligned}
\mathbf{g}(k) &= \mathbf{p}(k) - \mathbf{R}(k)\mathbf{w}(k) = \lambda\mathbf{p}(k-1) + d(k)\mathbf{x}(k) \\
&\quad - [\lambda\mathbf{R}(k-1) + \mathbf{x}(k)\mathbf{x}^H(k)][\mathbf{w}(k-1) + \alpha(k)\mathbf{c}(k)] \\
&= \lambda[\mathbf{p}(k-1) - \mathbf{R}(k-1)\mathbf{w}(k-1)] - \alpha(k)[\lambda\mathbf{R}(k-1) \\
&\quad + \mathbf{x}(k)\mathbf{x}^H(k)]\mathbf{c}(k) + \mathbf{x}(k)[d(k) - \mathbf{x}^H(k)\mathbf{w}(k-1)] \\
&= \lambda\mathbf{g}(k-1) - \alpha(k)\mathbf{R}(k)\mathbf{c}(k) + \mathbf{x}(k)e(k),
\end{aligned} \tag{2.11}$$

where $e(k) = d(k) - \mathbf{x}^H(k)\mathbf{w}(k-1)$.

As following described, the step size in equation (2.5) can be substituted by a new equation, computed by an inexact line search scheme. The descent property

valid in the CG algorithm ensures the convergence if [43]:

$$0 \leq \mathbf{c}^H(k)\mathbf{g}(k) \leq 0.5\mathbf{c}^H(k)\mathbf{g}(k-1). \tag{2.12}$$

Multiplying equation (2.11) by $\mathbf{c}^H(k)$, we obtain

$$\mathbf{c}^H(k)\mathbf{g}(k) = \lambda\mathbf{c}^H(k)\mathbf{g}(k-1) - \alpha(k)\mathbf{c}^H(k)\mathbf{R}(k)\mathbf{c}(k) + \mathbf{c}^H(k)\mathbf{x}(k)e(k). \tag{2.13}$$

By applying the expected value in both sides of equation (2.13), we have

$$\mathbb{E}[\mathbf{c}^H(k)\mathbf{g}(k)] = \lambda\mathbb{E}[\mathbf{c}^H(k)\mathbf{g}(k-1)] - \mathbb{E}[\alpha(k)]\mathbb{E}[\mathbf{c}^H(k)\mathbf{R}(k)\mathbf{c}(k)] + \mathbb{E}[\mathbf{c}^H(k)]\mathbb{E}[\mathbf{x}(k)e(k)], \tag{2.14}$$

where in the last term $\mathbf{c}(k)$ was considered uncorrelated with $\mathbf{x}(k)$ and $e(k)$ and the input and error are orthogonal in mean when the algorithm converges. Using the Wiener-Hopf equation $\mathbf{R}\mathbf{w}^* = \mathbf{p}$ and assuming that the algorithm converges, the last term tends to zero. Then, the expected value of the step size can be isolated as

$$\mathbb{E}[\alpha(k)] = \frac{\lambda\mathbb{E}[\mathbf{c}^H(k)\mathbf{g}(k-1)] - \mathbb{E}[\mathbf{c}^H(k)\mathbf{g}(k)]}{\mathbb{E}[\mathbf{c}^H(k)\mathbf{R}(k)\mathbf{c}(k)]}. \tag{2.15}$$

Therefore, using the inequality (2.12) and equation (2.15), we have

$$(\lambda - 0.5)\frac{\mathbb{E}[\mathbf{c}^H(k)\mathbf{g}(k-1)]}{\mathbb{E}[\mathbf{c}^H(k)\mathbf{R}(k)\mathbf{c}(k)]} \leq \mathbb{E}[\alpha(k)] \leq \lambda\frac{\mathbb{E}[\mathbf{c}^H(k)\mathbf{g}(k-1)]}{\mathbb{E}[\mathbf{c}^H(k)\mathbf{R}(k)\mathbf{c}(k)]}. \tag{2.16}$$

Finally, from inequalities (2.16), the modified step size in the coefficient update is obtained

$$\alpha(k) = \eta\frac{\mathbf{c}^H(k)\mathbf{g}(k-1)}{\mathbf{c}^H(k)\mathbf{R}\mathbf{c}(k)}, \qquad (\lambda - 0.5) \leq \eta \leq \lambda. \tag{2.17}$$

2.2.2 Problem Statement

In this subsection, the proposed data selection method is explained in more detail, and hence the Data Selective Conjugate Gradient algorithm is proposed [21, 42]. The main goal of the data selection technique is to reduce the computational cost and to show that the use of all available information may not be necessary. In many applications based on adaptive filtering, updating the filter coefficients with a certain probability produces almost the same results as when the coefficients are updated 100% of the time. Therefore, some coefficient updates can be avoided.

This method presents different configurations depending on the application we are dealing with. But regardless of the application, the filter output is formulated as

$$y(k) = \mathbf{w}^H(k-1)\mathbf{x}(k), \tag{2.18}$$

where $\mathbf{x}(k) = [x_0(k)\ x_1(k) \dots x_{N-1}(k)]^T$ is the input applied to the adaptive filter and $\mathbf{w}(k-1) = [w_0(k-1)\ w_1(k-1) \dots w_{N-1}(k-1)]^T$ represents the adaptive filter coefficients. The error is given by

$$e(k) = d(k) - y(k) = d(k) - \mathbf{w}^H(k-1)\mathbf{x}(k), \tag{2.19}$$

where $d(k)$ is the desired signal.

As illustrated in Figure 2.3, the data selection method uses the error signal $e(k)$ to determine if the current data carries innovative information, and depending on the result, the filter coefficients will be updated. The center interval (in light blue) corresponds to non-informative values, whereas the edge intervals (in red) represent possible outliers. Both regions are discarded throughout process iterations. The probability P_{up} defines the threshold established for deciding whether we update the algorithm coefficients, this value is explained in more detail shortly.

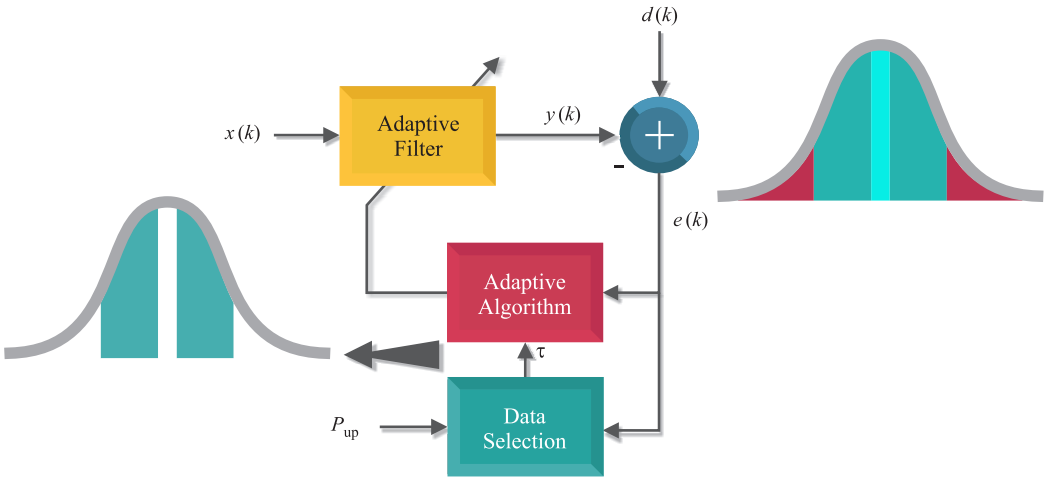


Figure 2.3: Data selection strategy.

In the previously mentioned applications, the error distribution is assumed as Gaussian,

$$\boldsymbol{e} \sim \mathcal{N}(0,\, \sigma_e^2) \tag{2.20}$$

where σ_e^2 is the error variance. By normalizing the error distribution, we obtain

$$\frac{\boldsymbol{e}}{\sigma_e} \sim \mathcal{N}(0,\, 1). \tag{2.21}$$

The objective function plays a crucial role in performing the coefficient update. One of the most common objective functions is the instantaneous squared error

$$J(\mathbf{w}(k-1)) = \frac{1}{2}|e(k)|^2, \tag{2.22}$$

where $|\cdot|$ denotes the absolute value.

Using the normalized error (2.21) and objective function (2.22), the update rule for the adaptive filter coefficients can be established. The decision criteria for not updating the coefficients are as follows: if the $\frac{|e(k)|}{\sigma_e}$ is greater than a given threshold $\sqrt{\tau(k)}$ (explained in more detail soon) or if the $\frac{|e(k)|}{\sigma_e}$ is below another threshold $\sqrt{\tau_{\max}}$. In equation (2.22) these rules are incorporated by

$$J'(\mathbf{w}(k-1)) = \begin{cases} \frac{1}{2}|e(k)|^2, \text{ if } \sqrt{\tau(k)} \leq \frac{|e(k)|}{\sigma_e} < \sqrt{\tau_{\max}} \\ 0, \text{ otherwise.} \end{cases} \tag{2.23}$$

Since the update depends of the objective function, the update rule for the coefficients is written as

$$\mathbf{w}(k) = \begin{cases} \mathbf{w}(k-1) + \mathbf{u}(k), \ \sqrt{\tau(k)} \leq \frac{|e(k)|}{\sigma_e} < \sqrt{\tau_{\max}} \\ \mathbf{w}(k-1), \text{ otherwise,} \end{cases} \tag{2.24}$$

where the term $\mathbf{u}(k)$ depends on the adaptive algorithm employed. Since we want to represent how often the data are updated, we describe the desired probability of update $P_{\text{up}}(k)$ as

$$P_{\text{up}}(k) = P\left\{\frac{|e(k)|}{\sigma_e} > \sqrt{\tau(k)}\right\} - P\left\{\frac{|e(k)|}{\sigma_e} > \sqrt{\tau_{\max}}\right\}. \tag{2.25}$$

By considering the distribution in (2.21), equation (2.25) in steady-state becomes

$$P_{\text{up}} = 2Q_e\left(\sqrt{\tau}\right) - 2Q_e\left(\sqrt{\tau_{\max}}\right), \tag{2.26}$$

where $Q_e(\cdot)$ is the complementary Gaussian cumulative distribution function, given by $Q_e(x) = 1/(2\pi)\int_x^\infty exp(-t^2/2)dt$ [44]. The probability $P\left\{\frac{|e(k)|}{\sigma_e} > \sqrt{\tau_{\max}}\right\}$ is omitted as a result of this value being too small, even when outliers are present in the dataset. Therefore, the parameter $\sqrt{\tau}$ can be obtained from the equation (2.26) as

$$\sqrt{\tau} = Q_e^{-1}\left(\frac{P_{\text{up}}}{2}\right), \tag{2.27}$$

where $Q_e^{-1}(\cdot)$ is the inverse of the $Q_e(\cdot)$ function. We point out that in the system identification application, the minimum MSE in steady-state is σ_n^2, the variance of the measurement noise $n(k)$. Furthermore, σ_e^2 is expressed as a function of the noise variance

$$\sigma_e^2 = (1+\rho)\sigma_n^2, \tag{2.28}$$

so the excess MSE is rewritten as $\rho\sigma_n^2$. Consequently, the update of coefficients is performed according to a scaled power noise, $\tau(k)\sigma_n^2$ [19–21], since the term of the

excess MSE is negligible. Therefore an equivalent expression to equation (2.25) for the identification system application is derived as

$$P_{\text{up}}(k) = P\left\{\frac{|e(k)|}{\sigma_n} > \sqrt{\tau(k)}\right\} - P\left\{\frac{|e(k)|}{\sigma_n} > \sqrt{\tau_{\max}}\right\} \tag{2.29}$$

resulting in the following modifications in equations (2.26) and (2.27)

$$P_{\text{up}} = 2Q_e\left(\frac{\sigma_n\sqrt{\tau}}{\sigma_e}\right) - 2Q_e\left(\frac{\sigma_n\sqrt{\tau_{\max}}}{\sigma_e}\right) \tag{2.30}$$

$$\sqrt{\tau} = \sqrt{(1+\rho)}Q_e^{-1}\left(\frac{P_{\text{up}}}{2}\right). \tag{2.31}$$

The threshold for outliers, $\tau_{\max}$, is planned from a statistical concept, called empirical rule [45]. The values exceeding the threshold are identified as outliers. Initially, in the first 20% of data, the threshold is not taken into account, hence obtaining an estimate of the error behavior. For the remaining iterations, it is calculated by

$$\sqrt{\tau_{\max}} = \mathbb{E}[|e(k)|/\sigma_e] + 3\text{Var}[|e(k)|/\sigma_e]. \tag{2.32}$$

Since the expression (2.21) represents a Gaussian distribution, the empirical rule is used in this problem to detect outliers.

An estimate to variance error is

$$\sigma_e^2 = (1-\lambda_e)e^2(k) + (\lambda_e)\sigma_e^2, \tag{2.33}$$

where λ_e is a forgetting factor.

Under our assumptions regarding the error distribution, $\mathbb{E}[e(k)] = 0$ and thus

$$\sigma_e^2 = \mathbb{E}[e^2(k)] = \xi(k), \tag{2.34}$$

where $\xi(k)$, as $k \to \infty$, is the steady-state MSE obtained by the algorithm employed. The filter application and the employed algorithm play fundamental role in the expression of $\xi(k)$. In the following pages, the steady-state MSE is computed for some adaptive filter applications.

A. System identification

System identification is one of the most important applications in adaptive filtering. The desired signal can be formulated as

$$d(k) = \mathbf{w}_{\text{o}}{}^H\mathbf{x}(k) + n(k), \tag{2.35}$$

where $\mathbf{w_o}$ is the optimal coefficient, $\mathbf{x}(k)$ is the input vector and $n(k)$ is the Gaussian noise with zero mean and variance σ_n^2. Therefore, replacing (2.35) in (2.19), the expression for the MSE can be expressed as

$$\begin{aligned}\xi(k) = \mathbb{E}[e^2(k)] = \mathbb{E}[n^2(k)] - 2\mathbb{E}[n(k)\Delta\mathbf{w}^H(k-1)\mathbf{x}(k)] \\ + \mathbb{E}[\Delta\mathbf{w}^H(k-1)\mathbf{x}(k)\mathbf{x}^H(k)\Delta\mathbf{w}(k-1)],\end{aligned} \tag{2.36}$$

in which $\Delta\mathbf{w}(k-1) = \mathbf{w}(k-1) - \mathbf{w}_o$. Assuming that the noise and inputs are uncorrelated, the second term in (2.36) is zero and we obtain the following result

$$\xi(k) = \sigma_n^2 + \xi_{\text{exc}}(k), \tag{2.37}$$

where $\xi_{\text{exc}}(k)$ is the excess MSE and $\mathbb{E}[n^2(k)] = \sigma_n^2$. Since the excess MSE is a function of the additional noise:

$$\xi(k) = \sigma_e^2 = (1+\rho)\sigma_n^2. \tag{2.38}$$

The expression suitable for ρ in the CG algorithm case is the same as in the RLS algorithm. This result is due to the equivalence between CG and RLS algorithms in steady state [1], as discussed below in more detail.

The filter coefficients in steady state satisfy $\mathbf{w}(k) \approx \mathbf{w}(k-1)$. Thus, one can obtain $\alpha(k)\mathbf{c}(k) \approx \mathbf{0}$ in (2.4). Therefore, multiplying both sides in equation (2.17) by $\alpha(k)$, it is possible to yield $\alpha(k) \approx 0$. Thereby, equation (2.6) becomes $\mathbf{g}(k) \approx \mathbf{g}(k-1)$ and, in equation (2.8) we have $\beta(k) \approx 0$. Following the theory, by using equation (2.7), we can verify that $\mathbf{c}(k+1) \approx \mathbf{g}(k)$ and replacing this result in (2.17), one can show that $\mathbf{g}(k) \approx 0$. Since $\mathbf{g}(k)$ is defined as the negative gradient of the objective function, we are led to infer that $\mathbf{w}(k) \approx \mathbf{R}^{-1}(k)\mathbf{p}(k)$, which is the same result obtained in RLS algorithm. In addition, CG and RLS algorithms estimate matrix $\mathbf{R}$ and vector $\mathbf{p}$ by using equations (2.9) and (2.10), respectively. As a result, it can be stated that the CG and RLS algorithms are equivalent in steady-state, giving rise to the following expression for misadjustment [1]:

$$\rho = \frac{\xi_{\text{exc}}}{\xi_{\text{min}}} \approx (N+1)\frac{P_{\text{up}}(1-\lambda)}{2 - P_{\text{up}}(1-\lambda)}, \tag{2.39}$$

where ξ_{min} is the minimum MSE.

As established above, $\mathbf{w}(k) = \mathbf{R}^{-1}(k)\mathbf{p}(k) = \mathbf{R}^{-1}\mathbf{p} = \mathbf{w}_o$, when k tends to infinity and equations (2.9) and (2.10) are used to estimate $\mathbf{R}$ and vector $\mathbf{p}$. Using these conclusions, the derivation of the equation (2.40) follows the same steps as the

RLS algorithm, see [1] (pp. 226):

$$\Delta\mathbf{w}(k) = \lambda\mathbf{R}^{-1}(k)\mathbf{R}(k-1)\Delta\mathbf{w}(k-1) + \mathbf{R}^{-1}(k)\mathbf{x}(k)e(k) \tag{2.40}$$

where $\Delta\mathbf{w}(k) = \mathbf{w}(k) - \mathbf{w}_o$.

Considering that the DS-CG updates take place only when there is informative data, we can apply an analytical model containing the desired probability of update P_{up}:

$$\begin{aligned}\Delta\mathbf{w}(k) &= \Delta\mathbf{w}(k-1) \\ &\quad + P_{\text{up}}[\lambda\mathbf{R}^{-1}(k)\mathbf{R}(k-1) - \mathbf{I}]\Delta\mathbf{w}(k-1) \\ &\quad + P_{\text{up}}\mathbf{R}^{-1}(k)\mathbf{x}(k)e_o(k).\end{aligned} \tag{2.41}$$

Using the equation above and a similar derivation of the excess MSE in [1], pp. 226-229, we obtain the equation (2.40).

B. Signal Prediction

In the signal prediction application, the desired signal $d(k)$ is an advanced version of the input signal $x(k)$. Therefore, from the error signal,

$$e(k) = x(k+L) - \mathbf{w}^H(k-1)\mathbf{x}(k), \tag{2.42}$$

and the MSE expression

$$\xi(k) = \mathbb{E}[e^2(k)] = \mathbb{E}[(x(k+L) - \mathbf{w}^H(k-1)\mathbf{x}(k))^2], \tag{2.43}$$

it is possible to derive an expression for the minimum MSE [1]:

$$\xi_{\min}(k) = r(0) - \mathbf{w}_{\text{o}}^H \begin{bmatrix} r(L) \\ r(L+1) \\ \vdots \\ r(L+N) \end{bmatrix}, \tag{2.44}$$

where $\mathbf{w}_{\text{o}}$ is the optimal coefficients of the predictor and $r(l) = \mathbb{E}[x(k)x(k-l)]$ for a stationary process. In the prediction case, equation (2.44) can estimate σ_e^2 at iteration k, simply by replacing $\mathbf{w}_{\text{o}}^H$ by $\mathbf{w}(k-1)$, since in steady-state this is a good estimate. We estimate $r(l)$ through $r(l) = \lambda_{pred}r(l-1) + (1-\lambda_{pred})x(k)x(k-l)$, in which $0 < \lambda_{pred} < 1$ is a forgetting factor.

C. Equalization

In the equalization case, the desired signal is a delayed version of the input signal and the adaptive filter input is composed by the received signal applied in a channel plus a noise as seen at the end of Subsection 2.1. Thus, the error signal can be written as

$$e(k) = d(k) - \mathbf{w}^H(k-1)\mathbf{x}(k) = s(k-L) - \mathbf{w}^H(k-1)(\mathbf{H}\mathbf{s}(k) + \mathbf{n}(k)), \quad (2.45)$$

where $\mathbf{H} \in \mathbb{C}^{N\times L}$ is the finite impulse response (FIR) channel convolution matrix, $\mathbf{s}(k) = [s_0(k)\ s_1(k)\dots s_{L-1}(k)]^T \in \mathbb{R}^L$ is the input signal and $\mathbf{n}(k) = [n_0(k)\ n_1(k)\dots n_{N-1}(k)]^T \in \mathbb{R}^N$ is the noise drawn from an independent Gaussian distribution with zero mean and variance σ_{n}^2. Therefore, we can compute the MSE as

$$\begin{aligned}
\xi(k) &= \mathbb{E}[e^2(k)] = \mathbb{E}[(d(k) - y(k))^2] = \mathbb{E}[(s(k-L) - \mathbf{w}^H(k-1)(\mathbf{H}\mathbf{s}(k) + \mathbf{n}(k)))^2] \\
&= \sigma_s^2 - 2\mathbb{E}[s^H(k-L)(\mathbf{w}^H(k-1)(\mathbf{H}\mathbf{s}(k) + \mathbf{n}(k)))] + \mathbb{E}[(\mathbf{w}^H(k-1)(\mathbf{H}\mathbf{s}(k) + \mathbf{n}(k)))^2] \\
&= \sigma_s^2 - 2\mathbf{w}^H(k-1)\mathbf{H}\mathbb{E}[s^H(k-L)\mathbf{s}(k)] + \mathbf{w}^H(k-1)\mathbf{H}\mathbb{E}[\mathbf{s}^H(k)\mathbf{s}(k)]\mathbf{H}^H\mathbf{w}(k-1) \\
&+ \mathbf{w}^H(k-1)\mathbb{E}[\mathbf{n}^H(k)\mathbf{n}(k)]\mathbf{w}(k-1) = \sigma_s^2 - 2\mathbf{w}^H(k)\mathbf{H}\mathbf{r}_l \\
&+ \mathbf{w}^H(k-1)(\mathbf{H}^H\mathbf{R}\mathbf{H} + \mathbf{I}_N\sigma_n^2)\mathbf{w}(k-1) \\
&\approx \sigma_s^2(1 - 2\mathbf{w}^H(k-1)\mathbf{h}_l + \mathbf{w}^H(k-1)\mathbf{H}^H\mathbf{H}\mathbf{w}(k-1)) + \sigma_n^2\mathbf{w}^H(k-1)\mathbf{w}(k-1),
\end{aligned} \quad (2.46)$$

where $\mathbf{R}$ is the autocorrelation matrix of the signal received in the input, $\mathbf{r}_l$ is the l-th column of the autocorrelation matrix, and we denote $\mathbf{h}_l = \mathbf{H}\mathbf{r}_l$. Furthermore, we are assuming that the inputs and the additional noise are uncorrelated.

When the channel model is unknown, a practical way of computing the data-selection threshold is estimating the output error variance by (2.33).

D. Signal enhancement

In the signal enhancement case, the desired signal is a signal corrupted by a noise

$$d(k) = s(k) + n_1(k). \quad (2.47)$$

Using another noise correlated with the noise established in (2.47) as being the adaptive filter input

$$\mathbf{x}(k) = \mathbf{n}_2(k), \quad (2.48)$$

the error signal $e(k)$ will be an enhancement version of $d(k)$ and the adaptive filter output $y(k)$ will be the actual error. For this reason, in this signal enhancement application, the MSE is calculated based on the variance of $y(k)$ instead of $e(k)$.

Hence, the MSE expression for signal enhancement is obtained as

$$\xi(k) = \sigma_y^2 = \mathbb{E}[y^2(k)] = \mathbb{E}[(\mathbf{w}^H(k-1)\mathbf{n}_2(k))^2] = \sigma_{n_2}^2(k)||\mathbf{w}(k-1)||_2^2. \tag{2.49}$$

Table 2.1: Data-Selective Conjugate Gradient algorithm

DS-CG algorithm

Initialize

λ, η with $(\lambda - 0.5) \leq \eta \leq \lambda$, $\mathbf{w}(0)$ = random vectors or zero vectors

$R(0) = \mathbf{I}$, $\mathbf{g}(0) = \mathbf{c}(1) = zeros(N+1,1)$, γ = small constant for regularization

Prescribe P_{up}, and choose $\tau_{\max}$

$\sqrt{\tau} = \sqrt{(1+\rho)}Q^{-1}(\frac{P_{\text{up}}}{2})$

For system identification use $\rho = (N+1)\frac{P_{\text{up}}(1-\lambda)}{2-P_{\text{up}}(1-\lambda)}$ and $\sigma_e^2 = (1+\rho)\sigma_n^2$.

For prediction, equalizer and enhancement use $\sigma_e^2 = (1-\lambda_e)e^2(k) + (\lambda_e)\sigma_e^2$,

where λ_e is chosen.

Do for $k > 0$

acquire $\mathbf{x}(k)$ and $d(k)$

$e(k) = d(k) - \mathbf{w}^H(k-1)\mathbf{x}(k)$

$$\delta(k) = \begin{cases} 0, \text{ if } -\sqrt{\tau} \leq \frac{|e(k)|}{\sigma_e} \leq \sqrt{\tau} \\ 0, \text{ if } \frac{|e(k)|}{\sigma_e} \geq \sqrt{\tau_{\max}} \\ 1, \text{ otherwise} \end{cases}$$

if $\delta(k) = 0$

$\mathbf{w}(k) = \mathbf{w}(k-1)$

if $\frac{|e(k)|}{\sigma_e} \geq \sqrt{\tau_{\max}}$

$e(k) = 0$, $d(k) = 0$

end if

else

$\mathbf{R}(k) = \lambda\mathbf{R}(k-1) + \mathbf{x}(k)\mathbf{x}^H(k)$

$\alpha(k) = \eta\frac{\mathbf{c}^H(k)\mathbf{g}(k-1)}{[\mathbf{c}^H(k)\mathbf{R}(k)\mathbf{c}(k)+\gamma]}$

$\mathbf{g}(k) = \lambda\mathbf{g}(k-1) - \alpha(k)\mathbf{R}(k)\mathbf{c}(k) + \mathbf{x}(k)e(k)$

$\mathbf{w}(k) = \mathbf{w}(k-1) + \alpha(k)\mathbf{c}(k)$

$\beta(k) = \frac{[\mathbf{g}(k)-\mathbf{g}(k-1)]^H\mathbf{g}(k)}{[\mathbf{g}^H(k-1)\mathbf{g}(k-1)+\gamma]}$

$\mathbf{c}(k+1) = \mathbf{g}(k) + \beta(k)\mathbf{c}(k)$

end if

end

We can observe from in Table 2.1 that in the DS-CG algorithm the error signal $e(k)$ is calculated before the coefficient update, avoiding extra computations in equations (2.9), (2.4), (2.11), (2.8) and (2.7). Moreover, the parameter ρ is null in prediction, equalizer and enhancement applications and defined in equation (2.39) for the system identification cases.

2.2.3 Simulation Results

In this subsection, in order to illustrate our contributions, the data selection method is applied in some examples, using both synthetic and real-world data to verify the performance of the DS-CG algorithm. The probability of update varies from 0% to 100% and it is compared to the estimated probability of update $\hat{P}_{\text{up}}$ to verify the efficiency of the estimate. All simulations presented in this chapter are obtained by the average of 200 independent Monte Carlo runs. The algorithms in this chapter are implemented in MATLAB and available online on GitHub [46]. The simulations are performed in a computer with Intel Core i7-7500U CPU 2.70GHz x4 processor and 15.5 GB of memory.

A. System identification

In this application, our goal is to identify an unknown system, described as:

$$\mathbf{h} = [0.1010\ 0.3030\ 0\ -0.2020\ -0.4040\ -0.7071\ -0.4040\ -0.2020]^T. \quad (2.50)$$

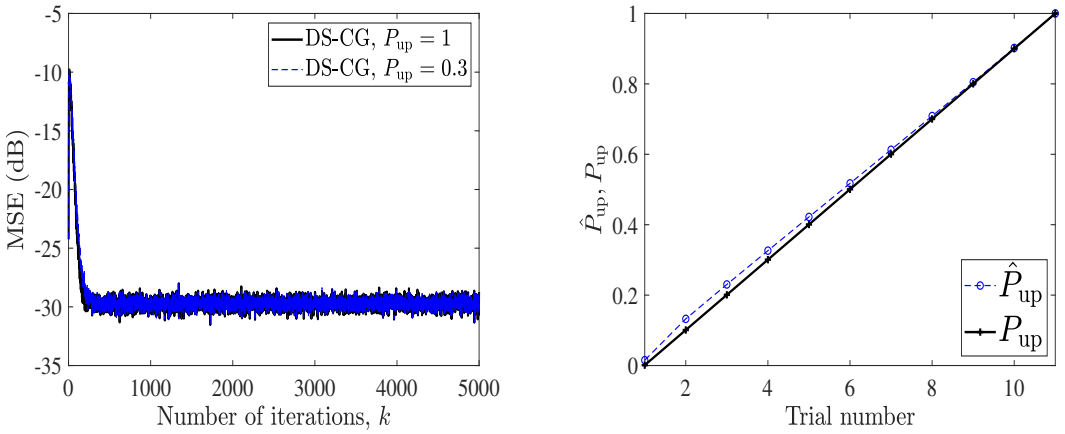
The desired output is written as $d(k) = \mathbf{h}^T\mathbf{x}(k) + n(k)$, wherein the input $\mathbf{x}(k)$ is obtained from a Gaussian distribution with zero mean and unit variance, and $n(k)$ consists of a Gaussian noise with zero mean and variance $\sigma_n^2 = 10^{-3}$. Two cases are considered in the experiment: first-order and fourth-order AR processes, given by

$$\begin{aligned} x(k) &= 0.88x(k-1) + n_1(k), \\ x(k) &= -0.55x(k-1) - 1.221x(k-2) - 0.49955x(k-3) \\ &\quad - 0.4536x(k-4) + n_2(k), \end{aligned}$$

where $n_1(k)$ and $n_2(k)$ are Gaussian noises with zero mean and uncorrelated with each other and with the additional noise $n(k)$. The variances $\sigma_{n_1}^2$ and $\sigma_{n_2}^2$ are chosen so that the input variance is unitary. The filter order is $N = 7$ to ensure the convergence of the filter coefficients. The parameters chosen for the system identification are $\lambda = 0.98$, $\eta = 0.48$ and $\gamma = 10^{-4}$.

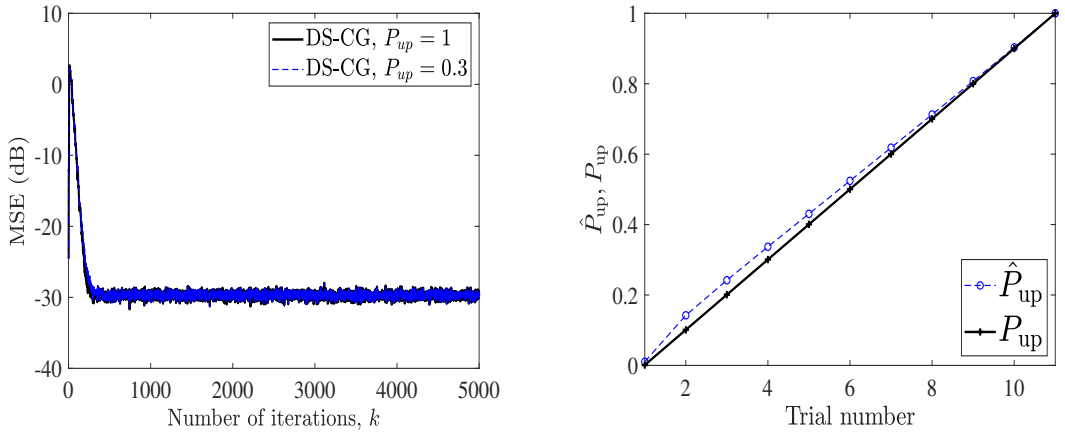
The results for the learning curves in the fourth-order and the first-order AR

processes are presented in Figure 2.4a and 2.5a, respectively. We can observe that the CG algorithm present fast convergence to the steady-state. In particular, the order increase of the input signal does not interfere with the result of the fourth-order case. The estimated probability of update depicted in Figures 2.4b and 2.5b, is closer to the prescribed probability of update.



(a) (b)

Figure 2.4: Simulation A: Fourth-order AR input signal, (a) Learning curves for the data selection and (b) Comparison between the desired P_{up} and achieved $\hat{P}_{\text{up}}$.



(a) (b)

Figure 2.5: Simulation A: First-order AR input signal, (a) Learning curves for the data selection and (b) Comparison between the desired P_{up} and achieved $\hat{P}_{\text{up}}$.

In another example, using the fourth-order input signal, we consider that the outliers are present in the desired signal $d(k)$. The outlier signal is included in 1% of the reference signal with an amplitude equal to five. The misalignment in the adaptive filter coefficients is measured and it is defined as $\frac{||\mathbf{w}(k)-\mathbf{w}_o||}{||\mathbf{w}_o||}$, where $\mathbf{w}_o$ represents the optimal coefficients for the system identification problem. As observed in Table 2.2, the misalignment is greater when outliers are present but ignored. The

level of misalignment achieved when considering outliers for $P_{\text{up}} = 0.3$ approaches the one in which no outliers are present with $P_{\text{up}} = 1$. As a result, we obtain a satisfactory average misalignment for $P_{\text{up}} = 0.1$ when compared with the case that no outliers are present and $P_{\text{up}} = 1$.

Table 2.2: Misalignment with outliers, in dBs.

	Outlier	**yes**	**yes**	**yes**	**no**
	$\tau_{\max}$ on	yes	no	yes	no
	P_{up}	0.3	0.3	0.1	1
Average Misalignment (dB)	DS-CG	-33.29	-15.25	-30.47	-33.37

B. Signal Prediction

The specification of the data considered in this subsection is provided by Google RE<C Initiative in [47]. The data consists of the wind speed recorded by five sensors on May 25th, 2011. The data are divided into 40 sets of size 8192 in order to use the Monte Carlo method.

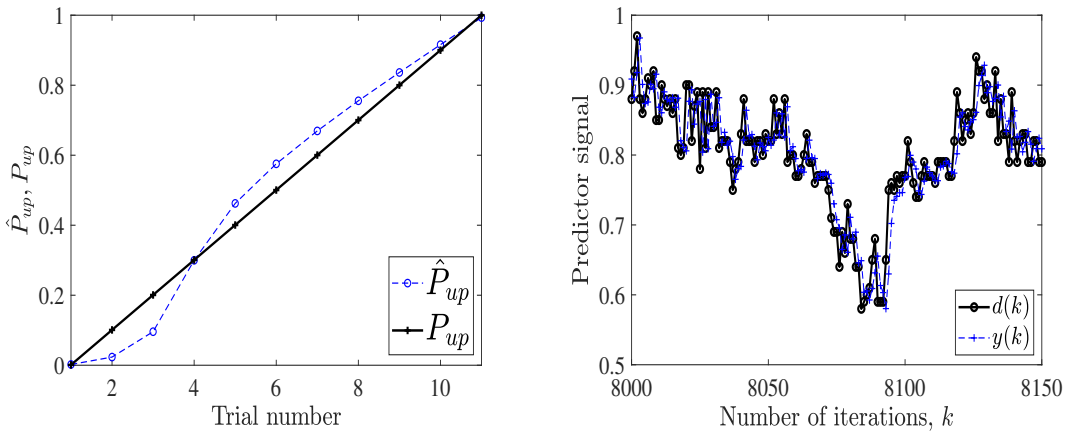
The parameters chosen for the application are $\lambda = 0.98$, $\eta = 0.48$ and $\gamma = 10^{-4}$. The filter order employed is $N = 7$. Figure 2.6*a* illustrates the comparison between the estimated probability of update $\hat{P}_{\text{up}}$ and the prescribed probability of update P_{up}. We can observe that the estimation obtained in the CG algorithm always achieves a value close to the prescribed probability. The comparison between the predicted signal $y(k)$ and reference signal $d(k)$ is shown in Figure 2.6*b* for $P_{\text{up}} = 0.4$. Indeed, a good performance is achieved even if the number of updates is reduced, thus it is not necessary to use all data to obtain an accurate prediction.

The evolution of the MSE can be seen in Figure 2.7. It is noted that the algorithm attains a fast convergence to the steady state.

C. Equalizer

In this subsection, finite impulse response (FIR) channel provided by The Signal Processing Information Base repository [48] are considered. Also, the complex channel taps were obtained from digital microwave radio systems measurements. The FIR model frequency response is illustrated in Figure 2.8a (in black).

A random Gaussian variable is attributed to the transmitted signal $\mathbf{s}(k)$ with zero mean and unitary variance. The signal traverses the complex channel and it is corrupted by a Gaussian noise with variance $\sigma_n^2 = 10^{-3}$. The parameters chosen in



(a) (b)

Figure 2.6: Simulation B: (a) Comparison between the desired P_{up} and achieved $\hat{P}_{\text{up}}$ by the DS-CG algorithm and (b) Comparison between the desired signal and the predicted by the DS-CG algorithm for $P_{up} = 0.4$.

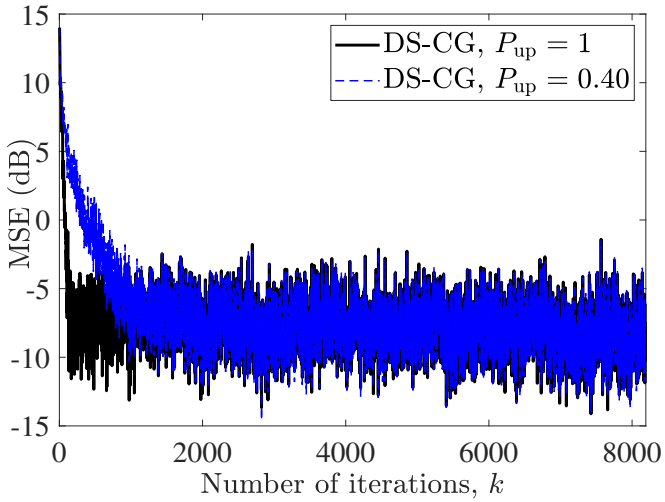


Figure 2.7: Simulation B: Learning curves for data selection in $P_{\text{up}} = 0.4$ and $P_{\text{up}} = 1$.

the configuration of the algorithm were $\lambda = 0.9995$, $\eta = 0.49$ and $\gamma = 10^{-4}$. In this case, we need a filter with higher order, so that we set $N = 100$. The error variance was estimated as in equation (2.33) for $b = 0.9999$.

As we are in a complex signal case, after the coefficients filter converge to their steady-state, the probability of update for white circularly-symmetric Gaussian input signals is modeled as Rayleigh probability distribution function given by

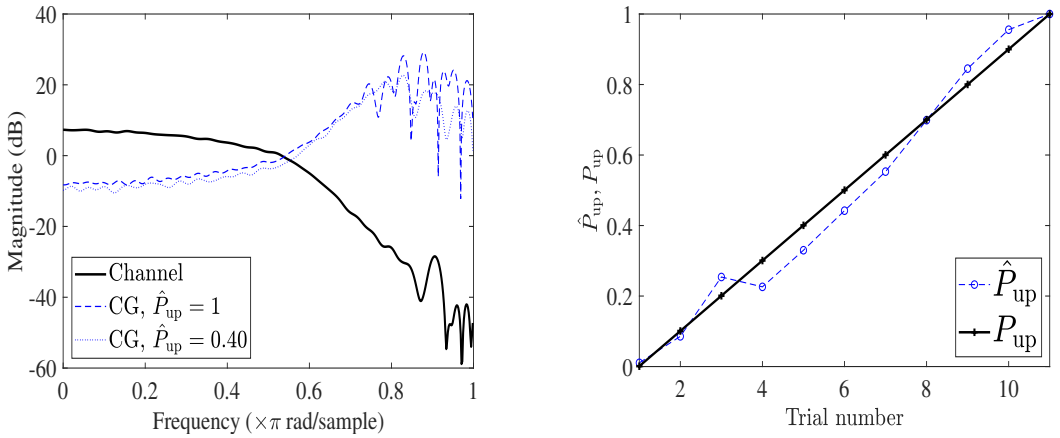
$$1 - P_{\text{up}} = F_E(x) = \left[1 - \exp^{-\frac{x^2}{2\sigma_e}}\right] u(x), \tag{2.51}$$

where the $u(x)$ is the unit step function and $x = \sqrt{\tau}\sigma_n$. Assuming that $\sigma_e \approx \sigma_{\text{n}}$,

the formula to the threshold is obtained as

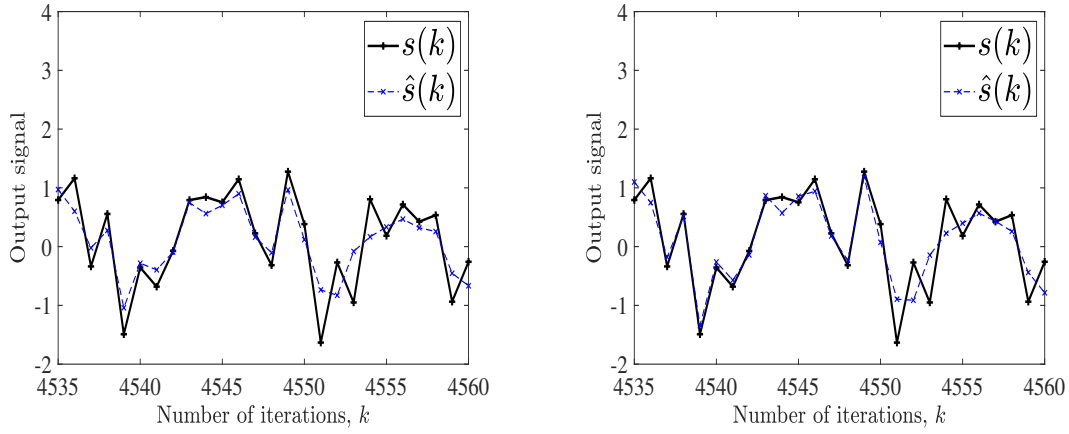
$$\tau = 2 \ln \left(\frac{1}{P_{\text{up}}} \right), \tag{2.52}$$

where $\rho = 0$. In Figure 2.8a the performance of the DS-CG algorithm is shown, for $P_{\text{up}} = 1$ and $P_{\text{up}} = 0.4$, and the filter output is an equalized version of the transmitted signal $\mathbf{s}(k)$, i.e., the algorithm tries to invert the channel frequency response. The estimated probability of update $\hat{P}_{\text{up}}$ is quite close to the real probability of update P_{up} as can be noticed in Figure 2.8b.



(a) (b)

Figure 2.8: Simulation C: (a) Frequency response of the channel and the data-selective filter (b) Comparison between the desired P_{up} and achieved $\hat{P}_{\text{up}}$ by the algorithm.



(a) (b)

Figure 2.9: Simulation C: Comparison between the transmitted and the recovered signals by the DS-CG algorithm for (a) $P_{\text{up}} = 0.4$ and (b) $P_{\text{up}} = 1$.

By comparing Figure 2.9a and 2.9b, it is possible to obtain the transmitted signal from only 40% of the input data with almost the same accuracy obtained as when

100% of the input data is used. Again we note that the data selection method is beneficial when jointly used in the CG algorithm. The result in Figure 2.10 further highlights the power of the data selection method since the learning curve in DS-CG with $P_{\text{up}} = 0.4$ yields similar results to the DS-CG algorithm with $P_{\text{up}} = 1$.

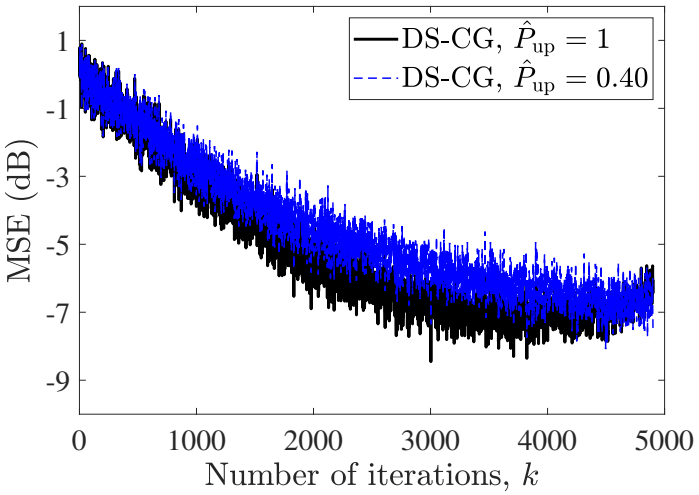


Figure 2.10: Simulation C: Learning curves for data selection in $P_{\text{up}} = 0.4$ and $P_{\text{up}} = 1$.

2.3 Concluding Remarks

In this chapter, the proposed data selection method is tested in the main adaptive filtering applications. The method was applied to the CG algorithm with some modifications depending on the application. In all the simulations, data selection achieves excellent performance even when less than 50% of times the coefficients are updated, leading to consistent results.

Chapter 3

Data Selection in Kernel Conjugate Gradient Algorithm

As seen in the previous chapter, the classical adaptive filtering algorithms consist of adapting the coefficients of linear filters in real-time. However, some applications includes phenomena that cannot be well modeled with linear adaptive systems. Although linear models may perform reasonable in some of these cases, the demand for high-speed communications and the improvements in the computer capacity in processing information drives the exploration of more sophisticated techniques.

Currently, one popular method used to improve performance in these applications is the kernel adaptive filter. In this chapter, the main concepts concerning kernel adaptive filtering are presented. Subsequently, the performance of the data selection method is verified in some applications exploiting the kernel conjugate gradient (KCG) algorithm. The data selection KCG (DS-KCG) algorithm is proposed originally in this work.

3.1 Kernel Conjugate Gradient

In the last two decades, the interest on applying kernel methods to nonlinear problems tackled by the Machine Learning and Signal Processing areas has been growing. In addition to adaptive filtering studied in this dissertation, we can name other nonlinear algorithms capable of operating with kernels, such as Support Vector Machines, Gaussian Processes, Kernel Principal Component Analysis (KPCA) and many others [2, 3, 28, 49, 50].

In this work, we propose the Kernel Conjugate Gradient (KCG) algorithm [51–54], in which the output filter consists of the linear combination of a function of the input vectors, applying a technique called kernel trick. This approach, in simple words, consists of employing an artifice to avoid the need for explicit mappings

that, when mapping a linear algorithm is transformed to work in a non-linear space by kernels (kernelization process). Our interest in high-dimensional (or infinite-dimensional) space is connected to Hilbert Spaces [55, 56], denoted by $\mathcal{H}$, the equivalent of Euclidean Space (finite-dimensional vector space) with its operations.

One advantage of using the KCG algorithm is that it converges faster than the kernel least mean squares (KLMS) algorithm [57, 58]. When compared to the kernel recursive least squares (KRLS) algorithm [59–61], in most applications both are equally fast. However, the computational cost of the KCG algorithm is lower than for the KRLS algorithm.

In this section, we study the KCG algorithm in online mode without the inclusion of the exponential decaying method. When working with online applications, we are concerned about the increase in the number of coefficients as the amount of samples grows. In order to mitigate this effect, the data selection method is employed in this chapter along with the possible application of some online sparsification method [59, 60, 62].

We introduce the Data Selection Kernel Conjugate gradient (DS-KCG) algorithm, which resembles the DS-CG algorithm by considering only relevant and non-redundant data to update the filter coefficients, thus reducing the computational cost. Before presenting the DS-KCG, we introduce the theory related to the kernel functions and Hilbert Spaces. Then, to verify the performance of the proposed DS-KCG algorithm, we perform simulations in various applications.

3.1.1 Concepts of the Kernel Method

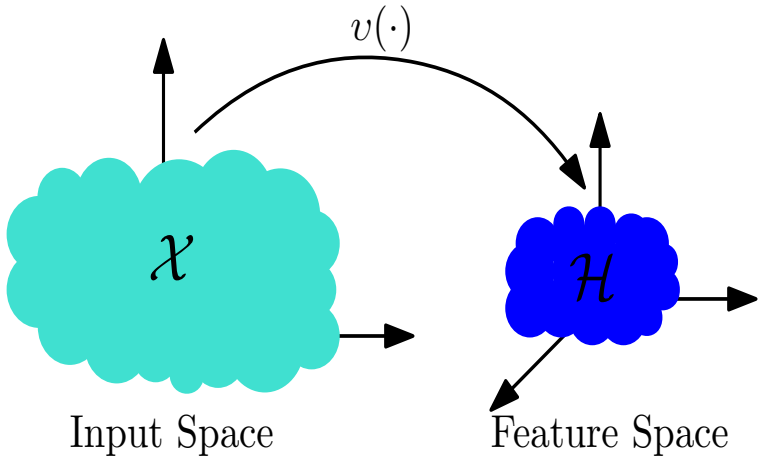
There are cases where the canonical inner product of the Euclidean Spaces is not suitable to measure the resemblance between two vectors. One possible solution is to apply nonlinear transformations on the input vector. The strategy consists of mapping the entries into a generalization of a Euclidean Space which might be infinite-dimensional, known as the Hilbert Space. In the function below, the input vector is mapped into the Hilbert Space $\mathcal{H}$, called feature space

$$\boldsymbol{\upsilon} : \mathcal{X} \to \mathcal{H} \tag{3.1}$$

where $\mathcal{X}$ is a subset of $\mathbb{R}^N$ and $\boldsymbol{\upsilon}$ is known as feature function (Figure 3.1).

In the kernel method, we make use of a function, $\kappa(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, where two vectors are mapped to a real value. This operation is known as the kernel function whose main task is to provide a measure of the similarity between two vectors mapped into $\mathcal{H}$, i.e.,

$$\kappa(\mathbf{x}_1, \mathbf{x}_2) = \langle \boldsymbol{\upsilon}(\mathbf{x}_1), \boldsymbol{\upsilon}(\mathbf{x}_2) \rangle_{\mathcal{H}} \tag{3.2}$$



(a)

Figure 3.1: Kernel mapping representation

where $\langle\cdot,\cdot\rangle_{\mathcal{H}}$ is the inner product in the Hilbert Space $\mathcal{H}$. A useful kernel function should present analogous properties of the inner product in Euclidean Spaces:

- Symmetry: $\kappa(\mathbf{x}_1, \mathbf{x}_2) = \kappa(\mathbf{x}_2, \mathbf{x}_1)$, for $\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$;
- Positive-definiteness: $\kappa(\mathbf{x}_1, \mathbf{x}_1) \geq 0$, for $\forall \mathbf{x}_1 \in \mathcal{X}$;
- Cauchy-Schwarz inequality: $|\kappa(\mathbf{x}_1,\mathbf{x}_2)| \leq \sqrt{\kappa(\mathbf{x}_1, \mathbf{x}_1)\kappa(\mathbf{x}_2, \mathbf{x}_2)}$, for $\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$.

Additionally, the kernel function is called reproducing kernel if it satisfies:

1. $\forall \mathbf{x} \in \mathcal{X}$, $\kappa(\mathbf{x},\cdot) \in \mathcal{H}$, meaning that all real valued functions of $\mathbf{x}$ are generated by this kernel;
2. The reproducing property, defined as

   Definition: For each function $\boldsymbol{f}(\cdot) \in \mathcal{H}$ and each $\mathbf{x} \in \mathcal{X}$, we have

$$f(\mathbf{x}) = \langle \boldsymbol{f}(\cdot), \boldsymbol{\kappa}(\cdot, \mathbf{x}) \rangle_{\mathcal{H}}. \tag{3.3}$$

   Particularly, if $\boldsymbol{f}(\cdot) = \boldsymbol{\kappa}(\cdot, \mathbf{x})$, then

$$\kappa(\mathbf{x}, \mathbf{x}') = \langle \boldsymbol{\kappa}(\cdot, \mathbf{x}'), \boldsymbol{\kappa}(\cdot, \mathbf{x}) \rangle_{\mathcal{H}}. \tag{3.4}$$

A Hilbert Space $\mathcal{H}$ is considered a reproducing kernel Hilbert Space (RKHS) if it fulfills these two properties. We also may conclude that $\boldsymbol{\upsilon}(\mathbf{x}) = \boldsymbol{\kappa}(\cdot, \mathbf{x})$.

Similarly to the correlation matrix in the classic adaptive filtering, the Gram matrix is defined as

$$\mathbf{K}(k) = \begin{bmatrix} \kappa(\mathbf{x}(k-I),\mathbf{x}(k-I)) & \kappa(\mathbf{x}(k-I),\mathbf{x}(k-I+1)) & \dots & \kappa(\mathbf{x}(k-I),\mathbf{x}(k)) \\ \kappa(\mathbf{x}(k-I+1),\mathbf{x}(k-I)) & \kappa(\mathbf{x}(k-I+1),\mathbf{x}(k-I+1)) & \dots & \kappa(\mathbf{x}(k-I+1),\mathbf{x}(k)) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}(k),\mathbf{x}(k-I)) & \kappa(\mathbf{x}(k-I),\mathbf{x}(k-1)) & \dots & \kappa(\mathbf{x}(k),\mathbf{x}(k)) \end{bmatrix}, \tag{3.5}$$

wherein $I+1$ is the number of input vectors included in the data dictionary until the iteration k. The Gram Matrix plays an important role in kernel adaptive filtering, since it assembles the similarity among $I+1$ input signal vectors. A Mercer Kernel [63] is a function $\kappa(\cdot,\cdot)$ whose Gram matrix is continuous, symmetric and positive-definite. In this text, any Gram matrix should be positive-definite. One of the main theorems in this area is the Mercer Theorem [63, 64]; it establishes that any reproducing kernel $\kappa(\mathbf{x},\mathbf{x}')$ can be rewritten as

$$\kappa(\mathbf{x},\mathbf{x}') = \sum_{i=1}^{\infty} a_i q_i(\mathbf{x}) q_i(\mathbf{x}'), \tag{3.6}$$

where the a_i and $q_i(\cdot)$, for $i = 1,2,\cdots$, correspond to the eigenvalues and eigenfunctions of the Gram matrix $\mathbf{K}$, respectively. Since the eigenvalues are non-negative, there exists a map $\boldsymbol{\phi}$ defined as

$$\boldsymbol{\phi}(\mathbf{x}) = [\sqrt{a_1}q_1(\mathbf{x}),\sqrt{a_2}q_2(\mathbf{x}),\cdots], \tag{3.7}$$

where $\boldsymbol{\phi}$ can be infinite-dimensional. Therefore, it is possible to conclude that

$$\kappa(\mathbf{x},\mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{x}') = \langle \boldsymbol{\phi}(\mathbf{x}),\boldsymbol{\phi}(\mathbf{x}') \rangle_{\mathcal{H}}. \tag{3.8}$$

Furthermore, the function $\boldsymbol{\phi}$ defined above is identical to the one introduced in equation (3.1), i.e., $\boldsymbol{\phi}(\mathbf{x}) = \boldsymbol{\upsilon}(\mathbf{x})$. Equation (3.8) is the fundamental key to the success of kernel methods. We use a technique known as kernel trick, in which the input data is mapped into a high-dimensional space (Figure 3.1) through a reproducing kernel, such that the inner product in the Hilbert Space is computed using this equation. The advantage is avoiding more complex computations in the high-dimension space while implementing the algorithm.

In our optimization problem related to Kernel Adaptive filtering, given the data samples $\{(\mathbf{x}(1),d(1)),(\mathbf{x}(2),d(2)),\cdots,(\mathbf{x}(I+1),d(I+1))\}$, the main goal is to find the optimal solution for a linear functional $\boldsymbol{\varphi}$ by seeking to minimize the mean

squared errors (MSE):

$$\min_{\varphi \in \mathcal{H}^*} \quad \sum_{i=1}^{I+1} |d(i) - \langle \boldsymbol{\upsilon}(\mathbf{x}(i)), \boldsymbol{\varphi} \rangle_{\mathcal{H}}|^2, \tag{3.9}$$

where $\mathcal{H}^*$ is the dual space of $\mathcal{H}$. It is shown by the representer theorem [65] that the optimal solution can be expressed as a linear combination of the functions mapped in the input data:

$$\boldsymbol{\varphi}_o = \sum_{i=1}^{I+1} \zeta_i \boldsymbol{\upsilon}(\mathbf{x}(i)), \tag{3.10}$$

where ζ_i are the kernel coefficients.

By replacing the optimal solution (3.10) in equation (3.9), we obtain a modified optimization problem

$$\min_{\boldsymbol{\zeta} \in \mathbb{R}^{I+1}} \quad \|\mathbf{d} - \mathbf{K}(I+1)\boldsymbol{\zeta}\|_2^2 \tag{3.11}$$

where $\mathbf{d} = [d(1), \cdots, d(M)]^T$ is the desired vector, $\boldsymbol{\zeta} = [\zeta_1, \zeta_2, \cdots, \zeta_{I+1}]$ are the coefficients to be minimized and $\mathbf{K}(I+1)$ is the Gram matrix defined in equation (3.5) whose (k,l)-th element is $\kappa(\mathbf{x}(I+1-k), \mathbf{x}(I+1-l))$.

Some examples of kernel functions used in applications are shown below:

- Cosine similarity kernel:

$$\kappa(\mathbf{x},\mathbf{x}') = \frac{\mathbf{x}^T\mathbf{x}'}{\|\mathbf{x}\|_2 \|\mathbf{x}'\|_2}, \tag{3.12}$$

  where this kernel function measures the angle between the vectors $\mathbf{x}$ and $\mathbf{x}'$.

- Sigmoid kernel:

$$\kappa(\mathbf{x},\mathbf{x}') = \tanh(a\mathbf{x}^T\mathbf{x}' + b), \tag{3.13}$$

  with a and b real numbers. This sigmoid kernel function does not have a positive-definite Gram matrix but is used in some situations such as Neural Networks.

- Polynomial kernel:

$$\kappa(\mathbf{x},\mathbf{x}') = (a\mathbf{x}^T\mathbf{x}' + b)^n, \tag{3.14}$$

  where $a \in \mathbb{R}$, $b \geq 0$, and $n \in \mathbb{N}$. If $b \neq 0$, this kernel is called inhomogeneous polynomial kernel. The value b is non-negative to guarantee that the Gram matrix is positive-definite.

- Gaussian kernel:

$$\kappa(\mathbf{x},\mathbf{x}') = e^{\frac{1}{2}(\mathbf{x}-\mathbf{x}')^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\mathbf{x}')} = e^{\frac{1}{2}\sum_{i=0}^{I}\frac{1}{\sigma_i^2}(x_i-x_i')^2}, \tag{3.15}$$

  where $\boldsymbol{\Sigma}$ is a diagonal matrix whose diagonal values are equal to σ_i^2, for $i = 0, 1, \cdots, I$. In some situations, $\sigma_i^2 = \sigma^2$ is used. The feature space for the Gaussian case is infinite dimensional.

3.1.2 Online Kernel Conjugate Gradient

In this subsection, to simplify mapping operations and computation of the inner product in the feature space, we introduce the Kernel Conjugate Gradient (KCG) solution based on a least squares solution, called Conjugate Gradient Least Squares (CGLS) [66]. In this case, the optimization problem consists of minimizing of the following least squares problem

$$\min_{\mathbf{w}} \quad \left\|\mathbf{d} - \mathbf{X}^T\mathbf{w}\right\|_2^2 \tag{3.16}$$

where $\mathbf{X} = [\mathbf{x}(1), \mathbf{x}(2), \cdots, \mathbf{x}(M)]$ is the input data matrix and $\mathbf{w}$ is the vector containing the adaptive filter coefficients. These algorithms work offline, that is, all information $\{\mathbf{X}, \mathbf{d}\}$ is available from the start. However, before applying the data selection method, we will formulate the algorithm so that the available information is updated at each iteration, deriving an online algorithm.

This version of the CG algorithm is derived through an algebraic rearrangement in basic conjugate gradient [29, 67, 68]. Hence, we start by formulating the basic CG algorithm and then the CGLS algorithm to finally arrive at the KCG algorithm. In the CG algorithm presented in the previous chapter, the modifications performed to obtain the basic CG algorithm are in the equations defining $\alpha(k)$, $\beta(k)$ and $\mathbf{g}(k)$. The first one is obtained in equation (3.17) using equation (2.7),

$$\begin{aligned}\alpha(k) &= \frac{\mathbf{c}^T(k)\mathbf{g}(k-1)}{\mathbf{c}^T(k)\mathbf{R}\mathbf{c}(k)} = \frac{(\mathbf{g}^T(k-1) + \beta(k-1)\mathbf{c}^T(k-1))\mathbf{g}(k-1)}{\mathbf{c}^T(k)\mathbf{R}\mathbf{c}(k)} \\ &= \frac{\mathbf{g}^T(k-1)\mathbf{g}(k-1)}{\mathbf{c}^T(k)\mathbf{R}\mathbf{c}(k)}\end{aligned} \tag{3.17}$$

where we define $\eta = 1$ and $\mathbf{c}^T(k-1)\mathbf{g}(k-1) = 0$, since the negative gradient $\mathbf{g}(k-1)$ is orthogonal to the directions $\mathbf{c}(i)$ for $1 \leq i \leq k-1$. In the second step factor $\beta(k)$, equation (3.18) has the following form,

$$\beta(k) = \frac{\mathbf{g}(k)^T\mathbf{g}(k)}{\mathbf{g}^T(k-1)\mathbf{g}(k-1)}. \tag{3.18}$$

The gradient equation adopted in this algorithm was introduced in the expression (2.6). Utilizing the derived equations (3.17), (2.4), (2.6), (3.18) and (2.7), the basic CG algorithm is illustrated in Table 3.1.

Table 3.1: Basic Conjugate Gradient algorithm

Basic CG algorithm
Initialize
$\mathbf{X} = [\mathbf{x}(1), \mathbf{x}(2), \cdots, \mathbf{x}(M)]$, $\mathbf{d} = [d(1), \cdots, d(M)]$, $\mathbf{R} = \mathbf{X}\mathbf{X}^T$, $\mathbf{p} = \mathbf{X}\mathbf{d}$
$\mathbf{w}(0) =$ random vectors or zero vectors
$\mathbf{g}(0) = \mathbf{c}(1) =$ zeros vectors or $(\mathbf{p} - \mathbf{R}\mathbf{w}(0))$, $\gamma =$ small constant for regularization
Do for $k > 0$
$\alpha(k) = \frac{\mathbf{g}^T(k-1)\mathbf{g}(k-1)}{[\mathbf{c}^T(k)\mathbf{R}\mathbf{c}(k)+\gamma]}$
$\mathbf{w}(k) = \mathbf{w}(k-1) + \alpha(k)\mathbf{c}(k)$
$\mathbf{g}(k) = \mathbf{g}(k-1) - \alpha(k)\mathbf{R}\mathbf{c}(k)$
$\beta(k) = \frac{\mathbf{g}^T(k)\mathbf{g}(k)}{[\mathbf{g}^T(k-1)\mathbf{g}(k-1)+\gamma]}$
$\mathbf{c}(k+1) = \mathbf{g}(k) + \beta(k)\mathbf{c}(k)$
end

As mentioned earlier, the CGLS algorithm is formulated from a small algebraic rearrangement in the basic CG algorithm. These modifications are performed in equations (3.19) and (3.20),

$$\alpha(k) = \frac{\mathbf{g}^T(k-1)\mathbf{g}(k-1)}{\mathbf{c}^T(k)\mathbf{R}\mathbf{c}(k)} = \frac{\mathbf{g}^T(k-1)\mathbf{g}(k-1)}{(\mathbf{c}^T(k)\mathbf{X})(\mathbf{X}^T\mathbf{c}(k))} = \frac{\mathbf{g}^T(k-1)\mathbf{g}(k-1)}{\mathbf{v}^T(k)\mathbf{v}(k)} \tag{3.19}$$

$$\mathbf{g}(k) = \mathbf{g}(k-1) - \alpha(k)\mathbf{R}\mathbf{c}(k) \implies \mathbf{z}(k) = \mathbf{z}(k-1) - \alpha(k)\mathbf{v}(k) \tag{3.20}$$

where $\mathbf{R} = \mathbf{X}\mathbf{X}^T$, $\mathbf{v}(k) = \mathbf{X}^T\mathbf{c}(k)$ and $\mathbf{g}(k) = \mathbf{X}\mathbf{z}(k)$. The CGLS algorithm is outlined below in Table 3.2.

These offline algorithms converge to the optimal solution in at most N iterations, but in some applications, it is desirable to finish the procedure before the N-th iteration. One of the stopping criteria for the basic CG and CGLS algorithms is the condition $\|\alpha(k)\mathbf{c}(k)\|_2 \leq \varepsilon$, where ε is a norm tolerance such that when this condition is satisfied, the algorithm stops [29].

Since both algorithms solve equivalent optimization problems, the main properties related to the CGLS algorithm are the same as the basic CG algorithm, as listed below [29]:

- Conjugate property: $\mathbf{v}^T(i)\mathbf{v}(j) = \mathbf{c}^T(i)\mathbf{R}\mathbf{c}(j) = 0$ for all $i \neq j$ and $\mathbf{R} = \mathbf{X}\mathbf{X}^T$;

Table 3.2: Conjugate Gradient Least Squares algorithm

CGLS algorithm

Initialize

$\mathbf{X} = [\mathbf{x}(1), \mathbf{x}(2), \cdots, \mathbf{x}(M)]$, $\mathbf{d} = [d(1), \cdots, d(M)]$, $\mathbf{w}(0) =$ random vectors or zero vectors

$\mathbf{z}(0) = (\mathbf{d}^T - \mathbf{X}^T\mathbf{w}(0))$, $\mathbf{g}(0) = \mathbf{X}\mathbf{z}(0)$, $\mathbf{c}(1) = \mathbf{g}(0)$

Do for $k > 0$

$\mathbf{v}(k) = \mathbf{X}^T\mathbf{c}(k)$

$\alpha(k) = \frac{\mathbf{g}^T(k-1)\mathbf{g}(k-1)}{\mathbf{v}^T(k)\mathbf{v}(k)}$

$\mathbf{w}(k) = \mathbf{w}(k-1) + \alpha(k)\mathbf{c}(k)$

$\mathbf{z}(k) = \mathbf{z}(k-1) - \alpha(k)\mathbf{v}(k)$

$\mathbf{g}(k) = \mathbf{X}\mathbf{z}(k)$

$\beta(k) = \frac{\mathbf{g}^T(k)\mathbf{g}(k)}{\mathbf{g}^T(k-1)\mathbf{g}(k-1)}$

$\mathbf{c}(k+1) = \mathbf{g}(k) + \beta(k)\mathbf{c}(k)$

end

- Orthogonality of the gradient vector $\mathbf{g}(k)$ to the previous gradient vectors: $\mathbf{g}^T(k)\mathbf{g}(i) = 0$ for $0 \leq i \leq k-1$;
- The gradient vector $\mathbf{g}(k)$ is orthogonal to the directions vectors $\mathbf{c}(i)$ previously obtained: $\mathbf{g}^T(k)\mathbf{c}(i)$ for $0 \leq i \leq k$;
- The spaces spanned by $\{\mathbf{g}(0), \mathbf{g}(1), \cdots, \mathbf{g}(k-1)\}$ and $\{\mathbf{c}(1), \mathbf{c}(2), \cdots, \mathbf{c}(k)\}$ are the same linear space.

As shown above, the CGLS and the basic CG algorithms have the same properties, and henceforth the CGLS will be referred to as CG.

The next step is to derive the offline KCG algorithm. The coefficient vector $\mathbf{w}(k)$ is transformed into the form $\sum_{i=1}^{M} \zeta_i \mathbf{x}(i)$, before we apply the mapping to the feature space to achieve the kernel approach in the CG algorithm and support the use of the kernel trick. Using the equations (2.4) and (2.7), we can rewrite the coefficient vector as a linear combination of the input vectors $\mathbf{X}$ in the following form:

$$\mathbf{w}(1) = \alpha(1)\mathbf{c}(1) = \alpha(1)\mathbf{g}(0), \tag{3.21}$$

$$\begin{aligned}\mathbf{w}(2) &= \mathbf{w}(1) + \alpha(2)\mathbf{c}(2) = \alpha(1)\mathbf{g}(0) + \alpha(2)(\mathbf{g}(1) + \beta(1)\mathbf{c}(1)) \\ &= (\alpha(1) + \alpha(2)\beta(1))\mathbf{g}(0) + \alpha(2)\mathbf{g}(1),\end{aligned} \tag{3.22}$$

$$\begin{aligned}\mathbf{w}(3) &= \mathbf{w}(2) + \alpha(3)\mathbf{c}(3) = \mathbf{w}(2) + \alpha(3)[\mathbf{g}(2) + \beta(2)(\mathbf{g}(1) + \beta(1)\mathbf{g}(0))] \\ &= (\alpha(1) + \alpha(2)\beta(1) + \alpha(3)\beta(1)\beta(2))\mathbf{g}(0) \\ &+ (\alpha(2) + \alpha(3)\beta(2))\mathbf{g}(1) + \alpha(3)\mathbf{g}(2).\end{aligned} \tag{3.23}$$

To simplify the notation, we define the auxiliary variable

$$b_i^j = \prod_{l=i}^{j} \beta(l) = \beta(i)\beta(i+1)\cdots\beta(j) \tag{3.24}$$

with $b_i^{i-1} = 1$. By induction, the general form of the coefficient vector is

$$\begin{aligned}
\mathbf{w}(k) &= (\alpha(1)b_1^0 + \alpha(2)b_1^1 + \cdots + \alpha(k)b_1^{k-1})\mathbf{g}(0) \\
&\quad + (\alpha(2)b_2^1 + \alpha(2)b_2^2 + \cdots + \alpha(k)b_2^{k-1})\mathbf{g}(1) \\
&\quad + \cdots + \alpha(k)b_k^{k-1}\mathbf{g}(k-1) \\
&= \sum_{i=1}^{k}\left(\sum_{j=i}^{k}\alpha(j)b_i^{j-1}\right)\mathbf{g}(i-1) \\
&= \mathbf{X}\left[\sum_{i=1}^{k}\left(\sum_{j=i}^{k}\alpha(j)b_i^{j-1}\right)\mathbf{z}(i-1)\right] \\
&= \mathbf{X}\left(\sum_{i=1}^{k} s_i(k)\mathbf{z}(i-1)\right) = \mathbf{X}(\mathbf{z}_{\mathrm{r}}(k)\mathbf{s}(k)) = \mathbf{X}\boldsymbol{\zeta}(k)
\end{aligned} \tag{3.25}$$

where $\mathbf{Z}(k) = [\mathbf{z}(0), \mathbf{z}(1), \cdots, \mathbf{z}(k-1)]$ and $\mathbf{s}(k) = [\mathbf{s}_1(k), \mathbf{s}_2(k), \cdots, \mathbf{s}_k(k)]$. In addition, the expression $s_i(k) = \sum_{j=i}^{k-1}\alpha(j)b_i^{j-1} + \alpha(k)b_i^k = s_i(k-1) + \alpha(k)b_i^k$ can follow from the derivation of equation (3.25).

In the next step, we define $\boldsymbol{\Upsilon}$ as the mapping of input matrix $\mathbf{X}$ into the feature space $\mathcal{H}$ through the function defined by equation (3.1),

$$\boldsymbol{\Upsilon} = \boldsymbol{\upsilon}(\mathbf{X}) = [\upsilon(\mathbf{x}(1)), \upsilon(\mathbf{x}(2)), \cdots, \upsilon(\mathbf{x}(M))]. \tag{3.26}$$

Therefore, the coefficient vector based on the kernel approach in the feature space is,

$$\mathbf{w}(k) = \boldsymbol{\upsilon}(\mathbf{X})\boldsymbol{\zeta}(k) \tag{3.27}$$

In addition to the equation (3.25), other expressions in the Table 3.2 require to be modified, since the dimension of the feature space is high, and possibly infinite as in the case of the Gaussian kernel. Therefore, it is not feasible to work with some expressions used in the input space. One of the equations is $\mathbf{v}(k)$, the strategy performed to deal with this obstacle consists of some algebraic manipulations and

thus derive a recursive relation in terms of $\mathbf{v}(k-1)$ and $\mathbf{z}(k-1)$,

$$\begin{aligned}\mathbf{v}(k) &= \boldsymbol{\upsilon}(\mathbf{X})^T\mathbf{c}(k) = \boldsymbol{\upsilon}(\mathbf{X})^T(\mathbf{g}(k-1) + \beta(k-1)\mathbf{c}(k-1)) \\ &= \boldsymbol{\upsilon}(\mathbf{X})^T\boldsymbol{\upsilon}(\mathbf{X})\mathbf{z}(k-1) + \beta(k-1)\boldsymbol{\upsilon}(\mathbf{X})^T\mathbf{c}(k-1) \\ &= \mathbf{K}\mathbf{z}(k-1) + \beta(k-1)\mathbf{v}(k-1) \end{aligned} \tag{3.28}$$

where $\mathbf{K} = \boldsymbol{\upsilon}(\mathbf{X})^T\boldsymbol{\upsilon}(\mathbf{X})$ is the Gram matrix defined in equation (3.5) when the algorithm has the complete data dictionary from the beginning, i.e., if the iteration is $k = i$ then $I = i$. The next expression to be modified is the inner product of $\mathbf{g}(k)$ and $\mathbf{g}(k)$,

$$\pi(k) = \mathbf{g}^T(k)\mathbf{g}(k) = (\boldsymbol{\upsilon}(\mathbf{X})\mathbf{z}(k))^T(\boldsymbol{\upsilon}(\mathbf{X})\mathbf{z}(k)) = \mathbf{z}^T(k)\mathbf{K}\mathbf{z}(k). \tag{3.29}$$

Due to the modifications performed in equations (3.25) and (3.28), the equation (2.7) is no longer required in the algorithm. The remaining equations (3.18), (3.19) and (3.20), arise slightly different variables,

$$\beta(k) = \frac{\mathbf{g}^T(k)\mathbf{g}(k)}{\mathbf{g}^T(k-1)\mathbf{g}(k-1)} = \frac{\pi(k)}{\pi(k-1)} \tag{3.30}$$

$$\alpha(k) = \frac{\mathbf{g}^T(k-1)\mathbf{g}(k-1)}{\mathbf{v}^T(k)\mathbf{v}(k)} = \frac{\pi(k-1)}{\mathbf{v}^T(k)\mathbf{v}(k)} \tag{3.31}$$

$$\mathbf{r}(k) = \mathbf{r}(k-1) - \alpha(k)\mathbf{v}(k) \tag{3.32}$$

Table 3.3: Offline Kernel Conjugate Gradient algorithm

Offline KCG algorithm

Initialize

Choose kernel function $(\mathbf{K})$, $\mathbf{z}(0) = \mathbf{d}^T = [d(1), \cdots, d(M)]^T$

$\mathbf{v}(1) = \mathbf{K}\mathbf{z}(0)$, $\pi(0) = \mathbf{z}^T(0)\mathbf{K}\mathbf{z}(0)$, $\beta(0) = 1$, $\boldsymbol{\zeta}(0) = 0$

Do for $k > 0$

$\alpha(k) = \frac{\pi(k-1)}{\mathbf{v}^T(k)\mathbf{v}(k)}$

$b = 1$

$s_k(k-1) = 0$

Do for $i = k, k-1, \cdots, 1$

$s_i(k) = s_i(k-1) + \alpha(k)b$

$b = b\beta(i)$

end

$\mathbf{z}(k) = \mathbf{z}(k-1) - \alpha(k)\mathbf{v}(k)$

$\pi(k) = \mathbf{z}(k)^T\mathbf{K}\mathbf{z}(k)$

$\beta(k) = \frac{\pi(k)}{\pi(k-1)}$

$\mathbf{v}(k+1) = \mathbf{K}\mathbf{z}(k) + \beta(k)\mathbf{v}(k)$

$$\zeta(k) = \begin{bmatrix} | & | & | \\ \mathbf{z}(1) & \cdots & \mathbf{z}(k) \\ | & | & | \end{bmatrix} \begin{bmatrix} s_1(k) \\ \vdots \\ s_k(k) \end{bmatrix}$$

end

The online version of the KCG algorithm can be derived from the offline KCG algorithm outlined in Table 3.3. At iteration k, we consider that the information available is

$$\mathbf{X}(k) = [\mathbf{x}(1), \mathbf{x}(2), \cdots, \mathbf{x}(k)] \quad \text{and} \quad \mathbf{d}(k) = [d(1), d(2), \cdots, d(k)]^T. \tag{3.33}$$

By mapping this matrix $\mathbf{X}(k)$ into the feature space $\mathcal{H}$ through the feature function υ,

$$\boldsymbol{\Upsilon}(k) = \boldsymbol{\upsilon}(\mathbf{X}(k)) = [\upsilon(\mathbf{x}(1)), \upsilon(\mathbf{x}(2)), \cdots, \upsilon(\mathbf{x}(k))]. \tag{3.34}$$

The matrix $\mathbf{K}(k)$ can be obtained from a recursive relation with $\mathbf{K}(k-1)$ and other factors, as follows

$$\mathbf{K}(k) = \boldsymbol{\Upsilon}^T(k)\boldsymbol{\Upsilon}(k) = \begin{bmatrix} \boldsymbol{K}(k-1) & \boldsymbol{\phi}(k) \\ \boldsymbol{\phi}^T(k) & q(k) \end{bmatrix} \tag{3.35}$$

where $q(k) = \kappa(\mathbf{x}(k), \mathbf{x}(k))$ and $\boldsymbol{\phi}(k)$ is defined as

$$\boldsymbol{\phi}(k) = [\kappa(\mathbf{x}(1), \mathbf{x}(k)), \kappa(\mathbf{x}(2), \mathbf{x}(k)), \cdots, \kappa(\mathbf{x}(k-1), \mathbf{x}(k))]^T. \tag{3.36}$$

Therefore, utilizing the kernel trick and equation (3.36), the error signal is computed in this case as

$$\begin{aligned} e(k) &= d(k) - \mathbf{w}^T(k-1)\upsilon(\mathbf{x}(k)) = d(k) - \boldsymbol{\zeta}^T(k-1)\boldsymbol{\upsilon}(\mathbf{X}(k-1))^T\upsilon(\mathbf{x}(k)) \\ &= d(k) - \boldsymbol{\zeta}^T(k-1)\boldsymbol{\phi}(k). \end{aligned} \tag{3.37}$$

The main issue related to the online KCG algorithm is how to update the coefficients $\boldsymbol{\zeta}(k)$, which is the vector of weights assigned to the kernel elements in equation (3.25). It should be noticed that whenever $\mathbf{x}(k)$ is added to the data dictionary, the vector $\boldsymbol{\zeta}(k)$ increases in size and also it updates each coefficient, resulting in a different approach when compared to the offline KCG algorithm. The proposal chosen to solve this problem begins by selecting the vector $[\boldsymbol{\zeta}^T(k-1), 0]^T$ as the initial value at the iteration k, and then k_{up} updates are performed with a fixed matrix Gram $\mathbf{K}(k)$ in order to achieve a satisfactory convergence in the coefficients of $\boldsymbol{\zeta}(k)$. At each iteration k, this initial vector is inserted into the residual $\mathbf{z}(0)$,

$$\mathbf{z}(0) = \mathbf{d}(k) - \mathbf{K}(k) \begin{bmatrix} \boldsymbol{\zeta}(k-1) \\ 0 \end{bmatrix}. \tag{3.38}$$

After performing k_{up} updates according to the KCG in iteration k, we obtain matrix $\mathbf{Z}(k) = [\mathbf{z}(0), \cdots, \mathbf{z}(k_{\text{up}})]$ and vector $\mathbf{s}(k)$ defined in equation (3.25), and hence we achieve the final value of the coefficients as

$$\boldsymbol{\zeta}(k) = \begin{bmatrix} \boldsymbol{\zeta}(k-1) \\ 0 \end{bmatrix} + \begin{bmatrix} | & | & | \\ \mathbf{z}(1) & \cdots & \mathbf{z}(k_{\text{up}}) \\ | & | & | \end{bmatrix} \begin{bmatrix} s_1(k) \\ \vdots \\ s_{k_{\text{up}}}(k) \end{bmatrix} = \begin{bmatrix} \boldsymbol{\zeta}(k-1) \\ 0 \end{bmatrix} + (\mathbf{Z}(k)\mathbf{s}(k)). \tag{3.39}$$

The second term in the above equation is calculated from k_{up} updates at the iteration k and then added to the initial value $[\boldsymbol{\zeta}^T(k-1), 0]^T$. The complete procedure for Online KCG is shown in Table 3.4.

Once we introduce the online KCG algorithm, we will incorporate the data selection method into the process. The data selection in the KCG is based on the same methodology explained in the subsection 2.2.2, such that we obtain a

Table 3.4: Online Kernel Conjugate Gradient algorithm

Online KCG algorithm

Initialize

$\mathbf{X}(1) = \mathbf{x}(1)$, $q(1) = \kappa(\mathbf{x}(1),\mathbf{x}(1))$, $\mathbf{K}(1) = [q(1)]$, $\zeta(0) = d(1)/q(1)$, $e(1) = 0$

Do for $k > 1$

$q(k) = \kappa(\mathbf{x}(k),\mathbf{x}(k))$

$\boldsymbol{\phi}(k) = [\kappa(\mathbf{x}(1), \mathbf{x}(k)), \kappa(\mathbf{x}(2), \mathbf{x}(k)), \cdots, \kappa(\mathbf{x}(k-1), \mathbf{x}(k))]^T$

$e(k) = d(k) - \boldsymbol{\zeta}^T(k-1)\boldsymbol{\phi}(k)$

$$\mathbf{K}(k) = \begin{bmatrix} \boldsymbol{K}(k-1) & \boldsymbol{\phi}(k) \\ \boldsymbol{\phi}^T(k) & q(k) \end{bmatrix}$$

$$\mathbf{z}(0) = \mathbf{d}(k) - \mathbf{K}(k) \begin{bmatrix} \boldsymbol{\zeta}(k-1) \\ 0 \end{bmatrix}$$

$\mathbf{v}(1) = \mathbf{K}(k)\mathbf{z}(0)$

$\pi(0) = \mathbf{z}^T(0)\mathbf{K}(k)\mathbf{z}(0)$

$\beta(0) = 1$

Do for $i = 1, \cdots, k_{\text{up}}$

$\alpha(i) = \frac{\pi(i-1)}{\mathbf{v}^T(i)\mathbf{v}(i)}$

$b = 1$, $s_i(k-1) = 0$

Do for $j = i, i-1, \cdots, 1$

$s_j(k) = s_j(k-1) + \alpha(i)b$

$b = b\beta(j)$

end

$\mathbf{z}(i) = \mathbf{z}(i-1) - \alpha(i)\mathbf{v}(i)$

$\pi(i) = \mathbf{z}^T(i)\mathbf{K}(i)\mathbf{z}(i)$

$\beta(i) = \frac{\pi(i)}{\pi(i-1)}$

$\mathbf{v}(i+1) = \mathbf{K}(i)\mathbf{z}(i) + \beta(i)\mathbf{v}(i)$

end

$$\boldsymbol{\zeta}(k) = \begin{bmatrix} \boldsymbol{\zeta}(k-1) \\ 0 \end{bmatrix} + \begin{bmatrix} | & | & | \\ \mathbf{z}(1) & \cdots & \mathbf{z}(k_{\text{up}}) \\ | & | & | \end{bmatrix} \begin{bmatrix} s_1(k) \\ \vdots \\ s_{k_{\text{up}}}(k) \end{bmatrix} = \begin{bmatrix} \boldsymbol{\zeta}(k-1) \\ 0 \end{bmatrix} + (\mathbf{Z}(k)\mathbf{s}(k))$$

end

threshold level from the error signal to determine if current data is relevant to modify the coefficients of the kernel problem. As the error distribution also is assumed as

Gaussian in this problem, we can conclude that the threshold is the same as in equation (2.27), $\sqrt{\tau} = Q^{-1}(\frac{P_{\text{up}}}{2})$.

For the estimation error variance σ_e^2, to simplify the computations, the chosen method is based on the equation (2.33), also repeated here for convenience,

$$\sigma_e^2 = (1 - \lambda_e)e^2(k) + (\lambda_e)\sigma_e^2, \tag{3.40}$$

where λ_e is the forgetting factor. The threshold $\sqrt{\tau_{\max}}$ to detect the outliers in dataset is defined as in equation (2.32), being repeated here for convenience as

$$\sqrt{\tau_{\max}} = \mathbb{E}[|e(k)|/\sigma_e] + 3\text{Var}[|e(k)|/\sigma_e]. \tag{3.41}$$

The data dictionary established from the data selection method is defined as

$$\mathbf{X}_I(k) = [\mathbf{x}_1(k), \mathbf{x}_2(k), \cdots, \mathbf{x}_I(k)] \tag{3.42}$$

where $\mathbf{x}_i(k)$ is the i-th input included in the data dictionary until the iteration k and I is the number of input vectors added in this set. The Data selection Kernel Conjugate Gradient (DS-KCG) algorithm is illustrated in Table 3.5

Table 3.5: Data Selection Kernel Conjugate Gradient algorithm

DS-KCG algorithm

Initialize

$\mathbf{X}_1(1) = \mathbf{x}(1)$, $q(1) = \kappa(\mathbf{x}(1),\mathbf{x}(1))$, $\mathbf{K}(1) = q(1)$, $\zeta(0) = d(1)/q(1)$, $e(1) = 0$

Prescribe P_{up}, and choose $\tau_{\max}$

$\sqrt{\tau} = Q^{-1}(\frac{P_{\text{up}}}{2})$, $I = 1$

Do for $k > 1$

$q(k) = \kappa(\mathbf{x}(k), \mathbf{x}(k))$

$\boldsymbol{\phi}(k) = [\kappa(\mathbf{x}_1(k-1), \mathbf{x}(k)), \kappa(\mathbf{x}_2(k-1), \mathbf{x}(k)), \cdots, \kappa(\mathbf{x}_I(k-1), \mathbf{x}(k))]^T$

$e(k) = d(k) - \boldsymbol{\zeta}^T(k-1)\boldsymbol{\phi}(k)$

$$\delta(k) = \begin{cases} 0, \text{ if } -\sqrt{\tau} \leq \frac{|e(k)|}{\sigma_e} \leq \sqrt{\tau} \\ 0, \text{ if } \frac{|e(k)|}{\sigma_e} \geq \sqrt{\tau_{\max}} \\ 1, \text{ otherwise} \end{cases}$$

if $\delta(k) = 0$

$\mathbf{K}(k) = \mathbf{K}(k-1)$

$\boldsymbol{\zeta}(k) = \boldsymbol{\zeta}(k-1)$

$\mathbf{X}_I(k) = \mathbf{X}_I(k-1)$

if $\frac{|e(k)|}{\sigma_e} \geq \sqrt{\tau_{\max}}$

$e(k) = 0$

$d(k) = 0$

end if

else

$I = I + 1$

$\mathbf{X}_I(k) = [\mathbf{X}_{I-1}(k-1), \mathbf{x}(k)]$

$$\mathbf{K}(k) = \begin{bmatrix} \boldsymbol{K}(k-1) & \boldsymbol{\phi}(k) \\ \boldsymbol{\phi}^T(k) & q(k) \end{bmatrix}$$

$$\mathbf{z}(0) = \mathbf{d}^T(k) - \mathbf{K}(k) \begin{bmatrix} \boldsymbol{\zeta}(k-1) \\ 0 \end{bmatrix}$$

$\mathbf{v}(1) = \mathbf{K}(k)\mathbf{z}(0)$

$\pi(0) = \mathbf{z}^T(0)\mathbf{K}(k)\mathbf{z}(0)$

$\beta(0) = 1$

Do for $i = 1, \cdots, k_{\text{up}}$

$\alpha(i) = \frac{\pi(i-1)}{\mathbf{v}^T(i)\mathbf{v}(i)}$

$b = 1$

$s_i(k-1) = 0$

Do for $j = i, i-1, \cdots, 1$

$s_j(k) = s_j(k) + \alpha(i)b$

$b = b\beta(j)$

end

$\mathbf{z}(i) = \mathbf{z}(i-1) - \alpha(i)\mathbf{v}(i)$

$\mathbf{g}(i) = \mathbf{K}(i)\mathbf{z}(i)$

$\pi(i) = \mathbf{z}^T(i)\mathbf{K}(i)\mathbf{z}(i)$

$\beta(i) = \frac{\pi(i)}{\pi(i-1)}$

$\mathbf{v}(i+1) = \mathbf{g}(i) + \beta(i)\mathbf{v}(i)$

end

$$\boldsymbol{\zeta}(k) = \begin{bmatrix} \boldsymbol{\zeta}(k-1) \\ 0 \end{bmatrix} + \begin{bmatrix} | & | & | \\ \mathbf{z}(1) & \cdots & \mathbf{z}(k_{\text{up}}) \\ | & | & | \end{bmatrix} \begin{bmatrix} s_1(k) \\ \vdots \\ s_{k_{\text{up}}}(k) \end{bmatrix} = \begin{bmatrix} \boldsymbol{\zeta}(k-1) \\ 0 \end{bmatrix} + (\mathbf{Z}(k)\mathbf{s}(k))$$

end if

end

In addition to the data selection, other methods can be used in online mode to avoid the expensive computational cost due to the increase in the amount of data. In some situations, other methods of sparsification may be applied, targeting to disregard data considered as significant. The coherence criterion, approximation linear dependency (ALD) and angle criterion are examples of these sparsification methods[1, 69–71].

3.1.3 Simulation Results

In this subsection, the results for the DS-KCG algorithm in some popular nonlinear adaptive filtering applications are presented. The prescribed probabilities of update are varied from 0% to 100% and compared to the estimated probability $\hat{P}_{\text{up}}$ to verify the performance of the algorithm when the data selection is employed. The MSE is obtained as the average of 100 independent Monte Carlo runs. The algorithms in this chapter are implemented in MATLAB and available online on GitHub [46]. The simulations are performed in a computer with Intel Core i7-7500U CPU 2.70GHz x4 processor and 15.5 GB of memory. In all problems the error variance was estimated as in equation (2.33) using $\lambda_e = 0.99$ and the number of updates in each iteration is $k_{\text{up}} = 2$.

A. System identification

In this example, we employ the online DS-KCG algorithm in the system identification problem. The input signal $x(k)$ is drawn from a uniform distribution (0,1) with variance $\sigma_x^2 = 0.1$, filtered by finite impulse response (FIR) filters specified by $\mathbf{h}_1 = [1,\ -0.5]^T$ in the simulation A1 and $\mathbf{h}_2 = [1,\ -0.5,\ 0.3, 0.7,\ -0.3]^T$ in the simulation A2. The reference signal is then built as the following nonlinear system

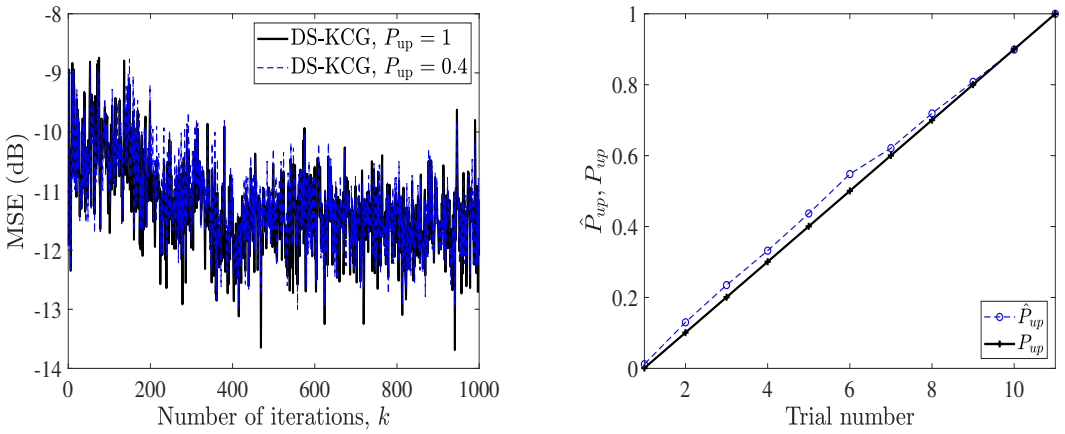
$$\begin{aligned} d(k) = &-0.76x(k) - 1.0x(k-1) + 1.0x(k-2) + 0.5x^2(k) \\ &+ 2.0x(k)x(k-2) - 1.6x^2(k-1) + 1.2x^2(k-2) \\ &+ 0.8x(k-1)x(k-2) + n(k), \end{aligned} \tag{3.43}$$

where the input noise $n(k)$ is an additional Gaussian distribution with zero mean and variance $\sigma_n^2 = 10^{-2}$.

In simulation A1, the filter order is N=10, whereas $N = 15$ in simulation A2 aiming at matching the linear filter and the nonlinear map. Using a polynomial kernel of order 3 in the simulation A1 and one of order 4 in simulation A2, we can obtain a suitable model for the problem.

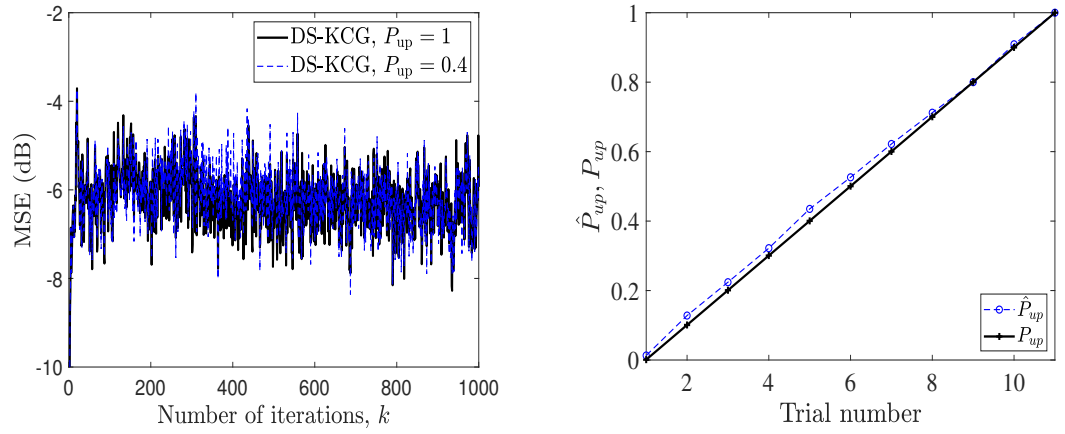
In Figures 3.2a and 3.3a are shown the learning curves for the input signal filtered by $\mathbf{h}_1$ and $\mathbf{h}_2$, respectively. The comparison between the performances for $P_{\text{up}} = 1$

and $P_{\text{up}} = 0.4$ leads us to conclude that data selection is an useful tool in nonlinear system identification. The estimated probability of update is presented in Figures 3.2b and 3.3b, where a result close to the prescribed probability of the update is observed.



(a) (b)

Figure 3.2: Simulation A1: (a) MSE learning curves for the data selection and (b) Comparison between the desired P_{up} and the achieved $\hat{P}_{\text{up}}$.



(a) (b)

Figure 3.3: Simulation A2: (a) MSE learning curves for the data selection and (b) Comparison between the desired P_{up} and the achieved $\hat{P}_{\text{up}}$.

In the following example, we use the filter impulse response $\mathbf{h}_1$ to obtain the input signal and we also consider that the outliers are present in the desired signal $d(k)$ In Table 3.6, we verify how the MSE is affected when outliers are present in the reference signal. The outlier signal is inserted in 2% of the output signal with an amplitude equal to five. Comparing the two first columns, we can conclude that when there is the presence of outliers and these are being ignored the result is

worse. There is an improvement in the MSE values for $P_{\text{up}} = 0.4$ and $P_{\text{up}} = 1$ when considering the presence of outliers.

Table 3.6: MSE (dB) for simulations with outliers

	Outlier	**yes**	**yes**	**yes**	**no**
	$\tau_{\max}$ on	yes	no	yes	no
	P_{up}	0.4	0.4	1	1
Average MSE (dB)	DS-KCG	−8.1207	−5.0197	−10.5748	−13.2229

B. Signal Prediction

In this application, the example considered is the Mackey-Glass time-delay differential equation [72],

$$\frac{dx(t)}{dt} = -\alpha_1 x(t) + \frac{\alpha_2 x(t - t_0)}{1 + x^{10}(t - t_0)}, \tag{3.44}$$

with $\alpha_1 = 0.1$, $\alpha_2 = 0.2$ and $t_0 = 17$. The differential equation solution is corrupted by Gaussian noise with zero mean and variance $\sigma_n^2 = 10^{-3}$. The filter order used is $N = 10$ and prediction parameter is $L = 3$.

The estimated probability of update $\hat{P}_{\text{up}}$ is somewhat close to the prescribed probability of update P_{up}, as can be observed in Figure 3.4*a*. In figure 3.4b the prediction results is illustrated using the Gaussian kernel with $\sigma = 0.5$ and with $P_{\text{up}} = 0.4$. One can observe that even when the number of updates is reduced, we still can obtain accurate results.

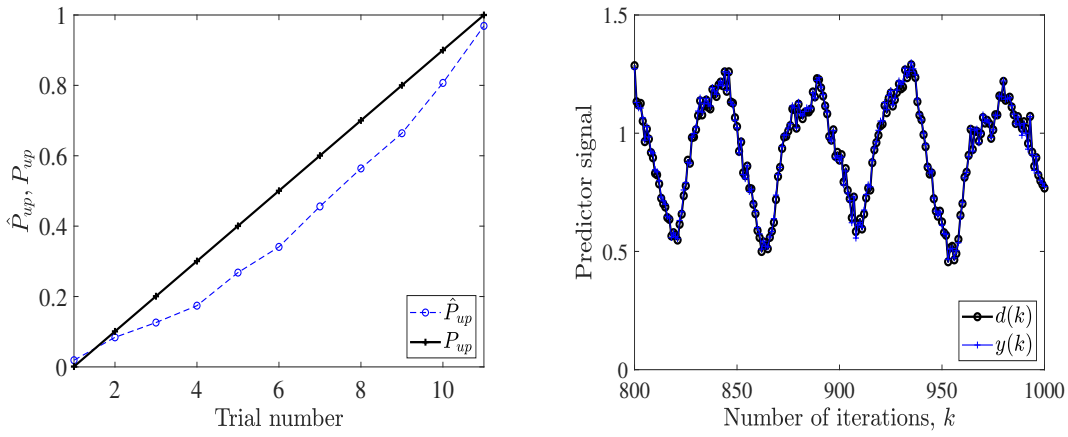
The performance of the MSE can be analyzed in figure 3.5. Just as in the previous section of the CG algorithm, a fast convergence is observed for the data selection, but there is a slight advantage for the full dataset at the steady state. This disadvantage can be explained by the trade-off between computational cost and algorithm performance when we apply the data selection method.

C. Channel Equalization

The problem chosen to represent the equalization is a digital channel modeled by the following system of equations:

$$x(k) = s(k) + 0.5s(k - 1), \tag{3.45}$$

$$y(k) = x(k) + 0.2x^2(k) + 0.1x^3(k) + n(k), \tag{3.46}$$



(a) (b)

Figure 3.4: Simulation B: (a) Comparison between the desired P_{up} and achieved $\hat{P}_{\text{up}}$ by the DS-CG algorithm and (b) Comparison between the desired signal and the predicted by the DS-CG algorithm for $P_{\text{up}} = 0.4$.

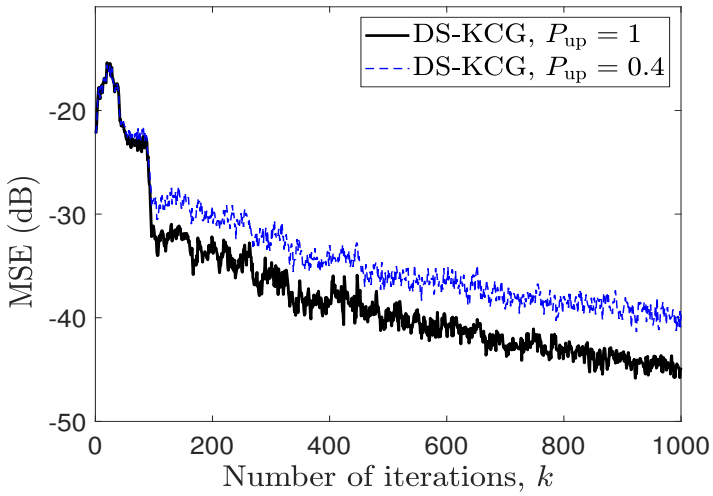
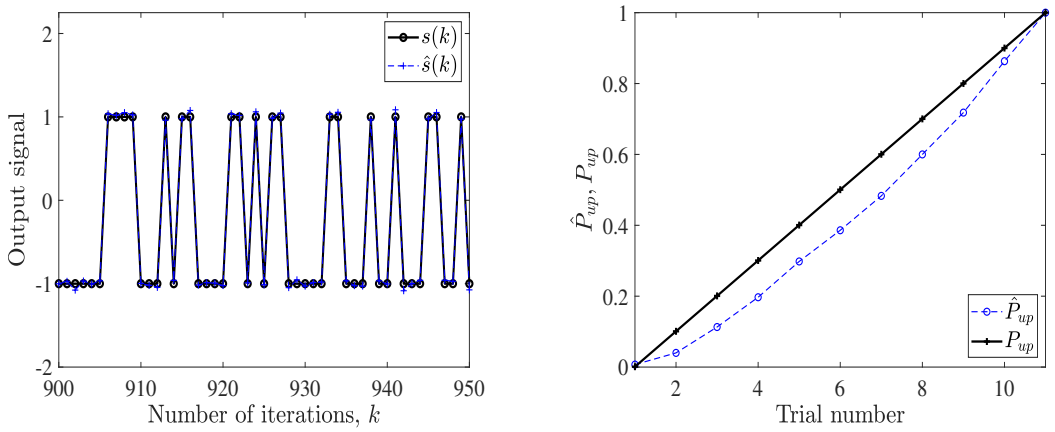


Figure 3.5: Simulation B: MSE learning curves for data selection in $P_{\text{up}} = 0.4$ and $P_{\text{up}} = 1$.

where the input signal $s(k)$ consists of independent binary random samples $\{-1,1\}$, the linearity and nonlinearity of the channel are modeled using the $x(k)$ and $y(k)$ and the channel is corrupted by a Gaussian distribution with zero mean and variance $\sigma_n^2 = 10^{-3}$. The order $N = 5$ is sufficient to achieve a good performance in the problem. In this problem, the standard deviation for the Gaussian kernel is $\sigma = 5$. The delayed step for the channel equalization considered is $L = 3$.

Figure in 3.6*a* presents the result for the $P_{\text{up}} = 0.4$ aiming to verify the performance of the adaptive algorithm by carrying out inverse filtering of the received signal to recover the input signal. As we can see, this result is satisfactory when using data selection. In Figure 3.6*b* is verified that the estimated probability

of update $\hat{P}_{\text{up}}$ is quite close to the probability of update P_{up} set beforehand.



(a) (b)

Figure 3.6: Simulation C: (a) Equalized signal of the channel and the data-selective filter (b) Comparison between the desired P_{up} and achieved $\hat{P}_{\text{up}}$ by the algorithm.

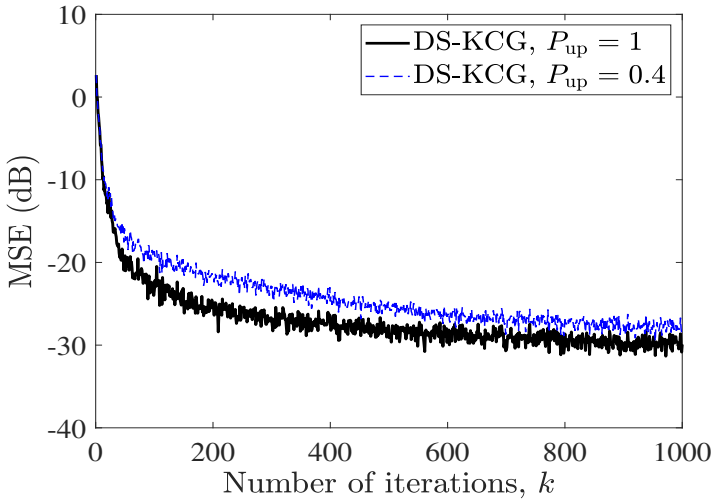


Figure 3.7: Simulation C: MSE learning curves for data selection in $P_{\text{up}} = 0.4$ and $P_{\text{up}} = 1$.

In terms of MSE, one can see in Figure 3.7 that an adaptive filtering problem using data selection method gets a result close to the simulation using 100% of the data.

3.2 Concluding Remarks

In this chapter, we proposed a data selection version of KCG algorithm. We observed a good performance when we updated the coefficients by less than 50% of the iterations, reducing the computational cost. The algorithm is also capable of identifying and discarding outliers and hence improving the filter performance. Therefore, data selection is an excellent tool in kernel adaptive filtering, mainly to reduce the computational cost.

Chapter 4

Data Selection in Neural Networks

Inspired by the good performance of the data selection method in linear and nonlinear adaptive filtering, the next goal is to adapt this method to the machine learning context. The technique chosen to explore data selection is the neural network (NN) learning due to its efficiency when dealing with large amounts of data.

In this chapter, we provide a brief introduction to the neural network for regression and classification problems by describing how the perceptron and back-propagation algorithms work. Then, the data selection method in neural network is formulated and tested with some datasets.

4.1 Introduction to Artificial Neural Networks

Neural networks arose from multidisciplinary efforts targeting to understand how the human brain works. In the human neural circuit, the main component in the structure is the neuron. A human brain has an average of 100 billion of neuron. Each neuron may be connected to up to 10.000 neurons, via synaptic connections. These links have the function of transferring information from one neuron to another [73].

In 1943, the neurophysiologist Warren McCulloch and the logician Walter Pitts created a computational model capable of behaving like a human brain. This paper is considered the beginning of the neural networks field [73].

An artificial neural network consists of several layers, each one with a specific number of neurons. The connection between its elements is made through the weights related to the synapses that provide the communication between the output of one neuron and the input of another. In this section, we begin by explaining the algorithm known as Perceptron [3, 28, 74], a simpler version for the neural network. Afterward, we describe the theoretical concepts and details of the learning algorithm to train a feed-forward neural network [2, 3, 28]. In this system, we

will propose a new data selection method in classification and regression problems. Then, we perform simulations in various applications to verify the performance of the proposed algorithm.

4.1.1 Perceptron Learning

In linearly separable two-classes (ω_1, ω_2), there exists at least one hyperplane, defined by the normal vector $\mathbf{w}_o$, such as $\mathbf{w}_o^T\mathbf{x} = 0$, which divides the dataspace in two regions useful for classifying the set of training samples correctly, $\{(\mathbf{x}(1), y(1)),(\mathbf{x}(2), y(2)), \cdots, (\mathbf{x}(M), y(M))\}$, such as

$$y(m) = \text{sign}(\mathbf{w}^T\mathbf{x}(m)) = \begin{cases} -1, & \text{if } \mathbf{x}(m) \in \omega_1, \\ +1, & \text{if } \mathbf{x}(m) \in \omega_2, \end{cases} \tag{4.1}$$

where $\text{sign}(\cdot)$ is the sign function.

The bias term of the hyperplane has been included in weight vector $\mathbf{w}$ to simplify the notation. Due to the above definition of the sign function, $y(m)(\mathbf{x}^T(m)\mathbf{w}) > 0$ if only if $\{\mathbf{x}(m), y(m)\}$ is correctly classified. One of the most popular examples of application of this algorithm is the credit scoring model [75], whose goal is to identify if a person can receive credit based on the information stored in a database.

The perceptron learning algorithm aims at achieving the previously mentioned hyperplane. The algorithm is initialized so that the value parameter $\mathbf{w}$ is a random or zero vector. The update rule per iteration is

$$\mathbf{w} = \begin{cases} \mathbf{w} + \mu y(m)\mathbf{x}(m) & \text{if } \mathbf{x}(m) \text{ is misclassified by } \mathbf{w}, \\ \mathbf{w} & \text{otherwise}, \end{cases} \tag{4.2}$$

where $\mu > 0$ is the step size parameter, which controls the convergence of the algorithm. A good way to understand this operation is by using its geometric interpretation, as depicted in Figure 4.1. Since $\mathbf{x}(m)$ is misclassified by $\mathbf{w}$, the term $\mu y(m)\mathbf{x}(m)$ is added and the hyperplane moves so that the weight vector $\mathbf{w}$ classifies $\mathbf{x}(m)$ correctly.

After a finite number of iterations, an optimal solution $\mathbf{w}_o$ is reached. This result holds regardless of which criterion is chosen to select the data at each iteration and also how the weight vector in the algorithm is initialized [76]. The perceptron algorithm is outlined in Table 4.1.

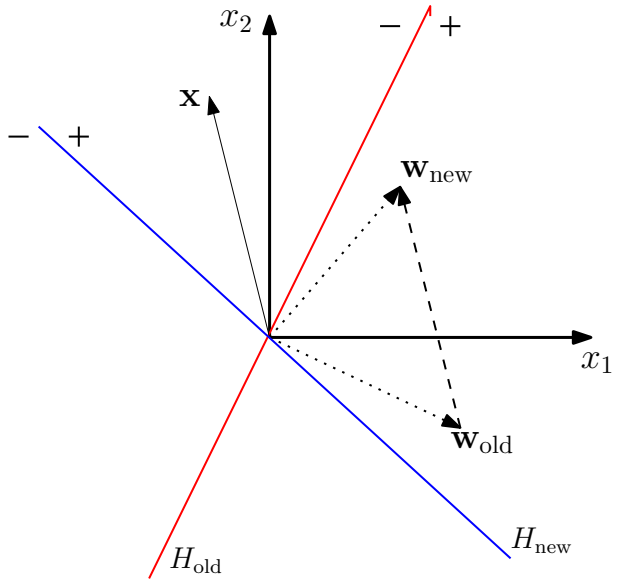


Figure 4.1: The vector $\mathbf{x}$ is misclassified by the hyperplane H_{old} (in red). The update rule adds $\mathbf{x}$ to the weight vector, $\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} + \mathbf{x}$, where in this case $\mu = 1$. The new hyperplane H_{new} (in blue) classifies correctly the vector $\mathbf{x}$.

Table 4.1: Perceptron algorithm

Perceptron algorithm
Initialize
$\mathbf{w}$ = random vectors or zero vectors,
set $\mu > 0$,
Do for $t > 0$ (epoch)
count $= 0$
Do for $m = 1 : M$ (for iteration m, select the data $\{\mathbf{x}(m), y(m)\}$)
if $y(m)(\mathbf{x}^T(m)\mathbf{w}) < 0$
$\mathbf{w} = \mathbf{w} + \mu y(m)\mathbf{x}(m)$
count = count + 1
end if
end
if count $= 0$
break; (If count $= 0$, then all data is classified correctly)
end if
end

4.1.2 Feed-Forward Multilayer Neural Network

The classification illustrated in Figure 4.2 cannot be implemented by the perceptron, even though this function is considered simple. For this problem, we can apply the algorithm known as Multi-layer Perceptron (MLP), a version of the perceptron with more layers. The layers added between input and output are called hidden layers. Each layer is composed of neurons, also known as nodes in this kind of problem. In each of the nodes belonging to the hidden and output layers, a sign function is calculated, as in the perceptron algorithm.

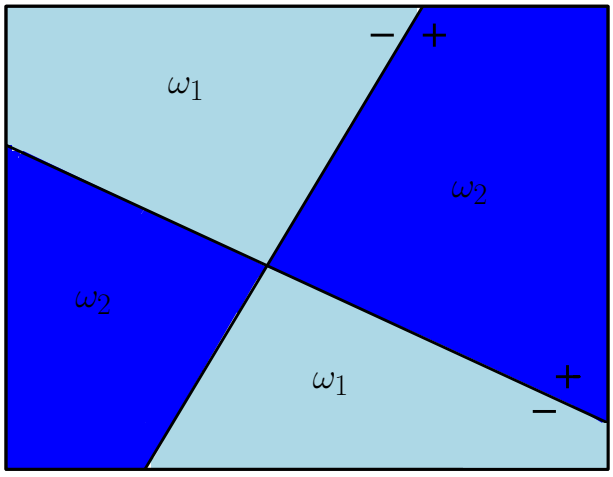


Figure 4.2: Two classes (ω_1,ω_2) are formed by the union of polyhedral regions. The configuration in this image is related to the Boolean XOR function.

In MLP, the process of learning the weights is challenging, since we are dealing with a nonlinear problem. One solution is the soft and differential approximation of the sign function that allow to use of numerical optimization methods to find the weights. These are called activation functions, which end up introducing a differentiability in the network. Some examples of activation functions that we can adopt here are the ReLU function, sigmoid logistic function, and the hyperbolic tangent function [77–79]. These models of networks are known as feed-forward because the data move forward from the initial layer to the last layer [80]. In addition, the larger is the number of layers and nodes, the greater is the complexity of the model.

Figure 4.3 illustrates the neural network scheme. The layers are denoted by $l = 0, 1, 2, \cdots, L$, where $l = 0$ is the input layer and $l = L$ is the output layer. The layers between them, $0 < l < L$, are known as hidden layers. In all layers, except in the output one, the first node is considered the bias node and its value is set to 1. In each layer, we have o^l nodes without counting the bias node, labeled as $o^1, \cdots, o^l$. The communication between the i-th node of a layer $l-1$ to the j-th node of the next layer l is made through the weight w_{ij}^l. It is important to point out that there

is no other type of communication between the layers. In each node of the hidden layers, except for the bias nodes, there is an activation function f applied to the weighted input values. In the output layer, the activation function is f_L.

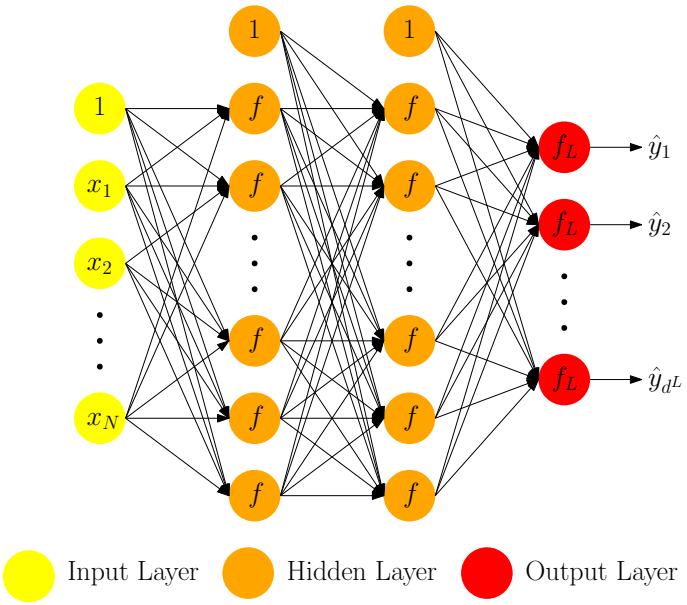


Figure 4.3: In this figure, the neural network has $L = 3$ layers (2 hidden and the output).

In regression problems, the number of nodes in the output layer is $o^L = 1$, returning a continuous numerical number $\hat{y}$ to be compared with the true value y. In classification problems, the value o^L is equal to the number of classes. In this case, the desired signal $\mathbf{y}$ is one-hot-encoded, meaning that if c is the correct class, $y_c = 1$ and $y_i = 0$ for $i \neq c$. Moreover, when the activation function at the output is softmax, the output signal $\hat{\mathbf{y}}$ returns a probability distribution on the classes, that is to say $\hat{y}_i = P(c = i)$, and this implies that $\sum_i \hat{y}_i = 1$.

In order to simplify the computations, we introduced vector and matrix notation for the network configuration. In each layer l, the input vector is denoted as $\mathbf{x}^l$ and the output vector is denoted as $\mathbf{y}^l$. In the data transfer from the layer $l-1$ to the layer l, we apply the weight matrix $\mathbf{W}^l$. This structure is shown in the Table 4.2,

Table 4.2: Configuration of an arbitrary neural network.

Input vector of layer l	$\mathbf{x}^l$	o^l-dimensional vector
Output vector of layer l	$\mathbf{y}^l$	$(o^l + 1)$-dimensional vector
Weight Matrix between layers $l-1$ and l	$\mathbf{W}^l$	$(o^l + 1) \times (o^{l+1})$-dimensional matrix

Since all the definitions and notations were introduced, we can present the feed-forward process that computes a parametric nonlinear function, $h_{\mathbf{W}}(\mathbf{x}) = \hat{\mathbf{y}}$, where

$\mathbf{W} = \{\mathbf{W}^1, \mathbf{W}^2, \cdots, \mathbf{W}^L\}$. The whole process is based on the repetition of two steps in each hidden layer, the sum of the weighted outputs of the previous layer and an activation function applied at a layer l to obtain the output vector,

$$\mathbf{x}^l = (\mathbf{W}^l)^T \mathbf{y}^{l-1}, \qquad \mathbf{y}^l = \begin{bmatrix} 1 \\ f(\mathbf{x}^l) \end{bmatrix}, \text{ for } 1 < l < L-1, \qquad \mathbf{y}^L = \begin{bmatrix} f_L(\mathbf{x}^L) \end{bmatrix} \quad (4.3)$$

where the expression $f(\mathbf{x}^l)$ is a vector whose components are $f(x_j^l)$ with $x_j^l = \sum_{i=0}^{o^{l-1}} w_{ij}^l y_i^{l-1}$, for $j = 1, \cdots, o^l$. Initializing the input layer as $\mathbf{y}^0 = [1; \mathbf{x}]$, the feed-forward process consists of the following chain

$$\mathbf{y}^0 \xrightarrow{\mathbf{W}^1} \mathbf{x}^1 \xrightarrow{f} \mathbf{y}^1 \xrightarrow{\mathbf{W}^2} \mathbf{x}^2 \xrightarrow{f} \mathbf{y}^2 \cdots \xrightarrow{\mathbf{W}^L} \mathbf{x}^L \xrightarrow{f_L} \mathbf{y}^L = \hat{\mathbf{y}}. \quad (4.4)$$

After obtaining the output value in the last layer, the next step is the back-propagation [81], in order to update the weights $\mathbf{W}$. Since we need a criterion to perform the weights update, the definition of an objective function, $J(\mathbf{W})$, is then required. Some examples of the objective function are Least-squares, Cross-Entropy, and Relative Entropy. Research works in the neural network has shown that a good combination of the activation function in the last layer and the objective function can lead to a better performance.

In the back-propagation step, the goal is to minimize the objective function with respect to the parameter $\mathbf{W}$, assuming the that set of training samples $\{(\mathbf{x}(1),\mathbf{y}(1)), (\mathbf{x}(2),\mathbf{y}(2)), \cdots, (\mathbf{x}(M),\mathbf{y}(M))\}$ is available, where M is the number of examples in this set. Here, the algorithm chosen for minimizing the objective function is the gradient descent. In this case, the weights $\mathbf{W}$ are iteratively adapted by using the negative gradient direction,

$$\mathbf{W}^l(k+1) = \mathbf{W}^l(k) - \frac{\mu}{b} \frac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}^l}\Big|_{\mathbf{W}(k)}, \quad (4.5)$$

where μ is the step size and b is the sample size of the data selected in each iteration. Here, the objective function assumed as the sum of the point-wise square error related to each training sample,

$$J(\mathbf{W}) = \frac{1}{M} \sum_{m=1}^{M} J_m(\mathbf{W}) = \frac{1}{M} \sum_{k=1}^{o^L} \sum_{m=1}^{M} J_m^k(\mathbf{W}), \quad (4.6)$$

where J_m^k is the objective function related to the output node k for the sample $\mathbf{x}(m)$. The back-propagation process deals with the difficulty of calculating all derivatives in (4.5) using a simple procedure. The method starts by calculating the derivative in the last layer obtaining the other values in other layers recursively in a backward

fashion, so that the value obtained in layer l influences in the one of the layer $l-1$, so the process is called back-propagation. In the partial derivative, the chain rule is applied so that the value obtained is partitioned into two new expressions,

$$\frac{\partial \mathbf{J}_m}{\partial \mathbf{W}^l} = \frac{\partial \mathbf{x}^l}{\partial \mathbf{W}^l}\frac{\partial \mathbf{J}_m}{\partial \mathbf{x}^l} = \mathbf{y}^{l-1}(\boldsymbol{\delta}^l)^T \tag{4.7}$$

where the first term is computed using equation (4.3) and the second term is obtained from the back-propagation process. Vector $\boldsymbol{\delta}^l$ is known as sensitivity vector for the layer l, which represents the gradient (sensitivity) of the cost J_m with respect to input $\mathbf{x}^l$.

The back-propagation has a process similar to feed-forward, but with some differences when calculating the value $\boldsymbol{\delta}^l$. The sensitivity vector $\boldsymbol{\delta}^{l+1}$ is multiplied by the weights $\mathbf{W}^{l+1}$ and the bias component is excluded. In the following, the applied transformation is the multiplication element by element of $\boldsymbol{\epsilon}^l = [\mathbf{W}^{l+1}\boldsymbol{\delta}^{l+1}]_1^{o^l}$ and the derivative of the activation function f in $\mathbf{x}^l$:

$$\boldsymbol{\delta}^l = f'(\mathbf{x}^l) \otimes \boldsymbol{\epsilon}^l. \tag{4.8}$$

The chain below shows how the procedure works:

$$[\boldsymbol{\delta}^L \xrightarrow{\times \mathbf{W}^L]\big|_1^{o^{L-1}}} \boldsymbol{\epsilon}^{L-1} \xrightarrow{\otimes f'(\mathbf{x}^{L-1})} \cdots [\boldsymbol{\delta}^2 \xrightarrow{\times \mathbf{W}^2]\big|_1^{o^1}} \boldsymbol{\epsilon}^1 \xrightarrow{\otimes f'(\mathbf{x}^1)} \boldsymbol{\delta}^1. \tag{4.9}$$

where "$\big|_1^{o^l}$" means that only the components $1, 2, \cdots, o^l$ of the vector $\mathbf{W}^{l+1}\boldsymbol{\delta}^{l+1}$ are selected.

Since the derivation of the algorithm is complete, the gradient descent back-propagation algorithm is shown in Table 4.3.

Remarks related to the algorithm [3, 28]:

- At each epoch, one can compute the objective function established in the algorithm for the training dataset J_{train} and the validation dataset J_{test}, if this set is previously defined.

- The joint choice of the output activation function and objective function is significantly important for NN learning tasks. In addition to simplifying the computations in updating weights, we can also improve algorithm performance. The most used combination in regression problems is the linear function with mean squared error; for multi-class classification problems is the softmax function with cross-entropy.

- For some time, the most widely used activation function was the sigmoid function, but it had some disadvantages, such as saturation in the tails and

centering on a nonzero value. Currently, the most widely employed functions are the tangent hyperbolic (tanh) and the "rectified linear unit" (ReLU).

- Gradient descent is the algorithm chosen in this text, but lately, more flexible optimizers were developed, such as Adagrad, Adam, RMSProp, among others [29, 82–85]. Some of these methods use adaptive step sizes that usually result in better and smoother convergence.

- The step size μ is a crucial parameter in the NN. Defining a value for the step size requires some care, since setting it too high or too low can cause the function to never converge to a (local) minimum. A widely used solution for this type of problem is to decrease gradually the rate after each iteration.

- The default procedure for initializing the weights $\mathbf{W}$ is by randomly selecting the values. If the weights are too large or too small numbers, the activation function will be saturated, resulting in low gradients and then a slower convergence. The current procedure for dealing with this drawback is to initialize the weights with uniform or normal distribution.

- A possible stopping criterion consists of training the model up to a certain epoch n_{ep}, and after that period, verify if the value of the objective function is less than a prescribed threshold. An alternative is to check if the objective function is decreasing from iteration to iteration in the validation set. Otherwise, the process may be interrupted.

- For each iteration i in the algorithm, only a sample of data of size b (mini-batch) is considered. Depending on how the algorithm draws these samples in the dataset, randomly or deterministically, the convergence can be faster.

- When selecting the number of hidden layers and neurons in a training network, the criterion used is to add layers and neurons until the validation error shows no improvement.

Table 4.3: Gradient descent back-propagation algorithm

Gradient descent back-propagation algorithm

Initialize

$\{(\mathbf{x}(1),\mathbf{y}(1)), (\mathbf{x}(2),\mathbf{y}(2)), \cdots, (\mathbf{x}(M),\mathbf{y}(M))\}$,

$\mathbf{W} = \{\mathbf{W}^1, \mathbf{W}^2, \cdots, \mathbf{W}^L\}$ (random vectors),

Select: step size $\mu > 0$, number of epoch n_{ep}, mini-batch size b, number of layers $L+1$, number of nodes $(o^1, \cdots, o^L)$, activation function f, output activation function f_L, objective function J

$iter = M/b$

Do for $t = 1 : n_{\text{ep}}$

Do for $i = 1 : iter$ (for each iteration, randomly select b examples in training dataset $\rightarrow \mathbf{X}_{(t,i)} = [\bar{\mathbf{x}}(1), \bar{\mathbf{x}}(2), \cdots, \bar{\mathbf{x}}(b)]$, $\mathbf{Y}_{(t,i)} = [\bar{\mathbf{y}}(1), \bar{\mathbf{y}}(2), \cdots, \bar{\mathbf{y}}(b)]$)

[Forward Propagation]:

$\mathbf{Y}^0 = [\bar{\mathbf{y}}^0(1), \cdots, \bar{\mathbf{y}}^0(b)] = [ones(1,b); \mathbf{X}_{(t,i)}]$ ($ones(1,b)$ are the bias term)

Do for $l = 1 : L-1$

$\mathbf{X}^l = (\mathbf{W}^l)^T \mathbf{Y}^{l-1}$

$\mathbf{Y}^l = [ones(1,b); f(\mathbf{X}^l)]$

end

$\mathbf{X}^L = (\mathbf{W}^L)^T \mathbf{Y}^{L-1}$

$\hat{\mathbf{Y}}_{(t,i)} = [\hat{\mathbf{y}}(1), \cdots, \hat{\mathbf{y}}(b)] = \mathbf{Y}^L = g(\mathbf{X}^L)$

[Back-propagation]:

$\boldsymbol{\Delta}^L = [\boldsymbol{\delta}^L(1), \cdots, \boldsymbol{\delta}^L(b)] = \frac{\partial J}{\partial \mathbf{X}^L}$

Do for $l = L-1 : -1 : 1$

$\boldsymbol{\Delta}^l = f'(\mathbf{X}^l) \otimes [\mathbf{W}^{l+1} \boldsymbol{\Delta}^{l+1}]_1^{o^l}$

end

[Updating the weights]:

Do for $l = 1 : L$

$\mathbf{W}^l = \mathbf{W}^l - \frac{\mu}{b} \mathbf{Y}^{l-1} (\boldsymbol{\Delta}^l)^T$

end

end

$J_{\text{train}}(\mathbf{W}) = \frac{1}{M} \sum_{k=1}^{o^L} \sum_{m=1}^{M} J_m^k(\mathbf{W})$ (and J_{test} if this set is previously defined)

end

4.2 Formulation of the Modified Data Selection in NN

In this section, a similar method to the data selection in adaptive filtering is modeled in NN for the regression and classification problems [86–88]. In the previous chapter, the decision criterion applied to update the coefficients is directly related to the objective function (2.22). As a rule, the closer this $e(k)$ value is to zero, the less informative or relevant will be the contribution of $(\mathbf{x}(k), \mathbf{y}(k))$ if it is included in the data dictionary. In the neural networks framework, we have an error measure e for each output neuron, which depends on the desired and estimated outputs, these values are defined according to the objective function (4.6). Therefore, the following error vector for the data $(\mathbf{x}, \mathbf{y})$ is obtained as

$$\mathbf{e}(\hat{\mathbf{y}}, \mathbf{y}) = [e(\hat{y}_1,y_1), e(\hat{y}_2,y_2), \cdots, e(\hat{y}_{o^L},y_{o^L})] \tag{4.10}$$

where $\mathbf{y} = [y_1, y_2, \cdots, y_{o^L}]$ is the desired value and $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_{o^L}]$ is the target value in the neural network. The total sum of this error in all classes of the problem is expressed as

$$E(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{k=1}^{o^L} e(\hat{y}_k,y_k). \tag{4.11}$$

The next step is to formulate how the data selection method applies to each problem, regression, and classification.

Regression

The approach in this subsection is quite similar to the one explained in the previous chapter for adaptive filtering. The objective function chosen in this case is the same, namely the mean squared error (MSE), and the output activation function is the linear function,

$$J(\mathbf{W}) = \frac{1}{M} \sum_{m=1}^{M} \left[\frac{1}{2} (\hat{y}(m) - y(m))^2 \right], \tag{4.12}$$

where $\hat{y}(m) = y_1^L(m) = x_1^L(m)$. In this approach, since the output has only one dimension, equation (4.11) can be rewritten as

$$E(\hat{y}, y) = e(\hat{y}, y) = y - \hat{y}, \tag{4.13}$$

where y is the desired value and $\hat{y}$ is the target value.

At each epoch t, the Gaussian distribution is assumed for the error signal,

$$\boldsymbol{e} \sim \mathcal{N}(0,\, (\sigma_e^t)^2) \tag{4.14}$$

where $(\sigma_e^t)^2$ is the error variance. By normalizing this error distribution, we obtain

$$\frac{\boldsymbol{e}}{\sigma_e^t} \sim \mathcal{N}(0,\, 1). \tag{4.15}$$

In this case, the error variance is calculated using all the training examples included in the mini-batch corresponding to the iteration. Therefore, proceeding with the same idea used in equation (2.23), at the epoch t, the decision criterion to update the coefficients $\mathbf{W}$ regarding the data $(\mathbf{x}, y)$ associated to iteration i is modified in the objective function as follows

$$J_i^1(\mathbf{W}) = \begin{cases} \frac{1}{2}(e(\hat{y}, y))^2, \text{ if } \sqrt{\tau} \leq \frac{|e(\hat{y},y)|}{\sigma_e^t} \\ 0, \text{ otherwise.} \end{cases} \tag{4.16}$$

The data selection method proposes to detect the non-informative values at each iteration after the feed-forward propagation process, eliminating the irrelevant data before the back-propagation process, thus reducing the computational cost in the NN algorithm. Since we are selecting an estimated portion $\hat{P}_{\text{up}}$ from the training dataset, the update equation of the weights (4.5) is modified to

$$\mathbf{W}^l(i+1) = \mathbf{W}^l(i) - \frac{\mu}{b}\mathbf{y}^{l-1}(\boldsymbol{\delta}^l)^T \rightarrow \mathbf{W}^l(i+1) = \mathbf{W}^l(i) - \frac{\mu}{b\hat{P}_{\text{up}}}\mathbf{y}_{\mathcal{R}}^{l-1}(\boldsymbol{\delta}_{\mathcal{R}}^l)^T, \tag{4.17}$$

where $\mathcal{R}$ is the set selected to update the weights at iteration i. The prescribed probability of update P_{up}, can be computed as follows

$$P_{\text{up}} = P\left\{\frac{|\boldsymbol{e}|}{\sigma_e^t} > \sqrt{\tau}\right\} = 2Q_e(\sqrt{\tau}), \tag{4.18}$$

where $Q_e(\cdot)$ is the complementary Gaussian cumulative distribution function, given by $Q_e(x) = 1/(2\pi)\int_x^\infty exp(-t^2/2)dt$ [44]. Therefore, the parameter $\sqrt{\tau}$ can be obtained from equation (4.18) as

$$\sqrt{\tau} = Q_e^{-1}\left(\frac{P_{\text{up}}}{2}\right), \tag{4.19}$$

where $Q_e^{-1}(\cdot)$ is the inverse of the $Q_e(\cdot)$ function.

At iteration i in the epoch t, the estimated error variance is calculated by

$$(\sigma_e^t(i))^2 = (1-\lambda_e)\sigma_e^2 + (\lambda_e)(\sigma_e^t(i-1))^2, \tag{4.20}$$

where σ_e^2 is the error variance related to the data in the i-th iteration, and λ_e is the forgetting factor. At the each start of the epoch t, the estimated error variance depends on the last error variance $(\sigma_e^{t-1}(b))^2$ from the previous epoch, thus the equation for this dependence is established by

$$(\sigma_e^t(0))^2 = P_{\text{up}}(\sigma_e^{t-1}(b))^2. \tag{4.21}$$

The main goal of this application in NN is to reduce the computational cost, with the possibility of improving algorithm performance. The NN algorithm for a regression problem is outlined in Table 4.4.

Table 4.4: Data Selection Feed-Forward Multilayer Neural Network algorithm in a regression problem

DS Feed-Forward Multilayer Neural Network algorithm in regression

Initialize

$\{(\mathbf{x}(1),\mathbf{y}(1)), (\mathbf{x}(2),\mathbf{y}(2)), \cdots, (\mathbf{x}(M),\mathbf{y}(M))\}$,

$\mathbf{W} = \{\mathbf{W}^1,\mathbf{W}^2,\cdots,\mathbf{W}^L\}$ (random vectors),

Select: step size $\mu > 0$, number of epoch n_{ep}, mini-batch size b, number of layers L,

number of nodes $(o^1,\cdots,o^L)$, activation function f, output function f_L, forgetting factor λ_e

Objective function J = Mean Squared Error

Define $iter = M/b$, $\sigma_e^0(b) = 0$ and prescribe P_{up}

$\sqrt{\tau} = Q_e^{-1}\left(\frac{P_{\text{up}}}{2}\right)$

Do for $t = 1 : n_{\text{ep}}$

$(\sigma_e^t(0))^2 = P_{\text{up}}(\sigma_e^{t-1}(b))^2$

Do for $i = 1 : iter$ (for each iteration, randomly select b examples

in a training dataset $\rightarrow \mathbf{X}_{(t,i)} = [\bar{\mathbf{x}}(1), \bar{\mathbf{x}}(2), \cdots, \bar{\mathbf{x}}(b)]$, $\mathbf{Y}_{(t,i)} = [\bar{y}(1), \bar{y}(2), \cdots, \bar{y}(b)]$)

[Forward Propagation]:

$\mathbf{Y}^0 = [\bar{y}^0(1),\bar{y}^0(2),\cdots,\bar{y}^0(b)] = [ones(1,b); \mathbf{X}_i^t]$ ($ones(1,b)$ are the bias term)

Do for $l = 1 : L-1$

$\mathbf{X}^l = (\mathbf{W}^l)^T \mathbf{Y}^{l-1}$

$\mathbf{Y}^l = [ones(1, b); f(\mathbf{X}^l)]$

end

$\mathbf{X}^L = (\mathbf{W}^L)^T \mathbf{Y}^{L-1}$

$\hat{\mathbf{Y}}_{(t,i)} = [\hat{y}(1), \hat{y}(2), \cdots, \hat{y}(b)] = \mathbf{Y}^L = g(\mathbf{X}^L)$

[Data Selection]:

$e(\hat{y}(k), \bar{y}(k)) = (\hat{y}(k) - \bar{y}(k))$, for $k = 1, \cdots, b$

$\mathcal{E} = [e(\hat{y}(1), \bar{y}(1)), \cdots, e(\hat{y}(b), \bar{y}(b))] = \hat{\mathbf{Y}}_{(t,i)} - \mathbf{Y}_{(t,i)}$

$\sigma_e^2 = \mathrm{Var}(\mathcal{E})$

$(\sigma_e^t(i))^2 = (1 - \lambda_e)\sigma_e^2 + (\lambda_e)(\sigma_e^t(i-1))^2$

$$\mathcal{R} \rightarrow \begin{cases} j \notin \mathcal{R}, \text{ if } \frac{|e(\hat{y}(k), \bar{y}(k))|}{(\sigma_e^t(i))} \leq \sqrt{\tau} \\ j \in \mathcal{R}, \text{ otherwise} \end{cases}, \quad \text{for } j = 1, \cdots, b$$

$\mathcal{R} = [j_1, j_2 \cdots, j_p]$

$\mathbf{Y}_{\mathcal{R}} = [\bar{y}(j_1), \bar{y}(j_2), \cdots, \bar{y}(j_p)]$

$\hat{\mathbf{Y}}_{\mathcal{R}} = [\hat{y}(j_1), \hat{y}(j_2), \cdots, \hat{y}(j_p)]$

$\hat{P}_{\text{up}} = \frac{|\mathcal{R}|}{b}$

[Back-propagation]:

$\boldsymbol{\Delta}_{\mathcal{R}}^L = [\boldsymbol{\delta}^L(j_1), \boldsymbol{\delta}^L(j_2), \cdots, \boldsymbol{\delta}^L(j_p)] = g'(\mathbf{X}_{\mathcal{R}}^L) \otimes (\hat{\mathbf{Y}}_{\mathcal{R}} - \mathbf{Y}_{\mathcal{R}})$

Do for $l = L-1 : -1 : 1$

$\boldsymbol{\Delta}_{\mathcal{R}}^l = f'(\mathbf{X}_{\mathcal{R}}^l) \otimes [\mathbf{W}^{l+1} \boldsymbol{\Delta}_{\mathcal{R}}^{l+1}]_1^{o^l}$

end

[Updating the weights]:

Do for $l = 1 : L$

$\mathbf{W}^l = \mathbf{W}^l - \frac{\mu}{b\hat{P}_{\text{up}}} \mathbf{Y}_{\mathcal{R}}^{l-1} (\boldsymbol{\Delta}_{\mathcal{R}}^l)^T$

end

end

$J_{\text{train}}(\mathbf{W}) = \frac{1}{M} \sum_{m=1}^{M} J_m^1(\mathbf{W})$ (and J_{test} if this set is previously defined)

end

Classification

In the classification problem, the performance of a NN can be is verified using two objective functions, the mean squared error

$$J(\mathbf{W}) = \frac{1}{M}\sum_{k=1}^{o^L}\sum_{m=1}^{M}\left[\frac{1}{2}(\hat{y}_k(m) - y_k(m))^2\right], \tag{4.22}$$

and the cross-entropy (CE),

$$J(\mathbf{W}) = \frac{1}{M}\sum_{k=1}^{o^L}\sum_{m=1}^{M}\left[-y_k(m)\ln(\hat{y}_k(m)) - (1-y_k(m))\ln(1-\hat{y}_k(m))\right], \tag{4.23}$$

where we adopts as outputs the 0, 1 values. The equation (4.22) is one of the most widely used in signal processing and NN, and the equation (4.23) obtains the best results in more recent research when combined with the softmax activation function in the output layer,

$$\hat{y}_k(m) = y_k^L(m) = \frac{\exp(x_k^L(m))}{\sum_{j=1}^{o^L}\exp(x_k^L(m))}, \quad \text{for } k = 1, \cdots, o^L. \tag{4.24}$$

The value e in equation (4.10) is defined for the mean squared error and cross entropy, respectively, as

$$e(\hat{y}_k, y_k) = (\hat{y}_k - y_k)^2, \tag{4.25}$$

$$e(\hat{y}_k, y_k) = -y_k\ln(\hat{y}_k) - (1-y_k)\ln(1-\hat{y}_k) \tag{4.26}$$

where $\hat{y}_k$ is the estimated output for the k-th class and y_k is the k-th desired value for the output $\mathbf{y}$.

As in classification problems, the last layer has multiple outputs, and it is difficult to infer a distribution for the error signal, as occurred in the regression problem, equation (4.14). We do not normalize the variance and directly use the equation (4.11).

Also due to this difficulty in obtaining this probabilistic function, as it happens in the regression problem, we formulated another procedure with a similar idea to obtain the threshold. This approach for the classification problems aims to detect the values smaller than a given threshold in the equation (4.11) such that the related errors are disregarded in the process of updating the coefficients. This proposal requires the selection of a threshold for each mini-batch per iteration, chosen from

a binomial distribution with $n = b$ and $p = P_{\text{up}}$,

$$t_{\text{bin}} \sim \text{Bin}(n, p). \tag{4.27}$$

The data associated with measures smaller than the t_{bin}-th largest value of E, equation (4.11), are eliminated before the back-propagation process. Defining $\mathcal{C}$ as the set related to the indexes of E values greater or equal to the t_{bin}-th largest value, we obtain

$$\mathcal{C} = [j_1, j_2, \cdots, j_{t_{\text{bin}}}]. \tag{4.28}$$

Figure 4.4 illustrate the full process in one epoch from the entry of the training samples to verification of the performance of the weights update in the test samples (Test error). The figure 4.5 present the schematic strategy of the data selection in regression and classification problems to a determined mini-batch. At each iteration per epoch, a sample of size b (mini-batch) is selected to update the weight vector, this input sample (yellow color) is inserted in forward propagation. Then we apply data selection, eliminating non-informative data (white color) and proceeding with the remainder (blue color) in back-propagation to update the weight vector.

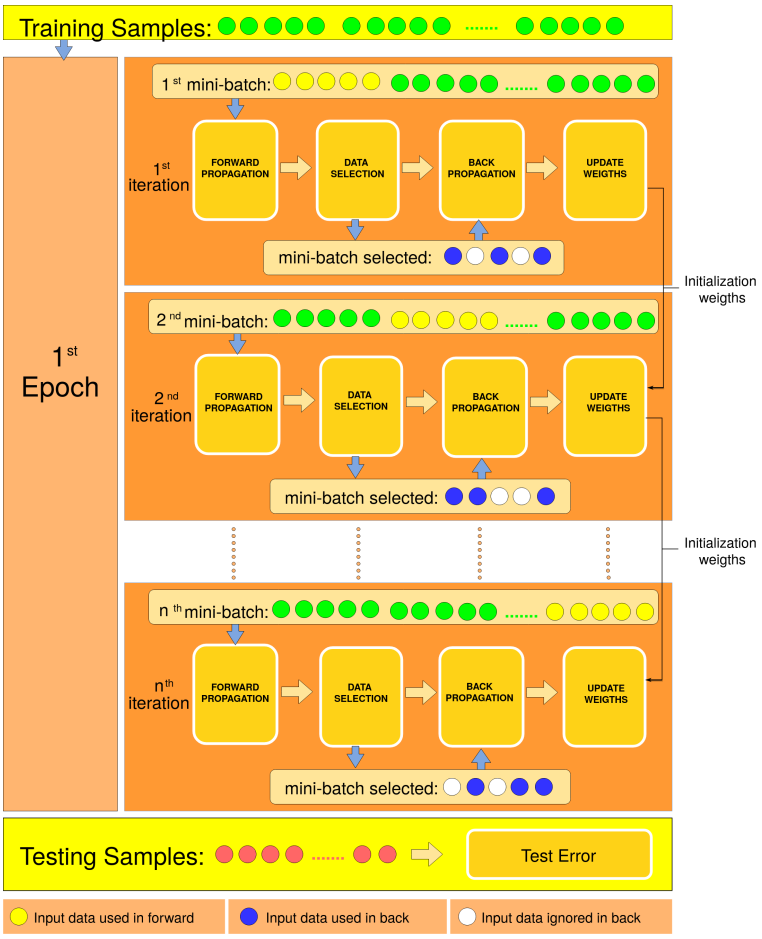


Figure 4.4: Epoch Scheme in neural network.

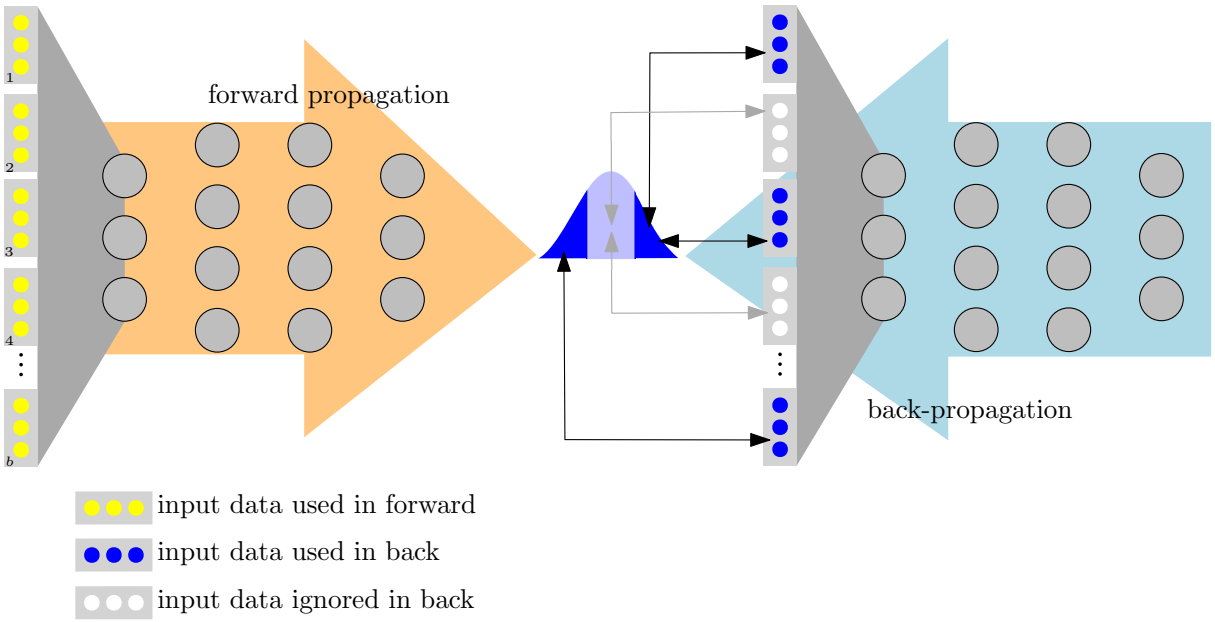


Figure 4.5: Data selection neural network diagram.

Again, this data selection method aims to identifying the non-informative values after the feed-forward propagation process at each iteration, eliminating some of the data before the back-propagation process. Therefore, as we are selecting an estimated portion $\hat{P}_{\text{up}}$ of data in each iteration, the update equation is rewritten as

$$\mathbf{W}^l(i+1) = \mathbf{W}^l(i) - \frac{\mu}{b}\mathbf{y}^{l-1}(\boldsymbol{\delta}^l)^T \rightarrow \mathbf{W}^l(i+1) = \mathbf{W}^l(i) - \frac{\mu}{b\hat{P}_{\text{up}}}\mathbf{y}_{\mathcal{C}}^{l-1}(\boldsymbol{\delta}_{\mathcal{C}}^l)^T, \quad (4.29)$$

The complete procedure for Data Selection Feed-Forward Multilayer Neural Network is described in algorithm 4.5.

Table 4.5: Data Selection Feed-Forward Multilayer Neural Network algorithm in classification problem

DS Feed-Forward Multilayer Neural Network algorithm in classification

Initialize

$\{(\mathbf{x}(1),\mathbf{y}(1)),(\mathbf{x}(2),\mathbf{y}(2)),\cdots,(\mathbf{x}(M),\mathbf{y}(M))\}$,

$\mathbf{W}=\{\mathbf{W}^1,\mathbf{W}^2,\cdots,\mathbf{W}^L\}$ (random vectors),

Select: step size $\mu>0$, number of epoch n_{ep}, mini-batch size b, number of layers L, number of nodes $(o^1,\cdots,o^L)$, activation function f

Option 1: Objective function $J=$ Mean Squared Error, output function $f_L=$ Linear or Hyperbolic tangent;

Option 2: Objective function $J=$ Cross-Entropy, output function $f_L=$ Softmax;

Define $iter=M/b$, prescribe P_{up}

Do for $t=1:n_{\text{ep}}$

Do for $i=1:iter$ (for each iteration, randomly select b examples in training dataset $\rightarrow \mathbf{X}_{(t,i)}=[\bar{\mathbf{x}}(1),\bar{\mathbf{x}}(2),\cdots,\bar{\mathbf{x}}(b)]$, $\mathbf{Y}_{(t,i)}=[\bar{\mathbf{y}}(1),\bar{\mathbf{y}}(2),\cdots,\bar{\mathbf{y}}(b)]$)

[Forward Propagation]:

$\mathbf{Y}^0=[\bar{\mathbf{y}}^0(1),\bar{\mathbf{y}}^0(2),\cdots,\bar{\mathbf{y}}^0(b)]=[ones(1,b);\mathbf{X}_{(t,i)}]$ ($ones(1,b)$ are the bias term)

Do for $l=1:L-1$

$\mathbf{X}^l=(\mathbf{W}^l)^T\mathbf{Y}^{l-1}$

$\mathbf{Y}^l=[ones(1,b);f(\mathbf{X}^l)]$

end

$\mathbf{X}^L=(\mathbf{W}^L)^T\mathbf{Y}^{L-1}$

$\hat{\mathbf{Y}}_{(t,i)}=[\hat{\mathbf{y}}(1),\hat{\mathbf{y}}(2),\cdots,\hat{\mathbf{y}}(b)]=\mathbf{Y}^L=g(\mathbf{X}^L)$

[Data Selection]:

$$e(\hat{y}_k(j), \bar{y}_k(j)) = \begin{cases} (\hat{y}_k(j) - \bar{y}_k(j))^2, & \text{if Mean Squared Error} \\ -\bar{y}_k(j)\ln(\hat{y}_k(j)) - (1-\bar{y}_k(j))\ln(1-\hat{y}_k(j)), & \text{if Cross Entropy} \end{cases}$$

for $j = 1, \cdots, b$ and $k = 1, \cdots, o^L$

$\mathcal{E}^k = [e(\hat{y}_k(1), \bar{y}_k(1)), \cdots, e(\hat{y}_k(b), \bar{y}_k(b))]$, for $k = 1, \cdots, o^L$

$\boldsymbol{E} = \left[\sum_{k=1}^{o^L} e(\hat{y}_k(1), \bar{y}_k(1)), \sum_{k=1}^{o^L} e(\hat{y}_k(2), \bar{y}_k(2)), \cdots, \sum_{k=1}^{o^L} e(\hat{y}_k(b), \bar{y}_k(b))\right]$

$t_{\text{bin}} \sim \text{Bin}(n,p)$,

$\mathcal{C} = [j_1, j_2 \cdots, j_{t_{\text{bin}}}]$, where $\mathcal{C}$ is the index set related to the t_{bin} largest values in the vector $\boldsymbol{E}$

$\mathbf{Y}_{\mathcal{C}} = [\bar{\mathbf{y}}(j_1), \mathbf{y}_{j_2}, \cdots, \mathbf{y}_{t_{\text{bin}}}]$

$\hat{\mathbf{Y}}_{\mathcal{C}} = [\hat{\mathbf{y}}(j_1), \hat{\mathbf{y}}(j_2), \cdots, \hat{\mathbf{y}}(t_{\text{bin}})]$

$\hat{P}_{\text{up}} = \frac{|\mathcal{C}|}{b}$

[Back-propagation]:

$$\boldsymbol{\Delta}_{\mathcal{C}}^L = [\boldsymbol{\delta}^L(j_1), \boldsymbol{\delta}^L(j_2), \cdots, \boldsymbol{\delta}^L(t_{\text{bin}})] = \begin{cases} g'(\mathbf{X}_{\mathcal{C}}^L) \otimes (\hat{\mathbf{Y}}_{\mathcal{C}} - \mathbf{Y}_{\mathcal{C}}), & \text{if is chosen the } \textit{option 1} \\ (\hat{\mathbf{Y}}_{\mathcal{C}} - \mathbf{Y}_{\mathcal{C}}), & \text{if is chosen the } \textit{option 2} \end{cases}$$

Do for $l = L-1 : -1 : 1$

$\boldsymbol{\Delta}_{\mathcal{C}}^l = f'(\mathbf{X}_{\mathcal{C}}^l) \otimes [\mathbf{W}^{l+1} \boldsymbol{\Delta}_{\mathcal{C}}^{l+1}]_1^{o^l}$

end

[Updating the weights]:

Do for $l = 1 : L$

$\mathbf{W}^l = \mathbf{W}^l - \frac{\mu}{b\hat{P}_{\text{up}}} \mathbf{Y}_{\mathcal{C}}^{l-1} (\boldsymbol{\Delta}_{\mathcal{C}}^l)^T$

end

end

$J_{\text{train}}(\mathbf{W}) = \frac{1}{M} \sum_{k=1}^{o^L} \sum_{m=1}^{M} J_m^k(\mathbf{W})$ (and J_{test} if this set is previously defined)

end

4.3 Simulations

In this section, the performance of the data selection method in the neural network is verified in different datasets. The proposed method is evaluated considering classification and regression problems. All algorithms in this chapter were implemented in MATLAB and are available online on the GitHub [46]. The simulations were performed in a computer with Intel Core i7-7500U CPU 2.70GHz x4 processor and with 15.5 GB of memory.

The activation function chosen for both problems was the ReLU function. As previously mentioned, the algorithm was tested in two combinations of the objective function and the output function: 1) cross-entropy error and softmax function for classification problems; 2) mean square error and linear function for regression and classification problems. The number of layers and the number of units per layer in the hidden layers varies according to the problem. The parameters in this section were established from the main neural network references followed by this text [2, 3, 28]. We vary the probability of update so that $P_{\text{up}} \in \{0.005, 0.1, 0.3, 0.5, 0.7\}$ and compare with the case where the data is always updated, $P_{\text{up}} = 1$, in order to infer the performance of the data selection method.

The reduction on the computational cost is the main benefit of the decrease of update probability. To verify whether there is also an improvement in the algorithm performance, an additional simulation is included in each problem, considering for each dataset an approximate mini-batch for the best performance. For example, in most cases, the best accuracy was observed when setting $P_{\text{up}} = 0.3$ and $b = 256$. Therefore, we compared this result with an alternative simulation assuming $P_{\text{up}} = 1$ and $b = 80$, as this value corresponds to approximately 30% of 256.

There are advantages and disadvantages of using a smaller mini-batch size (b) or applying the data selection to obtain a small batch size ($\hat{P}_{\text{up}} b$) in the back-propagation process. Both require less memory during the process, as we consider a small number of samples. On the other hand, we can say that a smaller mini-batch results in a less accurate estimate (stochastic), i.e., a larger variance.

In the next simulations, the data selection method is applied to the neural network to verify its performance in the testing set. We quantify the computational complexity of the proposed algorithms by counting the number of flops required to perform the NN computation in appendix $A.2$.

4.3.1 Regression - Problem 1: Superconductivity Dataset

The UC Irvine (UCI) Machine Learning Repository provides superconducting material dataset, which is considered in this subsection [89, 90]. This type of material has zero resistance and several practical applications. However, to reach

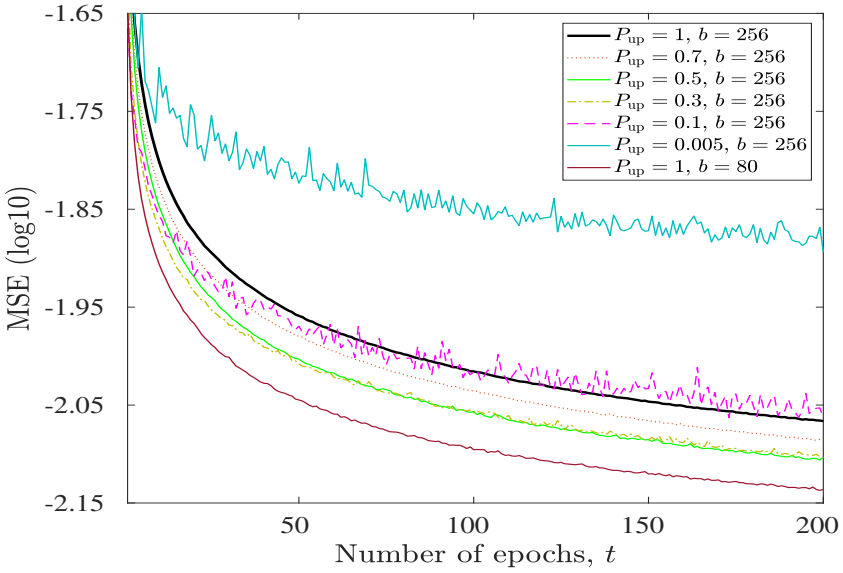
this property, the temperature (T) must be at or below its critical temperature (T_c). Moreover, the scientific model and theory that predicts the value (T_c) is an open problem. In this subsection using a neural network, a model for this superconducting critical temperature (T_c) is formulated from features extracted based on thermal conductivity, atomic radius, valence, electron affinity, and atomic mass.

This dataset has no missing values and contains 81 features (input of the NN), among which two are categorical variables. The One hot encoding[1] is used on the two categorical variables and, as a result, we replace them by 14 features in the dataset. When selecting the features, a basic criterion adopted is to eliminate a portion of the attributes which are not correlated enough to the output. Thus, we eliminate 6 attributes that correlate less than 0.05 to the output value, amounting a total of 89 features in the dataset. The last step in data preprocessing consists of normalizing the attributes in the range 0 to 1, which is also known as the min-max normalization. This procedure is useful in neural networks to scale the dataset that has columns with different magnitude of values. The output value also is scaled into the range from 0 to 1.

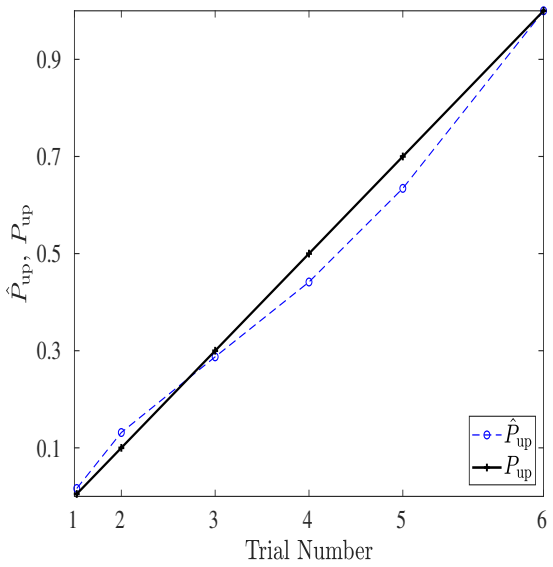
The superconductivity dataset has 18000 training examples and 3000 test examples. We use a step size of $\mu = 0.01$, the mini-batch size has $b = 256$ examples per iteration, and the epoch is equal to $n_{\text{up}} = 200$. The number of hidden layers is 2 and each hidden layer has exactly 128 nodes. The forgetting factor used in the data selection process to update the estimated error variance is evaluated as $\lambda_e = 0.9$.

The test MSE curves obtained by using different probabilities of update P_{up}, as shown above, are presented in Figure 4.6*a*. We may notice that there is always an improvement in two-layer NN performance ($b = 256$) when we decrease the amount of updates per epoch, but from a certain value P_{up}, the test curve acquire a stochastic behavior (high variance) and the algorithm performance decreases. We can also observe that between the values $P_{\text{up}} = 0.3$ and $P_{\text{up}} = 0.1$, we can achieve a minimum value for MSE by selecting a specific amount of data. The performance of $P_{\text{up}} = 1$ and $b = 80$ has a slight advantage in the accuracy of results over simulations using data selection, but with greater variance in some cases and highest number of flops (see Tables 4.6 and 4.7 at the end of the subsection 4.3.4). As one can see in the Figure 4.6*b*, the estimated probability of update are close to the prescribed probability of update.

[1] One hot encoding is a process by which categorical variables are converted into a format so that Machine Learning algorithms can do a better job of predicting. For example, suppose the values in a categorical variable are from 0 to $N - 1$ categories. We replace this column with other N columns and apply the following rule: 1 indicates occurrence of the related category and 0 otherwise [2].



(a)



(b)

Figure 4.6: Superconductivity simulation: (a) Test MSE curves comparing the case when the algorithm is always updated $P_{\text{up}} = 1$ ($b = 256$ and $b = 80$) and with the probability of update $P_{\text{up}} \in \{0.005, 0.1, 0.3, 0.5, 0.7\}$ (b) Comparison between the desired P_{up} and achieved $\hat{P}_{\text{up}}$.

4.3.2 Regression - Problem 2: Online News Popularity Dataset

Due to the widespread use of the web, the number of news shares is increasing each year. This number indicates how popular the news is. In this subsection, we propose to find a model to predict the popularity of online news using neural network and data selection. The dataset used in this subsection has a total of 39644 articles

published by Mashable website [91] in two years. This dataset is provided by UCI Machine Repository [89], originally acquired and preprocessed by K.Fernandes et al. [92]. The 60 features are extracted concerning different aspects such as words, links, and publication time. The goal is to help media companies to predict the number of shares on the web before the news becomes published.

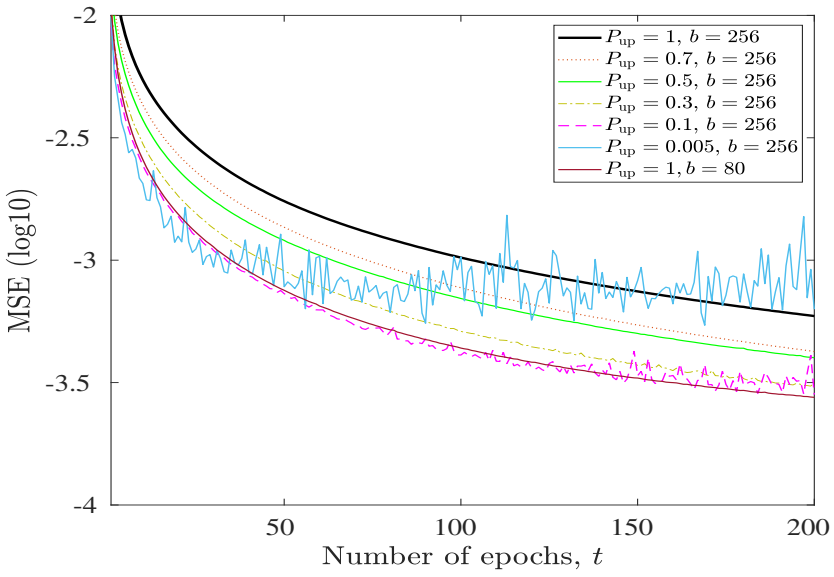
The dataset has no missing values, and all categorical variables are already processed. The columns related to the URL of the article and time delta (days between the article publication and the dataset acquisition) are excluded in this process. As in the previous subsection, we employ the basic criterion, in which all attributes that correlate less than 0.05 with the output value are eliminated, amounting a total of 50 features for the input in the dataset after exclusion. The min-max normalization is also used on features and output signal such that the values are in the range of [0,1].

The dataset is divided into 35000 examples for the training dataset and 4644 examples for the test dataset. The parameters chosen for this simulation are the $\mu = 0.01$, $b = 256$ and $n_{\text{up}} = 200$. The number of hidden layers is 2 and the number of nodes in each hidden layer is equal to 128. The forgetting factor for data selection in estimating error variance is equal to $\lambda_e = 0.99$.

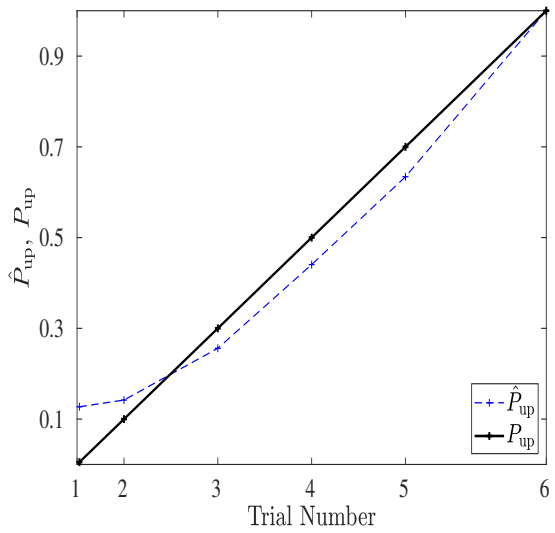
The evolutions of the MSE by varying the probability of update P_{up} are illustrated in Figure 4.7*a*. We can note an improvement in the evolution of MSE ($b = 256$) as we decrease the amount of update per epoch. With exception for the case $P_{\text{up}} = 0.005$ that obtain a result worse than $P_{\text{up}} = 1$ and with a high variance. Again the network with $P_{\text{up}} = 1$ and $b = 80$ presents a slightly better performance when compared with simulations using data selection, but with larger number of flops (see Tables 4.6 and 4.7 at the end of the subsection 4.3.4). Figure 4.7*b* shows the result of the estimated probability of update $\hat{P}_{\text{up}}$ when compared to the prescribed probability of update P_{up}. We can observe that the estimation obtained in NN always achieves a value close to the prescribed probability.

4.3.3 Regression - Problem 3: Facebook Comment Volume Dataset

In this subsection, we consider the Facebook comment volume dataset which is provided by UCI Machine Learning Repository [89]. Currently, the amount of data on social networking services is increasing exponentially day by day. So it is important to verify the dynamic behavior of users in media services. We propose a neural network model to the dataset to predict how many comments the post will receive. This dataset contains features extracted from Facebook posts, as page likes, page category, publication day, among others.



(a)



(b)

Figure 4.7: Online news popularity simulation: (a) Test MSE curves comparing the case when the algorithm is always updated $P_{\text{up}} = 1$ ($b = 256$ and $b = 80$) and with the probability of update $P_{\text{up}} \in \{0.005, 0.1, 0.3, 0.5, 0.7\}$ (b) Comparison between the desired P_{up} and achieved $\hat{P}_{\text{up}}$.

The dataset consists in the union of 2700 facebook pages for 57000 posts. The data cleaning was already performed by Kamaljot Singh and Dinesh Kumar [93]. After cleansing 5.892, posts are omitted and 51.108 posts are kept. This dataset was divided into two subsets, the training dataset with 40.988 examples and the test dataset with 10.120 examples.

Facebook dataset has no missing values. The categorical variables were already

preprocessed. The criterion correlation is the same as the one used in previous simulations, eliminating three variables that correlate less than 0.05 with the output value, subsequently amounting a total of 50 variables in the input. We also use the min-max normalization applied on features and output signal so that the values are in the range of [0,1].

The parameter chosen in this subsection are $\mu = 0.01$, $b = 256$ and $n_{\text{up}} = 200$. The number of hidden layers is 2 and the number of nodes in each hidden layer is equal to 128. The forgetting factor in the error variance estimation is $\lambda_e = 0.99$.

The evolutions of the MSE are illustrated in figure 4.8*a*, each curve represents a result for the probability of update P_{up} by varying its values to compare when all data are selected ($P_{\text{up}} = 1$). In this figure, we notice an improvement in algorithm performance ($b = 256$) each time we decrease the probability of update P_{up}. In addition, the $P_{\text{up}} = 0.3$ presents a better performance when compared to the $P_{\text{up}} = 1$ and $b = 80$ (see Tables 4.6 and 4.7 at the end of the subsection 4.3.4). The comparison between the estimated of probability $\hat{P}_{\text{up}}$ and the prescribed probability of update P_{up} is presented in Figure 4.8b, where we can conclude that the estimated of probability is close to the prescribed probability.

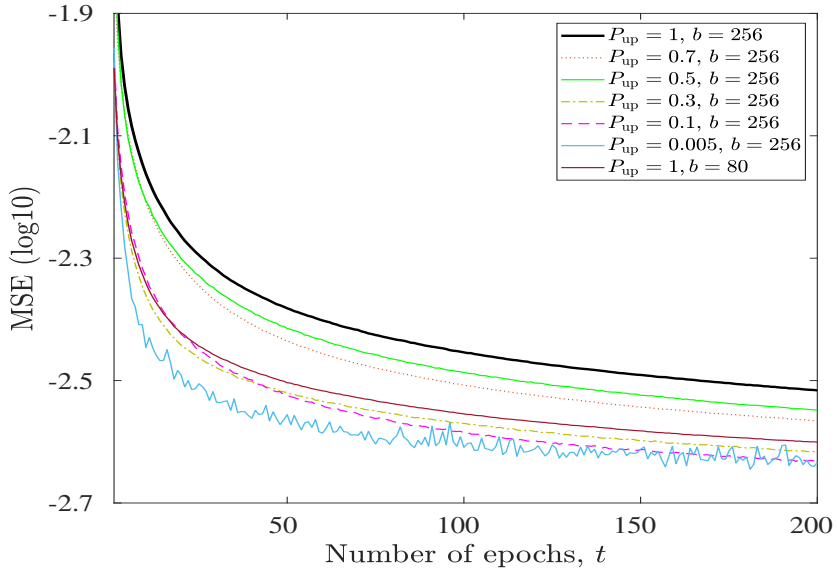
4.3.4 Regression - Problem 4: FIFA 19 Complete Player Dataset

The dataset considered in this subsection is the FIFA 19 Complete Player, provided by the site kaggle [94]. This dataset contains all statistics and playing attributes of all players in the version of FIFA 19.

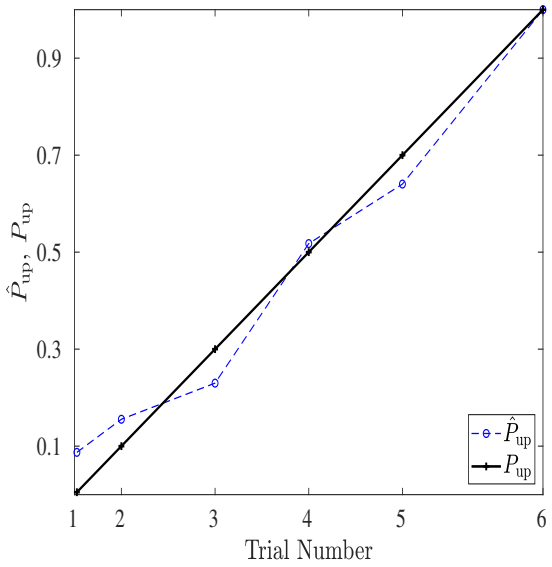
The dataset has no missing values. Initially it contains 89 features. In this set of data, it was necessary to do a simple preprocessing. The columns referring to the goalkeepers attributes are eliminated due to low correlation with the output. Excluding a few more insignificant variables like url photos, ID, and among others, we obtain a total of 73 features for the input. These variables are normalized using the min-max normalization on features and output signal.

The dataset considered in this simulation has a total of 12000 training examples and 2743 test examples. We chose the parameters as $\mu = 0.01$, $b = 256$ and $n_{\text{up}} = 200$. The number of hidden layers is 2 and the number of nodes in each hidden layer is 128. The forgetting factor chosen in this subsection to estimate the error variance online is equal to $\lambda_e = 0.9$.

The results of the performance for test MSE curves varying the probability of update P_{up} are shown in Figure 4.9a. We may note that the MSE improves ($b = 256$) when the number of updates in the learning process decreases, achieving an optimal value between $P_{\text{up}} = 0.3$ and $P_{\text{up}} = 0.1$. In this problem, the result with $P_{\text{up}} = 1$ and



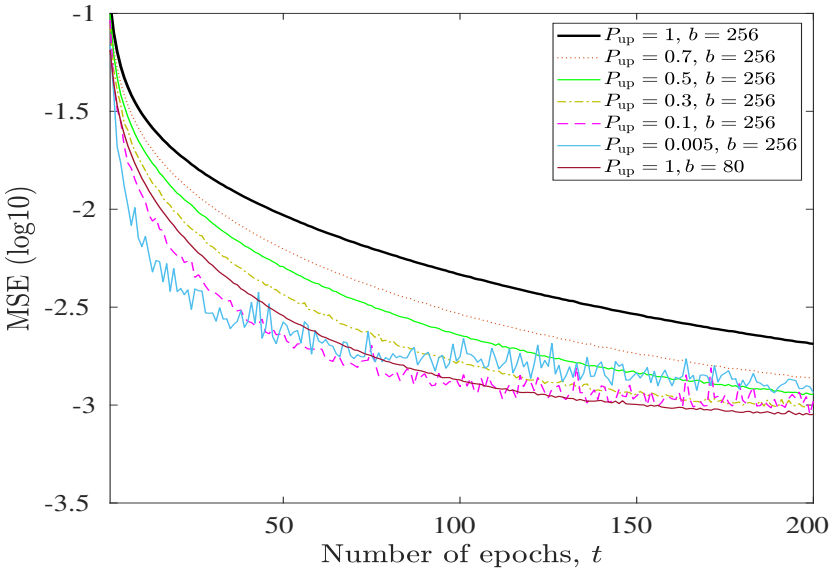
(a)



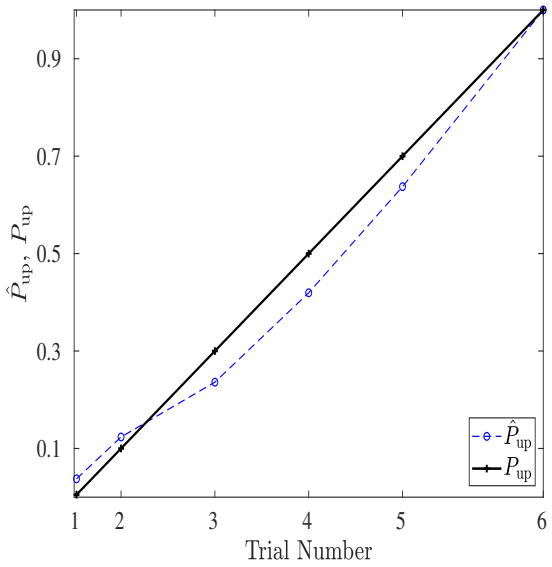
(b)

Figure 4.8: Facebook Comment Volume simulation: (a) test MSE curves comparing the case when the algorithm is always updated $P_{\text{up}} = 1$ ($b = 256$ and $b = 80$) and with the probability of update $P_{\text{up}} \in \{0.005, 0.1, 0.3, 0.5, 0.7\}$ (b) Comparison between the desired P_{up} and achieved $\hat{P}_{\text{up}}$.

$b = 80$ yields slightly better performance, but the value is close to the probability of update $P_{\text{up}} = 0.3$. The estimated probability of update for this simulation is presented in Figure 4.9b, where a result close to the prescribed probability of the update is observed.



(a)



(b)

Figure 4.9: FIFA 19 Complete Player simulation: (a) Test MSE curves comparing the case when the algorithm is always updated $P_{\text{up}} = 1$ ($b = 256$ and $b = 80$) and with the probability of update $P_{\text{up}} \in \{0.005, 0.1, 0.3, 0.5, 0.7\}$ (b) Comparison between the desired P_{up} and achieved $\hat{P}_{\text{up}}$.

In Table 4.6, the performance of the data selection method applied in regression problems inferred as the MSE average over the last 10 epochs. As noted in the table, using the data selection method the average MSE achieves a better result between $P_{\text{up}} = 0.3$ and $P_{\text{up}} = 0.1$ with mini-batch $b = 256$, leading us to conclude that using a small portion of the samples, the model prediction improves considerably in the final result. Comparing the best results in data selection and simulations with

$P_{\text{up}} = 1$ and $b = 80$, the values are close, but with a considerable reduction in the computational cost for the proposed method, Table 4.7.

Table 4.6: Comparison between the regression problems in the test error varying the probability of update P_{up} (blue is the best and red is the worst for each problem)

	Problem 1	Problem 2	Problem 3	Problem 4
$P_{\text{up}} = 1$	0.0086±2.79e-5	6.12e-04±7.52e-6	0.0031±1.06e-5	0.0021±3.91e-5
$P_{\text{up}} = 0.7$	0.0082±3.02e-5	4.41e-4±5.89e-6	0.0027±1.8e-4	0.0014±2.74e-5
$P_{\text{up}} = 0.5$	0.0079±5.76.e-5	4.12e-04±5.38e-6	0.0029±1.07e-5	0.0012±2.93e-5
$P_{\text{up}} = 0.3$	0.0079 ±9.96.e-5	3.20e-4±8.72e-6	0.0024±8.49e-6	0.0010±1.41e-4
$P_{\text{up}} = 0.1$	0.0088±3.18e-4	3.56e-04±1.50e-4	0.0023±1.43e-5	0.0011± 2.74e-4
$P_{\text{up}} = 0.005$	0.0132±6.4e-4	0.0014±0.0017	0.0024± 1.19e-4	0.0013± 2.84e-4
$P_{\text{up}} = 1, b = 80$	0.0073±3.97e-5	2.96e-4±2.79e-6	0.0025±6.79e-6	0.0009±3.34e-5

Table 4.7: Approximated number of flops in one epoch varying the probability of update P_{up}

	Problem 1	Problem 2	Problem 3	Problem 4
$P_{\text{up}} = 1$	2599×10^6	43549×10^6	50999×10^6	16349×10^6
$P_{\text{up}} = 0.7$	2119×10^6	35269×10^6	41309×10^6	13299×10^6
$P_{\text{up}} = 0.5$	1799×10^6	2974×10^6	3483×10^6	1125×10^6
$P_{\text{up}} = 0.3$	1479×10^6	2422×10^6	2837×10^6	922×10^6
$P_{\text{up}} = 0.1$	1159×10^6	1870×10^6	2190×10^6	719×10^6
$P_{\text{up}} = 0.005$	1007×10^6	1608×10^6	1883×10^6	622×10^6
$P_{\text{up}} = 1, b = 80$	2598×10^6	4354×10^6	5099×10^6	1634×10^6

4.3.5 Classification - Problem 5: MNIST Handwritten Digit Recognition Dataset

The Modified National Institute of Standards and Technology (MNIST) [95] database is a large data set containing digits hand written by students and employees of the United States Census Bureau, Figure 4.10. This dataset consists of 60,000 and 10,000 in training and test examples, respectively. The input of this set is a matrix 28×28, where each value represents a pixel of the image. The input signal is normalized to the range 0 to 1. The output dataset has integer values between 0 and 9, widely used in the field of machine learning, and it is one of the most commonly used in NN to explore learning techniques.

This dataset has been previously preprocessed and therefore has no missing values. Since we are modeling an image classification problem, there are no

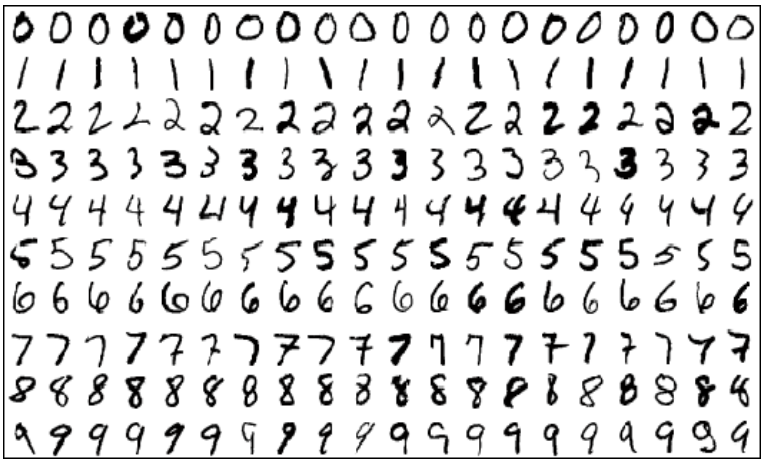


Figure 4.10: Sample images from MNIST dataset.

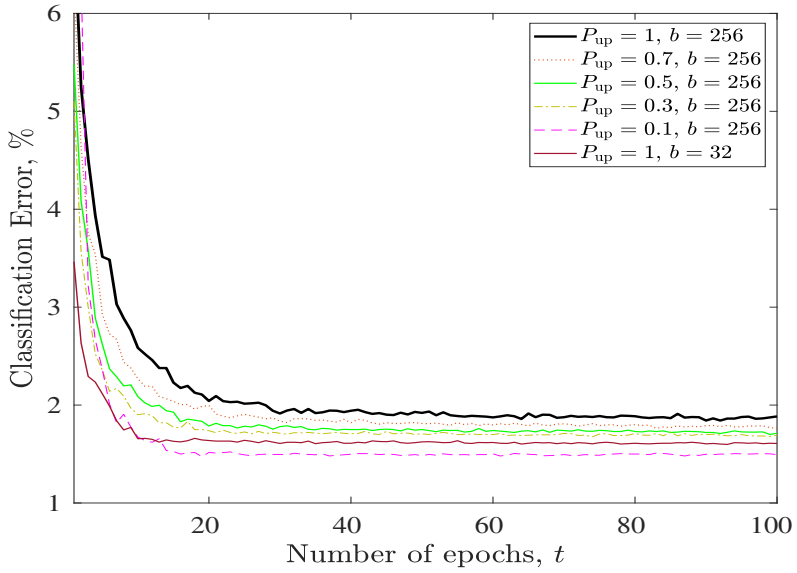
categorical variables, and we have not eliminated any variables before the learning process.

The step size chosen for this simulation is $\mu = 0.1$ when the objective function is the cross-entropy error and the softmax function is the output activation function, while for MSE in the objective function and linear function on output activation, the step size is equal to 0.01. The other parameters are $b = 256$ and $n_{\text{up}} = 100$ in the two cases. With the number of hidden layers equal to 2 and the number of nodes in each hidden layer equal to 1024.

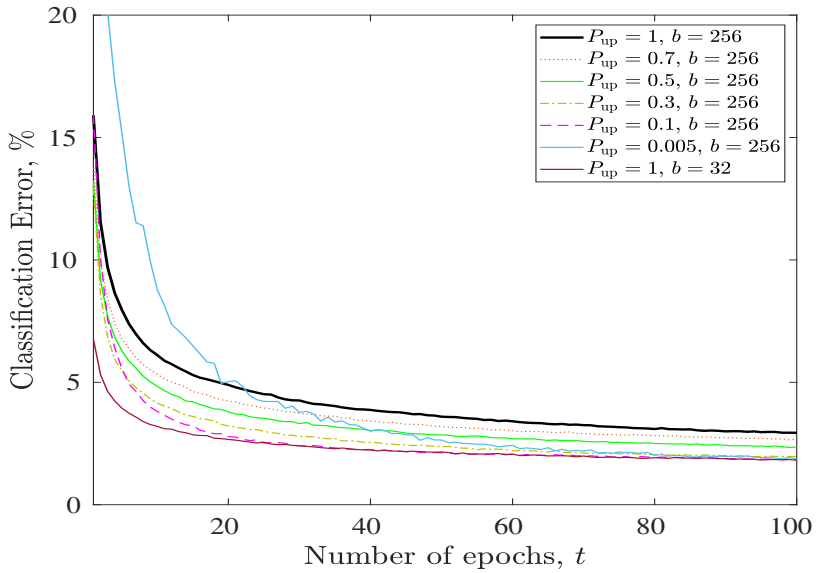
The figures 4.11*a* and 4.11*b* show the results with cross-entropy as the objective function and softmax as the output activation function, and results with MSE as the objective function and linear function as the output activation function, respectively. Again, the data selection method achieves a good performance in the NN, showing that when we decrease the amount of update per epoch it occurs an improvement in two layer NN performance ($b = 256$). Also, when compared with simulation $P_{\text{up}} = 1$ and $b = 32$, the best result in data selection achieves a better performance with the benefit of requiring a reduced number of flops. The test classification error curve for $P_{\text{up}} = 0.005$ is not depicted in first figure, because it has a higher value compared to other probabilities, approximately equal to 0.20 in the last epochs.

4.3.6 Classification - Problem 6: EMNIST Letters Dataset

The EMNIST letter dataset considered in this subsection is provided by the National Institute of Standards and Technology (NIST) [96]. This dataset can represent the 26 letters of the alphabet at the output, Figure 4.12. The letters EMNIST contain 120000 training examples and 25000 test examples, and the examples have been normalized between 0 and 1. The input is a 28×28 matrix, where each input represents a pixel in the image. It has no categorical variables, so



(a)



(b)

Figure 4.11: MNIST Handwritten Digit Recognition simulation: Test classification error (%) comparing the case when the algorithm is always updated $P_{\text{up}} = 1$ ($b = 256$ and $b = 32$) with the probability of update $P_{\text{up}} \in \{0.1, 0.3, 0.5, 0.7\}$ when (a) the output activation function is softmax and objective function is cross-entropy error and when (b) the output activation function is linear function and the objective function is MSE.

it does not need any transformation in the variables. This dataset has already been preprocessed [97].

The choice of the objective function and the output activation function influences the choice of the μ parameter, thus if we choose MSE as the objective function and the linear function as the output function, the parameters in this NN will be

Figure 4.12: Sample images from EMNIST letters dataset.

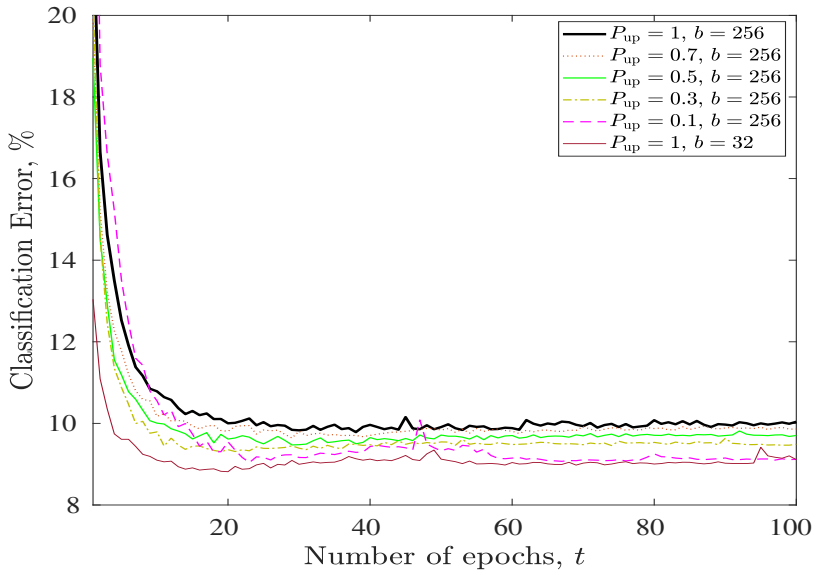
$\mu = 0.01$, $b = 256$ and $n_{\text{up}} = 100$. However, if the objective function is cross-entropy and the output function is softmax, the parameters will be $\mu = 0.1$, $b = 256$ and $n_{\text{up}} = 100$. The number of hidden layers is equal to 2. Also, the number of nodes in the hidden layers in this simulation is 1024.

The results regarding the test classification error when the probability of update is varied are shown in the Figures 4.13a and 4.13b for cross-entropy and MSE, respectively. The performance of simulation by setting $P_{\text{up}} = 1$ and $b = 32$ in CE as objective function and $P_{\text{up}} = 1$ and $b = 128$ in MSE as objective function have a slight advantage in the accuracy of results over simulations using data selection. Again, the test classification error curve for $P_{\text{up}} = 0.005$ is not included in the first figure due to higher classification errors, approximately 96% in the last epochs.

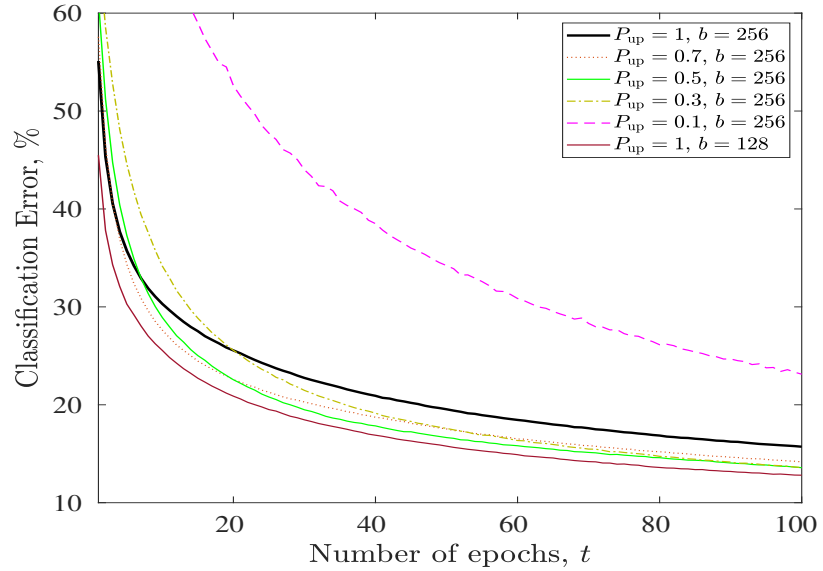
In Table 4.8, we show the performance for the data selection method in the classification problem. The value in each table entry is the averaged MSE over the last 10 epochs corresponding to a specific probability P_{up} and simulation problem. In all cases for $b = 256$, except in the last simulation, the method achieves the best result for $P_{\text{up}} = 0.1$. Also, we can note that the combination: cross-entropy as objective function combined with the softmax as output activation function achieve better results. In the Table 4.9, we conclude that the data selection reduces the computational cost. Comparing the best results in data selection and simulations with $P_{\text{up}} = 1$ and $b = 32$ ($b = 128$ in last column), the values of percentage in the test error are close.

4.3.7 Problem 7: Deep Neural Network (Transcoding Time and MNIST Datasets)

The UCI Machine Learning Repository [89] provides the dataset considered in this subsection. In this subsection, we propose a deep neural network (number



(a)



(b)

Figure 4.13: EMNIST letters simulation: Test classification error (%) comparing the case when the algorithm is always updated $P_{\text{up}} = 1$ with the probability of update $P_{\text{up}} \in \{0.1, 0.3, 0.5, 0.7\}$ when (a) the output activate function is softmax and objective function is cross-entropy error and when (b) the output activate function is linear function and objective function is MSE.

of hidden layers greater than 2) model to predict the transcoding time of videos given some basic characteristics as the set of features, including bit rate, frame rate, resolution, codec, among others. The basic idea of video transcoding is to convert unsupported video formats into supported ones.

This dataset has no missing values. Initially contains 20 columns, two of which are categorical variables (input and output coding, with 4 categories each), and using

Table 4.8: Comparison in test error between the classification problems varying the probability of update P_{up} (blue is the best and red is the worst for each problem)

	Problem 5 (CE)	Problem 5 (MSE)	Problem 6 (CE)	Problem 6 (MSE)
$P_{\text{up}} = 1$	1.87±0.023	2.97±0.037	9.99±0.069	15.95±0.163
$P_{\text{up}} = 0.7$	1.78±0.020	2.70±0.038	9.88±0.056	14.39±0.149
$P_{\text{up}} = 0.5$	1.72±0.021	2.39±0.04	9.72±0.059	13.8±0.151
$P_{\text{up}} = 0.3$	1.69±0.05	1.98±0.03	9.48±0.042	13.84±0.166
$P_{\text{up}} = 0.1$	1.50±0.014	1.86±0.035	9.19±0.034	23.82±0.493
$P_{\text{up}} = 0.005$	22.61±4.65	1.93±0.071	96.12±0.047	96.3±0.487
$P_{\text{up}} = 1$, $b = 32$	1.60±0.009	1.84±0.031	9.13±0.142	12.96±0.118

Table 4.9: Approximated number of flops in one epoch varying the probability of update

	Problem 5	Problem 6
$P_{\text{up}} = 1$	566×10^9	1145×10^9
$P_{\text{up}} = 0.7$	462×10^9	935×10^9
$P_{\text{up}} = 0.5$	393×10^9	794×10^9
$P_{\text{up}} = 0.3$	324×10^9	654×10^9
$P_{\text{up}} = 0.1$	254×10^9	514×10^9
$P_{\text{up}} = 0.005$	221×10^9	447×10^9
$P_{\text{up}} = 1$, $b = 32$	566×10^9	1145×10^9

the One Hot Encoding (OHE) results in 8 new columns. We also use the min-max normalization applied on features and output signals so that the values are in the range of [0,1].

The transcoding time dataset is divided into two subsets, training dataset with 55000 examples and test dataset with 13378 examples. The number of hidden layers is 4, and the number of nodes in each layer is 128. The parameters chosen in this subsection are $\mu = 0.01$, $b = 256$ and $n_{\text{up}} = 200$. The forgetting factor in the error variance estimation is $\lambda_e = 0.995$.

The test MSE curves are illustrated in figure 4.14, each curve represents the evolution of the MSE in relation to the fixed probability of update P_{up}. We may notice an improvement ($b = 256$) in the deep neural network performance when we decrease the amount of updates, but from a certain value P_{up} below 0.1, the algorithm performance degrades. The proposed methods performance is close to the case where $P_{\text{up}} = 1$ and $b = 80$, with the benefit of reduced computational complexity (Table 4.10). However, the proposed method has the advantage of requiring a reduced number of flops. The figure 4.15*a* shows the probability distribution for all the mini-batches size in 100-th epoch, which is similar to the Gaussian distribution

reinforcing our hypothesis of the equation (4.15). The comparison between the estimated probability $\hat{P}_{\text{up}}$ and the prescribed probability of update P_{up} is presented in Figure 4.15b, where we can conclude that the estimated of probability is close to the prescribed probability of update.

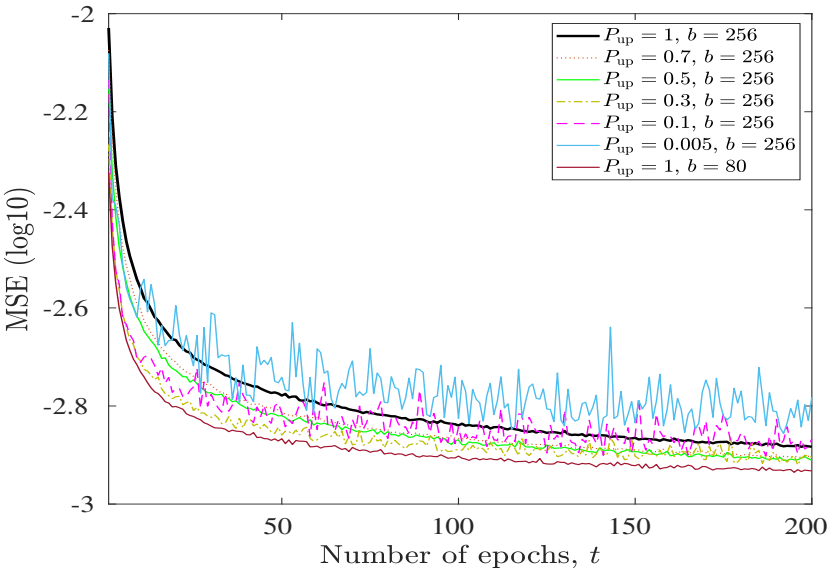
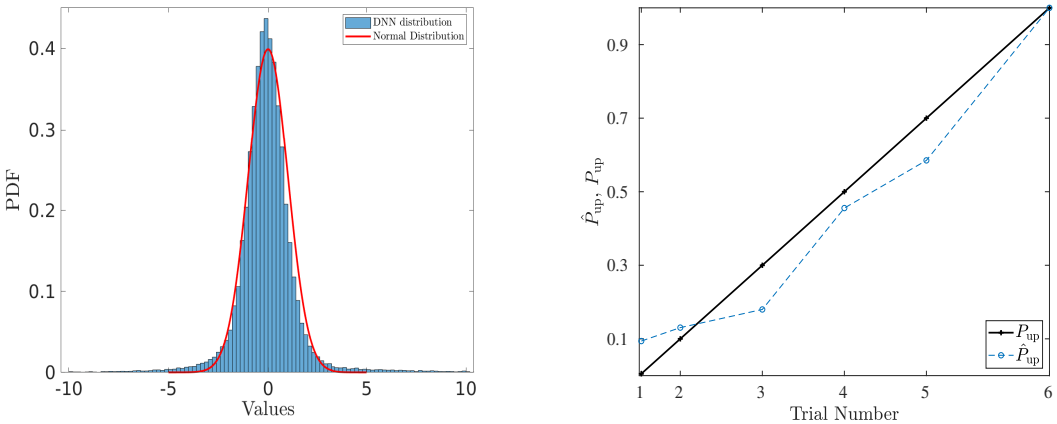


Figure 4.14: Transcoding time simulation: test MSE curves comparing the case when the algorithm is always updated $P_{\text{up}} = 1$ ($b = 512$ and $b = 64$) and with the probability of update $P_{\text{up}} \in \{0.005, 0.1, 0.3, 0.5, 0.7\}$.



(a) (b)

Figure 4.15: Transcoding time simulation: (a) Probability distribution for the samples selected in the 100-th epoch and (b) Comparison between the desired P_{up} and achieved $\hat{P}_{\text{up}}$.

The next example verifies the performance of the data selection method in classification deep learning. For this, we use the MNIST dataset detailed in subsection 4.3.5. The framework parameters are defined as: the number of hidden

layers equals 3, and the number of nodes in each layer is 1024. The other parameters are chosen as $\mu = 0.1$, $b = 128$ and $n_{\text{up}} = 100$. Moreover, we compared the data selection with another method called dropout [98] (explained in more detail in the appendix $A.1$), which purpose to make node selection in each epoch of the neural network, in both cases the main role is to reduce the computational cost. The results of this deep learning example is illustrated in 4.16, where the data selection with update probability P_{up} beat the dropout. In addition, a simulation with $P_{\text{up}} = 1$ and $b = 16$ is performed and compared to simulations with data selection, the results show that our proposed algorithms achieve a performance level close to the standard neural network, but with a reduction in computational cost when applying the data selection method in NN algorithm (Table 4.10).

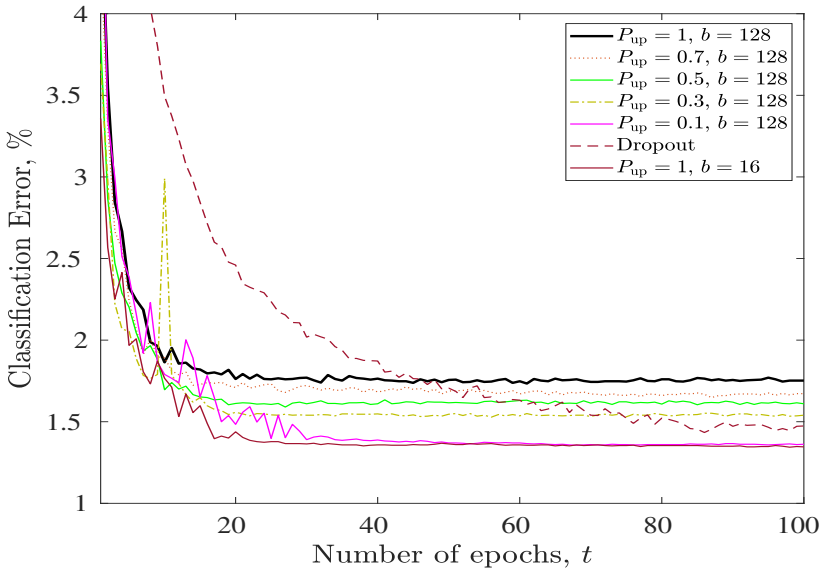


Figure 4.16: MNIST Handwritten Digit Recognition simulation: Test classification error (%) comparing the case when the deep learning algorithm utilizes the dropout technique, and when the data selection method is applied, varying the probability of update $P_{\text{up}} \in \{0.1,0.3,0.5,0.7,1\}$. The output activation function is given by softmax, and objective function is the cross-entropy error.

Table 4.10: Approximated number of flops in one epoch varying the probability of update

	MNIST Dataset	Transcoding Time Dataset
$P_{\text{up}} = 1$	944×10^9	16966×10^6
$P_{\text{up}} = 0.7$	764×10^9	13604×10^6
$P_{\text{up}} = 0.5$	645×10^9	11362×10^6
$P_{\text{up}} = 0.3$	525×10^9	9120×10^6
$P_{\text{up}} = 0.1$	405×10^9	6879×10^6
$P_{\text{up}} = 0.005$	348×10^9	5814×10^6
$P_{\text{up}} = 1$, $b = 16$	944×10^9	–
$P_{\text{up}} = 1$, $b = 80$	–	16966×10^6

4.4 Concluding Remarks

In this chapter, it was formulated a new method for data selection in neural selection. This method was applied in several dataset for classification and regression problems. In all simulation, the data selection achieves good performance even when only 30% of the data is considered, leading to consistent results. Therefore, data selection in NN is an excellent tool, mainly with the advantage of requiring a lower computational cost.

Chapter 5

Conclusions

5.1 Final Remarks

In this work, we proposed some data selection methods in signal processing and machine learning. The criterion employed is based on selecting only the innovative data in the iteration process, avoiding irrelevant and redundant information. The proposed theory was carefully constructed from a statistical point of view.

In signal processing, we verified the performance method in Conjugate Gradient (CG) and Kernel Conjugate Gradient (KCG) algorithms. Both algorithms achieve a good performance when the coefficients are updated less than 50% of times. Moreover, the simulations including outliers present satisfactory results when compared with the cases that no outliers are present.

In the neural network (NN), we evaluated the performance of the data selection method in regression and classification problems. The data-selective algorithms designed for neural networks have a similar result in most simulations where the updates are done 100% of times. Moreover, when the data selection method was incorporated into the algorithm, the computational cost decreased substantially. The number of flops is used to quantify this advantage. In addition, when compared to the dropout method, considering a Deep Neural Network (DNN), a better result was obtained.

5.2 Future Works

In adaptive filtering, the next goal is to apply the data selection to other algorithms, including linear and nonlinear problems.

As we concluded in the Chapter 4, the data selection method proposed in this dissertation is a promising technique for Machine Learning, the next step is to compare it other selection methods, as occurred with dropout. Then, we aim

at extending it to other neural architectures, such as the Convolutional Neural Networks (CNN), which is more suitable for image processing. Also, there is a possibility to improve the proposed method and establish new approaches in the NN learning methods.

Bibliography

[1] DINIZ, P. S. R. *Adaptive Filtering: Algorithms and Practical Implementation.* 4th ed. New York, Springer, 2013.

[2] BISHOP, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer, 2007.

[3] ABU-MOSTAFA, Y. S., MAGDON-ISMAIL, M., LIN, H.-T. *Learning From Data.* AMLBook, 2012.

[4] DURGESH, K. S., LEKHA, B. "Data classification using support vector machine", *Journal of theoretical and applied information technology*, v. 12, n. 1, pp. 1–7, 2010.

[5] LAI, C.-C., TSAI, M.-C. "An empirical performance comparison of machine learning methods for spam e-mail categorization". In: *Fourth International Conference on Hybrid Intelligent Systems (HIS'04)*, pp. 44–48. IEEE, 2004.

[6] KONONENKO, I. "Machine learning for medical diagnosis: history, state of the art and perspective", *Artificial Intelligence in medicine*, v. 23, n. 1, pp. 89–109, 2001.

[7] SWAN, A. L., MOBASHERI, A., ALLAWAY, D., et al. "Application of machine learning to proteomics data: classification and biomarker identification in postgenomics biology", *Omics: a journal of integrative biology*, v. 17, n. 12, pp. 595–610, 2013.

[8] BAJARI, P., NEKIPELOV, D., RYAN, S. P., et al. "Machine learning methods for demand estimation", *American Economic Review*, v. 105, n. 5, pp. 481–85, 2015.

[9] RUSSELL, S., NORVIG, P. *Artificial Intelligence: A Modern Approach.* 3rd ed. USA, Prentice Hall Press, 2009. ISBN: 0136042597.

[10] MCCORDUCK, P. *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*. AK Peters Ltd, 2004. ISBN: 1568812051.

[11] KURZWEIL, R. *The Singularity is near: When Humans Transcend Biology*. New York, Viking, 2005. ISBN: 978-0-670-03384-3.

[12] BOTTOU, L., BOUSQUET, O. "The tradeoffs of large scale learning". In: *Advances in neural information processing systems*, pp. 161–168, 2008.

[13] PAPADIMITRIOU, C. H. "Computational Complexity". In: *Encyclopedia of Computer Science*, p. 260–265, GBR, John Wiley and Sons Ltd., 2003. ISBN: 0470864125.

[14] WERNER, S., DINIZ, P. S. R. "Set-membership affine projection algorithm", *IEEE Signal Process. Lett.*, v. 8, n. 8, pp. 231–235, Aug. 2001. ISSN: 1070-9908. doi: 10.1109/97.935739.

[15] DINIZ, P. S. R., WERNER, S. "Set-membership binormalized data-reusing LMS algorithms", *IEEE Trans. Signal Process.*, v. 51, n. 1, pp. 124–134, Jan. 2003. ISSN: 1053-587X. doi: 10.1109/TSP.2002.806562.

[16] GALDINO, J. F., APOLINÁRIO, J. A., DE CAMPOS, M. L. R. "A set-membership NLMS algorithm with time-varying error bound". In: *IEEE Int. Symposium Circuits and Syst. (ISCS*, pp. 4 pp.–280, May 2006. doi: 10.1109/ISCAS.2006.1692576.

[17] BERBERIDIS, D., KEKATOS, V., GIANNAKIS, G. B. "Online Censoring for Large-Scale Regressions with Application to Streaming Big Data", *IEEE Trans. Signal Process.*, v. 64, n. 15, pp. 3854–3867, Aug. 2016. ISSN: 1053-587X. doi: 10.1109/TSP.2016.2546225.

[18] WANG, Z., YU, Z., LING, Q., et al. "Distributed recursive least-squares with data-adaptive censoring". In: *IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, pp. 5860–5864, March 2017. doi: 10.1109/ICASSP.2017.7953280.

[19] DINIZ, P. S. R. "On Data-Selective Adaptive Filtering", *IEEE Trans. Signal Process.*, v. 66, n. 16, pp. 4239–4252, Aug. 2018. ISSN: 1053-587X. doi: 10.1109/TSP.2018.2847657.

[20] TSINOS, C. G., DINIZ, P. S. R. "Data-Selective Lms-Newton And Lms-Quasi-Newton Algorithms". In: *ICASSP 2019 - IEEE International Conference*

on Acoustics, Speech and Signal Processing, pp. 4848–4852, Brighton, UK, May 2019.

[21] DINIZ, P. S. R., MENDONÇA, M. O. K., FERREIRA, J. O., et al. "Data-Selective Conjugate Gradient Algorithm", *Eusipco: European Signal Processing Conference*, pp. 707–711, 2018.

[22] GHADIKOLAEI, H. S., GHAUCH, H., FISCHIONE, C., et al. "Learning and Data Selection in Big Datasets", v. 97, pp. 2191–2200, 09–15 Jun 2019.

[23] COLEMAN, C., YEH, C., MUSSMANN, S., et al. "Selection Via Proxy: Efficient Data Selection For Deep Learning", *CoRR*, v. abs/1906.11829, 2019. Available at `http://arxiv.org/abs/1906.11829`.

[24] GOODFELLOW, I., BENGIO, Y., COURVILLE, A. *Deep Learning*. The MIT Press, 2016.

[25] HAYKIN, S. *Adaptive Filter Theory*. 4th ed. Upper Saddle River, NJ, Prentice Hall, 2002.

[26] SAYED, A. *Fundamentals of Adaptive Filtering*. 1st ed. Hoboken, NJ, Wiley, 2003.

[27] LIU, W., PRINCIPE, J. C., HAYKIN, S. *Kernel Adaptive Filtering: A Comprehensive Introduction*. 1st ed. Hoboken, NJ, Wiley Publishing, 2010.

[28] THEODORIDIS, S. *Machine Learning: A Bayesian and Optimization Perspective*. 1st ed. Orlando, FL, USA, Academic Press, Inc., 2015.

[29] ANTONIOU, A., LU, W. S. *Practical Optimization - Algorithms and Engineering Applications*. New York, NY, Springer, 2007.

[30] FLETCHER, R. *Practical methods of optimization*. 2nd ed. Cornwall, UK, John Wiley & Sons, 2013.

[31] APOLINÁRIO, J. A., DE CAMPOS, M. L. R., BERNAL O, C. P. "The constrained conjugate gradient algorithm", *IEEE Signal Processing Letters*, v. 7, n. 12, pp. 351–354, 2000.

[32] HULL, A. W., JENKINS, W. K. "Preconditioned conjugate gradient methods for adaptive filtering". In: *Circuits and Systems, 1991., IEEE International Sympoisum on*, pp. 540–543. IEEE, 1991.

[33] CHEN, Z., LI, H., RANGASWAMY, M. "Conjugate gradient adaptive matched filter", *IEEE Transactions on Aerospace and Electronic Systems*, v. 51, n. 1, pp. 178–191, 2015.

[34] WIDROW, B., MCCOOL, J. M., LARIMORE, M. G., et al. "Stationary and nonstationary learning characteristics of the LMS adaptive filters", *Proceedings of the IEEE*, v. 64, pp. 1151–1162, Aug 1976.

[35] WIDROW, B., WALACH, E. "On the statistical efficiency of the LMS algorithm with nonstationary inputs", *IEEE Trans. Information Theory*, v. 30, n. 2, pp. 211–221, Mar 1984. doi: 10.1109/TIT.1984.1056892.

[36] SLOCK, D. "On the Convergence Behavior of the LMS and the Normalized LMS Algorithms", *Trans. Sig. Proc.*, v. 41, n. 9, pp. 2811–2825, Sep 1993. doi: 10.1109/78.236504.

[37] ELEFTHERIOU, E., FALCONER, D. D. "Tracking properties and steady-state performance of RLS adaptive filter algorithms", *IEEE Trans. Acoustics, Speech, and Signal Processing*, v. 34, n. 5, pp. 1097–1110, Mar 1986. doi: 10.1109/TASSP.1986.1164950.

[38] SLOCK, D. T. M., KAILATH, T. "Numerically stable fast transversal filters for recursive least squares adaptive filtering", *IEEE Trans. Signal Processing*, v. 39, n. 1, pp. 92–114, Jan 1991. doi: 10.1109/78.80769.

[39] BORAY, G. K., SRINATH, M. D. "Conjugate gradient techniques for adaptive filtering", *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, v. 39, n. 1, pp. 1–10, January 1992.

[40] CHANG, P. S., WILLSON JR, A. N. "Analysis of conjugate gradient algorithms for adaptive filtering", *IEEE Transactions on Signal Processing*, v. 48, n. 2, pp. 409–418, Feb 2000.

[41] J. A. APOLINÁRIO, S. W., DINIZ, P. S. R. "Conjugate Gradient Algorithm with Data Selective Updating", 2001.

[42] MENDONCA, M. O. K., FERREIRA, J. O., TSINOS, C. G., et al. "On Fast Converging Data-Selective Adaptive Filtering", *Algorithms*, v. 12, n. 1, pp. 4, Jan 2019. doi: 10.3390/a12010004.

[43] AL-BAALI, M. "Descent Property and Global Convergence of the Fletcher–Reeves Method with Inexact Line Search", *IMA Journal of Numerical Analysis*, v. 5, Jan 1985. doi: 10.1093/imanum/5.1.121.

[44] PAPOULIS, A., PILLAI, S. U. *Probability, random variables, and stochastic processes*. Tata McGraw-Hill Education, 2002.

[45] GRAFAREND, E. *Linear and Nonlinear Models: Fixed Effects, Random Effects, and Mixed Models*. Springer, Jan 2006.

[46] FERREIRA, G. J. "Code Repository for Dissertation in GitHub". https://github.com/Jonathasof/Dissertation. Accessed: 2019-12.

[47] GOOGLE. "RE<C: Surface level wind data collection, Google Code, [Online] ". http://code.google.com/p/google-rec-csp/. Accessed: 2018-11.

[48] SPIB. "Signal Processing Information Base, [Online]". http://spib.linse.ufsc.br/microwave.html. Accessed: 2018-11.

[49] SCHOLKOPF, B., SMOLA, A. J. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA, MIT Press, 2001. ISBN: 0262194759.

[50] MURPHY, K. P. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN: 0262018020, 9780262018029.

[51] ZHANG, M., WANG, X., CHEN, X., et al. "The Kernel Conjugate Gradient Algorithms", *IEEE Transactions on Signal Processing*, v. 66, pp. 4377–4387, Jun 2018. doi: 10.1109/TSP.2018.2853109.

[52] RATLIFF, N. D., BAGNELL, J. A. "Kernel Conjugate Gradient for Fast Kernel Machines". In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pp. 1017–1022, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[53] BLANCHARD, G., KRÄMER, N. "Optimal Learning Rates for Kernel Conjugate Gradient Regression". In: *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 1*, NIPS'10, pp. 226–234, Vancouver, Canada, 2010. Curran Associates Inc.

[54] LIN, J., CEVHER, V. "Kernel Conjugate Gradient Methods with Random Projections." *CoRR*, v. abs/1811.01760, 2018.

[55] STONE, M. H. "Linear Transformations in Hilbert Space and their Applications to Analysis", *COLLOQUIUM PUBLICATIONS (AMER MATHEMATICAL SOC) (Book 15)*, v. 40, pp. A25–A26, Dec 1932. ISSN: 1436-5081.

[56] SANSONE, G. *Orthogonal Functions*. Dover Books on Mathematics. Dover, 1959. ISBN: 9780486667300.

[57] LIU, W., POKHAREL, P., PRINCIPE, J. "The Kernel Least-Mean-Square Algorithm", *Trans. Sig. Proc.*, v. 56, n. 2, pp. 543–554, fev. 2008. ISSN: 1053-587X. doi: 10.1109/TSP.2007.907881.

[58] PARREIRA, W., CARLOS M. BERMUDEZ, J., RICHARD, C., et al. "Stochastic Behavior Analysis of the Gaussian Kernel Least-Mean-Square Algorithm", *IEEE Transactions on Signal Processing - TSP*, v. 60, pp. 4116 – 4119, Jun 2011. doi: 10.1109/ICASSP.2011.5947258.

[59] ENGEL, Y., MANNOR, S., MEIR, R. "The Kernel Recursive Least-squares Algorithm", *Trans. Sig. Proc.*, v. 52, n. 8, pp. 2275–2285, ago. 2004. ISSN: 1053-587X. doi: 10.1109/TSP.2004.830985.

[60] LIU, W., PARK, I., WANG, Y., et al. "Extended Kernel Recursive Least Squares Algorithm", *Trans. Sig. Proc.*, v. 57, n. 10, pp. 3801–3814, out. 2009. ISSN: 1053-587X. doi: 10.1109/TSP.2009.2022007.

[61] VAN VAERENBERGH, S., VIA, J., SANTAMANA, I. "A Sliding-Window Kernel RLS Algorithm and Its Application to Nonlinear Channel Identification", *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on*, v. 5, pp. V – V, Jun 2006. doi: 10.1109/ICASSP.2006.1661394.

[62] RICHARD, C., BERMUDEZ, J., HONEINE, P. "Online Prediction of Time Series Data With Kernels", *Signal Processing, IEEE Transactions on*, v. 57, pp. 1058 – 1067, 04 2009. doi: 10.1109/TSP.2008.2009895.

[63] ARONSZAJN, N. "Theory of Reproducing Kernels", *Transactions of the American Mathematical Society*, v. 68, pp. 337–404, Mar 1950.

[64] BURGES, C. J. C. "A tutorial on support vector machines for pattern recognition", *Data Mining and Knowledge Discovery*, v. 2, pp. 121–167, 1998.

[65] SCHÖLKOPF, B., HERBRICH, R., SMOLA, A. J. *A Generalized Representer Theorem*. COLT '01/EuroCOLT '01. London, UK, UK, Springer-Verlag, 2001. ISBN: 3-540-42343-5.

[66] PAIGE, C. C., SAUNDERS, M. A. "LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares", *ACM Trans. Math. Softw.*, v. 8, n. 1, pp. 43–71, Mar 1982. ISSN: 0098-3500. doi: 10.1145/355984.355989.

[67] FLETCHER, R. *Practical Methods of Optimization; (2Nd Ed.)*. New York, NY, USA, Wiley-Interscience, 1987. ISBN: 0-471-91547-5.

[68] NOCEDAL, J., J. WRIGHT, S. *Numerical Optimization*. Springer Series, 01 2006.

[69] RICHARD, C., BERMUDEZ, J. C. M., HONEINE, P. "Online prediction of time series data with kernels", *IEEE Transactions on Signal Processing*, v. 57, n. 3, pp. 1058–1067, 2008.

[70] SMALE, S., ZHOU, D.-X. "Estimating the approximation error in learning theory", *Analysis and Applications*, v. 1, n. 01, pp. 17–41, 2003.

[71] CHEN, B., ZHAO, S., ZHU, P., et al. "Quantized kernel least mean square algorithm", *IEEE Transactions on Neural Networks and Learning Systems*, v. 23, n. 1, pp. 22–32, 2011.

[72] HONEINE, P. "Approximation errors of online sparcification criteria", *IEEE Trans. Signal Process*, v. 63, 2015.

[73] MCCULLOCH, W. S., PITTS, W. "A logical calculus of the ideas immanent in nervous activity", *The bulletin of mathematical biophysics*, v. 5, n. 4, pp. 115–133, 1943.

[74] MINSKY, M., PAPERT, S. *Perceptrons*. Cambridge, MA, MIT Press, 1969.

[75] WEST, D. "Neural network credit scoring models", *Computers & Operations Research*, v. 27, n. 11-12, pp. 1131–1152, 2000.

[76] NOVIKOFF, A. B. *On convergence proofs for perceptrons*. Relatório técnico, STANFORD RESEARCH INST MENLO PARK CA, 1963.

[77] GLOROT, X., BORDES, A., BENGIO, Y. "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, 2011.

[78] LECUN, Y., BOTTOU, L., ORR, G. B., et al. "Efficient BackProp." In: Montavon, G., Orr, G. B., Müller, K.-R. (Eds.), *Neural Networks: Tricks of the Trade (2nd ed.)*, v. 7700, *Lecture Notes in Computer Science*, Springer, pp. 9–48, 2012. ISBN: 978-3-642-35288-1.

[79] RAMACHANDRAN, P., ZOPH, B., LE, Q. V. "Searching for Activation Functions", *ArXiv*, v. abs/1710.05941, 2017.

[80] SCHMIDHUBER, J. "Deep learning in neural networks: An overview", *Neural networks*, v. 61, pp. 85–117, 2015.

[81] HECHT-NIELSEN, R. "Theory of the backpropagation neural network". In: *Neural networks for perception*, Elsevier, pp. 65–93, 1992.

[82] RUDER, S. "An overview of gradient descent optimization algorithms", *ArXiv*, v. abs/1609.04747, 2016.

[83] C. DUCHI, J., HAZAN, E., SINGER, Y. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", *Journal of Machine Learning Research*, v. 12, pp. 2121–2159, Jul 2011.

[84] ZEILER, M. D. "ADADELTA: an adaptive learning rate method", *arXiv preprint arXiv:1212.5701*, 2012.

[85] KINGMA, D., BA, J. "Adam: A Method for Stochastic Optimization", *International Conference on Learning Representations*, Dec 2014.

[86] BLUM, A. L., LANGLEY, P. "Selection of relevant features and examples in machine learning", *Artificial intelligence*, v. 97, n. 1-2, pp. 245–271, 1997.

[87] MACKAY, D. J. "Information-based objective functions for active data selection", *Neural computation*, v. 4, n. 4, pp. 590–604, Jul 1992.

[88] COHN, D. A., GHAHRAMANI, Z., JORDAN, M. I. "Active learning with statistical models", *Journal of artificial intelligence research*, v. 4, pp. 129–145, Mar 1996.

[89] DHEERU, D., KARRA TANISKIDOU, E. "UCI Machine Learning Repository". 2017. Available at `http://archive.ics.uci.edu/ml`.

[90] HAMIDIEH, K. "A data-driven statistical model for predicting the critical temperature of a superconductor", *Computational Materials Science*, v. 154, pp. 346–354, 2018.

[91] MASHABLE. "Site Mashable". https://mashable.com/. Accessed: 2019-09.

[92] FERNANDES, K., VINAGRE, P., CORTEZ, P. "A proactive intelligent decision support system for predicting the popularity of online news". In: *Portuguese Conference on Artificial Intelligence*, pp. 535–546. Springer, 2015.

[93] SINGH, K., SANDHU, R. K., KUMAR, D. "Comment Volume Prediction Using Neural Networks and Decision Trees". In: *IEEE UKSim-AMSS*

17th International Conference on Computer Modelling and Simulation, UKSim2015 (UKSim2015), Cambridge, United Kingdom, mar 2015.

[94] KAGGLE. "FIFA 19 Complete Player dataset". https://www.kaggle.com/karangadiya/fifa19. Accessed: 2019-09.

[95] YANN LECUN, CORINNA CORTES, C. J. B. "MNIST dataset of handwritten digit". http://yann.lecun.com/exdb/mnist/. Accessed: 2019-09.

[96] OF STANDARDS, N. I., (NIST), T. "Emnist Letters dataset". https://www.nist.gov/node/1298471/emnist-dataset. Accessed: 2019-09.

[97] COHEN, G., AFSHAR, S., TAPSON, J., et al. "EMNIST: an extension of MNIST to handwritten letters", *arXiv preprint arXiv:1702.05373*, 2017.

[98] SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *J. Mach. Learn. Res.*, v. 15, n. 1, pp. 1929–1958, jan. 2014. ISSN: 1532-4435. Available at http://dl.acm.org/citation.cfm?id=2627435.2670313.

[99] GOLUB, G. H., VAN LOAN, C. F. *Matrix Computations*. The Johns Hopkins University Press, 1996.

Appendix A

Appendix

A.1 Dropout

Dropout is a stochastic technique to avoid the overfitting during the training and provides a way of combining exponentially many different neural network architectures. The idea is based on dropping out nodes in the network, including all its connections as it is showed in Figure A.1. In the standard case, the node is retained with probability p (Bernoulli Distribution) independent of other nodes, that is, their values are temporarily set to zero in each iteration. This process is applied at each layer. Furthermore, the derivatives of the loss function are backpropagated through the network.

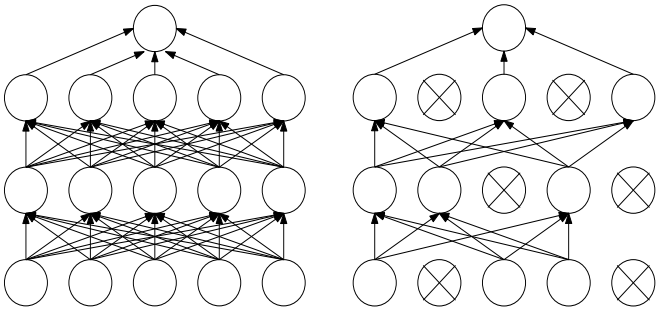


Figure A.1: (a) The neural network with two hidden layers and (b) Dropout applied in the network producing a thinned net

At test procedure, it is unfeasible to average all combinations of many thinned models. The solution is to use neural network without dropout but with a reduced version of the trained weights. If a node is dropped with probability p during training, the weights are multiplied by p at test as shown in Figure A.2.

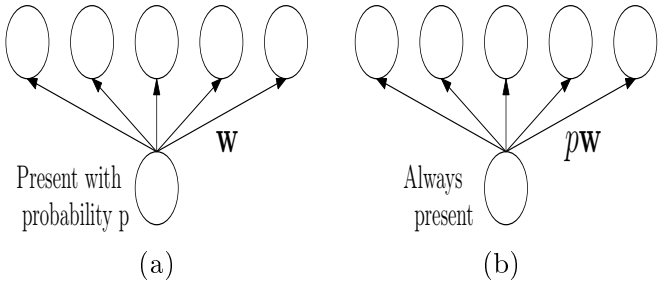


Figure A.2: (a) The training node present in process with probability p and (b) The test phase with node always present and weight multiplied by p.

A.2 Number of Flops

A flop is a floating point operation [99]. There are several difficulties associated with calculating a precise count of floating points operations. The addition and subtraction is always counted as a flop. The multiplication is usually counted as a flop. The division and exponential are more complicated because we are instructed to consider each one as a flop, but in some cases the values is defined as 4 flops for division and 8 for exponential (HPC community). In this text, we consider all operations as a flop, including division and exponential.

We compute the number of flops in two steps, for forward and back-propgation algorithms. As example, we compute the number of flops for a Neural Network with 4 layers (two hidden layers), where i denotes the number of nodes of the input layer, j the number of nodes in the second layer, k the number of nodes in the third layer and l the number of nodes in the output layer. The parameters are denoted as n_{ep} (epoch), b (mini batch size), *iter* (iteration), P_{up} (probability of update).

At each iteration for a certain epoch, we have b training examples. In the forward process, the following operations are performed from the layer to the next layer

$$\begin{aligned} \mathbf{X}^l &= (\mathbf{W}^l)^T \mathbf{Y}^{l-1}, \\ \mathbf{Y}^l &= [ones(1, b); f(\mathbf{X}^l)]. \end{aligned} \tag{A.1}$$

From the first layer to the second layer, the number of flops for a first operation, product between two matrices $\mathbf{W}^1 \in \mathbb{R}^{i \times j}$ and $\mathbf{Y}^0 \in \mathbb{R}^{i \times b}$, consist of jb inner products between vectors of size i. This inner product involves $i-1$ additions and i multiplications. Then, the resulting number of flops is $(2i-1)jb$. The activation function (ReLU) applied in second equation has 0 flops. Then, we have $(2i-1)jb$ flops.

Using the same logic from the second layer to third layer, we have $(2j-1)kb$ flops. While in the propagation from the third layer to last layer, the matrix multiplication results in $(2k-1)lb$ flops. In the last part, we obtain the estimated value in NN. In regression problem the output activation function is linear resulting in 0 flops. In

classification problem, the softmax funtion has $l-1$ additions, 1 division and $l+1$ exponential for b training examples, resulting a total of $(2l+1)b$.

Therefore, the total number of flops for feedforward propagation depends on the problem, for regression problem it is $(2ij+2jk+2kl-j-k-l)b$ flops and for classification problem it is $(2ij+2jk+2kl+l+1)b$ flops.

In the back-propagation, the data selection is considered in the procedure of counting the flops. To not include the proposed method in process, simply set probability of update $P_{\text{up}}=1$. Starting from the last layer to third layer, we compute the sensitivity vector in forth layer. To obtain $\boldsymbol{\Delta}^3=\hat{\mathbf{Y}}_{(t,i)}-\mathbf{Y}^L$, it is required $l(P_{\text{up}}b)$ flops. In the remainder layers, we compute the sensitive weights from the vectors previously obtained. For example, from the third layer to second layer, we have $\boldsymbol{\Delta}^2=f'(\mathbf{X}^2)\otimes[\mathbf{W}^3\boldsymbol{\Delta}^3]_1^{o^2}$. The number of flops in the derivative of activation function is zero. The multiplication matrix $\mathbf{W}^3\boldsymbol{\Delta}^3$ results in $k(P_{\text{up}}b)$ inner products that involves $l-1$ additions and l multiplications. The element-wise operation has $k(P_{\text{up}}b)$ flops. Then, the resulting number of flops is $2lk(P_{\text{up}}b)$ in this propagation. Finally, from second to first layer, we have the same procedure and the number of flops required is $2kj(P_{\text{up}}b)$.

The last step is the weight updating, for example from forth to third layer, we have the update $\mathbf{W}^3=\mathbf{W}^3-\frac{\mu}{b}\mathbf{Y}^2(\boldsymbol{\Delta}^3)^T$. The multiplication matrix results in a total of $(2(P_{\text{up}}b)-1)kl$ flops, beyond the sum of the matrices, with a number of flops equal to kl. By summing the multiplication and addition, we end-up with $2(P_{\text{up}}b)kl$ flops. The same process is performed in third to second layer, $2(P_{\text{up}}b)jk$, and second to first layer, $2(P_{\text{up}}b)ij$. Then, the total number of flops in back-propagation is

$$\begin{aligned}&2(P_{\text{up}}b)ij+2(P_{\text{up}}b)jk+2(P_{\text{up}}b)kl+2kj(P_{\text{up}}b)+2kl(P_{\text{up}}b)+l(P_{\text{up}}b)\\&=(P_{\text{up}}b)(2ij+2jk+2kl+2kj+2kl+l).\end{aligned}\tag{A.2}$$

In the back-propagation, the number of flops is a function of the probability of update, reducing the computational cost.