KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Deep Contractive Least Squares Support Vector Machines for associative memory

Octave Oliviers

Thesis submitted for the degree of
Master of Science in
Mathematical Engineering

Thesis supervisor:
Prof. dr. ir. Johan Suykens

Academic year 2019 - 2020

Master of Science in
Mathematical Engineering

KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Deep Contractive Least Squares Support Vector Machines for associative memory

Octave Oliviers

Thesis submitted for the degree of
Master of Science in
Mathematical Engineering

Thesis supervisor:
Prof. dr. ir. Johan Suykens

Assessors:
Prof. dr. ir. Hugo Van Hamme
Prof. dr. ir. Raf Vandebril

Mentors:
Ir. Henri De Plaen
Ir. Arun Pandey

Academic year 2019 - 2020

Preface

This thesis hopes to contribute to the broader research on adaptive AI, namely an AI that can adapt its behaviour as its environment changes. Think for instance, of a surgical robot that adapts to a patient's record, or an autonomous car that becomes *cautious* as soon as it starts raining.

For an AI to be able to adapt its behaviour, it should have memory. Therefore, I focused this thesis on developing an effective memory model by finding synergies between (a priori unrelated) deep learning models. The first semester involved a lot of reading to develop our memory model. The second semester was all about theoretically and empirically studying it.

Of course, this thesis is much more than a one-year individual effort. Therefore, I want to thank everyone who supported me along the way.

First of all, I thank professor Johan Suykens for the interest he showed in my research proposal about "neural networks that memorize things". I thank him for helping me turn this vague idea into a valuable research, for his accessibility this year, and for his sharp comments on my work. His help and expertise contributed a great deal to the work in this thesis.

Furthermore, I thank Arun and Henri for always being very encouraging, and for giving detailed and critical feedback when I presented new results. Our interesting discussions generated the ideas at the heart of this thesis.

Also, I thank everyone who answers questions on the mathematics' fora. Keep up with the good work, you are one of the pillars of today's research.

On a more personal note, I thank the Gianni's who made the last five years so entertaining. I look forward to our second Reis rond de Wereld, which will undoubtedly be preceded by a Tour de France as a warm-up. I thank Matthieu, Marie-Françoise, Gaspard, Cassiopée, and Constance for inspiring me everyday with their generosity and drive for excellence.

Octave Oliviers
Leuven, June 2020

Abstract

Despite the recent successes of artificial intelligence (AI) on diverse tasks, most of today's models still lack a key feature of human intelligence. Namely, they are trained to perform one particular task extremely well, but cannot spontaneously adapt themselves as their environment changes.

One way to implement adaptive behaviour is through associative memory, i.e. the model memorizes suitable behaviours for different environments. The model can then spontaneously adapt to its changing environment, by choosing the appropriate behaviour associated with that environment.

We examine a model for associative memory, called a Hopfield network. A Hopfield network stores memories as attractors of a dynamical system. Consequently, as the state of the dynamical system evolves step by step, it progressively converges to an attractor, and hence recalls a memory.

We improve Hopfield networks by combining two deep learning models: (1) Least Squares Support Vector Machines (LS-SVMs), which are a type of Support Vector Machine that uses a least squares loss function; and (2) contractive autoencoders (C-AEs), which are a type of autoencoder that uses contractive regularization, namely which induces contraction.

We call our model a Contractive Least Squares Support Vector Machine. This new model, abbreviated C-LS-SVM, relates to three existing models. Therefore, it contributes to the existing research on each of these models.

For the field of Hopfield networks, a C-LS-SVM is a new memory model whose memory capacity is independent of the dimension of the memories. It has a significantly larger memory capacity, and can store movements.

For the field of LS-SVMs, a C-LS-SVM is a new framework for modelling a discrete dynamical system from a set of desired locally stable equilibria. It can be stacked to obtain deep models, and has generative properties.

For the field of C-AEs, a C-LS-SVM is a new autoencoder architecture whose shallow version has a convex training process, and its deep version an alternating convex process with no vanishing or exploding gradients.

Overall, C-LS-SVMs suggest new synergies between Hopfield networks, Least Squares Support Vector Machines and contractive autoencoders.

Samenvatting

Ondanks de recente successen van artificiële intelligentie op diverse taken, missen veel modellen een belangrijk kenmerk van menselijke intelligentie. Ze zijn namelijk getraind om een bepaalde taak zeer goed uit te voeren, maar kunnen zich niet spontaan aanpassen als hun omgeving verandert.

Een manier om adaptief gedrag te bekomen is met associatief geheugen. Hierbij onthoudt het model een geschikt gedrag voor allerlei omgevingen. Het kan zich dan spontaan aanpassen aan zijn veranderende omgeving, door steeds het juiste gedrag te selecteren dat bij zijn omgeving hoort.

We onderzoeken een model voor associatief geheugen: Hopfield netwerken. Deze slaan herinneringen op als attractoren van een dynamisch systeem. Bijgevolg, naarmate de toestand van het dynamische systeem evolueert, convergeert het geleidelijk naar een attractor, en roept zo een herinnering op.

We verbeteren dit model door twee deep learning modellen te combineren: (1) Least Squares Support Vector Machines (LS-SVM's), een type Support Vector Machine dat gebruik maakt van een kleinste-kwadraten criterium; (2) contractieve autoencoders (C-AE's), een type autoencoder dat gebruik maakt van contractieve regularisatie, namelijk die contractie veroorzaakt.

Ons model heet een Contractive Least Squares Support Vector Machine, afgekort als C-LS-SVM. Het heeft betrekking op drie bestaande modellen. Daarom draagt het ook bij aan het onderzoek rond elk van deze modellen.

Voor Hopfield netwerken vormt een C-LS-SVM een nieuw geheugenmodel waarvan de geheugencapaciteit onafhankelijk is van de herinneringen. Het haalt zo een grotere geheugencapaciteit en kan bewegingen opslaan.

Voor LS-SVM's vormt een C-LS-SVM een nieuw framework om een dynamisch systeem te modeleren met gewenste lokaal stabiele evenwichten. Het kan worden gestapeld tot diepe modellen en werkt ook generatief.

Voor C-AE's vormt een C-LS-SVM een nieuwe autoencoderarchitectuur wiens ondiepe versie een convex trainingsproces heeft en de diepe versie een afwisselend convex zonder verdwijnende of exploderende gradiënten.

C-LS-SVM's suggereren dus nieuwe synergieën tussen Hopfield-netwerken, Least Squares Support Vector Machines en contractieve autoencoders.

Contents

Notation

Numbers and arrays

x	A scalar
$\mathbf{x}$	A column vector
$\mathbf{X}$	A matrix
$\boldsymbol{\mathbf{X}}$	A cell containing several matrices of type $\mathbf{X}$
$\mathbf{0}$	A column vector of zeros
$\mathbf{1}$	A column vector of ones
$\mathbf{I}$	An identity matrix

Linear Algebra Operators

$\mathbf{X}^\top$	The transpose of matrix $\mathbf{X}$
$\mathbf{X}^+$	The left pseudo-inverse of matrix $\mathbf{X}$, i.e. $(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top$
$\mathrm{tr}(\mathbf{X})$	The trace of matrix $\mathbf{X}$

Indexing

x^i	The i^{th} power of scalar x
x_i	The i^{th} element of vector $\mathbf{x}$, with indexing starting at 1
$x_{i,j}$	The $(i,j)^{\text{th}}$ element of matrix $\mathbf{X}$
$\mathbf{x}_i$	The i^{th} column of matrix $\mathbf{X}$
$\mathbf{x}^{(k)}$	The vector $\mathbf{x}$ in the k^{th} discrete time step
$\mathbf{x}_{(l)}$	The vector $\mathbf{x}$ in the l^{th} layer
$\mathbf{0}_N$	The vector of zeros of length N
$\mathbf{0}_{M\times N}$	The matrix of zeros of size M by N
$\mathbf{I}_N$	The identity matrix of size N

Sets and Spaces

$\{ \mathbf{x}_p \}_{p=1}^{P}$	The set containing P vectors $\mathbf{x}_p$
$[a,\ b]$	The real interval between a and b
$\{ a,\ b \}$	The set containing a and b
$\mathbb{R}$	The space of real scalars
$\mathbb{R}^+$	The space of real non-negative scalars
$\mathbb{R}^N$	The space of real column vectors of length N
$\mathbb{R}^{M \times N}$	The space of real matrices of size M by N
$\mathbb{R}^M \times \mathbb{R}^N$	The space of couples of real vectors of length M and N

Functions

$f(\cdot) : \mathbb{A} \to \mathbb{B}$	The function $f(\cdot)$ from domain $\mathbb{A}$ to range $\mathbb{B}$
$\sigma_n(\mathbf{X})$	The n^{th} largest singular value of matrix $\mathbf{X}$
$\| \mathbf{x} \|_2$	The 2-norm of vector $\mathbf{x}$
$\| \mathbf{X} \|_{\text{F}}$	The Frobenius norm of matrix $\mathbf{X}$

Calculus

$\dfrac{\mathrm{d}y}{\mathrm{d}x}$	The derivative of y with respect to x
$\dfrac{\partial y}{\partial x}$	The partial derivative of y with respect to x
$\dfrac{\partial f}{\partial \mathbf{x}}$	The Jacobian of function $f(\cdot)$ with respect to $\mathbf{x}$
$\dfrac{\partial^2 f}{\partial \mathbf{x} \partial \mathbf{y}}$	The second order partial derivative of function $f(\cdot)$ with respect to $\mathbf{x}$ and $\mathbf{y}$

Probability and Information Theory

$P(A \mid B)$	The probability of A given B
$\mathbb{E}[x]$	The expected value of x over the related input space
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	The normal distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

List of Abbreviations

AE	Autoencoder
AI	Artificial Intelligence
AM	Associative Memory
C-AE	Contractive Autoencoder
C-LS-SVM	Contractive Least Squares Support Vector Machine
DMS	Direct Multiple Shooting method
DNN	Deep Neural Network
KAM	Kernel Associative Memory
LS-SVM	Least Squares Support Vector Machine
MAC	Method of Auxiliary Coordinates
NN	(Artificial) Neural Network
SVD	Singular Value Decomposition
SVM	Support Vector Machine

Chapter 1

Introduction

1.1 Motivation for modelling memory

Artificial Intelligence (AI) models outperform experts on diverse tasks, for instance, for diagnosing cancer [51], spotting contractual issues [57], constructing game strategies [88], or developing fairer tax policies [111]. However, it is still a long way before Artificial General Intelligence [38], i.e. an AI that can understand or learn any task at the level of a human.

One of the keystones that today's models still miss is adaptive behaviour. Namely, they are trained to perform one particular task extremely well, but cannot spontaneously adapt if their environment suddenly changes. This is one of the principal reasons why there are still so few AI models in unpredictable environments, such as surgical rooms or the roadways.

For an AI to be able to adapt its behaviour, it should have memory, because memory enables generalizing to unknown environments [103], learning from experiences [77], or predicting from incomplete data [11]. Thus, an AI with memory integrates faster new data after training [85], can reach better results with less data [30], and can do multiple tasks [8].

This thesis hopes to contribute to the research on adaptive AI models, by developing a way to effectively implement memory in an AI model. In the long run, we envision a model that can generalize his knowledge to new environments, and thereby spontaneously switch between tasks, e.g. a robot that performs surgeries tailored to the patient's medical record.

1.2 Computational principles of memory

This section outlines the two fundamental principles of biological memory, and then presents a method to model memory with a dynamical system.

1.2.1 The two principles of biological memory

Chaudhuri and Fiete define memory as "a persistent change in activity or connectivity of a neural network, triggered by stimuli or brain states". This definition highlights the two fundamental principles of memory [18].

1. Memory is a persistent change

The duration of the change indicates if the memory is short- or long-term.

Short-term memory typically persists for several seconds or minutes. Therefore, it consists of a brief change in the *activity* of the network [24].

Long-term memory typically persists for several years or even decades. Since the cells that make up the brain have a lifespan of several days, long-term memory needs changes in the *connectivity* of the network [99].

The distinction between short-term and long-term is not clear-cut [18]. Even so, both types of memory involve a change that persists over time.

2. Memory is triggered by an event

The event's type indicates if the memory is auto- or hetero-associative.

Auto-associative memory is the ability to recall a particular memory when the triggering event is an incomplete version of that memory [44]. Thus, triggered by a noisy memory $\tilde{\mathbf{x}}_p$, we recall the original memory $\mathbf{x}_p$.

Hetero-associative memory is the ability to recall a particular memory when the triggering event is not an incomplete version of that memory [5]. Thus, triggered by a noisy memory $\tilde{\mathbf{x}}_p$, we recall the associated memory $\mathbf{y}_p$.

In this thesis, we consider long-term, auto-associative memory models. We study how to change the connectivity of a neural network so that, when triggered with a partial memory, it reconstructs the original one.

1.2.2 Modelling auto-associative memory with a dynamical system

The definition of auto-associative memory suggests that we can model it with a neural network that changes over time due to triggering events.

Principle

Let us represent the state of the neural network with a vector $\mathbf{x} \in \mathbb{R}^N$, and model the change of this state with an function $f(\cdot) : \mathbb{R}^N \to \mathbb{R}^N$. Thus, the neural network is described by the discrete dynamical system:

$$\mathbf{x}^{(k+1)} = f(\mathbf{x}^{(k)}).$$

The objective is then to learn the parameters of $f(\cdot)$ that guarantee that the memories coincide with the stable equilibria of the dynamical system. This way, as the state $\mathbf{x}$ evolves, it progressively converges to a memory.

Such a recurrent neural network is referred to as an *attractor network*, because the stable equilibria, i.e. the memories, *attract* the state [3, 81].

Example

Suppose we want a neural network to memorize images of digits so that, when presented with a noisy digit, it still recognizes the original one. This is a common task, for instance, in post offices or autonomous cars.

The state $\mathbf{x}$ is thus an image, and the state space resembles figure 1.1. We see that each initial condition that corresponds with a noisy digit, evolves to the equilibrium of that basin of attraction, i.e. the true digit.

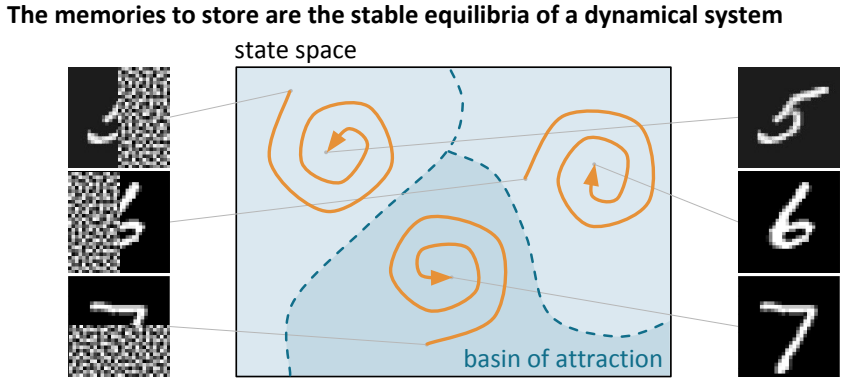


Figure 1.1: We model auto-associative memory with a dynamical system whose stable equilibria correspond with each of the images to memorize. So, when presented with a noisy image, it converges to the original one.

1.3 Previous work

Over the years, a wide range of memory models have been introduced. The first models, such as Associative Nets [108], Hopfield networks [45], or Sparse Distributed Memory [47], could only store binary memories. Still, they inspired more powerful ones, such as deep belief networks [41], or deep Boltzmann machines [1, 83], which can store more complex data. Recently, Bartunov *et al.* even developed an energy-based framework such that any neural network architecture could now model memory [7].

In this thesis, we focus in particular on a model called Hopfield networks, for their mathematical simplicity, and the wealth of research they inspired. We improve these networks by combining two deep learning frameworks: Least Squares Support Vector Machines and contractive autoencoders.

1.3.1 Hopfield networks

A Hopfield network is a neural network for auto-associative memory [45]. Thus, it is a dynamical system that stores memories as stable equilibria. This network has attractive features, like being asymptotically stable. However, its memory capacity is limited by the number of neurons it has, and hence also by the dimension of the memories that it tries to store.

1.3.2 Least Squares Support Vector Machines

Least Squares Support Vector Machines (LS-SVMs) are a specific type of Support Vector Machine that uses a least squares loss function [96]. In this thesis, we apply the LS-SVM framework for function estimation.

We chose LS-SVMs firstly because their optimization problem is convex which provides an analytical formulation for the optimal parameters. Also, they can apply a mathematical trick to train a large neural network without the computational burden of manipulating such a large network.

1.3.3 Contractive autoencoders

Contractive autoencoders (C-AEs) are a specific type of autoencoder that uses contractive regularization, i.e. which induces contraction [80]. In this thesis, we apply this contractive feature to build a memory model.

We chose C-AEs firstly because their optimization problem analytically expresses the goal of a memory model: reconstruct a noisy data point. Furthermore, they have been rigorously studied over the past decade, generating interesting extensions concerning deep and generative variants.

1.4 Contributions of this thesis

This thesis develops a model at the interface of three, a priori unrelated fields in Artificial Intelligence: Hopfield networks, LS-SVMs and C-AEs. We call this model a Contractive Least Squares Support Vector Machine. It is a dynamical system that integrates the typical contraction of C-AEs into the LS-SVM framework to build a model for auto-associative memory.

The principal contributions of our work to each of the fields are as follows.

For the field of Hopfield networks, a C-LS-SVM is a new memory model whose memory capacity is independent of the dimension of the memories. It has a significantly larger memory capacity, and can store movements.

For the field of LS-SVMs, a C-LS-SVM is a new framework for modelling a discrete dynamical system from a set of desired locally stable equilibria. It can be stacked to obtain deep models, and has generative properties.

For the field of C-AEs, a C-LS-SVM is a new autoencoder architecture whose shallow version has a convex training process, and its deep version an alternating convex process with no vanishing or exploding gradients.

Overall, C-LS-SVMs suggest new synergies between Hopfield networks, Least Squares Support Vector Machines and contractive autoencoders.

1.5 Structure of this thesis

Part I. Theoretical background for our developments

Part I introduces the theoretical background for the work in this thesis. In chapter 2, we summarize the mathematics behind a Hopfield network. In chapter 3, we discuss the LS-SVM framework for function estimation. In chapter 4, we describe the functioning of a contractive autoencoder.

Part II. Our developments

Part II introduces Contractive Least Squares Support Vector Machines. In chapter 5, we construct the optimization problem to train a C-LS-SVM, and then apply the Lagrange method to derive the optimal parameters. In chapter 6, we study C-LS-SVMs from the angle of Hopfield networks. Namely, we analyse the ability of C-LS-SVMs for modelling memory. In chapter 7, we study C-LS-SVMs from the perspective of LS-SVMs. Namely, we construct the Bayesian interpretation behind C-LS-SVMs. In chapter 8, we study C-LS-SVMs from the point of view of C-AEs. Namely, we construct deep and generative variants inspired by C-AEs. In chapter 9, we apply C-LS-SVMs on a benchmark data set: MNIST.

Our software is available at `github.com/OctaveOliviers/master-thesis`

Part I

Theoretical background for our developments

Chapter 2

Hopfield networks

A Hopfield network is a neural network for auto-associative memory [45]. So, it reconstructs a memory when given a partial or noisy version of it, provided that the network had stored that particular memory beforehand.

Specifically, a Hopfield network consists of a discrete dynamical system whose stable equilibria should correspond with the memories to store. This way, when given an initial condition that represents a noisy memory, the state of the system converges to the original memory step by step. For a brief review on discrete dynamical systems, we refer to Appendix B.

In this chapter, we outline the mathematics behind a Hopfield network.

Section 2.1 gives the update equation that describes a Hopfield network.

Section 2.2 presents a learning rule for the parameters: the Hebbian rule.

Section 2.3 studies the most memories that a Hopfield network can store.

Section 2.4 states the energy function that describes a Hopfield network.

Section 2.5 reviews the strengths and weaknesses of a Hopfield network.

Section 2.6 analyses an improved learning rule: the pseudo-inverse rule.

2.1 The update equation

The dynamics of a Hopfield network are described by the update equation

$$\mathbf{x}^{(k+1)} \;=\; \text{sign}(\; \mathbf{W}^\top \mathbf{x}^{(k)} \;). \tag{2.1}$$

The weights $\mathbf{W} \in \mathbb{R}^{N \times N}$ control the strength of the neural connections.

2.2 Training the model

Consider storing P bipolar memories $\mathbf{x}_p$ represented by vectors of length N:

$$\begin{aligned} \mathbf{x}_p \;=\; [\; x_{1,p} \;\; \cdots \;\; x_{N,p} \;]^\top \quad & \text{with} \quad x_{n,p} \in \{-1, +1\} \\ & \text{for} \quad 1 \le p \le P \\ & \qquad\;\; 1 \le n \le N. \end{aligned}$$

The Hebbian learning rule to assign the elements $w_{n,n'}$ of $\mathbf{W}$ then reads: *neurons that fire together wire together, neurons that fire apart wire apart.*

$$w_{n,n'} \;=\; \frac{1}{N} \sum_{p=1}^{P} x_{n,p}\, x_{n',p} \qquad \text{for } 1 \;\le\; n, n' \;\le\; N. \tag{2.2}$$

As a result, $w_{n,n'}$ is positive if the states of neurons n and n' are correlated, i.e. $x_{n,p}$ and $x_{n',p}$ are jointly $+1$ or -1 for most of the memories $\mathbf{x}_p$, $w_{n,n'}$ is negative if the states of neurons n and n' are anti-correlated, i.e. $x_{n,p}$ and $x_{n',p}$ have opposite signs for most of the memories $\mathbf{x}_p$, and $w_{n,n'}$ is close to 0 if the states of neurons n and n' are uncorrelated [40].

$\mathbf{W}$ is thus, up to a scaling constant, the covariance matrix of the memories

$$\mathbf{W} \;=\; \frac{1}{N} \sum_{p=1}^{P} \mathbf{x}_p\, \mathbf{x}_p^\top \;=\; \frac{1}{N}\, \mathbf{X}\, \mathbf{X}^\top, \tag{2.3}$$

where the mean $\overline{\mathbf{x}}$ is assumed to be zero since the memories are random, and the matrix $\mathbf{X} = [\; \mathbf{x}_1 \;\; \cdots \;\; \mathbf{x}_P \;]$ contains the memories $\mathbf{x}_p$ in columns.

With this additional formulation for $\mathbf{W}$, the update equation rewrites to

$$\mathbf{x}^{(k+1)} \;=\; \text{sign}\Big(\; \frac{1}{N} \sum_{p=1}^{P} \mathbf{x}_p\, \mathbf{x}_p^\top\, \mathbf{x}^{(k)} \;\Big), \tag{2.4}$$

which is a weighted average of the memories, with weights $\mathbf{x}_p^\top\, \mathbf{x}^{(k)}/N$. Therefore, the more $\mathbf{x}^{(k)}$ and $\mathbf{x}_p$ overlap, the larger their inner product, the larger the weight, and the more $\mathbf{x}^{(k)}$ is attracted towards $\mathbf{x}_p$ by (2.4).

2.3 Memory capacity

The memory capacity of a network is the maximal number of memories it can store, while ensuring that each memory is a stable equilibrium [4].

A memory $\mathbf{x}_p$ is an equilibrium of the update equation (2.1) if it satisfies

$$\begin{aligned} \mathbf{x}_p &= \text{sign}(\, \mathbf{W}^\top \mathbf{x}_p \,) \\ &= \text{sign}\Big(\, \frac{1}{N} \sum_{p'=1}^{P} \mathbf{x}_{p'}\, \mathbf{x}_{p'}^\top\, \mathbf{x}_p \,\Big) \\ &= \text{sign}\Big(\, \mathbf{x}_p \,+\, \underbrace{\frac{1}{N} \sum_{\substack{p'=1 \\ p' \neq p}}^{P} \mathbf{x}_{p'}\, \mathbf{x}_{p'}^\top\, \mathbf{x}_p}_{\text{Noise}} \,\Big). \end{aligned} \tag{2.5}$$

This relation holds if the absolute value of the noise is smaller than one. However, as the number of memories P increases, so does the the noise.

It can be shown that for a number of neurons N that approaches infinity the largest P for which relation (2.5) holds is $P = N/4\log(N)$ [65, 106]. This is thus the memory capacity of a Hopfield network for perfect recall.

Else if small errors are admitted, it can store up to $P = 0.14\ N$ memories before the percentage of reconstruction errors rises sharply to 50% [4, 89]. So if $P > 0.14\ N$, the network can no longer work as associative memory.

2.4 Energy function

The energy function of a discrete dynamical system $\mathbf{x}^{(k+1)} = f(\mathbf{x}^{(k)})$ is a continuous function $E(\cdot) : \mathbb{R}^N \to \mathbb{R}$ that satisfies the following conditions:

1. $E(f(\mathbf{x})) \;\leq\; E(\mathbf{x}) \quad \forall\, \mathbf{x} \in \mathbb{R}^N$,
2. $E(\mathbf{x}^{(k)}) \;=\; 0 \quad \Longrightarrow \quad \mathbf{x}^{(k)} \;=\; f(\mathbf{x}^{(k)})$.

If such an energy function exists, the system is asymptotically stable. This means that the state $\mathbf{x}^{(k)}$ always converges to an equilibrium [20].

Hopfield demonstrated that the energy function of a Hopfield network is

$$E(\mathbf{x}) \;=\; -\frac{1}{2}\mathbf{x}^\top \mathbf{W}\mathbf{x}. \tag{2.6}$$

Consequently, the state of a Hopfield network never diverges to infinity. Instead, it always converges to a local minimum of the energy function.

2.5 Strengths and Weaknesses

Over the last forty years, research on Hopfield networks has highlighted its strengths and weaknesses. This section, presents the principal ones.

In Part II of this thesis, we will then construct a new memory model with the intention to retain these strengths, but remove these weaknesses.

2.5.1 Strengths

A Hopfield network trained with the Hebbian rule has three main strengths.

1. Local learning rule

Expression (2.2) shows that the weight $w_{n,n'}$ between neurons n and n' is computed with information from only these neurons in each memory.

Furthermore, each neuron of the network can be updated independently. Hence, there is potential for parallelizing the training and the simulation.

2. Incremental learning rule

The weight matrix $\mathbf{W}$ is computed with a sum over all the memories. Hence, a trained Hopfield network can easily store a new memory $\mathbf{x}_p$ as

$$\mathbf{W} \leftarrow \mathbf{W} + \frac{1}{N} \mathbf{x}_p \mathbf{x}_p^\top,$$

or forget an old memory $\mathbf{x}_p$ as

$$\mathbf{W} \leftarrow \mathbf{W} - \frac{1}{N} \mathbf{x}_p \mathbf{x}_p^\top.$$

Since these updates do not need to refer to previously stored memories, it is computationally light to modify the memories in a Hopfield network.

3. One-step learning rule

The weights are the solution of a one-step computation: (2.2) or (2.3).

The training process thus requires a fixed $2N^2P$ floating point operations, which contrasts with most of today's gradient-based training algorithms where learning is a limit process that gradually converges to good weights.

2.5.2 Weaknesses

A Hopfield network trained with the Hebbian rule has three main weaknesses. Figure 2.1 illustrates these three weaknesses on a one-dimensional system.

1. Spurious attractors

The sign$(\cdot)$ function in (2.1) generates a symmetric update equation. Consequently, each time the network stores a memory $\mathbf{x}_p$, it automatically also stores $-\mathbf{x}_p$, and linear combinations of the memories $\sum \beta_{p'} \mathbf{x}_{p'}$ [6]. These unwanted attractors cause noise on the reconstructed memories.

2. Limited capacity

The sign$(\cdot)$ function in (2.1) generates a quite rigid update equation. Consequently, it cannot bend to store an unlimited number of memories.

Besides, the classical Hopfield network can only store bipolar memories.

3. Abrupt breakdown

Storing additional memories once the critical capacity $0.14\,N$ is reached, abruptly destroys the stability of all the existing memories [4, 49, 90]. As a result, none of the stored memories can no longer be reconstructed.

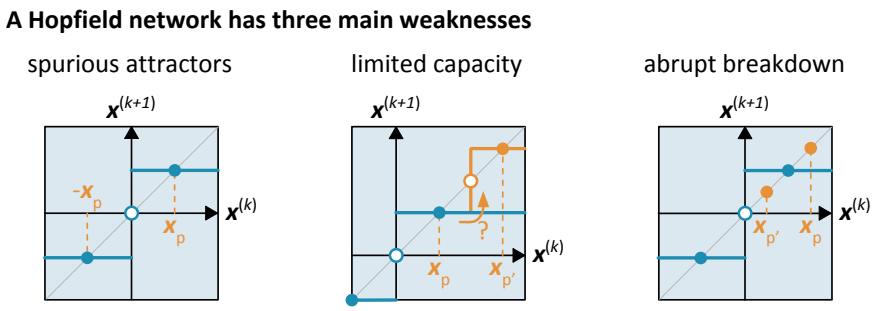


Figure 2.1: The update equation (2.1) (–) has three main weaknesses. First, its symmetric shape creates spurious attractors in the state space, which causes the reconstructed memories to be corrupted with noise. Second, it cannot bend, which limits the memory capacity of the model, and restricts the Hopfield network to storing only binary memories. Third, exceeding the capacity destroys the stability of each memory, which abruptly precludes any of the stored memories to be reconstructed.

2.6 Improvements

There exist improvements of Hopfield networks to address each weakness. For instance, to restrain the noise produced by the spurious attractors, we can use error correcting codes in the network's update equation [19], to increase the capacity, we can train the network with another rule [91], and to avoid the abrupt breakdown, we can bound the weights' size [70].

In this section, we discuss a new learning rule that has been particularly insightful for our work, namely the pseudo-inverse learning rule [48, 73].

2.6.1 The pseudo-inverse learning rule

The pseudo-inverse learning rule solves a problem of the Hebbian rule. Namely, the Hebbian rule (2.3) struggles to store similar memories, because the more similar the memories are, the larger the noise in (2.5). The attractors of the network are then noisy versions of the memories.

The pseudo-inverse learning rule solves it by computing the weights as

$$\mathbf{W} \;=\; \frac{1}{N}\,\mathbf{X}\left[\,\frac{1}{N}\,\mathbf{X}^\top\mathbf{X}\,\right]^{-1}\mathbf{X}^\top \;=\; \mathbf{X}\,\mathbf{X}^{+} \tag{2.7}$$

which guarantees that each memory $\mathbf{x}_p$ is an equilibrium as it holds that

$$\begin{aligned}\operatorname{sign}(\,\mathbf{W}^\top\mathbf{x}_p\,) \;&=\; \operatorname{sign}(\,\mathbf{X}\,\mathbf{X}^{+}\,\mathbf{x}_p\,)\\ &=\; \operatorname{sign}(\,\mathbf{X}\,\mathbf{e}_p\,)\\ &=\; \operatorname{sign}(\,\mathbf{x}_p\,)\\ &=\; \mathbf{x}_p.\end{aligned}$$

The vector $\mathbf{e}_p$ denotes the p^{th} column of the identity $\mathbf{I}_P$, since $\mathbf{X}^{+}\mathbf{X} = \mathbf{I}_P$.

The principal advantage of the pseudo-inverse learning rule (2.7) is that it raises the memory capacity of the model from $0.14\,N$ to N memories. If P is larger than N however, the matrix $\mathbf{X}^\top\mathbf{X}$ is not invertible anymore.

The principal disadvantage of the pseudo-inverse learning rule is that it is computationally more intensive to compute than the Hebbian rule, i.e. above the $2N^2P$ flops of the Hebbian rule, it needs to compute $\mathbf{X}^{+}$.

2.7 Conclusion

A Hopfield network is a neural network for auto-associative memory. Thus, it is a dynamical system that stores memories as stable equilibria. For a thorough explanation, we refer to Hopfield's original paper [45].

Chapter 3

Least Squares Support Vector Machines

Least Squares Support Vector Machines are Support Vector Machines.

Support Vector Machines (SVMs) were originally linear classifiers [102]. In the nineties, SVM's were generalized for nonlinear classification [12], for non-separable classes [25], and for different types of loss functions [97]. Around 2000, several researchers started developing SVM's for new tasks, for example, for function estimation [101], reinforcement learning [69], system identification [98], feature extraction [87], as well as clustering [9]. More recently, some deep learning frameworks also emerged [23, 93, 107].

Least Squares Support Vector Machines (LS-SVMs) are a specific type of Support Vector Machine that uses a least squares loss function [96]. As a result, the LS-SVM parameters are computed from a linear system.

In this chapter, we discuss the LS-SVM framework for function estimation. Function estimation consists of estimating a function $g(\cdot) : \mathbb{R}^N \to \mathbb{R}^M$ from a set $\{\mathbf{x}_p \in \mathbb{R}^N, \mathbf{y}_p \in \mathbb{R}^M\}_{p=1}^P$ that presumably satisfies $\mathbf{y}_p = g(\mathbf{x}_p)$.

Section 3.1 introduces the equation that LS-SVMs use to estimate $g(\cdot)$.

Section 3.2 first constructs the convex optimization to train LS-SVMs, and then applies the Lagrange method to derive the optimal parameters.

Section 3.3 presents an essential element for LS-SVMs: the kernel trick.

Section 3.4 analyses the principal strengths and weaknesses of LS-SVMs.

Section 3.5 discusses a model for associative memory based on LS-SVMs.

3.1 The model equation

LS-SVMs approximate the function $g(\cdot)$ with the parametric equation

$$f(\mathbf{x}) \;=\; \mathbf{W}^\top \varphi(\mathbf{x}) \;+\; \mathbf{b}, \tag{3.1}$$

where $\varphi(\cdot) : \mathbb{R}^N \to \mathbb{R}^{N_\mathcal{F}}$ is a nonlinear function, called the feature map, that maps the input vector $\mathbf{x}$ to a higher dimensional feature space $\mathbb{R}^{N_\mathcal{F}}$.

An LS-SVM is thus linear in its parameters $\mathbf{W} \in \mathbb{R}^{N_\mathcal{F} \times M}$ and $\mathbf{b} \in \mathbb{R}^M$. As a result, the optimization problem for finding $\mathbf{W}$ and $\mathbf{b}$ can be convex.

3.2 Training the model

Consider a data set $\{(\mathbf{x}_p \in \mathbb{R}^N,\ \mathbf{y}_p \in \mathbb{R}^M)\}_{p=1}^P$ that satisfies $\mathbf{y}_p \;=\; g(\mathbf{x}_p)$.

3.2.1 The optimization problem

Training an LS-SVM involves a tradeoff between two opposite goals: minimizing the error of the model (3.1) and minimizing its complexity.

1. Minimize the error

To prompt the model (3.1) to capture the relation in each pair $(\mathbf{x}_p, \mathbf{y}_p)$, the optimization problem minimizes the 2-norm of the following errors:

$$\| \mathbf{e}_p \|_2^2 = \| \mathbf{y}_p - \mathbf{W}^\top \varphi(\mathbf{x}_p) - \mathbf{b} \|_2^2 \qquad 1 \le p \le P. \tag{3.2}$$

2. Minimize the complexity

To make sure that the model (3.1) generalizes well on new data points, the optimization problem minimizes the Frobenius norm of the weights:

$$\| \mathbf{W} \|_\mathrm{F}^2 = \sum_{m=1}^{M} \| \mathbf{w}_m \|_2^2. \tag{3.3}$$

The intuition behind minimizing (3.3) is that if the model is too complex, it overfits, i.e. it learns characteristics that are specific for the data set but independent of the true input-output relation $g(\cdot)$, for instance noise. The model then generalizes poorly on new data as it has a different noise. Minimizing the complexity consequently minimizes the risk of overfitting.

The complete optimization problem

Altogether, we obtain the following least squares optimization problem:

$$\begin{aligned} \underset{\mathbf{w}_m,\mathbf{b},\mathbf{e}_p}{\text{minimize}} \quad & \frac{\lambda}{2}\overbrace{\sum_{p=1}^{P} \| \mathbf{e}_p \|_2^2}^{\text{Error}} + \frac{\eta}{2}\overbrace{\sum_{m=1}^{M} \| \mathbf{w}_m \|_2^2}^{\text{Complexity}} \\ \text{subject to} \quad & \mathbf{e}_p = \mathbf{y}_p - \mathbf{W}^\top \varphi(\mathbf{x}_p) - \mathbf{b} \\ & 1 \leq p \leq P. \end{aligned} \tag{3.4}$$

The hyper-parameters determine the relative importance of each term. If λ is much larger than η, (3.4) mainly minimizes the model's errors. The model then performs well on the data, but has a high risk to overfit. If η is much larger than λ, (3.4) mainly minimizes the model's complexity. The model then has a low risk to overfit, but performs badly on the data. Clearly, none of the two terms on their own would generate a good model.

3.2.2 The optimal parameters

To shorten the notation, let us collect the vectors in matrices as follows:

$$\begin{aligned} &\text{the feature matrix} && \boldsymbol{\Phi} \in \mathbb{R}^{N_\mathcal{F} \times P} && = \begin{bmatrix} \varphi(\mathbf{x}_1) & \cdots & \varphi(\mathbf{x}_P) \end{bmatrix} \\ &\text{the output matrix} && \mathbf{Y} \in \mathbb{R}^{M \times P} && = \begin{bmatrix} \mathbf{y}_1 & \cdots & \mathbf{y}_P \end{bmatrix} \\ &\text{the weight matrix} && \mathbf{W} \in \mathbb{R}^{N_\mathcal{F} \times M} && = \begin{bmatrix} \mathbf{w}_1 & \cdots & \mathbf{w}_M \end{bmatrix} \\ &\text{the bias matrix} && \mathbf{B} \in \mathbb{R}^{M \times P} && = \begin{bmatrix} \mathbf{b} & \cdots & \mathbf{b} \end{bmatrix} = \mathbf{b}\,\mathbf{1}_P^\top \\ &\text{the error matrix} && \mathbf{E} \in \mathbb{R}^{M \times P} && = \begin{bmatrix} \mathbf{e}_1 & \cdots & \mathbf{e}_P \end{bmatrix} \end{aligned}$$

The optimization problem (3.4) then rewrites in matrix-notation to

$$\begin{aligned} \underset{\mathbf{W},\mathbf{b},\mathbf{E}}{\text{minimize}} \quad & \frac{\lambda}{2}\operatorname{tr}(\mathbf{E}^\top\mathbf{E}) + \frac{\eta}{2}\operatorname{tr}(\mathbf{W}^\top\mathbf{W}) \\ \text{subject to} \quad & \mathbf{E} = \mathbf{Y} - \mathbf{W}^\top\boldsymbol{\Phi} - \mathbf{B}. \end{aligned} \tag{3.5}$$

The trace $\operatorname{tr}(\cdot)$ computes the sum of the diagonal elements of a matrix. Note that the P vector constraints are gathered in one matrix constraint.

Let us now apply the three steps of the Lagrange method to solve (3.5)[1].

[1]The following properties are used throughout this thesis: $\frac{\partial \operatorname{tr}}{\partial \mathbf{X}}(\mathbf{AXB}) = \mathbf{A}^\top\mathbf{B}^\top$, $\frac{\partial \operatorname{tr}}{\partial \mathbf{X}}(\mathbf{AX}^\top\mathbf{B}) = \mathbf{BA}$, $\frac{\partial \operatorname{tr}}{\partial \mathbf{X}}(\mathbf{AX}^\top\mathbf{X}) = \mathbf{XA}^\top + \mathbf{XA}$, $\frac{\partial \operatorname{tr}}{\partial \mathbf{X}}(\mathbf{X}^\top\mathbf{XA}) = \mathbf{XA}^\top + \mathbf{XA}$, $\frac{\partial \operatorname{tr}}{\partial \mathbf{X}}(\mathbf{AXBX}^\top\mathbf{C}) = \mathbf{A}^\top\mathbf{C}^\top\mathbf{XB}^\top + \mathbf{CAXB}$, as well as $\operatorname{tr}(\mathbf{A}+\mathbf{B}) = \operatorname{tr}(\mathbf{A}) + \operatorname{tr}(\mathbf{B})$ [74].

First, we compute the Lagrange function with dual variable $\mathbf{L} \in \mathbb{R}^{M \times P}$:

$$\begin{aligned}\mathcal{L}(\mathbf{W}, \mathbf{b}, \mathbf{E}, \mathbf{L}) \;=\; & \frac{\lambda}{2} \operatorname{tr}(\mathbf{E}^\top \mathbf{E}) \;+\; \frac{\eta}{2} \operatorname{tr}(\mathbf{W}^\top \mathbf{W}) \\ & +\; \operatorname{tr}(\mathbf{L}^\top (\mathbf{Y} - \mathbf{W}^\top \boldsymbol{\Phi} - \mathbf{B} - \mathbf{E})).\end{aligned} \tag{3.6}$$

Second, we minimize (3.6) with respect to the variables $\mathbf{W}$, $\mathbf{b}$ and $\mathbf{E}$:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} \;=\; \mathbf{0}_{N_\mathcal{F} \times M} \qquad \Longleftrightarrow \qquad \mathbf{W} \;=\; \frac{1}{\eta}\, \boldsymbol{\Phi}\, \mathbf{L}^\top \tag{3.7a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} \;=\; \mathbf{0}_M \qquad \Longleftrightarrow \qquad \mathbf{0}_M \;=\; \mathbf{L}\, \mathbf{1}_P \tag{3.7b}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{E}} \;=\; \mathbf{0}_{M \times P} \qquad \Longleftrightarrow \qquad \mathbf{E} \;=\; \frac{1}{\lambda} \mathbf{L} \tag{3.7c}$$

and obtain the dual function by filling in expressions (3.7a) and (3.7c), namely the fixed points of the Lagrange function, in the Lagrange function:

$$\begin{aligned}\mathcal{Q}(\mathbf{L}) \;=\; & \frac{\lambda}{2} \operatorname{tr}\Big(\frac{1}{\lambda^2}\, \mathbf{L}^\top \mathbf{L}\Big) \;+\; \frac{\eta}{2} \operatorname{tr}\Big(\frac{1}{\eta^2}\, \mathbf{L}\, \boldsymbol{\Phi}^\top \boldsymbol{\Phi}\, \mathbf{L}^\top\Big) \\ & +\; \operatorname{tr}\Big(\mathbf{L}^\top \Big(\mathbf{Y} - \frac{1}{\eta}\, \mathbf{L}\, \boldsymbol{\Phi}^\top \boldsymbol{\Phi} - \mathbf{B} - \frac{1}{\lambda}\, \mathbf{L}\Big)\Big).\end{aligned} \tag{3.8}$$

Third, we maximize (3.8) with respect to the variable $\mathbf{L}$, and obtain that

$$\frac{\partial \mathcal{Q}}{\partial \mathbf{L}} \;=\; \mathbf{0}_{M \times P} \qquad \Longleftrightarrow \qquad \mathbf{Y} \;=\; \frac{1}{\eta}\, \mathbf{L}\, \boldsymbol{\Phi}^\top \boldsymbol{\Phi} \;+\; \mathbf{B} \;+\; \frac{1}{\lambda}\, \mathbf{L} \tag{3.9}$$

The optimal parameters satisfy the optimality conditions (3.7) or (3.9).

Thus, to solve the optimality conditions in the primal variables $\mathbf{W}$ and $\mathbf{b}$, we replace $\mathbf{L}$ in (3.7a) and (3.7b) with $\lambda \mathbf{E}$ and obtain the linear system

$$\begin{bmatrix} \boldsymbol{\Phi}\, \boldsymbol{\Phi}^\top + \dfrac{\eta}{\lambda} \mathbf{I}_{N_\mathcal{F}} & \boldsymbol{\Phi}\, \mathbf{1}_P \\ \mathbf{1}_P^\top\, \boldsymbol{\Phi}^\top & P \end{bmatrix} \begin{bmatrix} \mathbf{W} \\ \mathbf{b}^\top \end{bmatrix} \;=\; \begin{bmatrix} \boldsymbol{\Phi}\, \mathbf{Y}^\top \\ \mathbf{1}_P^\top\, \mathbf{Y}^\top \end{bmatrix}. \tag{3.10}$$

Alternatively, to solve the optimality conditions in the dual variable $\mathbf{L}$, we couple the conditions (3.7b) and (3.9), and obtain the linear system

$$\begin{bmatrix} \dfrac{1}{\eta}\, \boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \dfrac{1}{\lambda}\, \mathbf{I}_P & \mathbf{1}_P \\ \mathbf{1}_P^\top & 0 \end{bmatrix} \begin{bmatrix} \mathbf{L}^\top \\ \mathbf{b}^\top \end{bmatrix} \;=\; \begin{bmatrix} \mathbf{Y}^\top \\ \mathbf{0}_M^\top \end{bmatrix}. \tag{3.11}$$

Expression (3.7a) implies that an LS-SVM has two equivalent formulations:

$$f(\mathbf{x}) \;=\; \mathbf{W}^\top \varphi(\mathbf{x}) \;+\; \mathbf{b}, \tag{3.12}$$

$$\;=\; \frac{1}{\eta}\, \mathbf{L} \boldsymbol{\Phi}^\top \varphi(\mathbf{x}) \;=\; \sum_{p=1}^{P} \mathbf{l}_p\, \varphi(\mathbf{x}_p)^\top \varphi(\mathbf{x}) \;+\; \mathbf{b}. \tag{3.13}$$

3.3 The kernel trick

The kernel trick is a shortcut to bypass the explicit computation of $\varphi(\mathbf{x})$. It relies on a theorem from functional analysis, called Mercer's theorem.

Theorem 3.1. *Denote $h(\cdot) : \mathbb{R}^N \to \mathbb{R}$ any square-integrable function and $\mathcal{S}$ a compact subset of $\mathbb{R}^N$. Then any symmetric, continuous function $k(\cdot,\cdot) : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$ that is positive-definite, i.e. satisfies*

$$\int_{\mathcal{S}} k(\mathbf{x},\mathbf{y})\; h(\mathbf{x})\; h(\mathbf{y})\; d\mathbf{x}\; d\mathbf{y} \geq 0 \tag{3.14}$$

is expandable as the sum of a uniformly-convergent series of the form

$$k(\mathbf{x},\mathbf{y}) \;=\; \sum_{n=1}^{N_{\mathcal{F}}} \lambda_n\; \phi_n(\mathbf{x})\; \phi_n(\mathbf{y}), \tag{3.15}$$

with positive numbers $\lambda_n \in \mathbb{R}^+$, as well as a function $\phi(\cdot) : \mathbb{R}^N \to \mathcal{F}$ that maps the vectors $\mathbf{x},\mathbf{y} \in \mathbb{R}^N$ to a Hilbert space of dimension $N_{\mathcal{F}}$.

Since each λ_n is positive, we can now define the feature map such that

$$\varphi_n(\mathbf{x}) \;=\; \sqrt{\lambda_n}\; \phi_n(\mathbf{x}). \tag{3.16}$$

Hence, the kernel function $k(\cdot,\cdot)$ can be expressed as the inner product

$$k(\mathbf{x},\mathbf{y}) \;=\; \sum_{n=1}^{N_{\mathcal{F}}} \varphi_n(\mathbf{x})\; \varphi_n(\mathbf{y}) \;=\; \varphi(\mathbf{x})^\top \varphi(\mathbf{y}). \tag{3.17}$$

The identity (3.17) is a fundamental result for Support Vector Machines. Namely, since the LS-SVM model (3.1) has two equivalent formulations, and that its dual formulation (3.13) as well as the parameters $\mathbf{L}$ and $\mathbf{b}$ can be computed by exclusively using inner products of the form (3.17), there are two alternatives for training and simulating an LS-SVM model. Either we train and simulate it with system (3.10) and formulation (3.12), by explicitly defining the feature map, which implies a kernel function. Or we train and simulate it with system (3.11) and formulation (3.13), by explicitly defining the kernel function, which implies a feature map.

Developing kernel functions is a field in itself. Some popular choices are:

$$\begin{aligned} k(\mathbf{x},\mathbf{y}) \;&=\; (\;\mathbf{x}^\top\mathbf{y} + t\;)^d && \text{(polynomial kernel of degree } d\text{)}, \\ k(\mathbf{x},\mathbf{y}) \;&=\; \exp\left[-\frac{||\mathbf{x}-\mathbf{y}||_2^2}{2\,\sigma^2}\right] && \text{(RBF kernel of bandwidth } \sigma\text{)}. \end{aligned}$$

3.4 Strengths and Weaknesses

3.4.1 Strengths

There are three principal reasons why this thesis works with LS-SVMs.

1. Convex optimization

The optimization is convex since an LS-SVM is linear in its parameters. Consequently, we can derive an analytical expression for the parameters.

An analytical expression for the parameters is useful for a memory model, because it enables us to build a model with a one-step learning rule. Further, it facilitates parameter tuning because training is deterministic, and it simplifies the analytical study of the characteristics of the model.

2. Dual formulation

The constrained optimization yields two formulations for an LS-SVM. Consequently, we can choose to solve only the smallest of the two systems.

It is quite valuable to have a dual formulation for a memory model, because it allows to train a network with a very large number of neurons without the computational burden of manipulating this large network.

3. Interpretable hyper-parameters

The hyper-parameters control the priority of the optimization problem. Consequently, we can explain their impact on the LS-SVM parameters.

It is key to grasp the effect of the hyper-parameters on a memory model to understand under which conditions the memories are stable equilibria.

3.4.2 Weaknesses

Three aspects of LS-SVMs could hinder the working of a memory model.

1. Limited scalability

LS-SVMs are heavy to train on large and high-dimensional data sets, because the size of both the primal and the dual system grows rapidly. This could complicate the storage of many high-dimensional memories.

A solution developed in Fixed-size LS-SVMs is to build a lower dimensional approximation of the feature map with the Nyström method [31]. The systems are then smaller, and the performance remains acceptable.

2. Offline training process

It is not evident to add or remove a data point from a trained LS-SVM, because the primal and dual parameters are the result of a linear system. This could impede online storing new memories or forgetting old ones.

Nevertheless, several incremental training algorithms have been developed either based on the Sherman-Morrison-Woodbury formula [62, 68, 110], or by exploiting the Karush-Kuhn-Tucker optimality conditions [17, 56].

3. Lack of robustness

The LS-SVM parameters are quite sensitive to outliers in the data set, because the optimization problem relies on a least squares loss function. An outlier could thus deteriorate the reconstruction of some memories.

Nevertheless, several robust training algorithms have been developed either by weighting the importance of each error $\mathbf{e}_p$ in (3.4) [2, 27, 94], or with robust loss functions, like the Hampel or the Myriad loss [26, 32].

3.5 LS-SVMs for associative memory (KAM)

Section 2.2 showed that a Hopfield network trained with the Hebbian rule to store bipolar memories $\{\mathbf{x}_p\}_{p=1}^{P}$, is described by the update equation

$$\mathbf{x}^{(k+1)} = \operatorname{sign}\Big(\frac{1}{N} \sum_{p=1}^{P} \mathbf{x}_p \, \mathbf{x}_p^\top \, \mathbf{x}^{(k)} \Big), \tag{3.18}$$

which is a weighted average of the memories, with weights $\mathbf{x}_p^\top \, \mathbf{x}^{(k)}/N$. Therefore, the more $\mathbf{x}^{(k)}$ and $\mathbf{x}_p$ overlap, the larger their inner product, the larger the weight, and the more $\mathbf{x}^{(k)}$ is pulled towards $\mathbf{x}_p$ by (3.18).

Since (3.18) relies on the inner product between the vectors $\mathbf{x}_p$ and $\mathbf{x}^{(k)}$, it is possible to apply the kernel trick to nuance the similarity measure, which could then increase the memory capacity of the Hopfield network.

Several models thus apply the kernel trick to extend the Hebbian rule, like Recurrent Correlation Associative Memories that introduce a weighting function $w(\cdot) : \mathbb{R} \to \mathbb{R}$ to replace $\mathbf{x}_p^\top \mathbf{x}^{(k)}$ in (3.18) by $w(\mathbf{x}_p^\top \mathbf{x}^{(k)})$ [21, 39]. Kernel Hopfield networks then later explicitly replace this weighting function $w(\cdot)$ with a kernel function $k(\cdot, \cdot) : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$ [34, 72]. However, Support Vector Memories as first build an optimization problem by framing a Hopfield network as N maximal margin classifiers [15, 16].

Further, we discuss a model that particularly inspired us in this thesis, because it can be stated as an LS-SVM: Kernel Associative Memory [67].

3.5.1 Kernel Associative Memory

Kernel Associative Memory (KAM) applies the kernel trick to extend the pseudo-inverse learning rule instead of the Hebbian learning rule. Namely, the KAM learning rule computes the weights of the network as

$$\mathbf{W} \;=\; \mathbf{\Phi} \underbrace{\Big[\, \mathbf{\Phi}^\top \mathbf{\Phi} \;+\; \mu\, \mathbf{I}_P \,\Big]^{-1}}_{\mathbf{S}} \mathbf{X}^\top \tag{3.19}$$

where matrix $\mathbf{\Phi} = [\, \varphi(\mathbf{x}_1) \;\cdots\; \varphi(\mathbf{x}_P) \,] \in \mathbb{R}^{N_\mathcal{F} \times P}$. So, compared to (2.7), a KAM replaces the pseudo-inverse of $\mathbf{X}$ with the pseudo-inverse of $\mathbf{\Phi}$, and adds the identity matrix to ensure that the inverse $\mathbf{S}$ is computable even if the memories are not linearly independent in the feature space. The update equation of a Hopfield network trained with (3.19) thus reads

$$\mathbf{x}^{(k+1)} \;=\; \operatorname{sign}(\; \mathbf{W}^\top \varphi(\mathbf{x}^{(k)}) \;) \;=\; \operatorname{sign}\Big(\; \sum_{p,p'=1}^{P} \mathbf{x}_p \; s_{p,p'} \; k(\mathbf{x}_{p'}, \mathbf{x}^{(k)}) \Big).$$

The weights in (3.19) are the result of the LS-SVM optimization problem

$$\begin{aligned} \underset{\mathbf{w}_n, \mathbf{e}_p}{\text{minimize}} \quad & \sum_{p=1}^{P} \| \, \mathbf{e}_p \, \|_2^2 + \frac{\mu}{2} \sum_{n=1}^{N} \| \, \mathbf{w}_n \, \|_2^2 \\ \text{subject to} \quad & \mathbf{e}_p \;=\; \mathbf{x}_p - \mathbf{W}^\top \varphi(\mathbf{x}_p) \\ & 1 \le p \le P. \end{aligned} \tag{3.20}$$

This optimization shows that a KAM essentially models the identity map, but does not push the memories to be attractors of the dynamical system. A possible solution proposed in [68] is to use special *piecewise* kernels, which also guarantee that the kernel matrix $\mathbf{\Phi}^\top \mathbf{\Phi}$ is always invertible.

The major advantage of a KAM over a traditional Hopfield network is that it increases the number of neurons in the network from N to $N_\mathcal{F}$. Since the dimension of the feature space $N_\mathcal{F}$ is usually larger than N, the KAM learning rule can store more memories, namely up to $N_\mathcal{F}$ [67], while the Hebbian learning rule could only store $0.14\, N$ memories [4], and the pseudo-inverse learning rule could only store N memories [48].

3.6 Conclusion

LS-SVMs are a category of SVM that uses a least squares loss function. For function estimation, LS-SVMs linearly model a nonlinear function. Consequently, their training process is a convex optimization problem. For a thorough explanation, we refer to the book of Suykens *et al.* [96].

Chapter 4

Contractive autoencoders

Contractive autoencoders are a particular type of autoencoders (AEs).

An autoencoder is a neural network trained to copy its input to its output, on a given data set. It is forced, either by regularization techniques or architectural choices, to prioritize which aspects of the input to copy. Thereby, the network learns properties of the data-generating distribution which are useful for feature extraction [36, 104], denoising [22, 35, 105], dimensionality reduction [42, 43], and even data generation [29, 53, 55].

A contractive autoencoder (C-AE) is a particular type of autoencoder that uses contractive regularization, i.e. which induces contraction [80]. Specifically, a C-AE penalizes the Jacobian of the reconstruction function, which pushes that function to contract space around each data point $\mathbf{x}_p$, i.e. $\mathbf{x}_p$, as well as all the points around it, should be reconstructed as $\mathbf{x}_p$.

In this chapter, we explain the functioning of a contractive autoencoder.

Section 4.1 presents the encoder-decoder architecture of an autoencoder.

Section 4.2 constructs the optimization problem for the C-AEs training, and reviews two optimization algorithms: gradient descent and the MAC.

Section 4.3 analyses the principal strengths and weaknesses of a C-AE.

Section 4.4 discusses a model for associative memory based on a C-AE.

4.1 The model equation

An autoencoder is a type of deep feedforward neural network (DNN). Thus, in the same way as a DNN, it consists of three nested components: neurons are combined into layers, and layers are stacked into networks.

The defining feature of an autoencoder lies in its twofold architecture, i.e. it holds two networks connected in series: an encoder and a decoder.

1. The encoder

The encoder converts the input vector $\mathbf{x}$ in a latent representation $\mathbf{h}$ as

$$\mathbf{h} = e(\mathbf{x}; \boldsymbol{\theta}_e)$$

where $\boldsymbol{\theta}_e$ represents all the parameters of the encoder neural network. So this network encodes the main features of the input $\mathbf{x}$ into a *code* $\mathbf{h}$.

2. The decoder

The decoder reconstructs the input $\mathbf{x}$ from the latent representation $\mathbf{h}$ as

$$\tilde{\mathbf{x}} = d(\mathbf{h}; \boldsymbol{\theta}_d) = d(e(\mathbf{x}; \boldsymbol{\theta}_e); \boldsymbol{\theta}_d) = f(\mathbf{x}; \boldsymbol{\theta})$$

where $\boldsymbol{\theta}_d$ represents all the parameters of the decoder neural network. So this network decodes $\mathbf{h}$ to construct a vector $\tilde{\mathbf{x}}$ similar to the input $\mathbf{x}$.

There are two sorts of autoencoders, based on the size of the latent space. In an *undercomplete* autoencoder $\mathbf{h}$ contains fewer dimensions than $\mathbf{x}$, and in an *overcomplete* autoencoder $\mathbf{h}$ contains more dimensions than $\mathbf{x}$.

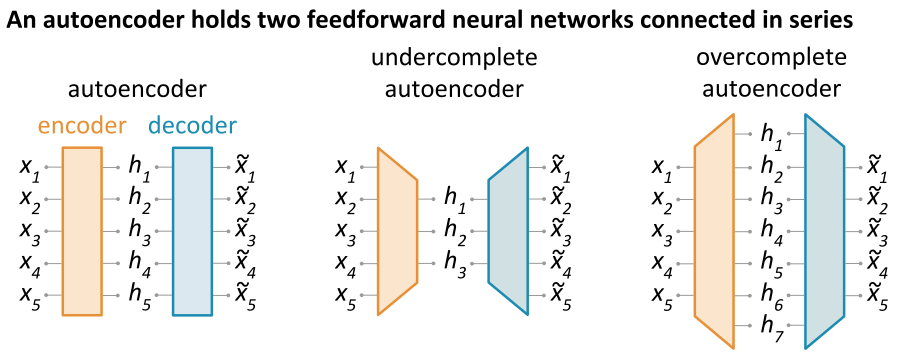


Figure 4.1: An autoencoder encodes its input in a latent representation, and then decodes that representation to reconstruct the original input.

4.2 Training the model

Consider a data set $\{ \mathbf{x}_p \in \mathbb{R}^N \}_{p=1}^P$ generated by a distribution $p(\mathbf{x})$. The goal of an autoencoder is then to identify the main features of $p(\mathbf{x})$.

A C-AE identifies them by contracting space around each data point $\mathbf{x}_p$, i.e. $\mathbf{x}_p$, as well as all the points around it, should be reconstructed as $\mathbf{x}_p$. In the following section, we introduce the C-AE optimization problem, and discuss two optimization algorithms: gradient descent and the MAC.

4.2.1 The optimization problem

Training a C-AE involves a balance between two complementary goals: minimizing the reconstruction errors and the sensitivity of the encoder.

1. Minimize the reconstruction errors

To restrain each reconstruction error, the optimization problem minimizes

$$L(\mathbf{x}_p, f(\mathbf{x}_p; \boldsymbol{\theta})) \;=\; \| \, \mathbf{x}_p \;-\; f(\mathbf{x}_p; \boldsymbol{\theta}) \, \| \qquad 1 \leq p \leq P \tag{4.1}$$

where the loss function $L(\cdot, \cdot) : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$ can be any chosen norm.

2. Minimize the encoder's sensitivity

To contract space around each $\mathbf{x}_p$, the optimization problem minimizes

$$\left\| \frac{\partial e}{\partial \mathbf{x}}(\mathbf{x}_p; \boldsymbol{\theta}_e) \right\|_{\mathrm{F}}^2 \qquad 1 \leq p \leq P. \tag{4.2}$$

This way, all the points close to $\mathbf{x}_p$ have a similar latent representation.

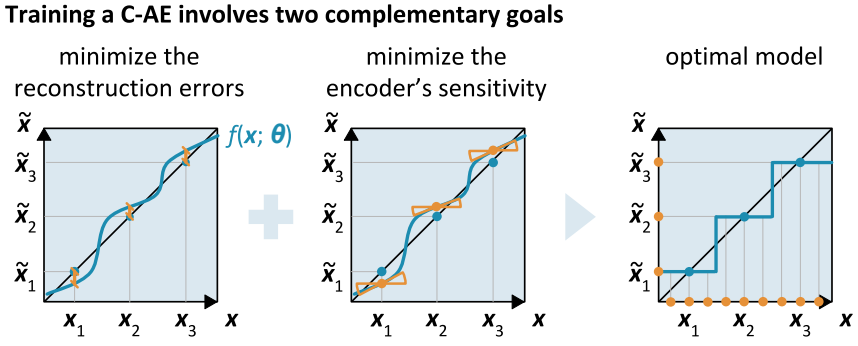


Figure 4.2: A C-AE intends to contract space around each data point by minimizing the reconstruction errors as well as the encoder's sensitivity.

3. The complete optimization problem

Altogether, we obtain the following optimization problem for a C-AE:

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad J(\boldsymbol{\theta}) \;=\; \lambda \overbrace{\sum_{p=1}^{P} L(\mathbf{x}_p, f(\mathbf{x}_p;\boldsymbol{\theta}))}^{\text{Reconstruction errors}} + \gamma \overbrace{\sum_{p=1}^{P} \left\| \frac{\partial e}{\partial \mathbf{x}}(\mathbf{x}_p;\boldsymbol{\theta}_e) \right\|_{\mathrm{F}}^{2}}^{\text{Encoder's sensitivity}} \tag{4.3}$$

The hyper-parameters determine the relative importance of each term. If λ is much larger than γ, (4.3) mainly ensures proper reconstructions. So, $\mathbf{x}_p$ will be reconstructed as $\mathbf{x}_p$, but space will not contract around it. If γ is much larger than λ, (4.3) mainly reduces the encoder's sensitivity. So, the points around $\mathbf{x}_p$ have the same reconstruction, but it is not $\mathbf{x}_p$. Clearly, none of the two terms on their own would yield a good model.

The C-AE optimization problem identifies the essential features of $p(\mathbf{x})$, because of the complementarity of both objectives in (4.3). Namely, minimizing the encoder's sensitivity contracts all directions around $\mathbf{x}_p$, but minimizing the reconstruction error ensures a good reconstruction. As a result, the only directions that resist the contracting pressure are the ones important for reconstruction, i.e. the essential features of $p(\mathbf{x})$.

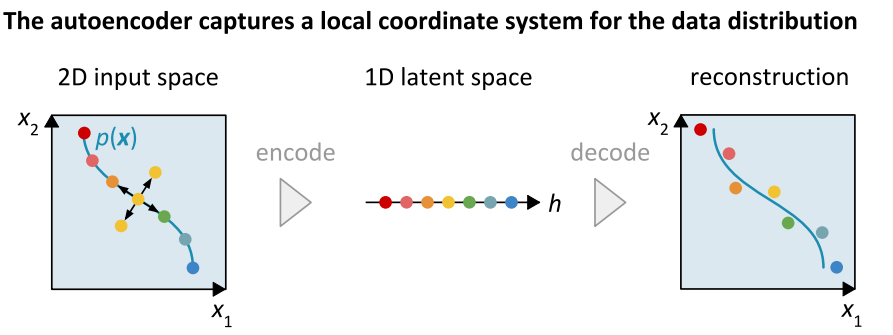


Figure 4.3: Consider a data set generated by a distribution $p(\mathbf{x})$ (–). The objective of an autoencoder is to identify the main features of $p(\mathbf{x})$. A C-AE does so, due to the complementarity of both objectives in (4.3). As a result, the latent representation $\mathbf{h}$ learns to be sensitive only to directions in the input space along which the features of $p(\mathbf{x})$ change. Hence, if a point (●) moves along $p(\mathbf{x})$ its latent representation changes, but if it moves perpendicular to $p(\mathbf{x})$ its latent representation is preserved. The latent space thus forms a local coordinate system for the distribution.

4.2.2 The optimization algorithm

Unlike LS-SVM's, the objective function of a C-AE (4.3) is non-convex. Therefore, the optimization is an iterative, usually gradient-based process that gradually converges towards good parameters for the autoencoder.

This section thus presents two optimization algorithms for autoencoders. For means of clarity, we cut down the optimization problem in (4.3) to

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad J(\boldsymbol{\theta}) \;=\; \sum_{p=1}^{P} L(\mathbf{x}_p, f(\mathbf{x}_p; \boldsymbol{\theta})). \tag{4.4}$$

Nevertheless, both algorithms are also suitable for the general case (4.3).

1. Gradient descent

Gradient descent [58, 82] updates the parameters of the neural network in the opposite direction of the gradient of (4.4). It works in three steps:

1. Forward-propagate the inputs to obtain the reconstructions $f(\mathbf{x}_p; \boldsymbol{\theta}^{(t)})$.
2. Compute for each data point the reconstruction error $L(\mathbf{x}_p, f(\mathbf{x}_p; \boldsymbol{\theta}^{(t)}))$.
3. Backpropagate the errors to update the network's parameters as:

$$\boldsymbol{\theta}^{(t+1)} \;=\; \boldsymbol{\theta}^{(t)} - \alpha \, \frac{\partial J}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}^{(t)}) \;=\; \boldsymbol{\theta}^{(t)} - \alpha \sum_{p=1}^{P} \frac{\partial L}{\partial \boldsymbol{\theta}}(\mathbf{x}_p, f(\mathbf{x}_p; \boldsymbol{\theta}^{(t)}))$$

with learning rate $\alpha \in \mathbb{R}^+$ to set the size of the parameter update.

It then repeats this procedure as long as $\partial J / \partial \boldsymbol{\theta}$ is larger than a tolerance.

This algorithm is quite straightforward, but it has three important issues.

First, gradient descent only uses first-order information, i.e. the gradient. As a result, it only knows about the local behaviour of the objective, which might be very different from the global behaviour of the objective. Better algorithms use higher-order information, or approximations of it.

Second, computing the gradient involves a sum over all the data points. As a result, gradient descent is heavy to perform on large data sets. Better algorithms estimate this first-order information stochastically [52].

Third, computing the gradient involves repeatedly using the chain-rule. As a result, the derivatives of the nonlinearities in each layer amplify, and the gradient quickly either vanishes to zero, or explodes to infinity. Better algorithms substitute the gradients in each layer with targets [61].

2. The method of auxiliary coordinates (MAC)

The method of auxiliary coordinates [14] is a coordinate descent [109] that introduces auxiliary coordinates $\mathbf{h}_{(l),p}$ to optimize nested functions:

$$f(\mathbf{x};\boldsymbol{\theta}) \;=\; f_{(L)}(\; f_{(L-1)}(\; \cdots \; f_{(1)}(\; \mathbf{x}\; ;\; \boldsymbol{\theta}_{(1)}\;)\; \cdots \; ;\; \boldsymbol{\theta}_{(L-1)}\;)\; ;\; \boldsymbol{\theta}_{(L)}\;).$$

It transforms the objective function of a deep feedforward neural network

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad J(\boldsymbol{\theta}) \;=\; \sum_{p=1}^{P} L(\mathbf{x}_p,\; f(\mathbf{x}_p;\boldsymbol{\theta})) \tag{4.5}$$

into

$$\begin{aligned} \underset{\boldsymbol{\theta},\mathbf{H}}{\text{minimize}} \quad & J(\boldsymbol{\theta},\mathbf{H}) \;=\; \sum_{p=1}^{P} L(\mathbf{x}_p,\; \mathbf{h}_{(L),p}) \\ \text{subject to} \quad & \mathbf{h}_{(l),p} \;=\; f_{(l)}(\mathbf{h}_{(l-1),p};\; \boldsymbol{\theta}_{(l)}) \\ & 1 \le l \le L \\ & 1 \le p \le P. \end{aligned} \tag{4.6}$$

The MAC then finds proper parameters by alternating optimization of the Lagrange function of (4.6)—once over the network's parameters $\boldsymbol{\theta}$ and the Lagrange variables $\mathbf{M}$, once over the auxiliary coordinates in $\mathbf{H}$:

$$\begin{aligned} \left[\; \boldsymbol{\theta}^{(t+1)},\; \mathbf{M}^{(t+1)} \;\right] \;&=\; \underset{\boldsymbol{\theta},\mathbf{M}}{\text{argmin}} \quad \mathcal{L}(\; \boldsymbol{\theta}^{(t)}, \mathbf{H}^{(t)}, \mathbf{M}^{(t)}\;) \\ \mathbf{H}^{(t+1)} \;&=\; \underset{\mathbf{H}}{\text{argmin}} \quad \mathcal{L}(\; \boldsymbol{\theta}^{(t+1)}, \mathbf{H}^{(t)}, \mathbf{M}^{(t+1)}\;) \end{aligned}$$

It repeats these steps while $\partial\mathcal{L}/\partial\boldsymbol{\theta}$ and $\partial\mathcal{L}/\partial\mathbf{H}$ are larger than a tolerance.

Minimizing the Lagrange functions is easier than the objective (4.5), because the optimization over each layer can be treated independently. The MAC thus reduces the complex problem of training a deep network, into uncoupled optimization problems coordinated by auxiliary variables.

This method has several advantages compared to simple gradient descent. Most importantly, it alleviates the vanishing and exploding gradients, because there is no need to backpropagate the error through the layers. Furthermore, this algorithm is straightforward to parallelize and it can use higher-order optimization algorithms developed for shallow networks. These faster computations and faster convergence thus speed up training.

4.3 Strengths and Weaknesses

4.3.1 Strenghts

There is one principal reason why we work with C-AEs in this thesis.

1. Analytically implements robustness

The autoencoder's objective is that the reconstruction function is robust to small perturbations around each data point, i.e. noise to be removed.

C-AEs achieve this objective analytically with contractive regularization.

There exist other methods to construct a robust reconstruction function, e.g. denoising AEs augment the data set with noisy versions of the data, and try to reconstruct the true data point, when given a noisy one [104]. The issue is that the data set grows, which is detrimental for LS-SVM's.

4.3.2 Weaknesses

Combining C-AEs with LS-SVMs can potentially solve three weaknesses.

1. Training has local optima

The optimization problem behind the training a C-AE (4.3) is non-convex. As a result, the training process can remain stuck in a bad local optimum.

In contrast, in LS-SVMs the optimization problem (3.4) is convex. The parameters are thus assuredly optimal, given the model architecture.

2. Needs tuning the number of neurons

To ensure that the latent representation is meaningful and generalizable, the number of neurons that form the latent space, should be tuned well.

In contrast, in LS-SVMs the number of neurons must not be tuned. It is implied in the feature map, or follows from the Lagrange method.

3. Unpredictable reconstruction function

C-AEs struggle to generalize in high-dimensional spaces with few data. They then behave well around the data, but are unpredictable elsewhere.

In contrast, in LS-SVMs the kernel matrix's size is independent of N. LS-SVMs can thus generalize in high-dimensional spaces with few data.

4.4 C-AEs for associative memory

An AE naturally incorporates the objective of auto-associative memory: to reconstruct a data point when presented with a noisy version of it. Several autoencoding architectures have thus been developed [63, 75]. Yet, we do not further discuss them since they do not relate to our work.

4.5 Conclusion

A C-AE is a type of autoencoder that uses contractive regularization. Specifically, a C-AE penalizes the Jacobian of the reconstruction function, which pushes that function to contract space around each data point $\mathbf{x}_p$. By doing so, it identifies the features of the data-generating distribution. For a thorough explanation, we refer to the papers of Rifai *et al.* [79, 80].

Part II

Our developments

Chapter 5

Design of a Contractive Least Squares Support Vector Machine

In the previous chapters, we presented the needed theoretical background.

In chapter 2, we showed that Hopfield networks model associative memory by storing a set of memories as stable equilibria of a dynamical system. Thus, when given a noisy memory, they can reconstruct the original one. These networks have attractive features, like being asymptotically stable, but their memory capacity is limited by the size of the neural network.

In chapter 3, we showed that LS-SVMs linearly model a nonlinear function by initially transforming the input data with a nonlinear feature map. Their assets are that they are trained with a convex optimization problem, and that they can apply the kernel trick to train a large neural network without the computational burden of manipulating such a large network.

In chapter 4, we showed that C-AEs identify the features of a distribution by building a reconstruction map that contracts around each data point. Thus, when given a noisy data point, they can reconstruct the original one. Their main asset is that they impose robustness in the model analytically. Yet, their optimization problem is non-convex, which complicates training.

In this chapter, we integrate the contractive properties of C-AEs into the LS-SVM framework to build a dynamical system for associative memory. We call this model a Contractive Least Squares Support Vector Machine.

Section 5.1 introduces the update equation of the new dynamical system. Similarly to LS-SVMs, this update equation is linear in the parameters.

Section 5.2 constructs the convex optimization to train a C-LS-SVM which balances an equilibrium objective and a local stability objective. Then it applies the Lagrange method to derive the optimal parameters.

5.1 The update equation

A C-LS-SVM is a dynamical system described by the update equation

$$\mathbf{x}^{(k+1)} = f(\mathbf{x}^{(k)}) = \mathbf{W}^\top \varphi(\mathbf{x}^{(k)}) + \mathbf{b}, \tag{5.1}$$

where the feature map $\varphi(\cdot) : \mathbb{R}^N \to \mathbb{R}^{N_\mathcal{F}}$ is either defined explicitly, or implicitly with a positive definite kernel function $k(\cdot, \cdot) : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$.

A C-LS-SVM is thus linear in its parameters $\mathbf{W} \in \mathbb{R}^{N_\mathcal{F} \times N}$ and $\mathbf{b} \in \mathbb{R}^N$. As a result, the optimization problem for finding $\mathbf{W}$ and $\mathbf{b}$ can be convex.

5.2 Training the model

The goal of the optimization is that a set of memories $\{\mathbf{x}_p \in \mathbb{R}^N\}_{p=1}^P$ becomes the locally stable equilibria of the C-LS-SVM update equation. In other words, the state $\mathbf{x}^{(k)}$ should converge to one of the memories $\mathbf{x}_p$. We now build the optimization problem and infer the optimal parameters.

5.2.1 The optimization problem

We decide to construct a constrained optimization problem in order to obtain a primal-dual interpretation of C-LS-SVMs, similarly to LS-SVMs. Thus, we convert the LS-SVM optimization problem for static regression into an optimization problem for associative memory with two objectives: an equilibrium and a local stability objective, as presented in figure 5.1.

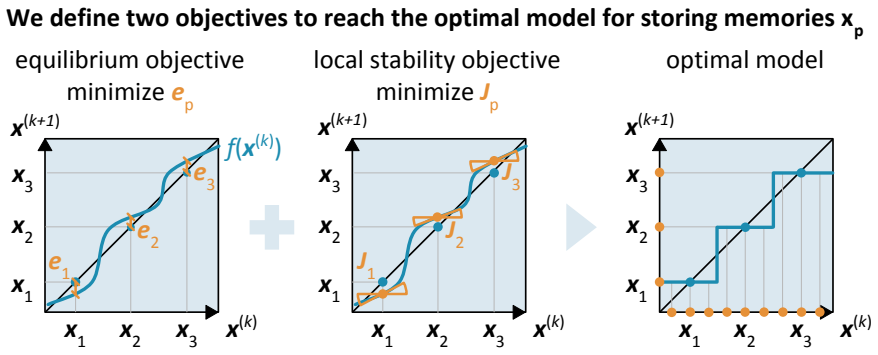


Figure 5.1: The optimal model maps each state onto the closest memory. In order to approximate this optimal model we formulate two objectives. The equilibrium objective intends to make each memory an equilibrium by minimizing the distance between the mapping of $\mathbf{x}_p$ and $\mathbf{x}_p$ itself. The local stability objective intends to make each memory locally stable by minimizing the derivative of the update equation in each memory $\mathbf{x}_p$.

1. The equilibrium objective

The equilibrium objective pushes each memory to become an equilibrium.

A memory $\mathbf{x}_p$ is an equilibrium of (5.1) if it satisfies $\mathbf{x}_p = \mathbf{W}^\top \varphi(\mathbf{x}_p) + \mathbf{b}$. The optimization problem should thus minimize the 2-norm of the errors:

$$\| \mathbf{e}_p \|_2^2 = \| \mathbf{x}_p - \mathbf{W}^\top \varphi(\mathbf{x}_p) - \mathbf{b} \|_2^2 \qquad 1 \leq p \leq P. \tag{5.2}$$

2. The local stability objective

The local stability objective pushes each memory to be locally attractive.

In the neighbourhood of $\mathbf{x}_p$, the first-order Taylor expansion of (5.1) is

$$\begin{aligned} \mathbf{x}^{(k+1)} &= f(\mathbf{x}_p) &&+ \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_p)\,(\mathbf{x}^{(k)} - \mathbf{x}_p) \\ &= \mathbf{W}^\top \varphi(\mathbf{x}_p) + \mathbf{b} &&+ \underbrace{\mathbf{W}^\top \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}_p)}_{\text{Jacobian } \mathbf{J}_p}\,(\mathbf{x}^{(k)} - \mathbf{x}_p). \end{aligned}$$

Thus, for any state $\mathbf{x}^{(k)}$ in the neighbourhood of $\mathbf{x}_p$ it should hold that

$$\begin{aligned} \| \mathbf{x}^{(k)} - \mathbf{x}_p \|_2^2 &> \| \mathbf{x}^{(k+1)} - \mathbf{x}_p \|_2^2 \\ &> \| \cancel{\mathbf{W}^\top \varphi(\mathbf{x}_p) + \mathbf{b}} + \mathbf{J}_p\,(\mathbf{x}^{(k)} - \mathbf{x}_p) - \cancel{\mathbf{x}_p} \|_2^2 \end{aligned}$$

where the two terms cancel out since $\mathbf{x}_p$ should already be an equilibrium. If $\mathbf{x}_p$ is not an equilibrium, the state cannot converge to it in the first place.

To ensure that $\|\mathbf{J}_p\,(\mathbf{x}^{(k)} - \mathbf{x}_p)\|_2^2 < \|\mathbf{x}^{(k)} - \mathbf{x}_p\|_2^2$ for any direction of $\mathbf{x}^{(k)}$, the singular values of the Jacobian $\mathbf{J}_p$ should be strictly smaller than 1. Hence, the optimization problem minimizes the norm of the Jacobians:

$$\| \mathbf{J}_p \|_\mathrm{F}^2 = \sum_{n=1}^{N} \sigma_n^2(\mathbf{J}_p) = \left\| \mathbf{W}^\top \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}_p) \right\|_\mathrm{F}^2 \qquad 1 \leq p \leq P. \tag{5.3}$$

Note that minimizing (5.3) in order to make each memory locally stable is similar to the regularization function of a contractive autoencoder. Essentially, both intend to make the space around each $\mathbf{x}_p$ contractive.

Alternatively, we could achieve local stability as a denoising autoencoder. This requires augmenting the data set with corrupted points $\mathbf{x}_p + \boldsymbol{\varepsilon}_p$, and afterwards, instead of minimizing (5.2), rather minimize the errors:

$$\| \mathbf{e}_p \|_2^2 = \| \mathbf{x}_p - \mathbf{W}^\top \varphi(\mathbf{x}_p + \boldsymbol{\varepsilon}_p) - \mathbf{b} \|_2^2 \qquad 1 \leq p \leq P. \tag{5.4}$$

However, this strategy would strongly increase the number of data points, which would increase the size of the dual linear system in an LS-SVM. This complicates training, and hinders exploiting the duality of the model. For this reason, we do not apply the strategy of denoising autoencoders.

Additional regularization

Additional regularization then controls the risk of over- and underfitting. As in LS-SVMs, we apply a parameter norm regularization on the weights. Thus, the optimization problem also minimizes the norm of the weights:

$$\| \mathbf{W} \|_{\mathrm{F}}^2 = \sum_{n=1}^{N} \| \mathbf{w}_n \|_2^2 = \operatorname{tr}(\mathbf{W}^\top \mathbf{W}). \tag{5.5}$$

Intuitively, this corresponds to minimizing the complexity of the model, which should restrain the number of unwanted equilibria in the system.

The complete optimization problem

Altogether, we obtain the following least squares optimization problem:

Definition 5.1. A Contractive Least Squares Support Vector Machine is a discrete dynamical system described by the update equation

$$\mathbf{x}^{(k+1)} = \mathbf{W}^\top \varphi(\mathbf{x}^{(k)}) + \mathbf{b}. \tag{5.6}$$

For a set of memories $\{\mathbf{x}_p \in \mathbb{R}^N\}_{p=1}^P$, hyper-parameters $\lambda, \gamma, \eta \in \mathbb{R}^+$, the parameters of (5.6) are the result of the optimization problem

$$\begin{aligned} \underset{\mathbf{w}_n, \mathbf{b}, \mathbf{e}_p, \mathbf{J}_p}{\text{minimize}} \quad & \frac{\lambda}{2} \overbrace{\sum_{p=1}^{P} \| \mathbf{e}_p \|_2^2}^{\text{Equilibrium}} + \frac{\gamma}{2} \overbrace{\sum_{p=1}^{P} \| \mathbf{J}_p \|_{\mathrm{F}}^2}^{\text{Local stability}} + \frac{\eta}{2} \overbrace{\sum_{n=1}^{N} \| \mathbf{w}_n \|_2^2}^{\text{Regularization}} \\ \text{subject to} \quad & \mathbf{e}_p = \mathbf{x}_p - \mathbf{W}^\top \varphi(\mathbf{x}_p) - \mathbf{b} \\ & \mathbf{J}_p = \mathbf{W}^\top \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}_p) \\ & 1 \leq p \leq P. \end{aligned} \tag{5.7}$$

The hyper-parameters determine the relative importance of each term. If λ is much larger than γ and η, (5.7) mainly minimizes the errors. Thus, each memory might be an equilibrium, but probably not attractive. If γ is much larger than λ and η, (5.7) mainly minimizes the Jacobians. Thus, each memory might be attractive, but probably not an equilibrium. If η is much larger than λ and γ, (5.7) mainly minimizes the weights. Thus, the model is flat, and the memories are certainly not equilibria. Clearly none of the three terms on their own would yield a good model. It is their complementarity that creates a model for associative memory.

5.2.2 The optimal parameters

To shorten the notation, let us define the matrix formulation as follows:

Definition 5.2. A Contractive Least Squares Support Vector Machine is a discrete dynamical system described by the update equation

$$\mathbf{x}^{(k+1)} \;=\; \mathbf{W}^\top \varphi(\mathbf{x}^{(k)}) \;+\; \mathbf{b}. \tag{5.8}$$

For a set of memories $\{\mathbf{x}_p \in \mathbb{R}^N\}_{p=1}^P$, hyper-parameters $\lambda, \gamma, \eta \in \mathbb{R}^+$, we can then collect the vectors from (5.7) into matrices defined as

$$\begin{aligned}
&\text{the data matrix} && \mathbf{X} \in \mathbb{R}^{N\times P} && = \begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_P \end{bmatrix} \\
&\text{the feature matrix} && \mathbf{\Phi} \in \mathbb{R}^{N_\mathcal{F}\times P} && = \begin{bmatrix} \varphi(\mathbf{x}_1) & \cdots & \varphi(\mathbf{x}_P) \end{bmatrix} \\
&\text{the derivative matrix} && \mathbf{F} \in \mathbb{R}^{N_\mathcal{F}\times NP} && = \begin{bmatrix} \dfrac{\partial\varphi}{\partial\mathbf{x}}(\mathbf{x}_1) & \cdots & \dfrac{\partial\varphi}{\partial\mathbf{x}}(\mathbf{x}_P) \end{bmatrix} \\
&\text{the weight matrix} && \mathbf{W} \in \mathbb{R}^{N_\mathcal{F}\times N} && = \begin{bmatrix} \mathbf{w}_1 & \cdots & \mathbf{w}_N \end{bmatrix} \\
&\text{the bias matrix} && \mathbf{B} \in \mathbb{R}^{N\times P} && = \begin{bmatrix} \mathbf{b} & \cdots & \mathbf{b} \end{bmatrix} = \mathbf{b}\,\mathbf{1}_P^\top \\
&\text{the error matrix} && \mathbf{E} \in \mathbb{R}^{N\times P} && = \begin{bmatrix} \mathbf{e}_1 & \cdots & \mathbf{e}_P \end{bmatrix} \\
&\text{the jacobian matrix} && \mathbf{J} \in \mathbb{R}^{N\times NP} && = \begin{bmatrix} \mathbf{J}_1 & \cdots & \mathbf{J}_P \end{bmatrix}
\end{aligned}$$

The parameters of (5.8) are the result of the optimization problem

$$\begin{aligned}
\underset{\mathbf{W},\mathbf{b},\mathbf{E},\mathbf{J}}{\text{minimize}} \quad & \frac{\lambda}{2}\overbrace{\operatorname{tr}(\mathbf{E}^\top\mathbf{E})}^{\text{Equilibrium}} + \frac{\gamma}{2}\overbrace{\operatorname{tr}(\mathbf{J}^\top\mathbf{J})}^{\text{Local stability}} + \frac{\eta}{2}\overbrace{\operatorname{tr}(\mathbf{W}^\top\mathbf{W})}^{\text{Regularization}} \\
\text{subject to} \quad & \mathbf{E} = \mathbf{X} - \mathbf{W}^\top\mathbf{\Phi} - \mathbf{B} \\
& \mathbf{J} = \mathbf{W}^\top\mathbf{F}.
\end{aligned} \tag{5.9}$$

The trace $\operatorname{tr}(\cdot)$ computes the sum of the diagonal elements of a matrix.

Importantly, a C-LS-SVM does not impose strict bounds on $\mathbf{e}_p$ or on $\mathbf{J}_p$. This could be done however, with inequality constraints on $\mathbf{e}_p$ and $\mathbf{J}_p$, but these few inequalities would complicate the optimization process. Namely, they require the use of Quadratic Programming algorithms, which are less efficient than the solvers we can use without inequalities.

Let us now apply the three steps of the Lagrange method to solve (5.9)[1].

First, we compute the Lagrange function with dual variables $\mathbf{L}$ and $\mathbf{M}$:

$$\begin{aligned}\mathcal{L}(\mathbf{W},\mathbf{b},\mathbf{E},\mathbf{J},\mathbf{L},\mathbf{M}) \;=\; & \frac{\lambda}{2}\operatorname{tr}(\mathbf{E}^\top\mathbf{E}) \;+\; \frac{\gamma}{2}\operatorname{tr}(\mathbf{J}^\top\mathbf{J}) \;+\; \frac{\eta}{2}\operatorname{tr}(\mathbf{W}^\top\mathbf{W}) \\ & +\; \operatorname{tr}(\mathbf{L}^\top(\mathbf{X}-\mathbf{W}^\top\boldsymbol{\Phi}-\mathbf{B}-\mathbf{E})) \\ & +\; \operatorname{tr}(\mathbf{M}^\top(\mathbf{J}-\mathbf{W}^\top\mathbf{F})). \end{aligned} \tag{5.10}$$

Second, we minimize (5.10) with respect to the variables $\mathbf{W}$, $\mathbf{b}$, $\mathbf{E}$ and $\mathbf{J}$

$$\frac{\partial\mathcal{L}}{\partial\mathbf{W}} \;=\; \mathbf{0}_{N_\mathcal{F}\times N} \quad\Longleftrightarrow\quad \mathbf{W} \;=\; \frac{1}{\eta}(\boldsymbol{\Phi}\mathbf{L}^\top+\mathbf{F}\mathbf{M}^\top), \tag{5.11a}$$

$$\frac{\partial\mathcal{L}}{\partial\mathbf{b}} \;=\; \mathbf{0}_N \quad\Longleftrightarrow\quad \mathbf{0}_N \;=\; \mathbf{L}\,\mathbf{1}_P, \tag{5.11b}$$

$$\frac{\partial\mathcal{L}}{\partial\mathbf{E}} \;=\; \mathbf{0}_{N\times P} \quad\Longleftrightarrow\quad \mathbf{E} \;=\; \frac{1}{\lambda}\mathbf{L}, \tag{5.11c}$$

$$\frac{\partial\mathcal{L}}{\partial\mathbf{J}} \;=\; \mathbf{0}_{N\times NP} \quad\Longleftrightarrow\quad \mathbf{J} \;=\; -\frac{1}{\gamma}\mathbf{M}, \tag{5.11d}$$

and obtain the dual function by filling in (5.11a), (5.11c) and (5.11d), namely the fixed points of the Lagrange function, in the Lagrange function:

$$\begin{aligned}\mathcal{Q}(\mathbf{L},\mathbf{M}) \;=\; & \frac{\lambda}{2}\operatorname{tr}\Big(\frac{1}{\lambda^2}\mathbf{L}^\top\mathbf{L}\Big) \;+\; \frac{\gamma}{2}\operatorname{tr}\Big(\frac{1}{\gamma^2}\mathbf{M}^\top\mathbf{M}\Big) \\ & +\; \frac{\eta}{2}\operatorname{tr}\Big(\frac{1}{\eta^2}(\mathbf{L}\boldsymbol{\Phi}^\top+\mathbf{M}\mathbf{F}^\top)(\boldsymbol{\Phi}\mathbf{L}^\top+\mathbf{F}\mathbf{M}^\top)\Big) \\ & +\; \operatorname{tr}\Big(\mathbf{L}^\top\Big(\mathbf{X}-\frac{1}{\eta}(\mathbf{L}\boldsymbol{\Phi}^\top+\mathbf{M}\mathbf{F}^\top)\boldsymbol{\Phi}-\mathbf{B}-\frac{1}{\lambda}\mathbf{L}\Big)\Big) \\ & +\; \operatorname{tr}\Big(\mathbf{M}^\top\Big(-\frac{1}{\gamma}\mathbf{M}-\frac{1}{\eta}(\mathbf{L}\boldsymbol{\Phi}^\top+\mathbf{M}\mathbf{F}^\top)\mathbf{F}\Big)\Big). \end{aligned} \tag{5.12}$$

Third, we maximize (5.12) with respect to the dual variables $\mathbf{L}$ and $\mathbf{M}$

$$\frac{\partial\mathcal{Q}}{\partial\mathbf{L}}=\mathbf{0} \quad\Longleftrightarrow\quad \mathbf{X}=\frac{1}{\eta}\mathbf{L}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}+\frac{1}{\eta}\mathbf{M}\mathbf{F}^\top\boldsymbol{\Phi}+\mathbf{B}+\frac{1}{\lambda}\mathbf{L} \tag{5.13a}$$

$$\frac{\partial\mathcal{Q}}{\partial\mathbf{M}}=\mathbf{0} \quad\Longleftrightarrow\quad \mathbf{0}=\frac{1}{\eta}\mathbf{L}\boldsymbol{\Phi}^\top\mathbf{F}+\frac{1}{\eta}\mathbf{M}\mathbf{F}^\top\mathbf{F}+\frac{1}{\gamma}\mathbf{M}. \tag{5.13b}$$

The optimal primal parameters $\mathbf{W}$ and $\mathbf{b}$, or dual parameters $\mathbf{L}$ and $\mathbf{M}$ are then the solutions to the optimality conditions in (5.11) or in (5.13).

[1] The following properties are used throughout this thesis: $\frac{\partial\operatorname{tr}}{\partial\mathbf{X}}(\mathbf{A}\mathbf{X}\mathbf{B})=\mathbf{A}^\top\mathbf{B}^\top$, $\frac{\partial\operatorname{tr}}{\partial\mathbf{X}}(\mathbf{A}\mathbf{X}^\top\mathbf{B})=\mathbf{B}\mathbf{A}$, $\frac{\partial\operatorname{tr}}{\partial\mathbf{X}}(\mathbf{A}\mathbf{X}^\top\mathbf{X})=\mathbf{X}\mathbf{A}^\top+\mathbf{X}\mathbf{A}$, $\frac{\partial\operatorname{tr}}{\partial\mathbf{X}}(\mathbf{X}^\top\mathbf{X}\mathbf{A})=\mathbf{X}\mathbf{A}^\top+\mathbf{X}\mathbf{A}$, $\frac{\partial\operatorname{tr}}{\partial\mathbf{X}}(\mathbf{A}\mathbf{X}\mathbf{B}\mathbf{X}^\top\mathbf{C})=\mathbf{A}^\top\mathbf{C}^\top\mathbf{X}\mathbf{B}^\top+\mathbf{C}\mathbf{A}\mathbf{X}\mathbf{B}$, as well as $\operatorname{tr}(\mathbf{A}+\mathbf{B})=\operatorname{tr}(\mathbf{A})+\operatorname{tr}(\mathbf{B})$ [74].

To solve the optimality conditions for the primal parameters $\mathbf{W}$ and $\mathbf{b}$, we replace the variables $\mathbf{L}$ and $\mathbf{M}$ in (5.11a) and (5.11b) by $\lambda\mathbf{E}$ and $-\gamma\mathbf{J}$, and write out $\mathbf{E}$ and $\mathbf{J}$ in terms of $\mathbf{W}$ and $\mathbf{b}$ to obtain the linear system

$$\begin{bmatrix} \boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \frac{\gamma}{\lambda}\mathbf{F}\mathbf{F}^\top + \frac{\eta}{\lambda}\mathbf{I}_{N_\mathcal{F}} & \boldsymbol{\Phi}\mathbf{1}_P \\ \mathbf{1}_P^\top\boldsymbol{\Phi}^\top & P \end{bmatrix} \begin{bmatrix} \mathbf{W} \\ \mathbf{b}^\top \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Phi}\mathbf{X}^\top \\ \mathbf{1}_P^\top\mathbf{X}^\top \end{bmatrix}. \tag{5.14}$$

Or, to solve the optimality conditions for the dual parameters $\mathbf{L}$ and $\mathbf{M}$, we couple the conditions in (5.11b) and (5.13) to obtain the linear system

$$\begin{bmatrix} \frac{1}{\eta}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \frac{1}{\lambda}\mathbf{I}_P & \frac{1}{\eta}\boldsymbol{\Phi}^\top\mathbf{F} & \mathbf{1}_P \\ \frac{1}{\eta}\mathbf{F}^\top\boldsymbol{\Phi} & \frac{1}{\eta}\mathbf{F}^\top\mathbf{F} + \frac{1}{\gamma}\mathbf{I}_{NP} & \mathbf{0}_{NP} \\ \mathbf{1}_P^\top & \mathbf{0}_{NP}^\top & 0 \end{bmatrix} \begin{bmatrix} \mathbf{L}^\top \\ \mathbf{M}^\top \\ \mathbf{b}^\top \end{bmatrix} = \begin{bmatrix} \mathbf{X}^\top \\ \mathbf{0}_{NP\times N} \\ \mathbf{0}_N^\top \end{bmatrix}. \tag{5.15}$$

As expected, these systems resemble the linear systems of an LS-SVM. In other words, (5.14) is similar to (3.10), and (5.15) is similar to (3.11).

The new terms, namely the ones that involve $\mathbf{F}$, introduce a complication, because $\mathbf{F}$ contains the Jacobians $\partial\varphi/\partial\mathbf{x}$ evaluated in each memory $\mathbf{x}_p$. Thus if the feature map is implicit because the model uses a kernel $k(\cdot,\cdot)$, we cannot explicitly calculate a formulation for the Jacobian $\partial\varphi/\partial\mathbf{x}$. Fortunately, (5.15) only holds $\mathbf{F}$ in the products $\mathbf{F}^\top\boldsymbol{\Phi}$, $\boldsymbol{\Phi}^\top\mathbf{F}$ and $\mathbf{F}^\top\mathbf{F}$. We can therefore extend the kernel trick to compute these products from

$$\begin{aligned} \frac{\partial\varphi}{\partial\mathbf{x}}(\mathbf{x})^\top\varphi(\mathbf{y}) &= \frac{\partial}{\partial\mathbf{x}}\Big(\varphi(\mathbf{x})^\top\varphi(\mathbf{y})\Big) &&= \frac{\partial k}{\partial\mathbf{x}}(\mathbf{x},\mathbf{y}) \\ \varphi(\mathbf{x})^\top\frac{\partial\varphi}{\partial\mathbf{y}}(\mathbf{y}) &= \left(\frac{\partial}{\partial\mathbf{y}}\Big(\varphi(\mathbf{x})^\top\varphi(\mathbf{y})\Big)\right)^\top &&= \left(\frac{\partial k}{\partial\mathbf{y}}(\mathbf{x},\mathbf{y})\right)^\top \\ \frac{\partial\varphi}{\partial\mathbf{x}}(\mathbf{x})^\top\frac{\partial\varphi}{\partial\mathbf{y}}(\mathbf{y}) &= \frac{\partial}{\partial\mathbf{y}}\left(\frac{\partial}{\partial\mathbf{x}}\Big(\varphi(\mathbf{x})^\top\varphi(\mathbf{y})\Big)\right) &&= \frac{\partial^2 k}{\partial\mathbf{x}\partial\mathbf{y}}(\mathbf{x},\mathbf{y}) \end{aligned}$$

Overall, like an LS-SVM, a C-LS-SVM has two equivalent formulations. In terms of the primal parameters $\mathbf{W}$ and $\mathbf{b}$, the update equation reads:

$$\mathbf{x}^{(k+1)} = \mathbf{W}^\top\varphi(\mathbf{x}^{(k)}) + \mathbf{b} \tag{5.16}$$

In terms of the dual parameters $\mathbf{L}$ and $\mathbf{M}$, the update equation reads:

$$\begin{aligned} \mathbf{x}^{(k+1)} &= \frac{1}{\eta}\left(\mathbf{L}\,\boldsymbol{\Phi}^\top + \mathbf{M}\,\mathbf{F}^\top\right)\varphi(\mathbf{x}^{(k)}) + \mathbf{b} \\ &= \frac{1}{\eta}\sum_{p=1}^{P}\left[\,\mathbf{l}_p\,k(\mathbf{x}_p,\mathbf{x}^{(k)}) + \mathbf{M}_p\,\frac{\partial k}{\partial\mathbf{x}}(\mathbf{x}_p,\mathbf{x}^{(k)})\,\right] + \mathbf{b}. \end{aligned} \tag{5.17}$$

5.2.3 The optimal space for training

There are two equivalent ways to train a C-LS-SVM: (5.14) or (5.15). However, the primal system requires computing $N(N_{\mathcal{F}}+1)$ parameters, whereas the dual system requires computing $N(P(N+1)+1)$ parameters. Thus depending on the memories to store, and the chosen feature map, one should solve the system that requires the less computational work.

As long as $N_{\mathcal{F}} < P(N+1)$ it is best to solve the primal linear system. This situation occurs when there is a large number of memories to store, when the memories to store are high-dimensional, for instance pictures, or when the feature space is low-dimensional, as in a tanh feature map. Also, if the feature map can only be explicit, we solve the primal system.

Contrarily, if $N_{\mathcal{F}} > P(N+1)$ it is best to solve the dual linear system. This situation occurs when there is a small number of memories to store, when the memories to store are low-dimensional, for instance sensor data, or when the feature space is high-dimensional, as with an RBF kernel. Also, if the feature map can only be implicit, we solve the dual system.

5.3 Conclusion

We presented Contractive Least Squares Support Vector Machines as dynamical systems that integrate the contractive properties of C-AEs into the LS-SVM framework in order to model auto-associative memory.

First, we introduced the update equation that describes a C-LS-SVM. Similarly to LS-SVMs, this update equation is linear in the parameters.

Afterwards, we constructed the convex optimization to train a C-LS-SVM as an equilibrium objective that pushes each memory to be an equilibrium, and a local stability objective that pushes each memory to be attractive.

Finally, we applied the Lagrange method to derive the optimal parameters. We obtained two alternatives for training and simulating a C-LS-SVM, Either we train and simulate it with system (5.14) and formulation (5.16), by explicitly defining the feature map, which implies a kernel function. Or we train and simulate it with system (5.15) and formulation (5.17), by explicitly defining the kernel function, which implies a feature map.

In the next chapters, we theoretically and empirically study C-LS-SVMs. By analysing them as Hopfield networks, we gauge the memory capacity, by analysing them as LS-SVMs, we derive their Bayesian interpretation, and by analysing them as C-AEs, we build deep and generative variants.

Chapter 6

Analysis from the perspective of Hopfield networks

In chapter 5, we presented C-LS-SVMs as models for associative memory. Their three main assets are, first, that they are linear in their parameters. Thus, the optimization yields an analytical expression for the parameters. Second, the C-LS-SVM update equation has a primal-dual interpretation. Thus, it can use the kernel trick to implicitly train a much larger network. Third, the C-LS-SVM optimization is controlled by hyper-parameters. Thus, we can modify them to control the behaviour of the memory model.

In this chapter, we study C-LS-SVMs from the angle of Hopfield networks. i.e. we analyse the ability of C-LS-SVMs to model associative memory. To that end, their linearity, duality and tractability will be of great help.

Section 6.1 derives a parallel formulation of the C-LS-SVM optimization which demonstrates that the weights of a C-LS-SVM are comparable to the weights of a Hopfield network trained with KAM on noisy memories. This noise explicitly encourages each memory to be a stable equilibrium.

Section 6.2 examines the memory capacity of a C-LS-SVM in two steps: the number of equilibria it can store, and the stability of each equilibrium. In section 6.2.1, we prove that a C-LS-SVM can store up to $N_{\mathcal{F}}$ equilibria. Yet as the amount of regularization increases, this capacity decreases. Even so, this result shows that a C-LS-SVM can apply the kernel trick to implicitly train a neural network that has a much larger capacity, without the computational burden of manipulating this larger network. In section 6.2.2, we derive conditions on the C-LS-SVM hyper-parameters to guarantee that the update equation contracts space around a point.

Section 6.3 visualizes C-LS-SVMs with different feature maps and kernels. It depicts the effect of the kernel parameter on the obtained C-LS-SVM.

6.1 A Hopfield network trained with KAM

Hopfield networks have been thoroughly studied over the last forty years, generating an important body of research on what we could call the Hopfield framework, i.e. storing memories with a dynamical system whose parameters can be computed through a one-step analytical expression. For that reason, by situating C-LS-SVMs within this Hopfield framework, there is a great potential to analyse C-LS-SVMs with existing theories.

This section thus situates C-LS-SVMs within the Hopfield framework. To do so, we derive a new formulation for the C-LS-SVM weights which closely relates to an existing learning rule, with the following proposition.

Proposition 6.1. *If $\gamma/\lambda \to 0$, the C-LS-SVM objective expressed as*

$$\begin{aligned} \underset{\mathbf{W},\mathbf{b},\mathbf{e},\mathbf{J}}{\text{minimize}} \quad & \frac{\lambda}{2}\,\mathbb{E}\Big[\,\|\,\mathbf{e}\,\|_2^2\,\Big] \;+\; \frac{\gamma}{2}\,\mathbb{E}\Big[\,\|\,\mathbf{J}\,\|_{\mathrm{F}}^2\,\Big] \;+\; \frac{\eta}{2}\,\mathbb{E}\Big[\,\|\,\mathbf{W}\,\|_{\mathrm{F}}^2\,\Big] \\ \text{subject to} \quad & \mathbf{e} = \mathbf{x} - \mathbf{W}^\top \varphi(\mathbf{x}) - \mathbf{b} \\ & \mathbf{J} = \mathbf{W}^\top \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}) \end{aligned} \tag{6.1}$$

is equivalent to the LS-SVM objective to estimate the identity map trained on corrupted data with noise $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}_N, \gamma/\lambda\, \mathbf{I}_N)$, which is

$$\begin{aligned} \underset{\mathbf{W},\mathbf{b},\mathbf{e}}{\text{minimize}} \quad & \frac{\lambda}{2}\,\mathbb{E}\Big[\,\|\,\mathbf{e}\,\|_2^2\,\Big] \;+\; \frac{\eta}{2}\,\mathbb{E}\Big[\,\|\,\mathbf{W}\,\|_{\mathrm{F}}^2\,\Big] \\ \text{subject to} \quad & \mathbf{e} = \mathbf{x} - \mathbf{W}^\top \varphi(\mathbf{x} + \boldsymbol{\varepsilon}) - \mathbf{b}. \end{aligned} \tag{6.2}$$

To facilitate the reading, we have moved the 1-page proof to Appendix A. Essentially, it consists of rewriting (6.2) with a first-order approximation.

Proposition 6.1 implies that the C-LS-SVM weights can be written as

$$\mathbf{W} \;=\; \frac{1}{\eta}\,\tilde{\boldsymbol{\Phi}}\,\mathbf{L}^\top \;=\; \tilde{\boldsymbol{\Phi}}\,\Big[\,\tilde{\boldsymbol{\Phi}}^\top\tilde{\boldsymbol{\Phi}} + \frac{\eta}{\lambda}\,\mathbf{I}_P\,\Big]^{-1}(\mathbf{X} - \mathbf{B})^\top \tag{6.3}$$

where $\tilde{\boldsymbol{\Phi}} = [\;\varphi(\mathbf{x}_1 + \boldsymbol{\varepsilon}_1) \;\; \cdots \;\; \varphi(\mathbf{x}_P + \boldsymbol{\varepsilon}_P)\;]$ with $\boldsymbol{\varepsilon}_p \sim \mathcal{N}(\,\mathbf{0}_N, \gamma/\lambda\,\mathbf{I}_N\,)$.

Expression (6.3) shows that a KAM is a specific case of a C-LS-SVM, since for $\gamma = 0$, we can define $\mu = \eta/\lambda$ such that (6.3) simplifies to (3.19). However, as noted in section 3.5.1, the KAM optimization problem only builds the identity map, but does not push the memories to be attractive. A potential solution proposed in [68] is to use special *piecewise* kernels. The alternative solution developed in C-LS-SVMs is to include the stability objective in the optimization problem, similarly to C-AE's. Overall, the weights computed with a C-LS-SVM thus correspond to a Hopfield network trained with the KAM learning rule on noisy memories.

6.2 Memory capacity of C-LS-SVMs

Now that we have situated C-LS-SVMs within the Hopfield framework, i.e. networks trained using the KAM learning rule on noisy memories, we can analyse C-LS-SVMs and relate our results to theories on KAMs.

This section specifically focuses on the memory capacity of C-LS-SVMs. In the same way as KAMs, we expect the memory capacity to scale with the size of the feature space $N_{\mathcal{F}}$ instead of the size of the input space N.

In general, the memory capacity of a network is defined by two aspects: the number of equilibria it can store, and the stability of each equilibrium. In this regard, we prove that C-LS-SVMs can store up to $N_{\mathcal{F}}$ equilibria, and derive conditions on λ, γ and η to ensure a point is locally attractive.

6.2.1 Number of equilibria that C-LS-SVMs can store

Expression (6.3) plainly separates the effect of γ and η on the weights. The ratio η/λ controls the amount of parameter norm regularization, whereas the ratio γ/λ controls the amount of regularization with noise.

Hence, with (6.3) we can study the number of equilibria in a C-LS-SVM for the four settings of the hyper-parameters: (1) $\gamma = 0$ with $\eta = 0$, (2) $\gamma = 0$ with $\eta \neq 0$, (3) $\gamma \neq 0$ with $\eta = 0$, and (4) $\gamma \neq 0$ with $\eta \neq 0$.

1. Setting $\gamma = 0$ and $\eta = 0$

Proposition 6.2. *A C-LS-SVM as defined in Definition 5.2 with hyper-parameters $\gamma = 0$, $\eta = 0$ can have up to $N_{\mathcal{F}}$ equilibria $\mathbf{x}_p$, provided that the vectors $\{ \varphi(\mathbf{x}_p) \}_{p=1}^{P}$ are linearly independent.*

Proof of Proposition 6.2. If $\gamma = \eta = 0$, the weights in (6.3) simplify to

$$\mathbf{W} \;=\; \boldsymbol{\Phi} \left[\boldsymbol{\Phi}^\top \boldsymbol{\Phi} \right]^{-1} (\mathbf{X} - \mathbf{B})^\top. \tag{6.4}$$

$\boldsymbol{\Phi}^\top \boldsymbol{\Phi}$ is invertible if the P vectors $\varphi(\mathbf{x}_p)$ in $\boldsymbol{\Phi}$ are linearly independent. Then, as long as $P \leq N_{\mathcal{F}}$ the C-LS-SVM maps each vector $\mathbf{x}_p$ onto itself:

$$\begin{aligned} \mathbf{W}^\top \varphi(\mathbf{x}_p) \;+\; \mathbf{b} \;&=\; (\mathbf{X} - \mathbf{B})\, \boldsymbol{\Phi}^{+}\, \varphi(\mathbf{x}_p) \;+\; \mathbf{b} \\ &=\; (\mathbf{X} - \mathbf{B})\, \mathbf{e}_p \;+\; \mathbf{b} \\ &=\; \mathbf{x}_p \end{aligned} \tag{6.5}$$

where $\mathbf{e}_p$ stands for the p^{th} column of the identity matrix of size P. ■

Proposition 6.2 proves that C-LS-SVMs can apply the kernel trick to implicitly work with neural networks that have a much larger capacity, without the computational burden of manipulating these large networks.

Figure 6.1a presents the maximal number of equilibria in a C-LS-SVM with various feature maps as a function of the size of the input space N.

2. Setting $\gamma = 0$ and $\eta \neq 0$

From equation (6.3), we derive that the error on a stored $\mathbf{x}_p$ is given by

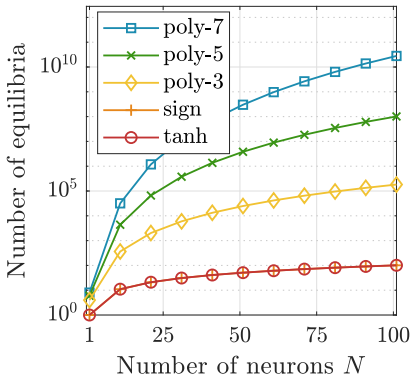
$$\left\| \mathbf{x}_p \; - \; (\mathbf{X} - \mathbf{B}) \left[\; \mathbf{\Phi}^\top \mathbf{\Phi} + \frac{\eta}{\lambda} \mathbf{I}_P \; \right]^{-1} \mathbf{\Phi}^\top \varphi(\mathbf{x}_p) \; - \; \mathbf{b} \right\|_2. \tag{6.6}$$

Figure 6.1b presents an experimental study of the effect of η/λ on (6.6), in which we trained a C-LS-SVM with polynomial kernel ($d = 3$, $t = 1$) on P memories $\mathbf{x}_p$ whose elements are uniformly distributed on $[-1, 1]$. We then measured the largest value of P for which the error (6.6) on each memory was smaller than a tolerance 10^{-4}. It yields two insights.

First, the capacity of the C-LS-SVM decreases as the ratio η/λ increases. Indeed, a stronger regularization yields a stiffer model to store each $\mathbf{x}_p$.

Second, in absolute terms the reduction for a fixed ratio η/λ is constant, i.e. the gap between $\eta/\lambda = 10^{-4}$ and $\eta/\lambda = 10^{-3}$ is $O(10^2)$ for all N, and the gap between $\eta/\lambda = 10^{-4}$ and $\eta/\lambda = 10^{-2}$ is $O(10^3)$ for all N. However, in relative terms the impact of η/λ decreases when N increases.

(a) Number of equilibria that a C-LS-SVM can store ($\gamma = 0$, $\eta = 0$)



(b) Number of equilibria that a C-LS-SVM (poly-3) can store ($\gamma = 0$, $\eta \neq 0$)

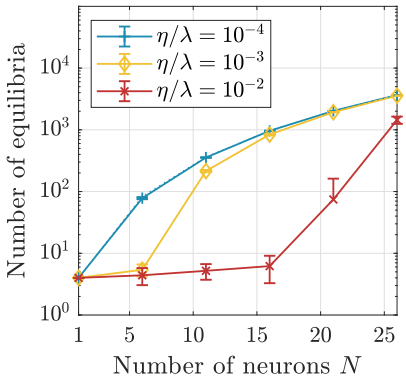


Figure 6.1: Analysis of the maximal number of equlibria in a C-LS-SVM. **(a)** In the case $\gamma = 0$, $\eta = 0$, a C-LS-SVM can store up to $N_{\mathcal{F}}$ equilibria. **(b)** In the case $\gamma = 0$, $\eta \neq 0$, the capacity of a C-LS-SVM is reduced.

3. Setting $\gamma \neq 0$ and $\eta = 0$

Proposition 6.3. *A C-LS-SVM as defined in Definition 5.2 with hyper-parameters $\gamma \neq 0$, $\eta = 0$ can have $N_{\mathcal{F}}/(N+1)$ equilibria $\mathbf{x}_p$, if the vectors $\left\{\varphi(\mathbf{x}_p),\ \left\{\frac{\partial \varphi}{\partial x_n}(\mathbf{x}_p)\right\}_{n=1}^{N}\right\}_{p=1}^{P}$ are linearly independent.*

Proof of Proposition 6.3. Since $\eta = 0$, we derive from system (5.15) that

$$\begin{aligned}\mathbf{W} &= \frac{1}{\eta}\begin{bmatrix}\boldsymbol{\Phi} & \mathbf{F}\end{bmatrix}\begin{bmatrix}\frac{1}{\eta}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \frac{1}{\lambda}\mathbf{I}_P & \frac{1}{\eta}\boldsymbol{\Phi}^\top\mathbf{F} \\ \frac{1}{\eta}\mathbf{F}^\top\boldsymbol{\Phi} & \frac{1}{\eta}\mathbf{F}^\top\mathbf{F} + \frac{1}{\gamma}\mathbf{I}_{NP}\end{bmatrix}^{-1}\begin{bmatrix}(\mathbf{X}-\mathbf{B})^\top \\ \mathbf{0}_{NP\times N}\end{bmatrix} \\ &= \begin{bmatrix}\boldsymbol{\Phi} & \mathbf{F}\end{bmatrix}\left(\begin{bmatrix}\boldsymbol{\Phi}^\top \\ \mathbf{F}^\top\end{bmatrix}\begin{bmatrix}\boldsymbol{\Phi} & \mathbf{F}\end{bmatrix}\right)^{-1}\begin{bmatrix}(\mathbf{X}-\mathbf{B})^\top \\ \mathbf{0}_{NP\times N}\end{bmatrix}.\end{aligned}$$

As long as $\begin{bmatrix}\boldsymbol{\Phi} & \mathbf{F}\end{bmatrix} \in \mathbb{R}^{N_{\mathcal{F}}\times(N+1)P}$ has linearly independent columns, i.e. $N_{\mathcal{F}} \geq (N+1)P$ and the columns of $\boldsymbol{\Phi}$, $\mathbf{F}$ are linearly independent, the inverse is computable. The C-LS-SVM then maps each $\mathbf{x}_p$ onto itself:

$$\begin{aligned}\mathbf{W}^\top\varphi(\mathbf{x}_p) + \mathbf{b} &= \begin{bmatrix}\mathbf{X}-\mathbf{B} & \mathbf{0}_{N\times NP}\end{bmatrix}\begin{bmatrix}\boldsymbol{\Phi} & \mathbf{F}\end{bmatrix}^+\varphi(\mathbf{x}_p) + \mathbf{b} \\ &= \begin{bmatrix}\mathbf{X}-\mathbf{B} & \mathbf{0}_{N\times NP}\end{bmatrix}\mathbf{e}_p + \mathbf{b} \\ &= \mathbf{x}_p\end{aligned}$$

where $\mathbf{e}_p$ is the p^{th} column of the identity matrix of size $(N+1)P$. ■

Proposition 6.3 highlights the importance of the ability of C-LS-SVMs to work with high dimensional feature spaces thanks to the kernel trick.

Namely, C-LS-SVMs with feature maps that have small feature spaces are strongly affected by the condition that $P \leq N_{\mathcal{F}}/(N+1)$ when $\eta = 0$. For instance, $\tanh(\cdot)$ and $\text{sign}(\cdot)$ have a feature space of size $N_{\mathcal{F}} = N$. Such a C-LS-SVM thus cannot store an equilibrium since $N/(N+1) < 1$.

Conversely, C-LS-SVMs with feature maps that have large feature spaces are slightly affected by the condition that $P \leq N_{\mathcal{F}}/(N+1)$ when $\eta = 0$. For instance, the capacity of a C-LS-SVM of size $N = 100$ with polynomial kernel of degree 5 decreases from 10^8 to 10^6, which is still much more than the $0.14\,N = 14$ memories in a Hopfield network of that size.

4. Setting $\gamma \neq 0$ and $\eta \neq 0$

Once both hyper-parameters are non-zero, their impact is less obvious. Nevertheless, experiments confirm that our earlier conclusions still hold: the hyper-parameter η progressively reduces the capacity as η/λ increases, while γ reduces the capacity by a fixed amount independently of its size.

6.2.2 Local stability of the equilibria stored in C-LS-SVMs

Because the update equation of a C-LS-SVM is linear in the weights $\mathbf{W}$, we can derive conditions on the hyper-parameters λ, γ, η to ensure that the largest singular value of the Jacobian in a point $\mathbf{x}^*$ is less that one.

We derive them, first for a C-LS-SVM with an explicit feature map $\varphi(\cdot)$, and then for one with an implicit feature map defined by a kernel $k(\cdot,\cdot)$.

Both derivations rely on an important relation, expressed in Lemma 6.1.

Lemma 6.1. *For matrices $\mathbf{A} \in \mathbb{R}^{M\times N}$ and $\mathbf{B} \in \mathbb{R}^{N\times P}$ it holds that*

$$\sigma_{\max}(\mathbf{A}\,\mathbf{B}) \;\leq\; \sigma_{\max}(\mathbf{A})\,\sigma_{\max}(\mathbf{B}), \tag{6.7}$$

where $\sigma_{\max}(\cdot) \in \mathbb{R}^+$ denotes the largest singular value of a matrix.

Proof of Lemma 6.1. For $\mathbf{A} \in \mathbb{R}^{M\times N}$, $\mathbf{B} \in \mathbb{R}^{N\times P}$, any $\mathbf{v} \in \mathbb{R}^P$ satisfies

$$\|\,\mathbf{A}\,\mathbf{B}\,\mathbf{v}\,\|_2 \leq\; \sigma_{\max}(\mathbf{A})\,\|\,\mathbf{B}\,\mathbf{v}\,\|_2 \leq\; \sigma_{\max}(\mathbf{A})\,\sigma_{\max}(\mathbf{B})\,\|\,\mathbf{v}\,\|_2.$$

If $\mathbf{v}$ is the first right singular vector of $\mathbf{AB}$, we obtain expression (6.7). ■

1. For an explicit feature map

In the case that the C-LS-SVM is designed with an explicit feature map, Proposition 6.4 presents a condition on the hyper-parameters to ensure that a C-LS-SVM locally contracts space around an arbitrary point $\mathbf{x}^*$.

Proposition 6.4. *Given a C-LS-SVM as defined in Definition 5.2. Then, a sufficient condition on the hyper-parameters λ, γ and η such that the C-LS-SVM would contract space around a point $\mathbf{x}^* \in \mathbb{R}^N$ is*

$$\lambda \;<\; \frac{\eta - \gamma\,\sigma_c}{\sigma_a \times \sigma_b + \sigma_d} \tag{6.8}$$

with

$$\sigma_a \;=\; \sigma_{\max}\Big(\;\frac{\partial\varphi}{\partial\mathbf{x}}(\mathbf{x}^*)\;\Big) \tag{6.9a}$$

$$\sigma_b \;=\; \sigma_{\max}\Big(\;\mathbf{\Phi}\Big(\mathbf{I}_P - \frac{1}{P}\mathbf{1}_{P\times P}\Big)\mathbf{X}^\top\;\Big) \tag{6.9b}$$

$$\sigma_c \;=\; \sigma_{\max}\Big(\;\mathbf{F}\mathbf{F}^\top\;\Big) \tag{6.9c}$$

$$\sigma_d \;=\; \sigma_{\max}\Big(\;\mathbf{\Phi}\Big(\mathbf{I}_P - \frac{1}{P}\mathbf{1}_{P\times P}\Big)\mathbf{\Phi}^\top\;\Big). \tag{6.9d}$$

To facilitate the reading, we have moved the 2-page proof to Appendix A. Essentially, it involves using Lemma 6.1 on the model's Jacobian in $\mathbf{x}^*$.

2. For an implicit feature map

In the case that the C-LS-SVM is designed with an implicit feature map, Proposition 6.5 presents a condition on the hyper-parameters to ensure that a C-LS-SVM locally contracts space around an arbitrary point $\mathbf{x}^*$.

Proposition 6.5. *Given a C-LS-SVM as defined in Definition 5.2. Then, a sufficient condition on the hyper-parameters λ, γ and η such that the C-LS-SVM would contract space around a point $\mathbf{x}^* \in \mathbb{R}^N$ is*

$$\max(\lambda, \gamma) \; < \; \frac{\eta}{\sigma_a \times \sigma_b + \sigma_c} \tag{6.10}$$

with

$$\sigma_a \; = \; \sigma_{\max}\left(\begin{bmatrix} \boldsymbol{\Phi}^\top \dfrac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}^*) \\ \mathbf{F}^\top \dfrac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}^*) \end{bmatrix} \right) \tag{6.11a}$$

$$\sigma_b \; = \; \sigma_{\max}\left(\begin{bmatrix} \left(\mathbf{I}_P - \dfrac{1}{P}\mathbf{1}_{P\times P}\right)\mathbf{X}^\top \\ \mathbf{0}_{NP\times N} \end{bmatrix} \right) \tag{6.11b}$$

$$\sigma_c \; = \; \sigma_{\max}\left(\begin{bmatrix} \left(\mathbf{I}_P - \dfrac{1}{P}\mathbf{1}_{P\times P}\right)\boldsymbol{\Phi}^\top\boldsymbol{\Phi} & \left(\mathbf{I}_P - \dfrac{1}{P}\mathbf{1}_{P\times P}\right)\boldsymbol{\Phi}^\top\mathbf{F} \\ \mathbf{F}^\top\boldsymbol{\Phi} & \mathbf{F}^\top\mathbf{F} \end{bmatrix} \right). \tag{6.11c}$$

To facilitate the reading, we have moved the 2-page proof to Appendix A. Essentially, it involves using Lemma 6.1 on the model's Jacobian in $\mathbf{x}^*$.

Proposition 6.4 thus bounds λ as a function of η and γ to ensure that space around a point $\mathbf{x}^*$ contracts under the C-LS-SVM update equation. This bound depends on the sensitivity of the feature map in $\mathbf{x}^*$ by σ_a, the place of each memory in the input and the feature space by σ_b, σ_d, and also the global sensitivity of the feature map in each memory by σ_c.

Proposition 6.5 then bounds λ and γ as a function of η to ensure that space around a point $\mathbf{x}^*$ contracts under the C-LS-SVM update equation. This bound depends on the sensitivity of the feature map in $\mathbf{x}^*$ by σ_a, the place of each memory in the input and the feature space by σ_b, σ_c, and also the global sensitivity of the feature map in each memory by σ_c.

Both bounds agree with the intuition that λ should vary inversely as σ_a. Namely, when the feature map is inherently sensitive in $\mathbf{x}^*$, σ_a is large. The model should then direct its efforts towards the local stability objective rather than the equilibrium objective. This implies a small λ. Conversely, if the feature map is inherently insensitive in $\mathbf{x}^*$, σ_a is small. The model can then allocate more efforts to the equilibrium objective.

6.3 Experiments

6.3.1 Comparing different feature maps and kernel functions

This experiment compares the impact of several feature maps and kernels.

Setup

We train eight models on a set $\{x_p \in \mathbb{R}\}_{p=1}^{3}$ ($\lambda = 10^2$, $\gamma = 10^2$, $\eta = 10^{-2}$). The first two models rely on the explicit sign and tanh feature maps. The next three models rely on a polynomial kernel of degree 3, 5 and 10. The last three models rely on an RBF kernel of bandwidth 0.5, 5 and 10.

Results

Figures 6.2a to 6.2h visualize the update equations of the eight models.

Discussion

The sign and tanh feature maps have the same flaws as Hopfield networks. First, the symmetry of the update equation generates unwanted attractors. For instance, if a memory x_p is an equilibrium, then $-x_p$ is one as well. Further, the rigidity of the update equation limits the memory capacity. For instance, it is hard to store non-symmetrically distributed memories. Besides, the memory abruptly breaks down once its capacity is exceeded.

The result of a polynomial kernel depends on the size of the degree d because a degree d polynomial can cross the identity at most d times, with a derivative that is half of the times < 1, and half of the times > 1. So with P memories, if $d < P$, not all of the memories can be equilibria. If $P \leq d < 2P - 1$, all of the memories can be equilibria, but not stable. If $d \geq 2P - 1$, all of the memories can be stable equilibria of the system. If $d \gg P$, the basin of attraction of the outer memories strongly shrinks. In general, the state can diverge since the update equation is not bounded.

The result of an RBF kernel depends on the size of the bandwidth σ because σ defines the scale at which two points are considered similar. If σ is too small, then the Gaussian in each memory is too narrow. Consequently, each memory attracts a narrow region of the state space and most states are mapped to the bias term, i.e. a form of mean memory. Else, if σ is too large, then the Gaussian in each memory is too wide. Consequently, each memory attracts a wide region of the state space and some memories lose their stability in favor of other more attractive ones. In general, the state cannot diverge since the update equation is bounded. Besides, the memory capacity of a model with RBF kernel is unlimited.

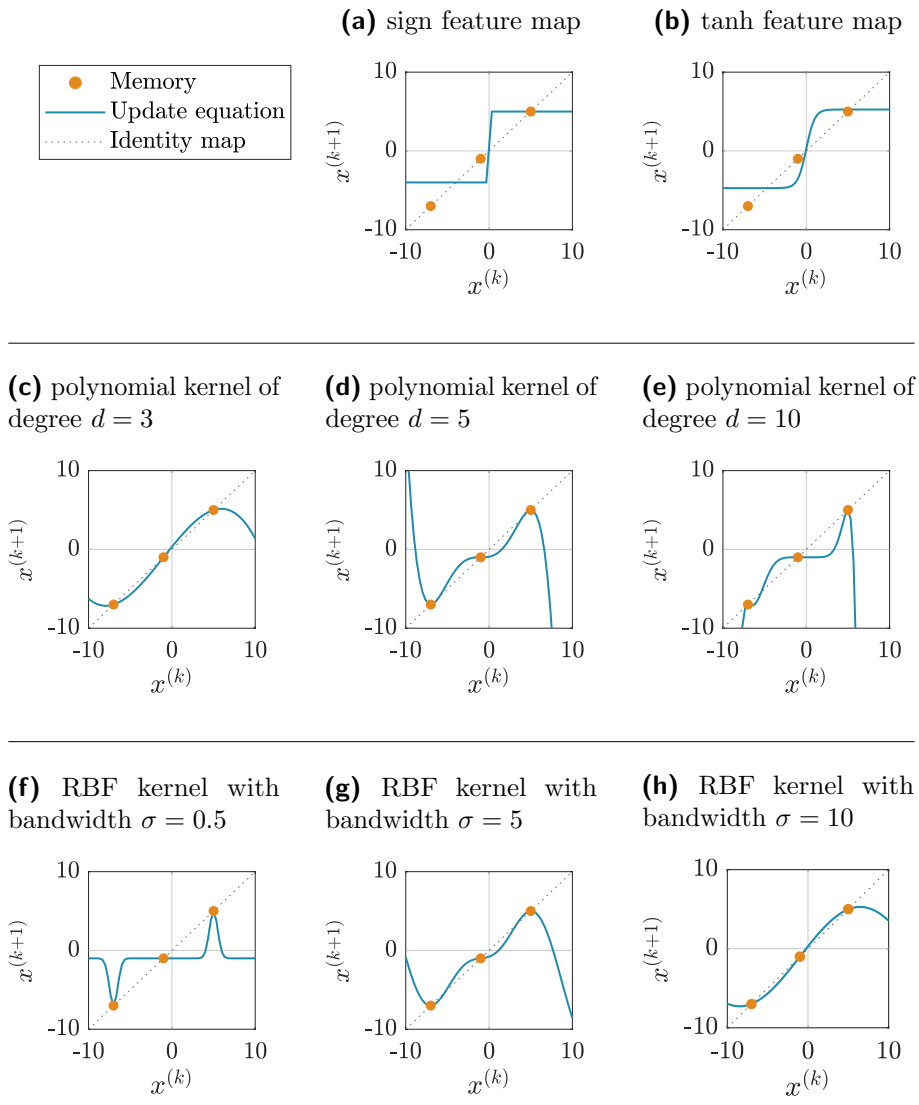


Figure 6.2: Update equations with different feature maps and kernels. **(a)-(b)** The sign and tanh feature maps behave like Hopfield networks. **(c)-(e)** The result with a polynomial kernel depends on the degree d. Also, the state could diverge because the update equation is unbounded. **(f)-(h)** The result with an RBF kernel depends on the bandwidth σ. Also, the state cannot diverge because the update equation is bounded.

6.4 Conclusion

We analysed C-LS-SVMs from the perspective of Hopfield networks. That is to say, we situated C-LS-SVMs within the Hopfield framework, we discussed the maximal number of equilibria a C-LS-SVM can store, and we derived conditions to guarantee that a point is locally attractive.

First, Proposition 6.1 showed that C-LS-SVMs can be interpreted as Hopfield networks trained with the KAM learning rule on noisy memories, which pushes the memories to be locally stable equilibria of the system. Additionally, this result provided a compact formulation for the weights in terms of η/λ controlling the amount of parameter norm regularization, and in terms of γ/λ controlling the amount of regularization with noise.

Then, we studied the memory capacity of a C-LS-SVM by considering the number of equilibria it can store, and the stability of each equilibrium.

Proposition 6.2 proved that C-LS-SVMs can store up to $N_{\mathcal{F}}$ equilibria. However, we demonstrated with an experiment as well as Proposition 6.3 that as the amount of regularization increases, this capacity decreases. Even so, this decrease is not a problem, as C-LS-SVMs can apply the kernel trick to train a neural network that has a much larger capacity, without the computational burden of manipulating this larger network.

Proposition 6.4 and Proposition 6.5 presented conditions on the hyper-parameters λ, η and γ to guarantee that a point $\mathbf{x}^*$ is locally attractive. These conditions depended on the sensitivity of the feature map in $\mathbf{x}^*$, the position of the stored memories in the input and the feature space, as well as the global sensitivity of the feature map in the stored memories. Also, they agreed with the intuition that the focus on the equilibrium objective must vary inversely as the sensitivity of the feature map in $\mathbf{x}^*$.

Finally, we discussed an experiment that contrasted the update equations of C-LS-SVMs trained with different feature maps and kernel functions. We saw that the sign and tanh feature maps behave as Hopfield networks, that the memory capacity of a polynomial kernel scales with its degree, and that the memory capacity of an RBF kernel relies on its bandwidth. Further, with a polynomial kernel the state of the system could diverge, but with an RBF kernel the update equation is bounded, so the state is too.

In the next chapter, we take a closer look at the C-LS-SVM optimization. In doing so, we obtain a new interpretation for the hyper-parameters, not in terms of controlling a parameter norm and a noise regularization, but instead, in terms of controlling a uniform and biased regularization.

Chapter 7

Analysis from the perspective of LS-SVMs

In this chapter, we study C-LS-SVMs from the perspective of LS-SVMs, i.e. we investigate the working of the C-LS-SVM optimization problem.

Section 7.1 constructs the Bayesian interpretation behind C-LS-SVMs. It provides insights on the regularization in the optimization problem. Specifically, it shows that the problem holds two types of regularization: a uniform regularization that drives all the weights uniformly to zero, as well as a biased regularization that mainly drives to zero the weights that correspond with the most sensitive dimensions of the feature map. The full regularization is a sum of both, weighted by the hyper-parameters.

Section 7.2 presents two experiments to illustrate both regularizations. In experiment 1, we visualize biased regularization on a simple data set. It confirms that the weights associated with the most sensitive dimensions of the feature map have a larger pressure towards zero than the others. In experiment 2, we compare the use of biased and uniform regularization. It indicates that both reduce the sensitivity of the update equation: uniform regularization reduces the sensitivity over the entire input space, but biased regularization reduces the sensitivity only near the memories.

7.1 The Bayesian interpretation of C-LS-SVMs

The Bayesian interpretation of LS-SVMs yields compelling insights on the regularization term in the LS-SVM optimization problem. Specifically, Van Gestel *et al.* [100] show that, given a data set and hyper-parameters, solving the LS-SVM optimization problem for function estimation (3.4) corresponds to finding the parameters with maximal posterior probability if one assumes the prior probability distributions over the parameters as

$$\mathbf{w}_m \sim \mathcal{N}\Big(\mathbf{0}_{N_\mathcal{F}}, \frac{1}{\eta}\, \mathbf{I}_{N_\mathcal{F}} \Big) \tag{7.1a}$$

$$\mathbf{b} \sim \mathcal{N}\Big(\mathbf{0}_M, \sigma_b^2\, \mathbf{I}_M \Big) \text{ with } \sigma_b \to \infty. \tag{7.1b}$$

Hence, regularization generates a pressure on the weights towards zero.

Because C-LS-SVMs are similar to LS-SVMs for function estimation, we can develop a similar Bayesian interpretation for C-LS-SVMs which also provides insights on the effect of regularization on the weights. Proposition 7.1 summarizes the Bayesian interpretation of C-LS-SVMs.

Proposition 7.1. *Given a data set* $\mathcal{D} = \{(\mathbf{x}_p \in \mathbb{R}^N,\ \mathbf{y}_p \in \mathbb{R}^M)\}_{p=1}^P$, *hyper-parameters* $\lambda, \gamma, \eta \in \mathbb{R}^+$, *and a model* $\mathcal{M}$ *with feature map* $\varphi(\cdot) : \mathbb{R}^N \to \mathbb{R}^{N_\mathcal{F}}$ *or kernel* $k(\cdot,\cdot) : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$, *then solving*

$$\begin{aligned} \underset{\mathbf{w}_m, \mathbf{b}, \mathbf{e}_p, \mathbf{J}_p}{\text{minimize}} \quad & \frac{\lambda}{2} \sum_{p=1}^P \| \mathbf{e}_p \|_2^2 + \frac{\gamma}{2} \sum_{p=1}^P \| \mathbf{J}_p \|_\mathrm{F}^2 + \frac{\eta}{2} \sum_{m=1}^M \| \mathbf{w}_m \|_2^2 \\ \text{subject to} \quad & \mathbf{e}_p = \mathbf{y}_p - \mathbf{W}^\top \varphi(\mathbf{x}_p) - \mathbf{b} \\ & \mathbf{J}_p = \mathbf{W}^\top \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}_p) \\ & 1 \leq p \leq P \end{aligned} \tag{7.2}$$

corresponds to finding the maximum a posteriori (MAP) parameters if one assumes the following prior distributions over the parameters

$$\mathbf{w}_m \sim \mathcal{N}\left(\mathbf{0}_{N_\mathcal{F}}, \left[\gamma \sum_{p=1}^P \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}_p)\, \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}_p)^\top + \eta\, \mathbf{I}_{N_\mathcal{F}} \right]^{-1} \right) \tag{7.3a}$$

$$\mathbf{b} \sim \mathcal{N}\Big(\mathbf{0}_M, \sigma_b^2\, \mathbf{I}_M \Big) \text{ with } \sigma_b \to \infty. \tag{7.3b}$$

To facilitate the reading, we have moved the 3-page proof to Appendix A. Essentially, it involves inserting the priors in (7.3) into the MAP formula.

Proposition 7.1 indicates that the prior over the bias term is uniform. Hence, before the data is known each vector $\mathbf{b}$ has the same probability.

The prior over the weights on the other hand, is normal with mean zero. Therefore, the regularization is a pressure on the weights towards zero. The covariance matrix then indicates how the pressure is distributed over each of the elements in $\mathbf{w}_m$. This matrix contains two components:

$$\mathbf{w}_m \sim \mathcal{N}\left(\mathbf{0}_{N_\mathcal{F}}, \left[\gamma \overbrace{\sum_{p=1}^{P} \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}_p) \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}_p)^\top}^{\text{Biased}} + \eta \overbrace{\mathbf{I}_{N_\mathcal{F}}}^{\text{Uniform}} \right]^{-1} \right).$$

The total regularization of the C-LS-SVM weights is thus a combination of a uniform regularization caused by the identity matrix, like in (7.1a), as well as a biased regularization caused by the product of the Jacobians.

1. The uniform regularization

Similarly to the LS-SVM framework, uniform regularization pushes all the elements of $\mathbf{w}_m$ to zero with the same pressure in all the directions.

In other words, if the prior distribution only contained the matrix $\mathbf{I}_{N_\mathcal{F}}$, the ellipses formed by the weights with equal prior probability, which are

$$\mathcal{E} = \{ \mathbf{w} \in \mathbb{R}^{N_\mathcal{F}} \mid \mathbf{w}^\top \mathbf{I}_{N_\mathcal{F}} \mathbf{w} = c \text{ for } c \in \mathbb{R}^+ \}$$

would be spheres in $\mathbb{R}^{N_\mathcal{F}}$ centered in $\mathbf{0}_{N_\mathcal{F}}$, like on the left of figure 7.1.

2. The biased regularization

Biased regularization pushes the elements of $\mathbf{w}_m$ to zero with a different pressure in the different directions. Concretely, let us define the matrix

$$\begin{aligned} \mathbf{S} &= \sum_{p=1}^{P} \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}_p) \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}_p)^\top \\ &= \begin{bmatrix} \sum_p \frac{\partial \varphi_1}{\partial \mathbf{x}}(\mathbf{x}_p)^\top \frac{\partial \varphi_1}{\partial \mathbf{x}}(\mathbf{x}_p) & \cdots & \sum_p \frac{\partial \varphi_1}{\partial \mathbf{x}}(\mathbf{x}_p)^\top \frac{\partial \varphi_{N_\mathcal{F}}}{\partial \mathbf{x}}(\mathbf{x}_p) \\ \vdots & \ddots & \vdots \\ \sum_p \frac{\partial \varphi_{N_\mathcal{F}}}{\partial \mathbf{x}}(\mathbf{x}_p)^\top \frac{\partial \varphi_1}{\partial \mathbf{x}}(\mathbf{x}_p) & \cdots & \sum_p \frac{\partial \varphi_{N_\mathcal{F}}}{\partial \mathbf{x}}(\mathbf{x}_p)^\top \frac{\partial \varphi_{N_\mathcal{F}}}{\partial \mathbf{x}}(\mathbf{x}_p) \end{bmatrix}. \end{aligned}$$

This sample covariance $\mathbf{S}$ thus measures the sensitivity of each dimension of the feature map $\varphi(\cdot)$ over the space spanned by the data points $\mathbf{x}_p$.

In particular, if $\varphi_1(\cdot)$ is more sensitive to $\mathbf{x}$ than the other dimensions, then the terms in matrix $\mathbf{S}$ that contain $\partial\varphi_1/\partial\mathbf{x}$ are relatively larger. Consequently, the probability distribution with covariance matrix $\mathbf{S}^{-1}$ is relatively thinner along the direction of w_1. This indicates that w_1 experiences a higher pressure towards zero than the other elements of $\mathbf{w}$.

The biased regularization thus pressures most towards zero the weights that correspond to the most sensitive dimensions of the feature map. This is sensible because the C-LS-SVM optimization problem in (5.7) aims for a model that is insensitive to variations around each data point. So reducing the weights associated with the sensitive dimensions of $\varphi(\cdot)$, reduces the sensitivity of the update equation $\mathbf{x}^{(k+1)} = \mathbf{W}^\top\varphi(\mathbf{x}^{(k)}) + \mathbf{b}$.

In other words, if the prior distribution only contained the matrix $\mathbf{S}^{-1}$, the ellipses formed by the weights with equal prior probability, which are

$$\mathcal{E} \;=\; \{\, \mathbf{w} \in \mathbb{R}^{N_\mathcal{F}} \;\mid\; \mathbf{w}^\top\, \mathbf{S}^{-1}\, \mathbf{w} \;=\; c \text{ for } c \in \mathbb{R}^+ \,\}$$

would be wide along the directions of $\mathbb{R}^{N_\mathcal{F}}$ associated with the least sensitive dimensions of the feature map, and thin along the directions of $\mathbb{R}^{N_\mathcal{F}}$ associated with the most sensitive dimensions of the feature map.

The total regularization

The total regularization of the C-LS-SVM weights is a sum of uniform and biased regularization, weighted by the hyper-parameters γ and η. When $\eta \gg \gamma$, it is mostly uniform. So all the weights are pushed to zero. When $\eta \ll \gamma$, it is mostly biased. So mainly the weights associated with the most sensitive dimensions of the feature map are pushed to zero.

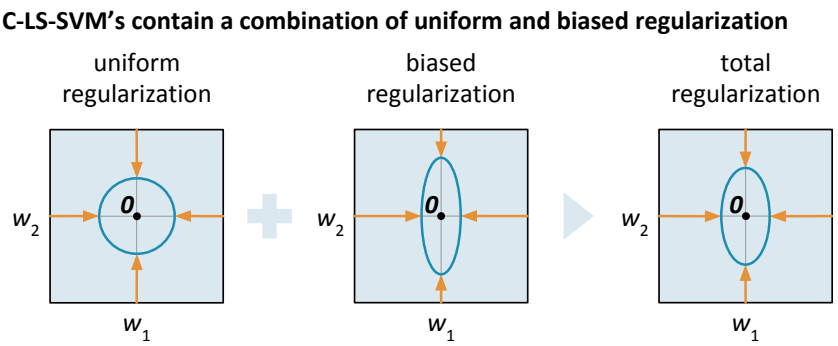


Figure 7.1: Ellipses (–) formed by weights with equal prior probability. Uniform regularization reduces all the weights with the same pressure (–). Biased regularization reduces different weights with different pressures.

7.2 Experiments on the Bayesian interpretation

Further, two experiments illustrate biased and uniform regularization. Experiment 1 shows the impact of biased regularization on the weights. Experiment 2 compares the effect of biased and uniform regularization.

7.2.1 Visualizing the biased regularization

This experiment shows that the biased regularization minimizes the weights associated with the most sensitive dimensions of the feature map.

Setup

We generate three data sets $\{\mathbf{x}_p \in \mathbb{R}\}_{p=1}^{5}$ and choose the feature map

$$\varphi(x) = \begin{bmatrix} x \\ x^2 \end{bmatrix}. \tag{7.4}$$

We then consider a model with only biased regularization ($\gamma = 1$, $\eta = 0$), and plot the ellipses formed by the weights with equal prior probability. These ellipses are the points for which $\mathbf{w}^\top (1\,\mathbf{S} + 0\,\mathbf{I}_2)^{-1}\mathbf{w}$ is constant.

$$\mathbf{S} \;=\; \mathbf{F}\mathbf{F}^\top \;=\; \sum_{p=1}^{P} \frac{\partial \varphi}{\partial x}(\mathbf{x}_p)\, \frac{\partial \varphi}{\partial x}(\mathbf{x}_p)^\top \;=\; \begin{bmatrix} P & \sum_p 2\, x_p \\ \sum_p 2\, x_p & \sum_p 4\, x_p^2 \end{bmatrix}.$$

Results

The top row in figure 7.2 shows the feature space for the three data sets, and the bottom row shows the weight space with the resulting ellipse.

Discussion

The border of the ellipses are weight vectors with equal prior probability. So, a small axis indicates a direction with a large pressure towards 0, and a large axis indicates a direction with a small pressure towards 0.

First, each ellipse is wider along the w_1-axis, than along the w_2-axis. This indicates that w_2 has a larger incentive to approach zero than w_1. That is because $\varphi_2(\cdot)$ is more sensitive to small changes in x than $\varphi_1(\cdot)$. As a result, the derivatives $\mathrm{d}\varphi_2/\mathrm{d}x$ are larger than the derivatives $\mathrm{d}\varphi_1/\mathrm{d}x$, which yields a covariance $\mathbf{S}$ with large variance in the second dimension. The prior with covariance $\mathbf{S}^{-1}$ then has few variance in that dimension.

Further, the inclination of the ellipses confirms that the model minimizes

$$\underbrace{w_1 \frac{\mathrm{d}\varphi_1}{\mathrm{d}x}}_{a} + \underbrace{w_2 \frac{\mathrm{d}\varphi_2}{\mathrm{d}x}}_{b}.$$

Namely, the prior over the weights intends for a and b to compensate. Such as, in figure 7.2a $\mathrm{d}\varphi_1/\mathrm{d}x$ and $\mathrm{d}\varphi_2/\mathrm{d}x$ always have opposite sign. Therefore, to ensure that a and b have opposite sign and compensate, the prior probability is larger for same-sign weights than for opposite-sign. Contrarily, in figure 7.2c $\mathrm{d}\varphi_1/\mathrm{d}x$ and $\mathrm{d}\varphi_2/\mathrm{d}x$ always have the same sign. Therefore, to ensure that a and b have opposite sign and compensate, the prior probability is larger for opposite-sign weights than for same-sign.

Overall, the optimization problem aims for a model whose output is insensitive to small changes in the input space around each data point. For that reason, the weights that correspond to the most sensitive dimensions of the feature map have the largest pressure towards zero. Further, the model drives the dimensions of the feature map whose derivatives have the same sign to have weights of opposite signs, whereas dimensions with opposite-sign derivatives should have same-sign weights.

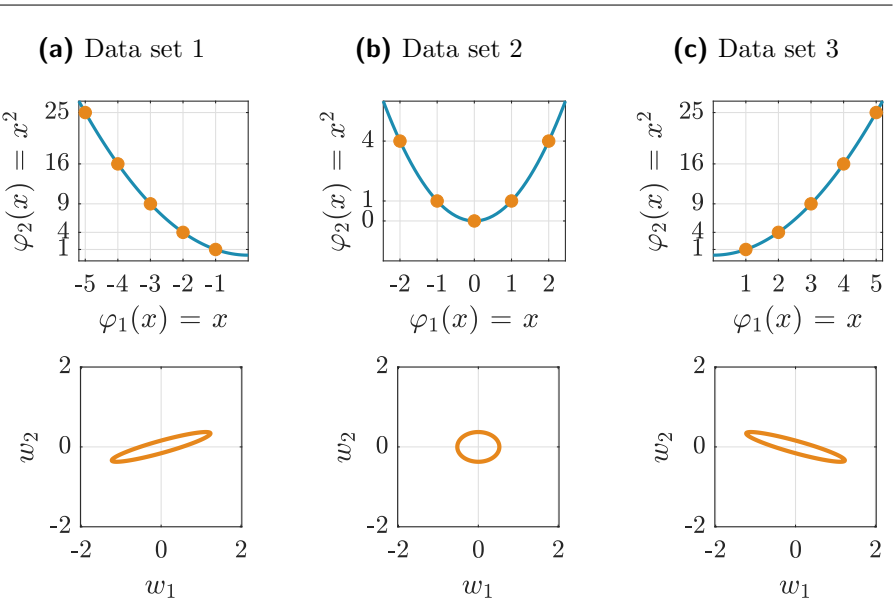


Figure 7.2: Study of the prior distribution over the weights w_1 and w_2 for a model that contains only biased regularization ($\gamma = 1$, $\eta = 0$). **Top.** The one-dimensional data points (●) are represented in the two-dimensional feature space, along with the feature map (–) from (7.4). **Bottom.** The ellipses (–) connect the weights with equal prior probability. **Overall.** The weight associated with the most sensitive dimension of the feature map undergoes a larger pressure towards zero than the other, because each ellipse is thinner along the w_2-axis, than along the w_1-axis.

7.2.2 The effect of the biased and uniform regularization

This experiment shows that both uniform and biased regularization flatten the update equation of a C-LS-SVM, though at different scales.

Setup

We train a C-LS-SVM with RBF kernel ($\sigma = 5$) on a data set $\{x_p \in \mathbb{R}\}_{p=1}^{3}$. We then fix $\lambda = 10^2$, and compare the cases $\gamma \ll \eta$, $\eta = \gamma$ and $\gamma \gg \eta$.

Results

Figure 7.3 visualizes the update equation of the three different models.

Discussion

A large γ or a large η flatten the update equation, but at different scales.

When $\gamma \ll \eta$, the update equation is flattened on the entire domain, because there is a uniform pressure on all the weights to approach zero. If η increases to 10^7, the update equation even becomes completely flat.

When $\gamma \gg \eta$, the update equation is flattened only in the data point, because only the weights that impact the Jacobian in x_p are regularized. If γ increases to 10^7, the update equation does not change anymore, as it is already flat around each data point (local stability objective is zero).

Thus, the memory capacity of the C-LS-SVM decreases as η increases. i.e. the model in figure 7.3a loses two equilibria, compared to figure 7.3b. Conversely, the capacity of the C-LS-SVM is not affected as γ increases, i.e. the three equilibria even become stable for the model in figure 7.3c.

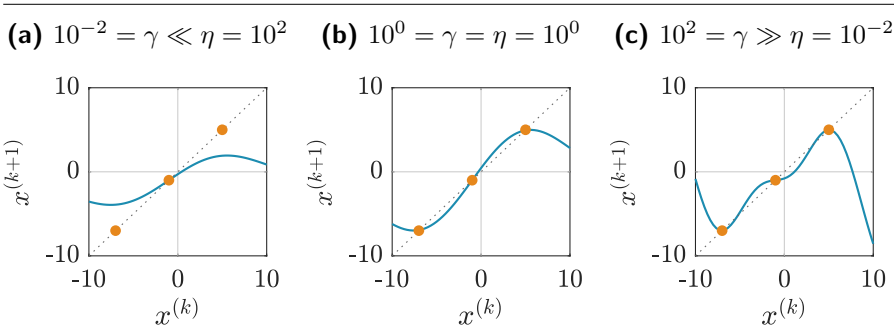


Figure 7.3: Comparison of **(a)** uniform and **(c)** biased regularization. **Overall.** Both flatten the update equation (–), but at a different scale. Uniform regularization flattens the update equation on the entire domain since it stems from the regularization term in the optimization (5.7). Biased regularization flattens the update equation in the data points (●) since it stems from the local stability objective in the optimization (5.7).

7.3 Conclusion

We have analysed C-LS-SVMs from the point of view of LS-SVMs. Namely, we derived the Bayesian interpretation behind C-LS-SVMs, which clarified the regularization in the C-LS-SVM optimization problem.

First, we showed that C-LS-SVMs contain two types of regularization: a uniform regularization that drives all the weights uniformly to zero, as well as a biased regularization that mainly drives to zero the weights that correspond with the most sensitive dimensions of the feature map. The full regularization is a sum of both, weighted by the hyper-parameters.

Afterwards, two experiments illustrated biased and uniform regularization. In experiment 1, we constructed a feature map to a 2D feature space. Because the second dimension was the one most sensitive to the input, the weight w_2 experienced a larger pressure to zero than the weight w_1. In experiment 2, we showed that both uniform and biased regularization reduce the sensitivity of the update equation, but at different scales. Uniform regularization reduces the sensitivity over the entire input space, while biased regularization reduces the sensitivity only in the memories.

In the next chapter, we construct deep and generative C-LS-SVMs. Specifically, we will stack C-LS-SVMs to obtain more expressive models, and we will generate new memories that are similar to the stored ones.

Chapter 8

Analysis from the perspective of C-AEs

In this chapter, we study C-LS-SVMs from the point of view of C-AEs, i.e. we build deep and generative variants inspired by theory on C-AEs.

Section 8.1 first builds the optimization problem for deep C-LS-SVMs, and then works out the optimal parameters with the Lagrange method. Since shallow C-LS-SVMs are convex, we can train a deep C-LS-SVM by an alternating convex process with no vanishing or exploding gradients.

Section 8.2 presents two experiments that illustrate deep C-LS-SVMs. In experiment 1, we show the potential advantages of deep C-LS-SVMs. Namely, deeper models can compactly represent more complex functions. In experiment 2, we confirm that the training algorithm works properly.

Section 8.3 constructs the random walk behind a generative C-LS-SVM, i.e. a C-LS-SVM able to generate new data points on the data manifold. It relies on the fact that the perpendiculars to the manifold are contracted, whereas the tangents to the manifold are nor contracted, nor expanded.

Section 8.4 presents two experiments to illustrate generative C-LS-SVMs. In experiment 1, we show that the Jacobian captures the data manifold, as its first and last singular vectors are tangent and perpendicular to it. In experiment 2, we confirm that the generating algorithm works properly.

8.1 Deep C-LS-SVMs

Today, most neural networks contain tens or hundreds of layers [59], because deeper networks can compactly represent more complex functions. This also holds for autoencoders: the deeper they are, the more expressive their reconstruction function is, and the better they perform [54, 78, 105].

A difficulty that remains however is the training of these deep networks. As discussed in chapter 4, stacking layers generates a composition of many nonlinear functions which hinders gradient-based training algorithms.

In this section we develop deep C-LS-SVMs in order to increase the expressiveness and so improve the performance of shallow C-LS-SVMs. We do so, first by defining the deep C-LS-SVM optimization problem, then by working out the optimal parameters with the Lagrange method.

Eventually, we show that because of the convexity of shallow C-LS-SVMs, we can train deep C-LS-SVMs through an alternating convex process that is not hindered by the composition of the many nonlinear functions.

8.1.1 The update equation

By stacking L shallow C-LS-SVMs, we obtain the deep update equation

$$\begin{aligned} \mathbf{h}_{(0)} &= \mathbf{x}^{(k)} \\ \mathbf{h}_{(l)} &= \mathbf{W}_{(l)}^{\top}\varphi_{(l)}(\mathbf{h}_{(l-1)}) + \mathbf{b}_{(l)} \qquad 1 \leq l \leq L \\ \mathbf{x}^{(k+1)} &= \mathbf{h}_{(L)}. \end{aligned} \tag{8.1}$$

The vector $\mathbf{h}_{(l)} \in \mathbb{R}^{N_{(l)}}$ represents the hidden state in the l^{th} C-LS-SVM.

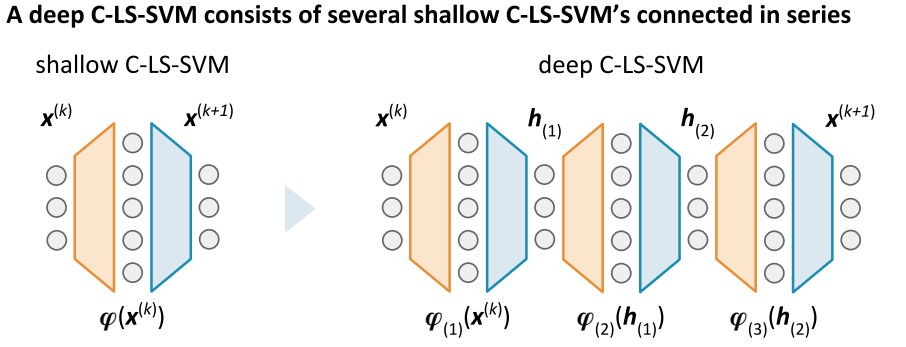


Figure 8.1: Example of a deep C-LS-SVM that contains three layers.

8.1.2 The optimization problem

The generalization of the optimization problem derived in section 5.2.1 for a deep C-LS-SVM that aims to store P memories $\{\mathbf{x}_p \in \mathbb{R}^N\}_{p=1}^P$ is

$$\begin{aligned}
\underset{\substack{\mathbf{W}_{(l)},\mathbf{b}_{(l)},\\ \mathbf{e}_p,\mathbf{J}_p}}{\text{minimize}} \quad & \frac{\lambda}{2}\overbrace{\sum_{p=1}^{P} \| \mathbf{e}_p \|_2^2}^{\text{Equilibrium}} + \frac{\gamma}{2}\overbrace{\sum_{p=1}^{P} \| \mathbf{J}_p \|_{\mathrm{F}}^2}^{\text{Local stability}} + \sum_{l=1}^{L} \frac{\eta_{(l)}}{2} \overbrace{\sum_{n=1}^{N_{(l)}} \| \mathbf{w}_{(l),n} \|_2^2}^{\text{Regularization}} \\
\text{subject to} \quad & \mathbf{e}_p = \mathbf{x}_p - \mathbf{W}_{(L)}^\top \varphi_{(L)}(\cdots(\mathbf{W}_{(1)}^\top \varphi_{(1)}(\mathbf{x}_p) + \mathbf{b}_{(1)})\cdots) - \mathbf{b}_{(L)} \\
& \mathbf{J}_p = \mathbf{W}_{(L)}^\top \frac{\partial \varphi_{(L)}}{\partial \mathbf{x}}(\mathbf{h}_{(L-1)}) \times \cdots \times \mathbf{W}_{(1)}^\top \frac{\partial \varphi_{(1)}}{\partial \mathbf{x}}(\mathbf{x}_p) \\
& 1 \leq p \leq P.
\end{aligned} \tag{8.2}$$

However this optimization problem is relatively complex and not convex, even though it is made of L layers that have a convex training process.

Therefore in order to leverage the convexity of each of the L C-LS-SVMs, we uncouple the optimization problem (8.2) into L optimization problems by introducing, for each memory $\mathbf{x}_p$, L *hidden states* $\{\mathbf{h}_{(l),p} \in \mathbb{R}^{N_{(l)}}\}_{l=1}^L$. $\mathbf{h}_{(l),p}$ then acts as a target for the representation of $\mathbf{x}_p$ in the l^{th} layer.

Further, let us denote the hidden states $\mathbf{h}_{(0),p} = \mathbf{x}_p$ and $\mathbf{h}_{(L),p} = \mathbf{x}_p$. The optimization problem for training a deep C-LS-SVM then becomes

$$\begin{aligned}
\underset{\substack{\mathbf{W}_{(l)},\mathbf{b}_{(l)},\\ \mathbf{e}_{(l),p},\mathbf{J}_{(l),p}}}{\text{minimize}} \quad & \sum_{l=1}^{L} \Bigg[\frac{\lambda_{(l)}}{2} \overbrace{\sum_{p=1}^{P} \| \mathbf{e}_{(l),p} \|_2^2}^{\text{Equilibrium}} + \frac{\gamma_{(l)}}{2} \overbrace{\sum_{p=1}^{P} \| \mathbf{J}_{(l),p} \|_{\mathrm{F}}^2}^{\text{Local stability}} \\
& \qquad + \frac{\eta_{(l)}}{2} \overbrace{\sum_{n=1}^{N_{(l)}} \| \mathbf{w}_{(l),n} \|_2^2}^{\text{Regularization}} \Bigg] \\
\text{subject to} \quad & \mathbf{e}_{(l),p} = \mathbf{h}_{(l),p} - \mathbf{W}_{(l)}^\top \varphi_{(l)}(\mathbf{h}_{(l-1),p}) - \mathbf{b}_{(l)} \\
& \mathbf{J}_{(l),p} = \mathbf{W}_{(l)}^\top \frac{\partial \varphi_{(l)}}{\partial \mathbf{x}}(\mathbf{h}_{(l-1),p}) \\
& 1 \leq l \leq L \\
& 1 \leq p \leq P.
\end{aligned} \tag{8.3}$$

It thus consists of training L C-LS-SVMs to map each $\mathbf{h}_{(l-1),p}$ onto $\mathbf{h}_{(l),p}$, and ensure that the mapping is as contractive as possible around $\mathbf{h}_{(l-1),p}$. So the new equilibrium objective and the new local stability objective have a slightly different interpretation than the one developed in section 5.2.1.

The new equilibrium objective

The update equation (8.1) should map each memory $\mathbf{x}_p$ onto itself. However, instead of directly minimizing the 2-norm of the deep error $\mathbf{e}_p$, the optimization minimizes the 2-norm of the error in each layer $\mathbf{e}_{(l),p}$.

This approach is similar to the Direct Multiple Shooting method (DMS). DMS is a numerical method for solving boundary value problems [28, 50]. It divides the interval over which a solution is sought into smaller intervals, then solves an initial value problem over each of these smaller intervals, and imposes continuity conditions to connect each of the partial solutions.

In a deep C-LS-SVM, each layer constitutes one of the smaller intervals, and the equilibrium objective then imposes the continuity conditions.

The new local stability objective

The update equation (8.1) should be contractive near each memory $\mathbf{x}_p$. Yet instead of minimizing the Frobenius norm of the deep Jacobian $\mathbf{J}_p$, (8.3) minimizes the Frobenius norm of the Jacobian in each layer $\mathbf{J}_{(l),p}$.

Lemma 6.1 now indicates that the new local stability objective minimizes an upper bound for the local stability objective in (8.2) as it holds that

$$\sigma_{\max}(\mathbf{J}_p) \leq \sigma_{\max}(\mathbf{J}_{(L),p}) \times \cdots \times \sigma_{\max}(\mathbf{J}_{(1),p}).$$

So if each layer is contractive, the deep C-LS-SVM is contractive too.

The equilibrium and local stability objectives in a deep C-LS-SVM with two layers

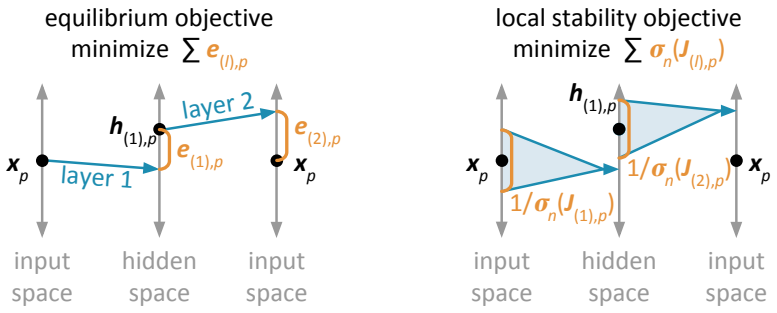


Figure 8.2: Equilibrium and local stability objectives in a deep model. **Left.** The equilibrium objective pushes each layer to map $\mathbf{h}_{(l-1),p}$ to $\mathbf{h}_{(l),p}$. **Right.** The local stability objective pushes each layer to be contractive. **Overall.** If the hidden states are right, $\mathbf{x}_p$ is a locally stable equilibrium.

8.1.3 The optimal parameters

The optimization problem (8.3) consists of L C-LS-SVMs of the form

$$\begin{aligned}
\underset{\substack{\mathbf{W}_{(l)},\mathbf{b}_{(l)},\\ \mathbf{E}_{(l)},\mathbf{J}_{(l)}}}{\text{minimize}} \quad & \frac{\lambda_{(l)}}{2}\,\mathrm{tr}(\mathbf{E}_{(l)}^\top\mathbf{E}_{(l)}) \;+\; \frac{\gamma_{(l)}}{2}\,\mathrm{tr}(\mathbf{J}_{(l)}^\top\mathbf{J}_{(l)}) \;+\; \frac{\eta_{(l)}}{2}\mathrm{tr}(\mathbf{W}_{(l)}^\top\mathbf{W}_{(l)}) \\
\text{subject to} \quad & \mathbf{E}_{(l)} \;=\; \mathbf{H}_{(l)} - \mathbf{W}_{(l)}^\top\boldsymbol{\Phi}_{(l)} - \mathbf{B}_{(l)} \\
& \mathbf{J}_{(l)} \;=\; \mathbf{W}_{(l)}^\top\mathbf{F}_{(l)}.
\end{aligned}$$

By collecting the parameters and hidden states of each layer, in the sets

- $\boldsymbol{\theta} \;=\; \{\, \mathbf{W}_{(l)},\; \mathbf{b}_{(l)},\; \mathbf{L}_{(l)},\; \mathbf{M}_{(l)} \,\}_{l=1}^{L}$
- $\mathbf{H} \;=\; \{\, \{\, \mathbf{h}_{(l),p} \in \mathbb{R}^{N_{(l)}} \,\}_{p=1}^{P} \,\}_{l=1}^{L} \;=\; \{\, \mathbf{H}_{(l)} \,\}_{l=1}^{L}$

we obtain that the Lagrange function of a deep C-LS-SVM with L layers equals the sum of the Lagrange functions of each of the L C-LS-SVMs:

$$\mathcal{L}_{\text{deep}}(\boldsymbol{\theta}, \mathbf{H}) \;=\; \sum_{l=1}^{L} \mathcal{L}_{(l)}(\, \mathbf{W}_{(l)}, \mathbf{b}_{(l)}, \mathbf{E}_{(l)}, \mathbf{J}_{(l)}, \mathbf{L}_{(l)}, \mathbf{M}_{(l)} \,) \tag{8.4}$$

Since the deep Lagrange function equals the sum of the L shallow ones, we reuse the results of section 5.2.2 to obtain two systems for each layer. If the feature map of layer l is explicit, the primal parameters result from

$$\begin{bmatrix} \boldsymbol{\Phi}_{(l)}\boldsymbol{\Phi}_{(l)}^\top + \dfrac{\gamma_{(l)}}{\lambda_{(l)}}\mathbf{F}_{(l)}\mathbf{F}_{(l)}^\top + \dfrac{\eta_{(l)}}{\lambda_{(l)}}\mathbf{I}_{N_{\mathcal{F}(l)}} & \boldsymbol{\Phi}_{(l)}\mathbf{1}_P \\ \mathbf{1}_P^\top\boldsymbol{\Phi}_{(l)}^\top & P \end{bmatrix} \begin{bmatrix} \mathbf{W}_{(l)} \\ \mathbf{b}_{(l)}^\top \end{bmatrix} \;=\; \begin{bmatrix} \boldsymbol{\Phi}_{(l)}\mathbf{H}_{(l)}^\top \\ \mathbf{1}_P^\top\mathbf{H}_{(l)}^\top \end{bmatrix} \tag{8.5}$$

Else if the feature map is implicit, the dual parameters are the result of

$$\begin{bmatrix} \dfrac{1}{\eta_{(l)}}\boldsymbol{\Phi}_{(l)}^\top\boldsymbol{\Phi}_{(l)} + \dfrac{1}{\lambda_{(l)}}\mathbf{I}_P & \dfrac{1}{\eta_{(l)}}\boldsymbol{\Phi}_{(l)}^\top\mathbf{F}_{(l)} & \mathbf{1}_P \\ \dfrac{1}{\eta_{(l)}}\mathbf{F}_{(l)}^\top\boldsymbol{\Phi}_{(l)} & \dfrac{1}{\eta_{(l)}}\mathbf{F}_{(l)}^\top\mathbf{F}_{(l)} + \dfrac{1}{\gamma_{(l)}}\mathbf{I}_{N_{(l)}P} & \mathbf{0}_{N_{(l)}P} \\ \mathbf{1}_P^\top & \mathbf{0}_{N_{(l)}P}^\top & 0 \end{bmatrix} \begin{bmatrix} \mathbf{L}_{(l)}^\top \\ \mathbf{M}_{(l)}^\top \\ \mathbf{b}_{(l)}^\top \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{(l)}^\top \\ \mathbf{0}_{N_{(l)}P\times N_{(l)}} \\ \mathbf{0}_{N_{(l)}}^\top \end{bmatrix} \tag{8.6}$$

8.1.4 The hidden states in H

Training a deep C-LS-SVM thus consists of solving L linear systems, coupled by the hidden states $\mathbf{H}$, i.e. each system holds $\mathbf{h}_{(l-1),p}$ and $\mathbf{h}_{(l),p}$. Therefore, to solve (8.5) or (8.6) the hidden states $\mathbf{H}$ have to be known. In this thesis, we have developed three ways to determine these matrices: (1) explicitly assign them, (2) implicitly learn them with the parameters, or (3) choose them as the different steps of a movement to memorize.

1. Explicitly assigning the hidden states

Explicitly assigning the hidden states consists of optimizing them at another level, and then inserting them in the linear system (8.5) or (8.6).

Algorithm 1 presents the explicit training process of a deep C-LS-SVM.

The major advantages of this approach are that it is easily parallelizable, and that the training process of the deep C-LS-SVM remains convex. Admittedly, the difficulty that remains lies in finding good hidden states.

2. Implicitly learning the hidden states

Implicitly learning the hidden states consists of jointly learning them with the parameters. We do so with the method of auxilliary coordinates, i.e we minimize the deep Lagrange function with coordinate descent, once over the C-LS-SVM parameters, and once over the hidden states. This algorithm provably converges to a local optimum of (8.3) [14, 109].

Algorithm 2 presents the implicit training process of a deep C-LS-SVM. For the optimization of the deep Lagrange function over the parameters, we can apply algorithm 1 with the previously optimized hidden states. For the optimization step over the hidden states $\mathbf{H}$ on the other hand, we can reuse any existing optimization algorithm, e.g. gradient descent.

The principal advantages of this approach over classical deep C-AEs are that the training is alternating convex because algorithm 1 is convex, and that it does not suffer from the composition of the nonlinear functions because it does not apply the chain rule to compute the deep Jacobian. Also, deep C-LS-SVMs can smoothly combine layers with explicit and implicit feature maps which extends the space of models to choose from. Finally, the training algorithm has a high potential for parallelization.

Algorithm 1: Explicit training of a deep C-LS-SVM

Input: Memories $\mathbf{X}$ and hidden states $\mathbf{H}$
Output: Trained deep C-LS-SVM
/* Train each layer independently */
for $l = 1 \rightarrow L$ **do**
  if $\varphi_{(l)}(\cdot)$ *is an explicit feature map* **then**
    /* Construct the matrices with the explicit formulation for $\varphi_{(l)}(\cdot)$ */
    $\boldsymbol{\Phi}_{(l)} = \left[\varphi_{(l)}(\mathbf{h}_{(l-1),p}) \right]_{p=1}^{P}$;
    $\mathbf{F}_{(l)} = \left[\dfrac{\partial \varphi_{(l)}}{\partial \mathbf{x}}(\mathbf{h}_{(l-1),p}) \right]_{p=1}^{P}$;
    Solve (8.5) ;
  else
    /* Construct the matrices with the implicit formulation for $\varphi_{(l)}(\cdot)$ */
    $\boldsymbol{\Phi}_{(l)}^{\top}\boldsymbol{\Phi}_{(l)} = \left[k_{(l)}(\mathbf{h}_{(l-1),p}, \mathbf{h}_{(l-1),p'}) \right]_{p,p'=1}^{P}$;
    $\boldsymbol{\Phi}_{(l)}^{\top}\mathbf{F}_{(l)} = \left[\dfrac{\partial k_{(l)}}{\partial \mathbf{y}}(\mathbf{h}_{(l-1),p}, \mathbf{h}_{(l-1),p'})^{\top} \right]_{p,p'=1}^{P}$;
    $\mathbf{F}_{(l)}^{\top}\boldsymbol{\Phi}_{(l)} = \left[\dfrac{\partial k_{(l)}}{\partial \mathbf{x}}(\mathbf{h}_{(l-1),p}, \mathbf{h}_{(l-1),p'}) \right]_{p,p'=1}^{P}$;
    $\mathbf{F}_{(l)}^{\top}\mathbf{F}_{(l)} = \left[\dfrac{\partial^2 k_{(l)}}{\partial \mathbf{x} \partial \mathbf{y}}(\mathbf{h}_{(l-1),p}, \mathbf{h}_{(l-1),p'}) \right]_{p,p'=1}^{P}$;
    Solve (8.6) ;
  end
end

Algorithm 2: Implicit training of a deep C-LS-SVM

Input: Memories $\mathbf{X}$
Output: Trained deep C-LS-SVM
Initialize $\mathbf{H}^{(0)}$;
/* Perform alternating optimization at most T times */
for $t = 1 \rightarrow T$ **do**
  /* Optimize the parameters with algorithm 1 */
  $\boldsymbol{\theta}^{(t)} = \underset{\boldsymbol{\theta}}{\text{argmin}} \quad \mathcal{L}_{\text{deep}}(\boldsymbol{\theta}^{(t-1)}, \mathbf{H}^{(t-1)})$;
  /* Optimize the hidden states with algorithm of one's choice */
  $\mathbf{H}^{(t)} = \underset{\mathbf{H}}{\text{argmin}} \quad \mathcal{L}_{\text{deep}}(\boldsymbol{\theta}^{(t)}, \mathbf{H}^{(t-1)})$;
  /* Stop training if the algorithm has converged */
  if $\left\| \dfrac{\partial \mathcal{L}_{\text{deep}}}{\partial \mathbf{H}}(\boldsymbol{\theta}^{(t)}, \mathbf{H}^{(t)}) \right\|_2^2 \leq$ *tolerance* **then**
    Stop training ;
  end
end

3. Choosing the steps of a movement as hidden states

By choosing the different steps of discretized movements as hidden states, a deep C-LS-SVM can memorize movements through the input space.

Specifically, consider a set of P movements, each discretized in L steps:

$$\begin{aligned}
\text{movement } 1: &\quad \mathbf{x}_{(1),1} \rightarrow \ldots \rightarrow \mathbf{x}_{(l),1} \rightarrow \ldots \rightarrow \mathbf{x}_{(L),1}, \\
\text{movement } p: &\quad \mathbf{x}_{(1),p} \rightarrow \ldots \rightarrow \mathbf{x}_{(l),p} \rightarrow \ldots \rightarrow \mathbf{x}_{(L),p}, \\
\text{movement } P: &\quad \mathbf{x}_{(1),P} \rightarrow \ldots \rightarrow \mathbf{x}_{(l),P} \rightarrow \ldots \rightarrow \mathbf{x}_{(L),P}.
\end{aligned}$$

The key idea is then that the l^{th} layer of the deep C-LS-SVM memorizes the l^{th} step of each of the movements. Figure 8.3 illustrates this principle. The l^{th} layer thus associates with position $\mathbf{x}_{(l),p}$, the next position $\mathbf{x}_{(l+1),p}$.

The deep C-LS-SVM therefore consists of $L-1$ shallow C-LS-SVMs. It can be trained with algorithm 1 on the hidden states $\mathbf{h}_{(l),p} = \mathbf{x}_{(l),p}$. As a result, the training process is easily parallelized, and remains convex.

Figure 8.4 presents a concrete example, namely writing the word *hello*. Figures 8.4a-8.4b show the movement to memorize, and its discretization. We then train a 46-layered C-LS-SVM with an RBF kernel in each layer, and assign the hyper-parameters to $\sigma = 2$, $\lambda = 10^2$, $\gamma = 10^1$, $\eta = 10^{-2}$. Figures 8.4c-8.4d show that a C-LS-SVM can memorize one or several movements, and then properly reconstruct the memorized movement. Namely, the initial conditions (●) are attracted to the closest movement, and then follow the discretized movement all the way up to the end (✕).

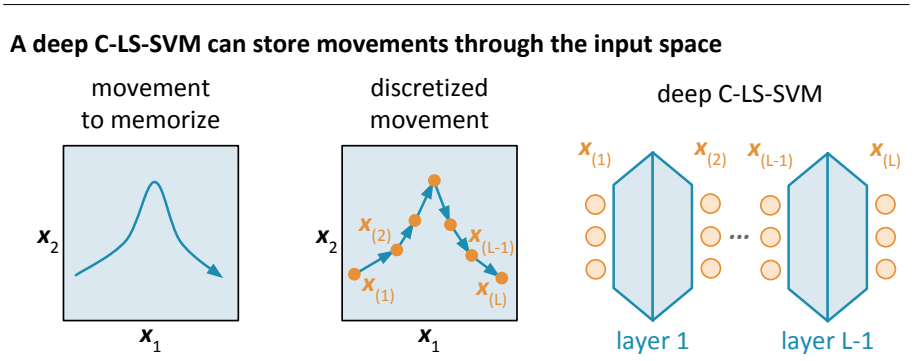


Figure 8.3: A C-LS-SVM stores movements through the input space by choosing the hidden states as the different steps of discretized movements. Thus, each layer of the C-LS-SVM memorizes one step of the movement. In other words, the first layer learns to associate position $\mathbf{x}_{(1)}$ with $\mathbf{x}_{(2)}$, all the way up to the last layer that associates position $\mathbf{x}_{(L-1)}$ with $\mathbf{x}_{(L)}$.

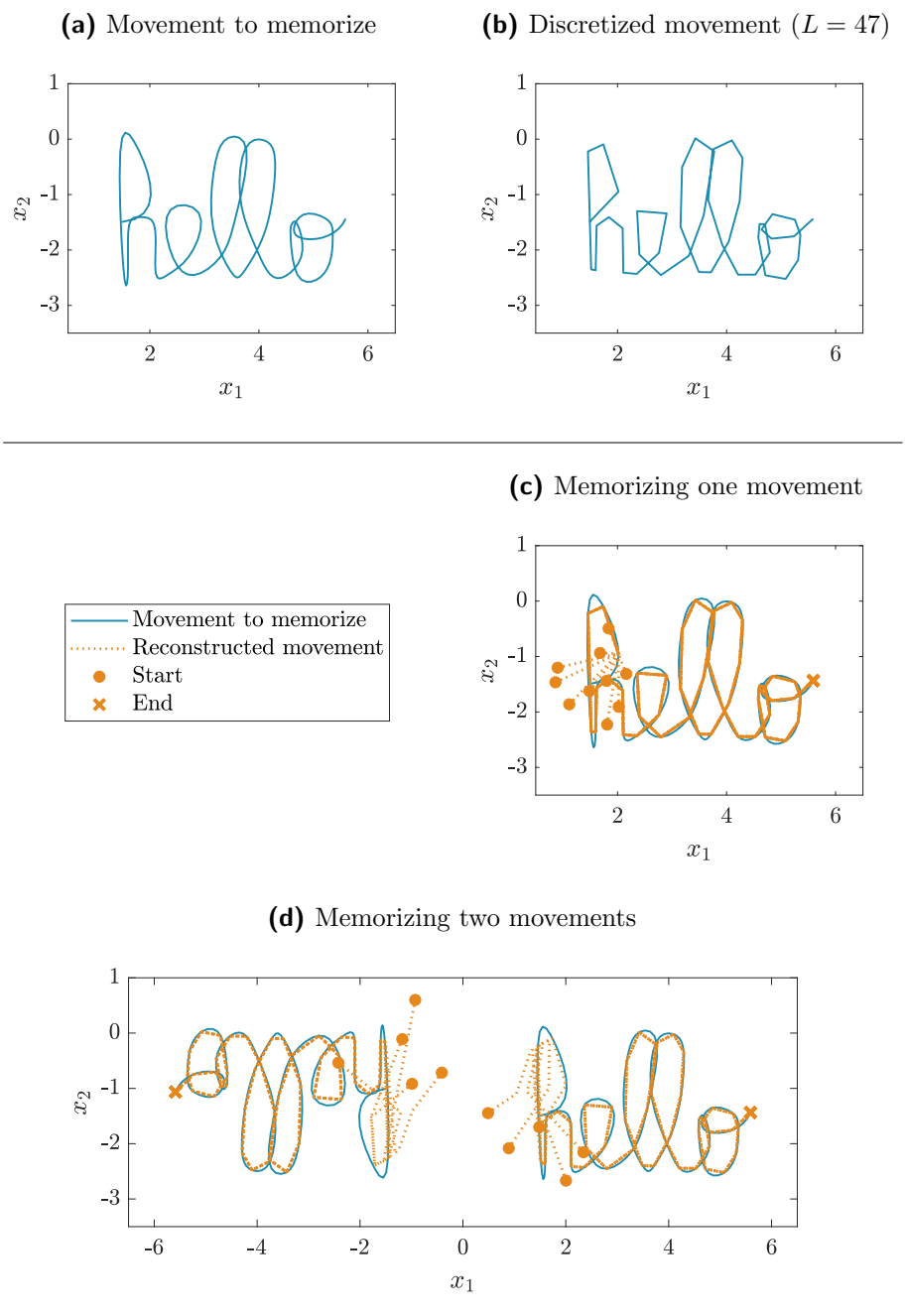


Figure 8.4: Choosing the steps of a discrete movement as hidden states. **(a),(b)** The movement to memorize, and its discretization (–) ($L = 47$). **(c),(d)** The C-LS-SVM correctly reconstructs the memorized movement, because initial conditions (●) close to a movement are attracted to it, and then follow the discretized movement all the way up to the end (×).

8.2 Experiments on deep C-LS-SVMs

Two experiments confirm that deep C-LS-SVMs function as expected. Experiment 1 trains deep models by explicitly assigning the hidden states. Experiment 2 illustrates that algorithm 2 computes good hidden states.

8.2.1 Explicitly assigning the hidden states

This experiment illustrates the potential advantages of deep C-LS-SVMs.

Setup

We generate two data sets that are subdivided in four and six groups. On each set, we train a shallow model that cannot store the memories. We then improve it by constructing a deep model and explicitly assigning the hidden states to corners of the cube spanned between $(\pm 1, \cdots, \pm 1)$.

Results

Figures 8.5 and 8.6 show the shallow (left), and the deep models (right).

Discussion

The shallow model in figure 8.5a cannot store the four groups of memories, because it cannot put a stable equilibrium on the x_1 or on the x_2-axis. The model can however place four stable equilibria on the diagonal axes. Therefore, we add a layer in front of it that rotates space by 90 degrees. Concretely, we choose a layer with a polynomial kernel of degree one, and assign the hidden states of all the memories in each group to the same point on the diagonal axis, either (-1, -1), (-1, 1), (1, -1) or (1, 1). The resulting two-layered model in figure 8.5b stores the memories well.

The shallow model in figure 8.6a cannot store the six groups of memories, as it lacks the flexibility to put six stable equilibria in the state space. Therefore, we append several layers to it so as to increase its flexibility. Concretely, we add three layers with polynomial kernels of degree three, and assign the hidden states of the memories in each group to the same point (-1, -1, -1), (-1, -1, 1), (-1, 1, -1), (-1, 1, 1), (1, -1, -1) or (1, -1, 1). The resulting four-layered model in figure 8.6b stores the memories well. The model thus reaches the flexibility of a high-degree polynomial kernel, while effectively only training several low-degree polynomial kernels, which is advantageous because their system matrix is better conditioned.

Overall, deeper models can compactly represent more complex functions.

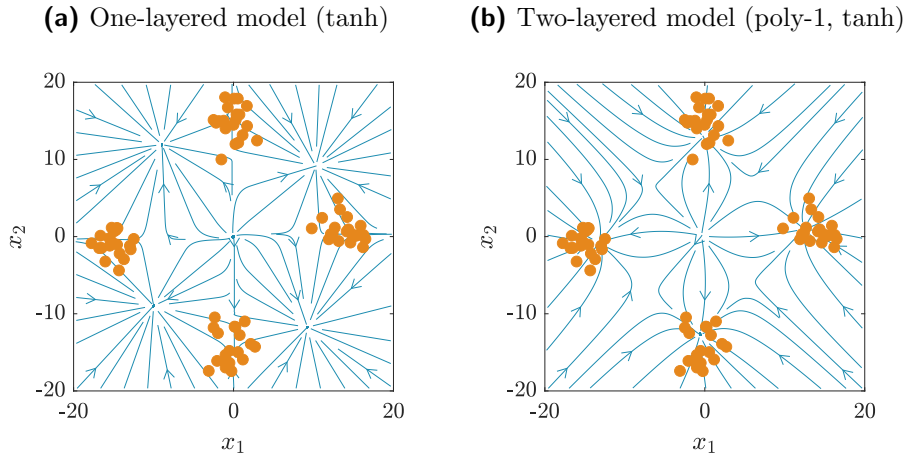


Figure 8.5: Deeper models offer more flexibility for storing memories. **(a)** The one-layered model cannot store the four groups of memories (●). **(b)** By adding one layer in front of it that rotates space by 90 degrees, the resulting two-layered model is able to properly store the memories.

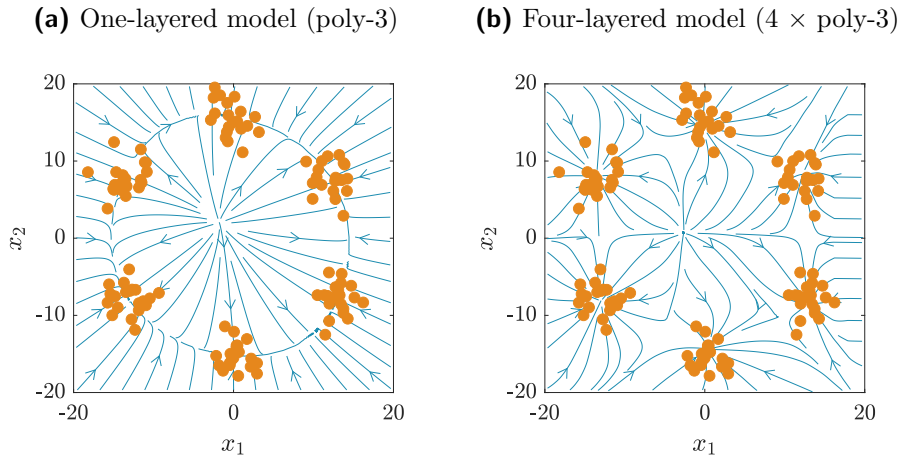


Figure 8.6: Deeper models offer more flexibility for storing memories. **(a)** The one-layered model cannot store the six groups of memories (●). **(b)** By appending three layers to it, which increase the model's flexibility, the resulting four-layered model is able to properly store the memories.

8.2.2 Implicitly learning the hidden states

This experiment indicates that algorithm 2 computes good hidden states. Besides, it shows that deep models learn more flexible update equations.

Setup

We train two deep models with algorithm 2 on three different data sets.

The first model has a polynomial kernel in the first layer ($d = 1$, $t = 0$), and a tanh feature map in the second layer. Its hyper-parameters are $\lambda_{(1)} = 10^2$, $\gamma_{(1)} = 10^{-2}$, $\eta_{(1)} = 10^{-2}$, $\lambda_{(2)} = 10^2$, $\gamma_{(2)} = 10^4$, $\eta_{(2)} = 10^{-2}$.

The second model has a polynomial kernel in the first layer ($d = 3$, $t = 1$), and a tanh feature map in the second layer. Its hyper-parameters are $\lambda_{(1)} = 10^2$, $\gamma_{(1)} = 10^{-2}$, $\eta_{(1)} = 10^0$, $\lambda_{(2)} = 10^2$, $\gamma_{(2)} = 10^2$, $\eta_{(2)} = 10^{-2}$.

Results

Figures 8.7a, 8.7b and 8.7c present the update equation of the models.

Discussion

The model in figures 8.7a and 8.7b has an update equation of the form

$$x^{(k+1)} = w_{(2)} \tanh(w_{(1)} x^{(k)} + b_{(1)}) + b_{(2)}.$$

The first layer thus shifts and scales the tanh function horizontally, whereas the second layer shifts and scales the tanh function vertically. Hence, the two-layered model can store memories whose mean is not zero, whereas a one-layered model with explicit tanh feature map could not. Also, for grouped memories, algorithm 2 stores their mean as equilibrium.

The update equation in figure 8.7c is more complex since the first layer has an implicit feature map, and the second layer has an explicit one. Still, algorithm 2 builds a model with each memory as stable equilibrium. Furthermore, in section 7.2 we showed that a one-layered model with tanh feature map or with polynomial kernel of degree 3, could not store three memories. Interestingly, the deep model that combines them can.

Overall, the state cannot diverge because the update equation is bounded.

(a) Two-layered model (poly-1, tanh) **(b)** Two-layered model (poly-1, tanh) **(c)** Two-layered model (poly-3, tanh)

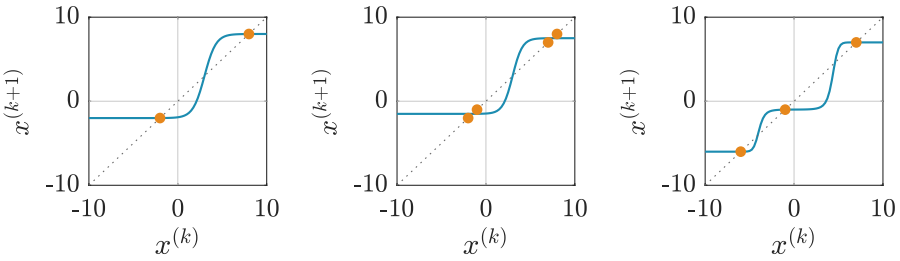


Figure 8.7: Test of algorithm 2 for training C-LS-SVMs with two layers. **(a)** The deep C-LS-SVM can store memories (●) with non-zero mean. **(b)** For grouped memories, algorithm 2 stores their mean as equilibrium. **(c)** Algorithm 2 works when linking implicit and explicit feature maps. **Overall.** The three models store the memories as stable equilibria. Furthermore, deep models can learn more flexible update equations (—), than any of their layers could individually, e.g. compare with figure 6.2.

8.3 Generative C-LS-SVMs

Research on generative models has grown considerably in recent years. Such models, able to generate new data points from the data distribution, have been applied for image generation [37], image denoising [105], reinforcement learning [33], and collaborative filtering [84], among others.

Autoencoders are a quite popular class of generative models [10, 29, 76]. In particular, C-AEs generate new data points from the data distribution by performing a random walk within the space of the latent variable [78].

Even though C-LS-SVMs do not have an explicit latent representation, we can build generative C-LS-SVMs by applying insights from C-AEs. We do so, first by identifying the manifold on which the data concentrates, then by generating new data points with a random walk on that manifold.

8.3.1 Identifying the manifold from the SVD of the Jacobian

A C-LS-SVM identifies the data manifold due to the opposition between the local stability objective that contracts all directions around each $\mathbf{x}_p$, and the equilibrium objective that ensures a good reconstruction of $\mathbf{x}_p$. As a result, the only directions that resist the contracting pressure are the ones relevant for reconstruction, namely the tangents to the manifold.

Figure 8.8 illustrates the result of the interaction between both objectives. On the tangent axis, the pressures of both objectives (–) are orthogonal. Therefore, the tangent direction becomes nor contractive, nor expansive. On the perpendicular axis, the pressures of both objectives are similar. Therefore, the perpendicular direction becomes strongly contractive.

The SVD of the Jacobian indicates the contraction in each direction. Namely, consider the first order Taylor approximation around a point $\mathbf{x}_p$

$$\mathbf{W}^\top \varphi(\mathbf{x}_p + \boldsymbol{\varepsilon}) + \mathbf{b} \;=\; \Big[\, \mathbf{W}^\top \varphi(\mathbf{x}_p) \;+\; \mathbf{b} \,\Big] \;+\; \underbrace{\mathbf{W}^\top \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}_p)}_{\text{Jacobian } \mathbf{J}_p} \; \boldsymbol{\varepsilon}.$$

Hence, the perturbation $\boldsymbol{\varepsilon}$ will be scaled along the directions of the singular vectors of the Jacobian $\mathbf{J}_p$ by the corresponding singular values. A singular value around one marks a direction with a weak contraction. A singular value around zero marks a direction with a strong contraction.

Therefore the directions tangent to the manifold in a data point $\mathbf{x}_p$ correspond with the singular vectors of $\mathbf{J}_p$ whose singular value is one, while the directions perpendicular to the manifold in a data point $\mathbf{x}_p$ correspond with the singular vectors of $\mathbf{J}_p$ whose singular value is zero.

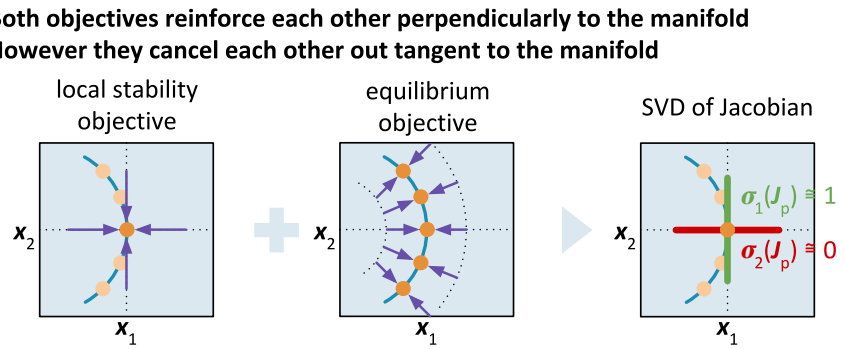


Figure 8.8: When the data points (●) concentrate near a manifold (–), the direction perpendicular to the manifold (–) is strongly contractive, whereas the direction tangent to the manifold (–) resists the contraction because it contains relevant information to reconstruct a data point well.

8.3.2 Generating data points with a random walk on the manifold

The algorithm for generation consists of a random walk on the manifold. It relies on the fact that the perpendiculars to the manifold are contracted, whereas the tangents to the manifold are nor contracted, nor expanded.

Suppose the walker is on the manifold in $\mathbf{x}^{(t)}$. He then walks in two steps:

1. He moves in a random direction $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}_N, s^2\mathbf{I}_N)$ with step size s:

$$\tilde{\mathbf{x}}^{(t)} \;=\; \mathbf{x}^{(t)} \;+\; \boldsymbol{\varepsilon}.$$

2. The C-LS-SVM update equation brings him back onto the manifold:

$$\mathbf{x}^{(t+1)} \;=\; \mathbf{W}^\top \varphi(\tilde{\mathbf{x}}^{(t)}) \;+\; \mathbf{b}.$$

If the step $\boldsymbol{\varepsilon}$ remains small, we can linearly approximate the update as

$$\mathbf{x}^{(t+1)} \;=\; \underbrace{\mathbf{W}^\top \varphi(\mathbf{x}^{(t)}) \;+\; \mathbf{b}}_{①} \;+\; \underbrace{\mathbf{W}^\top \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}^{(t)})\, \boldsymbol{\varepsilon}}_{②} \tag{8.7}$$

Part ① makes sure that if $\mathbf{x}^{(t)}$ is not on the manifold yet, then $\mathbf{x}^{(t+1)}$ is moved towards the manifold with the classic C-LS-SVM update equation. Part ② eliminates the directions in $\boldsymbol{\varepsilon}$ perpendicular to the manifold. Altogether, the walker always moves towards and tangent to the manifold.

Algorithm 3 outlines the random walk behind a generative C-LS-SVM.

Algorithm 3: Generative C-LS-SVM

Input: The start of the walk $\mathbf{x}^{(0)}$ and the step size s
Output: A random walk on the manifold $(\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(T)})$

```
/* Let the random walker perform T steps                    */
for t = 0 → T-1 do
    /* Sample a random step ε                               */
    ε ~ N(0_N, s^2 I_N) ;
    /* Update the position of the random walker             */
    x^(t+1) = W^T φ(x^(t)) + b + W^T ∂φ/∂x (x^(t)) ε ;
end
```

8.4 Experiments on generative C-LS-SVMs

Two experiments confirm that generative C-LS-SVMs work as expected. Experiment 1 inspects the singular value decomposition of the Jacobian. Experiment 2 generates points on complex manifolds with algorithm 3.

8.4.1 Singular value decomposition of the Jacobian

This experiment shows that the Jacobian estimates the manifold well.

Setup

We create two sets of 8 and 100 points $\mathbf{x}_p \in \mathbb{R}^2$ on an S-shaped manifold, and analyse the singular value decomposition of the Jacobian in each $\mathbf{x}_p$. That is to say, we visualize the two left singular vectors in the state space, and study the distribution of the two singular values in each data point. The model consists of a single layer with an RBF kernel of bandwidth 8, with the following hyper-parameters: $\lambda = 10^2$, $\gamma = 10^{-1}$, and $\eta = 10^{-2}$.

Further, we denote the Jacobian in a data point $\mathbf{x}_p$ as $\mathbf{J}_p = \mathbf{W}^\top \dfrac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}_p)$, and sort the singular values of each $\mathbf{J}_p$ in decreasing order, i.e. $\sigma_1 \geq \sigma_2$.

Results

Figures 8.9a-8.9c show the data, the scaled vectors, and the state space. Figures 8.9d-8.9d show the distribution of the singular values of each $\mathbf{J}_p$.

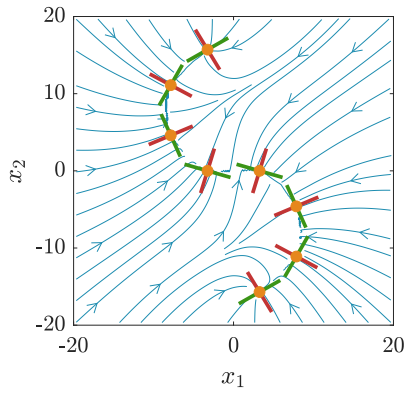
Discussion

To begin with, the left singular vectors properly follow the data-manifold, because the first left singular vectors (–) are tangent to the manifold, while the second left singular vectors (–) are always perpendicular to it.

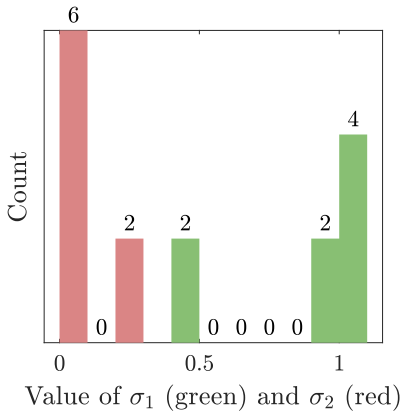
Further, the first singular values are much larger than the second ones. For instance, figure 8.9d indicates that on the data set with 100 points all the first singular values of each $\mathbf{J}_p$ lie around 1 (weak contraction), whereas all the second singular values lie around 0 (strong contraction).

Overall, the tangents to the manifold are nor contractive nor expansive, whereas the perpendiculars to the manifold are strongly contractive. Consequently, the streamlines (–) enter the manifold perpendicular to it.

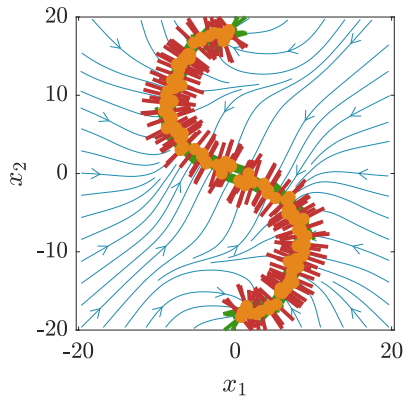
(a) State space with 8 data points and the singular vectors of each $\mathbf{J}_p$.



(b) Distribution of the singular values of the Jacobian in each data point.



(c) State space with 100 data points and the singular vectors of each $\mathbf{J}_p$.



(d) Distribution of the singular values of the Jacobian in each data point.

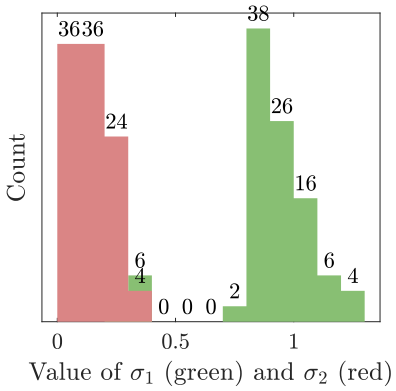


Figure 8.9: Study of the SVD of the Jacobian $\mathbf{J}_p$ in each data point. **(a),(c)** The data points (●) concentrate near an S-shaped manifold. Hence, the first left singular vectors of $\mathbf{J}_p$ (–) are tangent to the manifold, whereas the second left singular vectors of $\mathbf{J}_p$ (–) are perpendicular to it. **(b),(d)** The first singular values reveal that the tangents to the manifold are nor contractive nor expansive. The second singular values reveal that the perpendicular directions to the manifold are strongly contractive. **Overall.** The Jacobian properly approximates the manifold in the data, because the directions with a strong contraction (small singular value) correspond to the directions where the density in the data drops quickly, whereas the directions with a weak contraction (large singular value) correspond to the directions where the density in the data is constant.

8.4.2 Generating new data on complex manifolds

This experiment illustrates that C-LS-SVMs can generate new data well.

Setup

We generate three different data sets, each containing 250 data points. The data points in each data set concentrate near a complex manifold, but are slightly corrupted with Gaussian noise that has unit variance. We then train a model with RBF kernel, visualize the learnt state space, and generate 1000 new data points on each manifold with algorithm 3.

Results

Figures 8.10a, 8.11a and 8.12a show the data sets and the state spaces. Figures 8.10b, 8.11b and 8.12b show the 1000 generated new data points.

Discussion

As desired, the model is most contractive perpendicular to the manifold because the stream lines (–) approach the manifold perpendicular to it.

Further, the generated data points (●) precisely populate the manifold. The model thus approximates the data generating distributions well. Note that the generative process is a random walk on the manifold. Consequently, the different groups in figure 8.10b need to be connected.

(a) State space with 250 data points divided into 8 groups on a circle .

(b) State space with 1000 data points generated by algorithm 3.

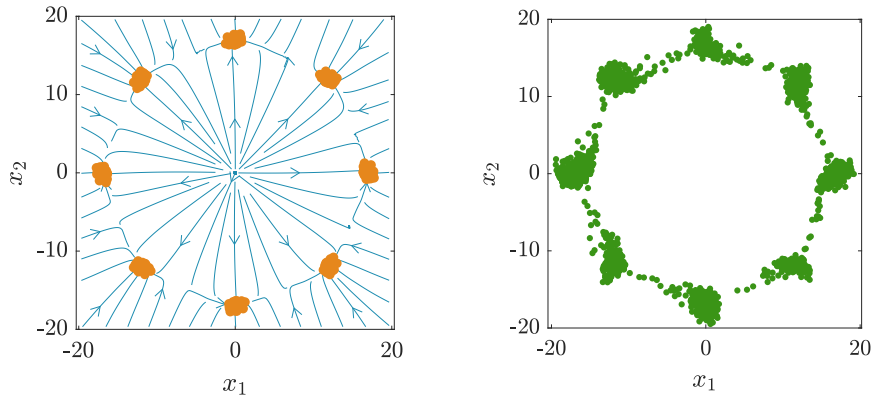


Figure 8.10: **(a)** 250 data points (●) on a fragmented circular manifold. **(b)** Algorithm 3 properly generates 1000 new points (●) on the manifold.

(a) State space with 250 data points divided into 2 concentric circles.

(b) State space with 1000 data points generated by algorithm 3.

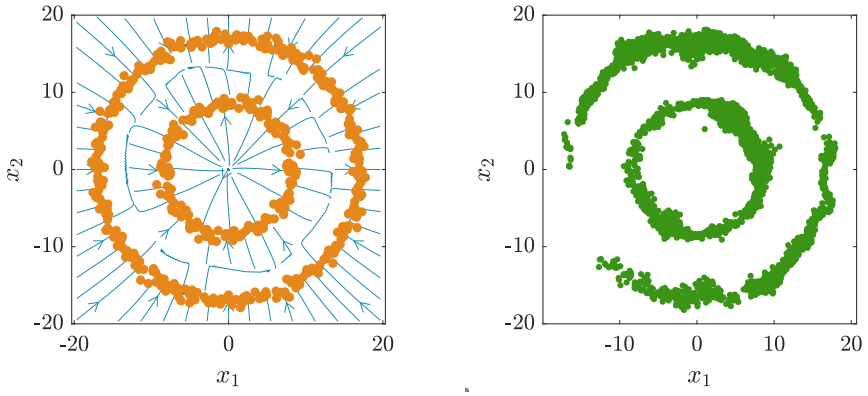


Figure 8.11: **(a)** 250 data points (●) on two concentric circular manifolds. **(b)** Algorithm 3 properly generates 1000 new points (●) on the manifold.

(a) State space with 250 data points divided into 5 branches of a star.

(b) State space with 1000 data points generated by algorithm 3.

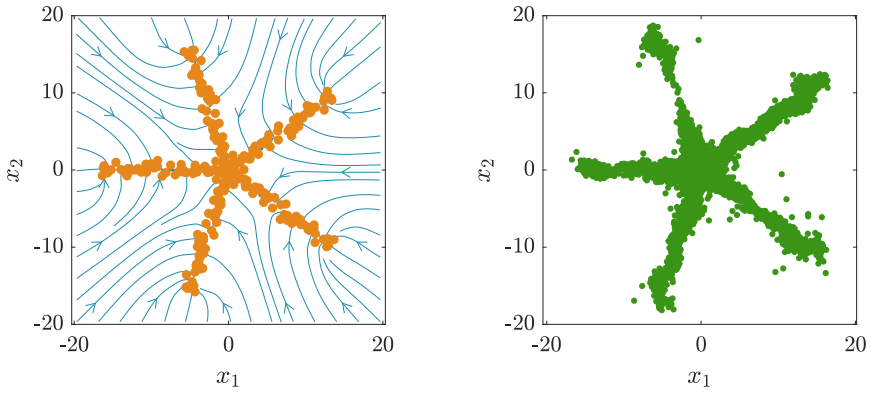


Figure 8.12: **(a)** 250 data points (●) on 5 branches of a star manifold. **(b)** Algorithm 3 properly generates 1000 new points (●) on the manifold.

8.5 Conclusion

We built deep and generative variants of C-LS-SVMs inspired by C-AEs.

First, we derived the optimization problem to train a deep C-LS-SVM, by defining targets for the representation of each memory in each layer. We called them *hidden states* and suggested three ways to compute them: (1) explicitly assign them, (2) implicitly learn them with the parameters, or (3) choose them as the different steps of a movement to memorize. Afterwards, we developed a training algorithm for deep C-LS-SVMs which is alternating convex and has no vanishing or exploding gradients.

Then, we showed the potential of deep C-LS-SVMs with two experiments. In the first experiment, we tested explicitly assigning the hidden states, and saw that deep models are more flexible than their individual layers. In the second experiment, we let algorithm 2 learn the hidden states, and saw that it produces deep models that properly store the memories.

In the third part of this chapter, we developed generative C-LS-SVMs, by noting that the Jacobian contains information on the data manifold. Specifically, the first singular vectors of the Jacobian are tangent to it, while the last singular vectors of the Jacobian are perpendicular to it. Herewith, we developed an algorithm to randomly walk on the manifold, and so generate new data points from the data-generating distribution.

Eventually, we illustrated generative C-LS-SVMs with two experiments. In the first experiment, we decomposed the Jacobian in the data points, and saw that the left singular vectors, indeed, capture the data manifold. In the second experiment, we let algorithm 3 generate new data points, and saw that these generated points precisely populate the data manifold.

In the next chapter, we apply C-LS-SVMs on a benchmark data set, which illustrates that they can properly memorize and generate images.

Chapter 9

Application on the MNIST data set

In the previous chapters, we derived theoretical results on C-LS-SVMs. By analysing them as Hopfield networks, we gauged the memory capacity. By analysing them as LS-SVMs, we derived a Bayesian interpretation. By analysing them as C-AEs, we built deep and generative variants. Each time, we supported these results with experiments on toy data sets.

In this chapter, we apply C-LS-SVMs on a benchmark data set: MNIST. This data set contains black and white images of handwritten digits [60]. An image holds 20×20 pixels, and is padded with 4-pixel-wide borders. However, we removed this padding. Thus, each image is 400-dimensional.

We perform two experiments to analyse the performance of C-LS-SVMs.

Experiment 1 shows that C-LS-SVMs can store high-dimensional images. Namely, we train a C-LS-SVM on fifty images, and show in figure 9.2 that if we give it a corrupted image, it reconstructs the original one. Besides, in figure 9.1 we study the effect of the RBF kernel parameter σ. We see that if σ is too small, the reconstruction converges to the bias, but if σ is too large, the reconstruction converges to the wrong memory.

Experiment 2 shows that C-LS-SVMs can properly generate new images. Namely, we train a C-LS-SVM on fifty different images of a same digit, and show in figure 9.4 that it generates diverse and realistic new images. Besides, in figure 9.3 we study the effect of the step size s on the walk. We see that if s is too small, the generated images are similar yet realistic, but if s is too large, the generated images are diverse yet relatively noisy.

9.1 Memorizing images

This experiment shows that C-LS-SVMs can store considerable images.

Setup

We train a C-LS-SVM with RBF kernel to memorize fifty MNIST images. This yields a dual linear system with 8.02 million parameters to estimate. We then pick the parameters that yield the smallest reconstruction error: a bandwidth $\sigma = 30$, and hyper-parameters $\lambda = 10^2$, $\gamma = 10^0$, $\eta = 10^{-2}$. Finally, we present the C-LS-SVM with images corrupted by Gaussian noise of unit variance, and report to which memory the state converges.

Results

Figure 9.1 pictures the effect of the bandwidth σ on the reconstruction. Figure 9.2 pictures the reconstruction with a C-LS-SVM that has $\sigma = 30$.

Discussion

Figure 9.1 highlights the importance of properly tuning the bandwidth σ. If σ is too small, a memory attracts only a narrow region around itself. As a result, the state converges to the bias, instead of the true memory. If σ is too large, all the memories cannot be attractors simultaneously. As a result, the state converges to another memory, though step 1 is fine. These results coincide with our earlier conclusions in experiment 6.3.1.

In figure 9.2, the C-LS-SVM with $\sigma = 30$ reconstructs the correct memory. We note that the first step reconstructs the principal part of the memory, and that the next steps then gradually remove the noise that remains.

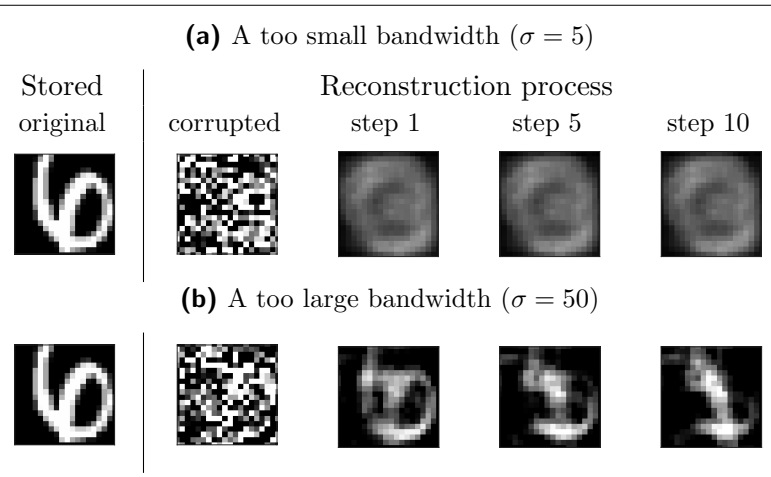


Figure 9.1: With a too small bandwidth the state converges to the bias, but with a too large bandwidth the state converges to the wrong memory.

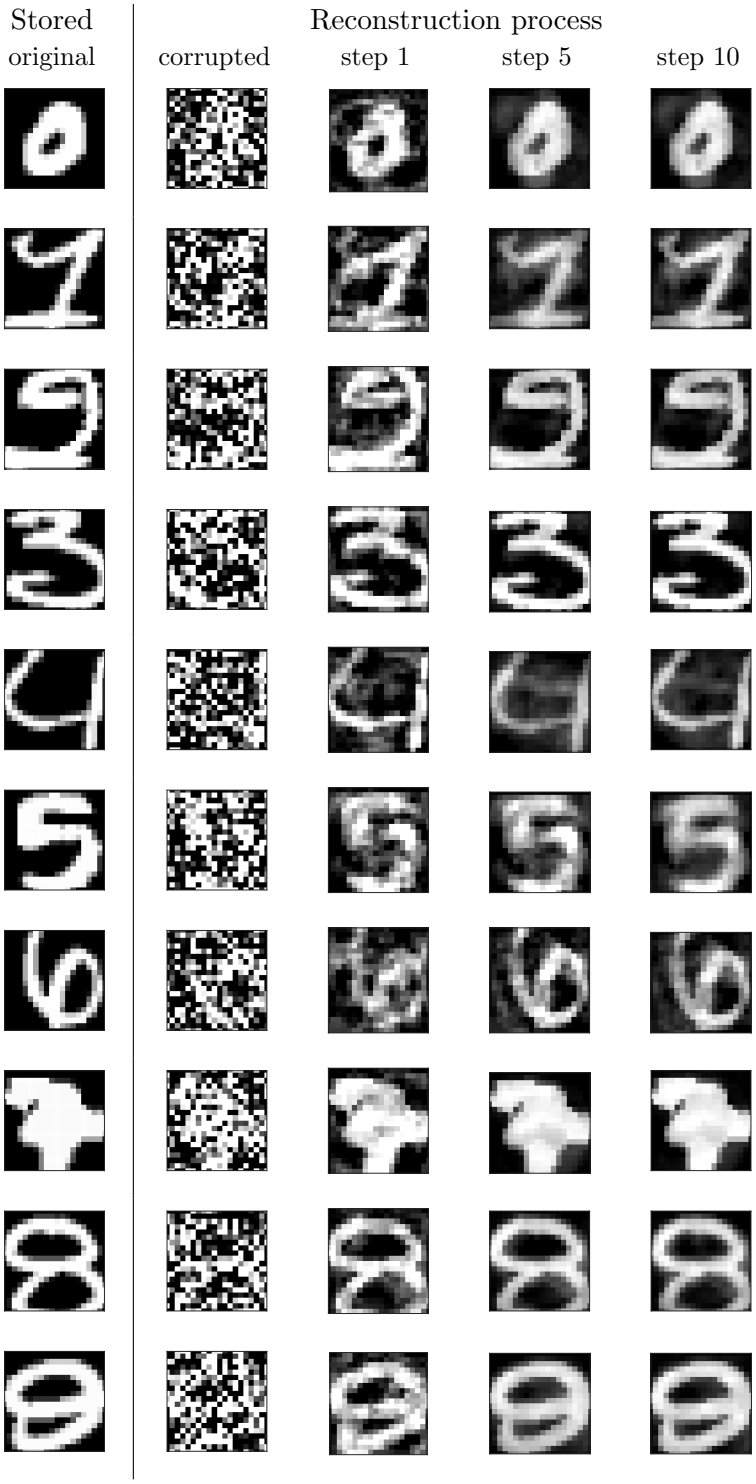


Figure 9.2: With $\sigma = 30$ the state converges to the correct memory.

9.2 Generating images

This experiment shows that C-LS-SVMs can generate new images well.

Setup

We generate ten data sets that contain fifty images of the same digit. We then train a C-LS-SVM with RBF kernel ($\sigma = 20$) on each data set, and assign the hyper-parameters to $\lambda = 10^2$, $\gamma = 10^2$ and $\eta = 10^{-2}$. Finally we generate 100 new images on each manifold with algorithm 3, i.e. perform a random walk on the manifold starting in a stored image.

Results

Figure 9.3 depicts the effect of the step size s on the generated images. Figure 9.4 shows four generated images with each C-LS-SVM and $s = 2$.

Discussion

Figure 9.1 illustrates the importance of properly tuning the step size s. If s is too small, the random walk remains trapped in the same position. As a result, the generated images look realistic, but are all very similar. If s is too large, the first-order approximation (8.7) does not fully denoise. As a result, generated images still contain noise, but are all quite different.

In figure 9.4, the C-LS-SVM generates diverse and realistic new images. The Jacobians are thus contractive along the right directions, otherwise the generated images would resemble the second column of figure 9.2.

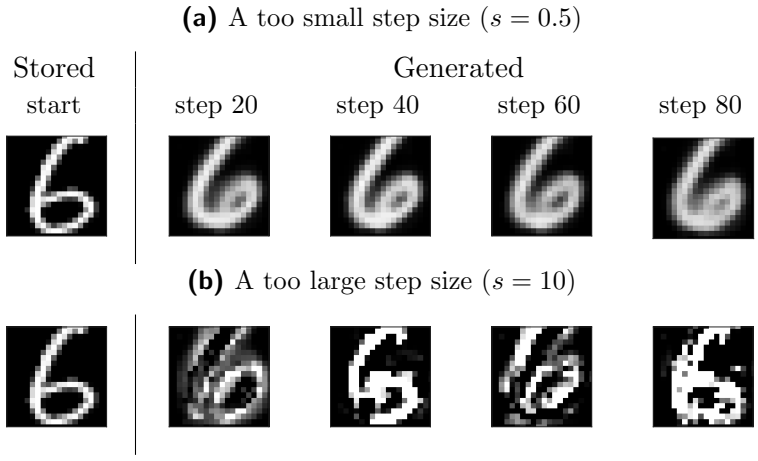


Figure 9.3: With a too small step size the generated images are similar, but with a too large step size the generated images contain much noise.

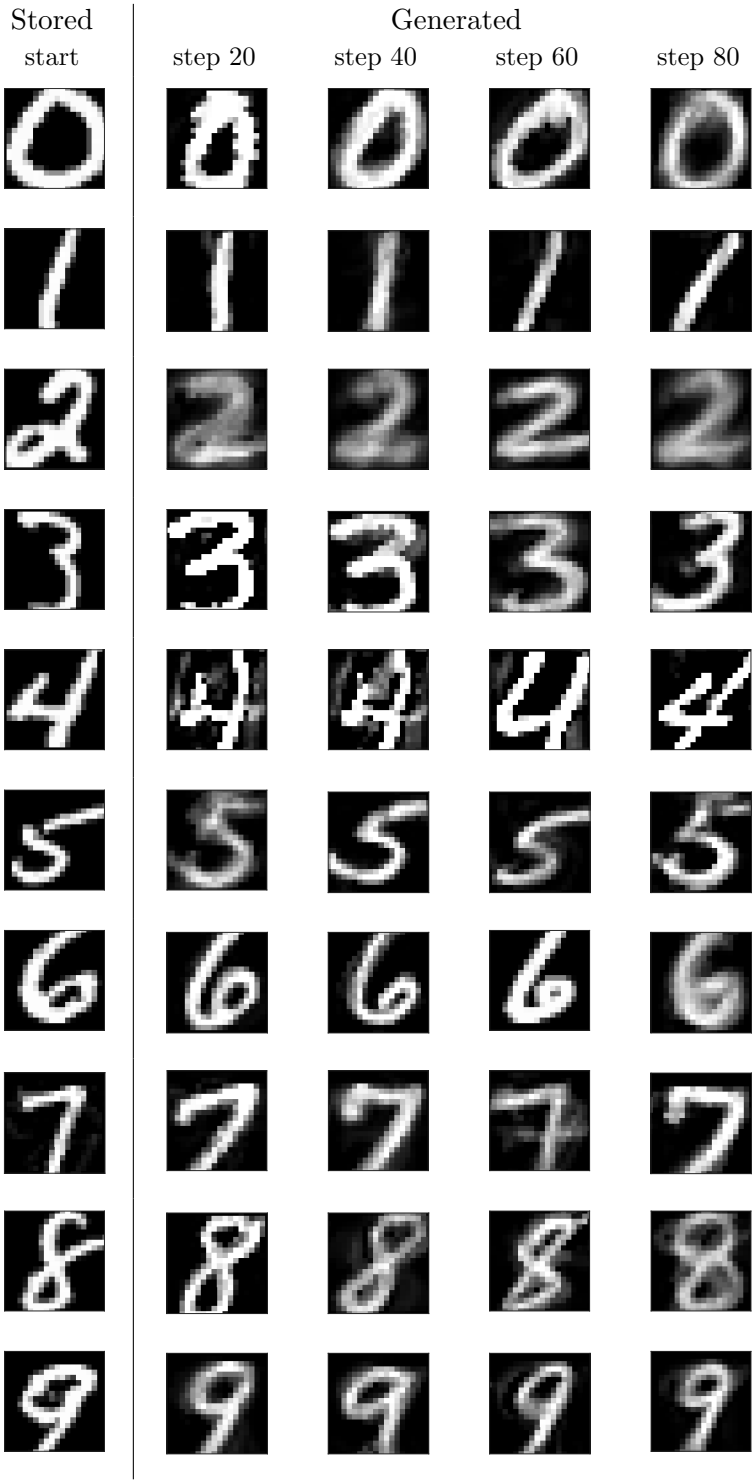


Figure 9.4: With $s = 2$ the generated images are diverse and realistic.

9.3 Conclusion

We have applied C-LS-SVMs on the MNIST data set in two experiments.

Experiment 1 trained a C-LS-SVM to store images of different digits. We noticed that if the (hyper-)parameters of the model are tuned well, when presented with a corrupted image, it reconstructs the original one. Also, we studied the effect of the RBF parameter σ on the reconstruction. We saw that if σ is too small, the reconstruction converges to the bias, but if σ is too large, the reconstruction converges to the wrong memory.

Experiment 2 trained a C-LS-SVM to store images of the same digit. We noticed that if the (hyper-)parameters of the model are tuned well, the random walk on the manifold generates diverse and realistic images. Further, we investigated the effect of the step size s on the random walk. We saw that if s is too small, the generated images are similar yet realistic, but if s is too large, the generated images are diverse yet relatively noisy.

These experiments show that C-LS-SVMs can store and generate images. However, they also highlight an important weakness of C-LS-SVMs: C-LS-SVMs have a limited scalability, because the systems grow rapidly. Specifically, the primal system requires computing $N(N_{\mathcal{F}}+1)$ parameters, and the dual system requires computing $N(P(N+1)+1)$ parameters. Thus, storing high-dimensional memories (large N) is always intensive, whether we train the C-LS-SVM with the primal or the dual system. Additionally, if the C-LS-SVM has to store many memories (large P), the computational work for solving the dual system further increases.

C-LS-SVMs thus struggle to store many high-dimensional memories, because their training is either slow, or the software runs out of memory, e.g. with 16GB of RAM Matlab runs out of memory when training a C-LS-SVM with RBF kernel on 60 MNIST images ($N = 400$, $P = 60$).

In the next chapter, we summarize our work, and propose future work.

Chapter 10

Conclusion

The goal of this thesis was to improve a model for associative memory, namely a Hopfield network, with insights from deep learning methods. We chose LS-SVMs and C-AEs, and presented our findings in two parts.

Part I introduced the theoretical background for the work in this thesis.

In chapter 2, we showed that Hopfield networks model associative memory by storing a set of memories as stable equilibria of a dynamical system. Thus, when given a noisy memory, they can reconstruct the original one. These networks have attractive features, like being asymptotically stable, but their memory capacity is limited by the size of the neural network.

In chapter 3, we showed that LS-SVMs linearly model a nonlinear function by initially transforming the input data with a nonlinear feature map. Their assets are that they are trained with a convex optimization problem, and that they can apply the kernel trick to train a large neural network without the computational burden of manipulating such a large network.

In chapter 4, we showed that C-AEs identify the features of a distribution by building a reconstruction map that contracts around each data point. Thus, when given a noisy data point, they can reconstruct the original one. Their main asset is that they impose robustness in the model analytically. Yet, their optimization problem is non-convex, which complicates training.

Part II introduced Contractive Least Squares Support Vector Machines, i.e. dynamical systems that integrate the typical contraction of C-AEs into the LS-SVM framework to build a model for auto-associative memory.

In chapter 5, we presented the overall framework behind C-LS-SVMs. First, we built the convex optimization problem to train a C-LS-SVM which balances an equilibrium objective and a local stability objective. Then, we applied the Lagrange method to derive the optimal parameters. We obtained two alternatives for training and simulating a C-LS-SVM, either in terms of the primal parameters, or in terms of the dual ones.

In chapter 6, we studied the ability of C-LS-SVMs to model memory. First, we demonstrated that the weights of a C-LS-SVM are comparable to the weights of a Hopfield network trained with KAM on noisy memories. Then, we proved that C-LS-SVMs can store $N_{\mathcal{F}}$ equilibria, like KAM, and derived conditions to ensure that space contracts around a point. Finally, we visualized C-LS-SVMs trained with different feature maps.

In chapter 7, we examined the optimization problem of a C-LS-SVM. First, we built the Bayesian interpretation of the optimization problem, which indicated that the optimization contains two types of regularization: a uniform regularization that reduces the overall sensitivity of the model, and a biased regularization that reduces the sensitivity in the memories. Then, we confirmed these results with two experiments on toy examples.

In chapter 8, we constructed deep and generative versions of C-LS-SVMs. First, we built deep C-LS-SVMs that have an alternating convex training, and confirmed that this training yields good models with two experiments. Then, we built generative C-LS-SVMs through a form of random walk, by observing that the Jacobian of the model captures the data manifold. We checked that generative C-LS-SVMs work well with two experiments.

In chapter 9, we applied C-LS-SVMs on a benchmark data set: MNIST. It showed that C-LS-SVMs can properly memorize and generate images. However, C-LS-SVMs are still heavy to train on high-dimensional data, because the size of the primal and dual linear systems increases quickly, as well as the size of the Hessian needed for training deep C-LS-SVMs. For this reason, we were limited to storing at most fifty MNIST images.

In this chapter, we put the developments of this thesis into perspective. Section 10.1 outlines the main strengths and weaknesses of C-LS-SVMs. Section 10.2 proposes ten ideas to analyse C-LS-SVMs in future work.

10.1 Strengths and Weaknesses of C-LS-SVMs

10.1.1 Strengths

There are three main strengths which make C-LS-SVMs valuable models.

1. Convex optimization problem

The optimization problem for training a C-LS-SVM is convex. Therefore, the model can generalize with few data points in high-dimensional spaces. Furthermore, it yields an analytical expression for the optimal weights, which makes it possible to analytically study the behaviour of the model. At last, it enabled us to construct a training process for deep C-LS-SVMs which is alternating convex and has no vanishing or exploding gradients.

2. Primal-dual interpretation

A C-LS-SVM has two formulations due to its primal-dual interpretation. This duality provides the possibility to optimize the training process depending on the memories to store as well as the model's architecture. Furthermore, it enables a C-LS-SVM to apply the kernel trick, and so implicitly work with a neural network that has a much larger capacity, without the computational burden of manipulating such a large network.

3. Relates to existing models

C-LS-SVMs lie at the interface of Hopfield networks, LS-SVMs and C-AEs. As a result, we can reuse existing theories to analyse and enhance them. By analysing them as Hopfield networks, we gauged the memory capacity. By analysing them as LS-SVMs, we derived a Bayesian interpretation. By analysing them as C-AEs, we developed deep and generative variants. C-LS-SVMs thus also show the potential synergies between these models.

10.1.2 Weaknesses

There are two major weaknesses that could be the start for future work.

1. Limited scalability

C-LS-SVMs are heavy to train on large and high-dimensional data sets, because the size of the primal system and the dual system grows rapidly. This hinders storing many high-dimensional memories, as in chapter 9.

2. Offline training process

It is not evident to add or remove a memory from a trained C-LS-SVM, because the primal and dual parameters are the result of a linear system. Consequently, it is not yet straightforward to adapt the model over time.

10.2 Future work

Future work could extend the results of chapter 5 in the following ways:

- Generalize C-LS-SVMs for continuous time dynamical systems. This might relate to methods for simulating initial value problems, and provide the possibility to also memorize continuous movements.
- State the C-LS-SVM optimization problem in matrix notation as

$$\begin{aligned} \underset{\mathbf{W},\mathbf{b},\mathbf{E}}{\text{minimize}} \quad & \frac{\lambda}{2}\operatorname{tr}(\mathbf{E}^\top\mathbf{E}) \;+\; \frac{\gamma}{2}\operatorname{tr}(\mathbf{W}^\top(\mathbf{F}\mathbf{F}^\top + \frac{\eta}{\gamma}\mathbf{I}_{N_\mathcal{F}})\mathbf{W}) \\ \text{subject to} \quad & \mathbf{E} = \mathbf{X} - \mathbf{W}^\top\mathbf{\Phi} - \mathbf{B} \end{aligned}$$

  to obtain a dual linear system of size $O(P)$ instead of $O(NP)$. However, the difficulty lies in computing $\mathbf{\Phi}^\top(\mathbf{F}\mathbf{F}^\top + \eta/\gamma\mathbf{I}_{N_\mathcal{F}})^{-1}\mathbf{\Phi}$.

Future work could extend the results of chapter 6 in the following ways:

- Examine how the memory capacity increases in deep C-LS-SVMs.
- Investigate the stability of deep C-LS-SVMs with NL_q theory [95].
- Find conditions on λ, γ and η such that a memory is an equilibrium, and see if they are compatible with the conditions for local stability.
- Develop a framework for storing new and forgetting old memories, based on Flexible Kernel Memory [68] or weighted LS-SVMs [94].

Future work could extend the results of chapter 7 in the following ways:

- Relate C-LS-SVMs to Kernel Principal Component Analysis [86].
- Facilitate the training process of deep C-LS-SVMs by combining layers with C-LS-SVMs and with LS-SVMs for function estimation.

Future work could extend the results of chapter 8 in the following ways:

- Construct and evaluate deep C-LS-SVMs for generative tasks, because deeper C-AEs can generate more accurate new data [78].
- Integrate higher order contractive regularization in C-LS-SVMs, i.e. additionally penalizing the Hessian of the update equation, because higher-order C-AEs can reach a higher performance [79].

Altogether, C-LS-SVMs suggest new synergies between Hopfield networks, Least Squares Support Vector Machines and contractive autoencoders. This thesis thus contributes to the research on each of these three models.

We believe that C-LS-SVMs offer many possibilities for future research, such as comparing them with existing models on benchmark data sets, or applying them on new tasks, e.g. robot control or reinforcement learning.

Hopefully, this thesis will inspire researchers to study memory models, and thereby speed up the development of adaptive Artificial Intelligence.

Appendices

Appendix A

Theorems and proofs

This appendix gathers all the propositions we have proved in this thesis.

Chapter 6

Proposition 6.1. *If $\gamma/\lambda \to 0$, the C-LS-SVM objective expressed as*

$$\begin{aligned} \underset{\mathbf{W},\mathbf{b},\mathbf{e},\mathbf{J}}{\text{minimize}} \quad & \frac{\lambda}{2}\,\mathbb{E}\Big[\,\|\,\mathbf{e}\,\|_2^2\,\Big] \;+\; \frac{\gamma}{2}\,\mathbb{E}\Big[\,\|\,\mathbf{J}\,\|_{\mathrm{F}}^2\,\Big] \;+\; \frac{\eta}{2}\,\mathbb{E}\Big[\,\|\,\mathbf{W}\,\|_{\mathrm{F}}^2\,\Big] \\ \text{subject to} \quad & \mathbf{e} = \mathbf{x} - \mathbf{W}^\top \varphi(\mathbf{x}) - \mathbf{b} \\ & \mathbf{J} = \mathbf{W}^\top \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}) \end{aligned} \tag{6.1}$$

is equivalent to the LS-SVM objective to estimate the identity map trained on corrupted data with noise $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}_N, \gamma/\lambda\, \mathbf{I}_N)$, which is

$$\begin{aligned} \underset{\mathbf{W},\mathbf{b},\mathbf{e}}{\text{minimize}} \quad & \frac{\lambda}{2}\,\mathbb{E}\Big[\,\|\,\mathbf{e}\,\|_2^2\,\Big] \;+\; \frac{\eta}{2}\,\mathbb{E}\Big[\,\|\,\mathbf{W}\,\|_{\mathrm{F}}^2\,\Big] \\ \text{subject to} \quad & \mathbf{e} = \mathbf{x} - \mathbf{W}^\top \varphi(\mathbf{x} + \boldsymbol{\varepsilon}) - \mathbf{b}. \end{aligned} \tag{6.2}$$

Proof of Proposition 6.1. Let us define a function $f(\cdot) : \mathbb{R}^N \to \mathbb{R}^N$ as

$$f(\mathbf{x}) \;=\; \mathbf{W}^\top \varphi(\mathbf{x}) \;+\; \mathbf{b}.$$

As the variance of the noise $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}_N, \gamma/\lambda\, \mathbf{I}_N)$ tends to 0, it holds that

$$f(\mathbf{x}+\boldsymbol{\varepsilon}) \;=\; f(\mathbf{x}) \;+\; \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x})\, \boldsymbol{\varepsilon} \;+\; O(\gamma/\lambda).$$

Then by substituting this Taylor expansion into the objective (6.2), and noting that $\mathbb{E}[\boldsymbol{\varepsilon}] = \mathbf{0}_N$, $\mathbb{E}[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top] = \gamma/\lambda\, \mathbf{I}_N$, $\mathrm{tr}(\boldsymbol{\varepsilon}^\top \boldsymbol{\varepsilon}) = \mathrm{tr}(\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top)$, we obtain:

$$\begin{aligned}
&\underset{\mathbf{W},\mathbf{b},\mathbf{e}}{\text{minimize}} \quad \frac{\lambda}{2}\, \mathbb{E}\Big[\, \|\, \mathbf{e}\, \|_2^2 \,\Big] \;+\; \frac{\eta}{2}\, \mathbb{E}\Big[\, \|\, \mathbf{W}\, \|_{\mathrm{F}}^2 \,\Big] \\
&\text{subject to} \quad \mathbf{e} \;=\; \mathbf{x} - f(\mathbf{x}+\boldsymbol{\varepsilon}) \\
\equiv \quad &\underset{\mathbf{W},\mathbf{b}}{\text{minimize}} \quad \frac{\lambda}{2}\, \mathbb{E}\Big[\, \|\, \mathbf{x} - f(\mathbf{x}) - \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x})\, \boldsymbol{\varepsilon} - O(\gamma/\lambda)\, \|_2^2 \,\Big] \;+\; \frac{\eta}{2}\, \mathbb{E}\Big[\, \|\, \mathbf{W}\, \|_{\mathrm{F}}^2 \,\Big] \\
\equiv \quad &\underset{\mathbf{W},\mathbf{b}}{\text{minimize}} \quad \frac{\lambda}{2}\Bigg(\mathbb{E}\Big[\, \|\, \mathbf{x} - f(\mathbf{x})\, \|_2^2 \,\Big] - 2\, \mathbb{E}\Big[\, (\mathbf{x} - f(\mathbf{x}))^\top \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}) \,\Big]\, \mathbb{E}[\, \boldsymbol{\varepsilon}\,] \\
&\qquad\qquad + \;\mathbb{E}\Big[\, \Big\|\, \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x})\, \boldsymbol{\varepsilon}\, \Big\|_2^2 \,\Big] \Bigg) \;+\; \frac{\eta}{2}\, \mathbb{E}\Big[\, \|\, \mathbf{W}\, \|_{\mathrm{F}}^2 \,\Big] \;+\; O(\gamma/\lambda) \\
\equiv \quad &\underset{\mathbf{W},\mathbf{b}}{\text{minimize}} \quad \frac{\lambda}{2}\Bigg(\mathbb{E}\Big[\, \|\, \mathbf{x} - f(\mathbf{x})\, \|_2^2 \,\Big] + \mathrm{tr}\Big(\mathbb{E}[\, \boldsymbol{\varepsilon}\, \boldsymbol{\varepsilon}^\top\,]\, \mathbb{E}\Big[\, \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x})^\top \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}) \,\Big] \Big) \Bigg) \\
&\qquad\qquad + \;\frac{\eta}{2}\, \mathbb{E}\Big[\, \|\, \mathbf{W}\, \|_{\mathrm{F}}^2 \,\Big] \\
\equiv \quad &\underset{\mathbf{W},\mathbf{b}}{\text{minimize}} \quad \frac{\lambda}{2}\, \mathbb{E}\Big[\, \|\, \mathbf{x} - f(\mathbf{x})\, \|_2^2 \,\Big] \;+\; \frac{\lambda}{2}\, \mathrm{tr}\Big(\frac{\gamma}{\lambda}\, \mathbb{E}\Big[\, \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x})^\top \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}) \,\Big] \Big) \\
&\qquad\qquad + \;\frac{\eta}{2}\, \mathbb{E}\Big[\, \|\, \mathbf{W}\, \|_{\mathrm{F}}^2 \,\Big] \\
\equiv \quad &\underset{\mathbf{W},\mathbf{b},\mathbf{e},\mathbf{J}}{\text{minimize}} \quad \frac{\lambda}{2}\, \mathbb{E}\Big[\, \|\, \mathbf{e}\, \|_2^2 \,\Big] \;+\; \frac{\gamma}{2}\, \mathbb{E}\Big[\, \|\, \mathbf{J}\, \|_{\mathrm{F}}^2 \,\Big] \;+\; \frac{\eta}{2}\, \mathbb{E}\Big[\, \|\, \mathbf{W}\, \|_{\mathrm{F}}^2 \,\Big] \\
&\text{subject to} \quad \mathbf{e} = \mathbf{x} - \mathbf{W}^\top \varphi(\mathbf{x}) - \mathbf{b} \\
&\qquad\qquad \mathbf{J} = \mathbf{W}^\top \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}).
\end{aligned}$$

■

Proposition 6.2. *A C-LS-SVM as defined in Definition 5.2 with hyper-parameters $\gamma = 0$, $\eta = 0$ can have up to $N_{\mathcal{F}}$ equilibria $\mathbf{x}_p$, provided that the vectors $\{\, \varphi(\mathbf{x}_p) \,\}_{p=1}^{P}$ are linearly independent.*

Proof of Proposition 6.2. If $\gamma = \eta = 0$, the weights in (6.3) simplify to

$$\mathbf{W} \;=\; \boldsymbol{\Phi} \left[\, \boldsymbol{\Phi}^\top \boldsymbol{\Phi} \,\right]^{-1} (\mathbf{X} - \mathbf{B})^\top. \tag{A.1}$$

$\boldsymbol{\Phi}^\top \boldsymbol{\Phi}$ is invertible if the P vectors $\varphi(\mathbf{x}_p)$ in $\boldsymbol{\Phi}$ are linearly independent. Then, as long as $P \leq N_{\mathcal{F}}$ the C-LS-SVM maps each vector $\mathbf{x}_p$ onto itself:

$$\begin{aligned} \mathbf{W}^\top \varphi(\mathbf{x}_p) \;+\; \mathbf{b} \;&=\; (\mathbf{X} - \mathbf{B})\, \boldsymbol{\Phi}^{+}\, \varphi(\mathbf{x}_p) \;+\; \mathbf{b} \\ &=\; (\mathbf{X} - \mathbf{B})\, \mathbf{e}_p \;+\; \mathbf{b} \\ &=\; \mathbf{x}_p \end{aligned} \tag{A.2}$$

where $\mathbf{e}_p$ stands for the p^{th} column of the identity matrix of size P. ■

Proposition 6.3. *A C-LS-SVM as defined in Definition 5.2 with hyper-parameters $\gamma \neq 0$, $\eta = 0$ can have $N_{\mathcal{F}}/(N+1)$ equilibria $\mathbf{x}_p$, if the vectors $\left\{ \varphi(\mathbf{x}_p),\; \left\{ \dfrac{\partial \varphi}{\partial x_n}(\mathbf{x}_p) \right\}_{n=1}^{N} \right\}_{p=1}^{P}$ are linearly independent.*

Proof of Proposition 6.3. Since $\eta = 0$, we derive from system (5.15) that

$$\begin{aligned} \mathbf{W} \;&=\; \frac{1}{\eta} \begin{bmatrix} \boldsymbol{\Phi} & \mathbf{F} \end{bmatrix} \begin{bmatrix} \frac{1}{\eta} \boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \frac{1}{\lambda} \mathbf{I}_P & \frac{1}{\eta} \boldsymbol{\Phi}^\top \mathbf{F} \\ \frac{1}{\eta} \mathbf{F}^\top \boldsymbol{\Phi} & \frac{1}{\eta} \mathbf{F}^\top \mathbf{F} + \frac{1}{\gamma} \mathbf{I}_{NP} \end{bmatrix}^{-1} \begin{bmatrix} (\mathbf{X} - \mathbf{B})^\top \\ \mathbf{0}_{NP \times N} \end{bmatrix} \\ &=\; \begin{bmatrix} \boldsymbol{\Phi} & \mathbf{F} \end{bmatrix} \left(\begin{bmatrix} \boldsymbol{\Phi}^\top \\ \mathbf{F}^\top \end{bmatrix} \begin{bmatrix} \boldsymbol{\Phi} & \mathbf{F} \end{bmatrix} \right)^{-1} \begin{bmatrix} (\mathbf{X} - \mathbf{B})^\top \\ \mathbf{0}_{NP \times N} \end{bmatrix}. \end{aligned}$$

As long as $\begin{bmatrix} \boldsymbol{\Phi} & \mathbf{F} \end{bmatrix} \in \mathbb{R}^{N_{\mathcal{F}} \times (N+1)P}$ has linearly independent columns, i.e. $N_{\mathcal{F}} \geq (N+1)P$ and the columns of $\boldsymbol{\Phi}$, $\mathbf{F}$ are linearly independent, the inverse is computable. The C-LS-SVM then maps each $\mathbf{x}_p$ onto itself:

$$\begin{aligned} \mathbf{W}^\top \varphi(\mathbf{x}_p) \;+\; \mathbf{b} \;&=\; \begin{bmatrix} \mathbf{X} - \mathbf{B} & \mathbf{0}_{N \times NP} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Phi} & \mathbf{F} \end{bmatrix}^{+} \varphi(\mathbf{x}_p) \;+\; \mathbf{b} \\ &=\; \begin{bmatrix} \mathbf{X} - \mathbf{B} & \mathbf{0}_{N \times NP} \end{bmatrix} \mathbf{e}_p \;+\; \mathbf{b} \\ &=\; \mathbf{x}_p \end{aligned}$$

where $\mathbf{e}_p$ is the p^{th} column of the identity matrix of size $(N+1)P$. ■

Proposition 6.4. *Given a C-LS-SVM as defined in Definition 5.2. Then, a sufficient condition on the hyper-parameters λ, γ and η such that the C-LS-SVM would contract space around a point $\mathbf{x}^* \in \mathbb{R}^N$ is*

$$\lambda \;<\; \frac{\eta - \gamma\,\sigma_c}{\sigma_a \times \sigma_b + \sigma_d} \tag{6.8}$$

with

$$\sigma_a \;=\; \sigma_{\max}\Big(\; \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}^*)\;\Big) \tag{6.9a}$$

$$\sigma_b \;=\; \sigma_{\max}\Big(\; \mathbf{\Phi}\Big(\mathbf{I}_P - \frac{1}{P}\mathbf{1}_{P\times P}\Big)\mathbf{X}^\top\;\Big) \tag{6.9b}$$

$$\sigma_c \;=\; \sigma_{\max}\Big(\; \mathbf{F}\mathbf{F}^\top\;\Big) \tag{6.9c}$$

$$\sigma_d \;=\; \sigma_{\max}\Big(\; \mathbf{\Phi}\Big(\mathbf{I}_P - \frac{1}{P}\mathbf{1}_{P\times P}\Big)\mathbf{\Phi}^\top\;\Big). \tag{6.9d}$$

Proof of Proposition 6.4. For space to contract around $\mathbf{x}^*$, the Jacobian

$$\mathbf{W}^\top \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}^*)$$

should have all its singular values $\sigma_{\min} \leq \cdots \leq \sigma_{\max}$ smaller than one. So the proof involves deriving an upper bound B, and bounding B by 1:

$$\sigma_{\max}\Big(\; \mathbf{W}^\top \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}^*)\;\Big) \;\leq\; B \;<\; 1.$$

Lemma 6.1 already yields a first bound on the largest singular value as

$$\sigma_{\max}\Big(\; \mathbf{W}^\top \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}^*)\;\Big) \;\leq\; \sigma_{\max}(\mathbf{W}^\top)\; \underbrace{\sigma_{\max}\Big(\; \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}^*)\;\Big)}_{\sigma_a}. \tag{A.3}$$

Because the feature map $\varphi(\cdot)$ is explicit, we can explicitly compute σ_a. Next, we bound $\sigma_{\max}(\mathbf{W})$ with the primal system derived in section 5.2.2:

$$\begin{bmatrix} \mathbf{\Phi}\mathbf{\Phi}^\top + \frac{\gamma}{\lambda}\mathbf{F}\mathbf{F}^\top + \frac{\eta}{\lambda}\mathbf{I}_{N_\mathcal{F}} & \mathbf{\Phi}\mathbf{1}_P \\ \mathbf{1}_P^\top \mathbf{\Phi}^\top & P \end{bmatrix} \begin{bmatrix} \mathbf{W} \\ \mathbf{b}^\top \end{bmatrix} \;=\; \begin{bmatrix} \mathbf{\Phi}\mathbf{X}^\top \\ \mathbf{1}_P^\top \mathbf{X}^\top \end{bmatrix}.$$

From the second row, we obtain the expression for $\mathbf{b}^\top$ in terms of $\mathbf{W}$

$$\mathbf{b}^\top \;=\; \frac{1}{P}\mathbf{1}_P^\top(\mathbf{X}^\top - \mathbf{\Phi}^\top\mathbf{W}).$$

Inserting this expression in the first row yields the expression for $\mathbf{W}$

$$\mathbf{W} \;=\; \Big[\mathbf{\Phi}\Big(\mathbf{I}_P - \frac{1}{P}\mathbf{1}_{P\times P}\Big)\mathbf{\Phi}^\top + \frac{\gamma}{\lambda}\mathbf{F}\mathbf{F}^\top + \frac{\eta}{\lambda}\mathbf{I}_{N_\mathcal{F}}\Big]^{-1}\mathbf{\Phi}\Big(\mathbf{I}_P - \frac{1}{P}\mathbf{1}_{P\times P}\Big)\mathbf{X}^\top.$$

By applying Lemma 6.1 on this new expression for $\mathbf{W}$ we obtain that

$$\sigma_{\max}(\mathbf{W}) \leq \sigma_{\max}\left(\left[\mathbf{\Phi}\Big(\mathbf{I}_P - \frac{1}{P}\mathbf{1}_{P\times P}\Big)\mathbf{\Phi}^\top + \frac{\gamma}{\lambda}\mathbf{F}\mathbf{F}^\top + \frac{\eta}{\lambda}\mathbf{I}_{N_\mathcal{F}}\right]^{-1} \right) \times \underbrace{\sigma_{\max}\Big(\mathbf{\Phi}\Big(\mathbf{I}_P - \frac{1}{P}\mathbf{1}_{P\times P}\Big)\mathbf{X}^\top \Big)}_{\sigma_b}. \tag{A.4}$$

Because the feature map $\varphi(\cdot)$ is explicit, we can explicitly compute σ_b.

Next, we bound the largest singular value of the inverse by introducing

$$\mathbf{P} = \mathbf{\Phi}\Big(\mathbf{I}_P - \frac{1}{P}\mathbf{1}_{P\times P}\Big)\mathbf{\Phi}^\top,$$
$$\mathbf{S} = \mathbf{F}\mathbf{F}^\top.$$

Because the feature map $\varphi(\cdot)$ is explicit, we can explicitly compute them. Herewith, the largest singular value of the inverse in (A.4) rewrites to

$$\sigma_{\max}\Big(\Big[\mathbf{P} + \frac{1}{\lambda}(\gamma\mathbf{S} + \eta\mathbf{I}_{N_\mathcal{F}})\Big]^{-1} \Big) = \sigma_{\min}^{-1}\Big(\mathbf{P} + \frac{1}{\lambda}(\gamma\mathbf{S} + \eta\mathbf{I}_{N_\mathcal{F}}) \Big).$$

With the formula derived in [64], we work out a lower bound on $\sigma_{\min}$ as

$$\begin{aligned}\sigma_{\min}\Big(\mathbf{P} + \frac{1}{\lambda}(\gamma\mathbf{S} + \eta\mathbf{I}_{N_\mathcal{F}}) \Big) &\geq \frac{1}{\lambda}\sigma_{\min}(\gamma\mathbf{S} + \eta\mathbf{I}_{N_\mathcal{F}}) - \sigma_{\max}(\mathbf{P}) \\ &\geq \frac{1}{\lambda}\Big(\eta - \gamma \underbrace{\sigma_{\max}(\mathbf{S})}_{\sigma_c}\Big) - \underbrace{\sigma_{\max}(\mathbf{P})}_{\sigma_d}.\end{aligned} \tag{A.5}$$

If the right-hand side of (A.5) is positive, i.e. $\lambda \leq (\eta - \gamma\,\sigma_c)/\sigma_d$, then

$$\sigma_{\max}\Big(\Big[\mathbf{P} + \frac{1}{\lambda}(\gamma\mathbf{S} + \eta\mathbf{I}_{N_\mathcal{F}})\Big]^{-1} \Big) \leq \frac{1}{(\eta - \gamma\,\sigma_c)/\lambda - \sigma_d}. \tag{A.6}$$

In the end, from (A.3), (A.4) and (A.6), we obtain the upper bound

$$\sigma_{\max}\Big(\mathbf{W}^\top \frac{\partial\varphi}{\partial\mathbf{x}}(\mathbf{x}^*) \Big) \leq \sigma_a \times \sigma_b \times \frac{1}{(\eta - \gamma\,\sigma_c)/\lambda - \sigma_d}. \tag{A.7}$$

To conlude the proof, bounding the right-hand side in (A.7) by 1 yields:

$$\lambda < \frac{\eta - \gamma\,\sigma_c}{\sigma_a \times \sigma_b + \sigma_d}.$$

■

Proposition 6.5. *Given a C-LS-SVM as defined in Definition 5.2. Then, a sufficient condition on the hyper-parameters λ, γ and η such that the C-LS-SVM would contract space around a point $\mathbf{x}^* \in \mathbb{R}^N$ is*

$$\max(\lambda, \gamma) \;<\; \frac{\eta}{\sigma_a \times \sigma_b + \sigma_c} \tag{6.10}$$

with

$$\sigma_a \;=\; \sigma_{\max}\left(\begin{bmatrix} \boldsymbol{\Phi}^\top \dfrac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}^*) \\ \mathbf{F}^\top \dfrac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}^*) \end{bmatrix} \right) \tag{6.11a}$$

$$\sigma_b \;=\; \sigma_{\max}\left(\begin{bmatrix} \left(\mathbf{I}_P - \dfrac{1}{P}\mathbf{1}_{P\times P}\right)\mathbf{X}^\top \\ \mathbf{0}_{NP\times N} \end{bmatrix} \right) \tag{6.11b}$$

$$\sigma_c \;=\; \sigma_{\max}\left(\begin{bmatrix} \left(\mathbf{I}_P - \dfrac{1}{P}\mathbf{1}_{P\times P}\right)\boldsymbol{\Phi}^\top\boldsymbol{\Phi} & \left(\mathbf{I}_P - \dfrac{1}{P}\mathbf{1}_{P\times P}\right)\boldsymbol{\Phi}^\top\mathbf{F} \\ \mathbf{F}^\top\boldsymbol{\Phi} & \mathbf{F}^\top\mathbf{F} \end{bmatrix} \right). \tag{6.11c}$$

Proof of Proposition 6.5. For space to contract around $\mathbf{x}^*$, the Jacobian

$$\mathbf{W}^\top \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}^*)$$

should have all its singular values $\sigma_{\min} \leq \cdots \leq \sigma_{\max}$ smaller than one. So the proof involves deriving an upper bound B, and bounding B by 1:

$$\sigma_{\max}\left(\mathbf{W}^\top \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}^*) \right) \;\leq\; B \;<\; 1.$$

First, an explicit formulation for $\mathbf{W}$ might not exist since $\varphi(\cdot)$ is implicit. Therefore, we work with its expression in terms of the dual variables:

$$\mathbf{W} \;=\; \frac{1}{\eta}\begin{bmatrix}\boldsymbol{\Phi} & \mathbf{F}\end{bmatrix}\begin{bmatrix}\mathbf{L}^\top \\ \mathbf{M}^\top\end{bmatrix}.$$

Lemma 6.1 already yields a first bound on the largest singular value as

$$\sigma_{\max}\left(\frac{1}{\eta}\begin{bmatrix}\mathbf{L} & \mathbf{M}\end{bmatrix}\begin{bmatrix} \boldsymbol{\Phi}^\top \dfrac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}^*) \\ \mathbf{F}^\top \dfrac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}^*) \end{bmatrix} \right) \;\leq\; \sigma_{\max}\left(\frac{1}{\eta}\begin{bmatrix}\mathbf{L} & \mathbf{M}\end{bmatrix} \right) \tag{A.8}$$

$$\times \underbrace{\sigma_{\max}\left(\begin{bmatrix} \boldsymbol{\Phi}^\top \dfrac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}^*) \\ \mathbf{F}^\top \dfrac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}^*) \end{bmatrix} \right)}_{\sigma_a}.$$

Since the kernel $k(\cdot,\cdot)$ is explicit, we can explicitly compute the value a.

Next, we bound $\sigma_{\max}\Big(\left[\begin{array}{cc}\mathbf{L} & \mathbf{M}\end{array}\right]/\eta\Big)$ with the dual system of section 5.2.2:

$$\begin{bmatrix} \frac{1}{\eta}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}+\frac{1}{\lambda}\mathbf{I}_P & \frac{1}{\eta}\boldsymbol{\Phi}^\top\mathbf{F} & \mathbf{1}_P \\ \frac{1}{\eta}\mathbf{F}^\top\boldsymbol{\Phi} & \frac{1}{\eta}\mathbf{F}^\top\mathbf{F}+\frac{1}{\gamma}\mathbf{I}_{NP} & \mathbf{0}_{NP} \\ \mathbf{1}_P^\top & \mathbf{0}_{NP}^\top & 0 \end{bmatrix} \begin{bmatrix} \mathbf{L}^\top \\ \mathbf{M}^\top \\ \mathbf{b}^\top \end{bmatrix} = \begin{bmatrix} \mathbf{X}^\top \\ \mathbf{0}_{NP\times N} \\ \mathbf{0}_N^\top \end{bmatrix}.$$

From the third row, we obtain the expression for $\mathbf{b}^\top$ in terms of $\left[\begin{array}{cc}\mathbf{L} & \mathbf{M}\end{array}\right]$

$$\mathbf{b}^\top = \frac{1}{P}\mathbf{1}_P^\top(\mathbf{X}^\top - \frac{1}{\eta}\begin{bmatrix}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} & \boldsymbol{\Phi}^\top\mathbf{F}\end{bmatrix}\begin{bmatrix}\mathbf{L}^\top \\ \mathbf{M}^\top\end{bmatrix}).$$

Inserting this expression in the first rows yields the expression for $\left[\begin{array}{cc}\mathbf{L} & \mathbf{M}\end{array}\right]$

$$\begin{bmatrix}\mathbf{L}^\top \\ \mathbf{M}^\top\end{bmatrix} = \begin{bmatrix} \frac{1}{\eta}\Big(\mathbf{I}_P-\frac{1}{P}\mathbf{1}_{P\times P}\Big)\boldsymbol{\Phi}^\top\boldsymbol{\Phi}+\frac{1}{\lambda}\mathbf{I}_P & \frac{1}{\eta}\Big(\mathbf{I}_P-\frac{1}{P}\mathbf{1}_{P\times P}\Big)\boldsymbol{\Phi}^\top\mathbf{F} \\ \frac{1}{\eta}\mathbf{F}^\top\boldsymbol{\Phi} & \frac{1}{\eta}\mathbf{F}^\top\mathbf{F}+\frac{1}{\gamma}\mathbf{I}_{NP} \end{bmatrix}^{-1} \times \begin{bmatrix}\Big(\mathbf{I}_P-\frac{1}{P}\mathbf{1}_{P\times P}\Big)\mathbf{X}^\top \\ \mathbf{0}_{NP\times N}\end{bmatrix}.$$

By applying Lemma 6.1 on this expression for $\left[\begin{array}{cc}\mathbf{L} & \mathbf{M}\end{array}\right]$ we obtain that

$$\sigma_{\max}\Big(\frac{1}{\eta}\begin{bmatrix}\mathbf{L} & \mathbf{M}\end{bmatrix}\Big) \leq \sigma_{\max}\left(\begin{bmatrix}\Big(\mathbf{I}_P-\frac{1}{P}\mathbf{1}_{P\times P}\Big)\boldsymbol{\Phi}^\top\boldsymbol{\Phi}+\frac{\eta}{\lambda}\mathbf{I}_P & \Big(\mathbf{I}_P-\frac{1}{P}\mathbf{1}_{P\times P}\Big)\boldsymbol{\Phi}^\top\mathbf{F} \\ \mathbf{F}^\top\boldsymbol{\Phi} & \mathbf{F}^\top\mathbf{F}+\frac{\eta}{\gamma}\mathbf{I}_{NP}\end{bmatrix}^{-1}\right) \times \underbrace{\sigma_{\max}\left(\begin{bmatrix}\Big(\mathbf{I}_P-\frac{1}{P}\mathbf{1}_{P\times P}\Big)\mathbf{X}^\top \\ \mathbf{0}_{NP\times N}\end{bmatrix}\right)}_{\sigma_b}. \tag{A.9}$$

Next, we bound the largest singular value of the inverse by introducing

$$\mathbf{P} = \begin{bmatrix}\Big(\mathbf{I}_P-\frac{1}{P}\mathbf{1}_{P\times P}\Big)\boldsymbol{\Phi}^\top\boldsymbol{\Phi} & \Big(\mathbf{I}_P-\frac{1}{P}\mathbf{1}_{P\times P}\Big)\boldsymbol{\Phi}^\top\mathbf{F} \\ \mathbf{F}^\top\boldsymbol{\Phi} & \mathbf{F}^\top\mathbf{F}\end{bmatrix},$$

$$\mathbf{S} = \begin{bmatrix}\frac{1}{\lambda}\mathbf{I}_P & \mathbf{0}_{P\times NP} \\ \mathbf{0}_{NP\times P} & \frac{1}{\gamma}\mathbf{I}_{NP}\end{bmatrix}.$$

Since the kernel $k(\cdot,\cdot)$ is explicit, we can explicitly compute matrix $\mathbf{P}$. Herewith, the largest singular value of the inverse in (A.9) rewrites to

$$\sigma_{\max}\Big(\Big[\mathbf{P}+\eta\mathbf{S}\Big]^{-1}\Big) = \sigma_{\min}^{-1}\Big(\mathbf{P}+\eta\mathbf{S}\Big).$$

With the formula derived in [64], we work out a lower bound on $\sigma_{\min}$ as

$$\begin{aligned}\sigma_{\min}\Big(\ \mathbf{P}+\eta\mathbf{S}\ \Big) \ &\geq\ \eta\,\sigma_{\min}(\mathbf{S})\ -\ \sigma_{\max}(\mathbf{P}) \\ &\geq\ \eta/\max(\lambda,\gamma)\ -\ \underbrace{\sigma_{\max}(\mathbf{P})}_{\sigma_c}.\end{aligned} \tag{A.10}$$

If the right-hand side of (A.10) is positive, i.e. $\max(\lambda,\gamma)\ \leq\ \eta/\sigma_c$, then

$$\sigma_{\max}\Big(\ \Big[\mathbf{P}+\eta\mathbf{S}\Big]^{-1}\ \Big)\ \leq\ \frac{1}{\eta/\max(\lambda,\gamma)-\sigma_c}. \tag{A.11}$$

In the end, from (A.8), (A.9) and (A.11), we obtain the upper bound

$$\sigma_{\max}\Big(\ \mathbf{W}^\top\frac{\partial\varphi}{\partial\mathbf{x}}(\mathbf{x}^*)\ \Big)\ \leq\ \sigma_a\times\sigma_b\times\frac{1}{\eta/\max(\lambda,\gamma)-\sigma_c}. \tag{A.12}$$

To conlude the proof, bounding the right-hand side in (A.12) by 1 yields:

$$\max(\lambda,\gamma)\ <\ \frac{\eta}{\sigma_a\times\sigma_b+\sigma_c}.$$

■

Chapter 7

Proposition 7.1. *Given a data set* $\mathcal{D}=\{(\mathbf{x}_p\in\mathbb{R}^N,\ \mathbf{y}_p\in\mathbb{R}^M)\}_{p=1}^P$, *hyper-parameters* $\lambda,\gamma,\eta\in\mathbb{R}^+$, *and a model* $\mathcal{M}$ *with feature map* $\varphi(\cdot):\mathbb{R}^N\to\mathbb{R}^{N_\mathcal{F}}$ *or kernel* $k(\cdot,\cdot):\mathbb{R}^N\times\mathbb{R}^N\to\mathbb{R}$, *then solving*

$$\begin{aligned}\underset{\mathbf{w}_m,\mathbf{b},\mathbf{e}_p,\mathbf{J}_p}{\text{minimize}}\quad & \frac{\lambda}{2}\sum_{p=1}^P\|\ \mathbf{e}_p\ \|_2^2+\frac{\gamma}{2}\sum_{p=1}^P\|\ \mathbf{J}_p\ \|_\mathrm{F}^2+\frac{\eta}{2}\sum_{m=1}^M\|\ \mathbf{w}_m\ \|_2^2 \\ \text{subject to}\quad & \mathbf{e}_p=\mathbf{y}_p-\mathbf{W}^\top\varphi(\mathbf{x}_p)-\mathbf{b} \\ & \mathbf{J}_p=\mathbf{W}^\top\frac{\partial\varphi}{\partial\mathbf{x}}(\mathbf{x}_p) \\ & 1\leq p\leq P\end{aligned} \tag{7.2}$$

corresponds to finding the maximum a posteriori (MAP) parameters if one assumes the following prior distributions over the parameters

$$\mathbf{w}_m\ \sim\ \mathcal{N}\Bigg(\ \mathbf{0}_{N_\mathcal{F}},\Bigg[\ \gamma\sum_{p=1}^P\frac{\partial\varphi}{\partial\mathbf{x}}(\mathbf{x}_p)\,\frac{\partial\varphi}{\partial\mathbf{x}}(\mathbf{x}_p)^\top\ +\ \eta\,\mathbf{I}_{N_\mathcal{F}}\ \Bigg]^{-1}\ \Bigg) \tag{7.3a}$$

$$\mathbf{b}\ \sim\ \mathcal{N}\Big(\ \mathbf{0}_M,\ \sigma_b^2\,\mathbf{I}_M\ \Big)\ \text{with}\ \sigma_b\to\infty. \tag{7.3b}$$

Proof of Proposition 7.1. By definition, the MAP parameters maximize

$$P(\,\mathbf{W},\mathbf{b}\mid\mathcal{D},\lambda,\gamma,\eta,\mathcal{M}\,) \;=\; \frac{P(\,\mathcal{D}\mid\mathbf{W},\mathbf{b},\lambda,\gamma,\eta,\mathcal{M}\,)\,P(\,\mathbf{W},\mathbf{b}\mid\lambda,\gamma,\eta,\mathcal{M}\,)}{P(\,\mathcal{D}\mid\lambda,\gamma,\eta,\mathcal{M}\,)}. \tag{A.13}$$

Since the evidence $P(\,\mathcal{D}\mid\lambda,\gamma,\eta,\mathcal{M}\,)$ is a normalizing constant that is independent of $\mathbf{W}$ and $\mathbf{b}$, we can omit it from the optimization problem. The parameters with maximal posterior probability thus also maximize

$$\underbrace{P(\,\mathcal{D}\mid\mathbf{W},\mathbf{b},\lambda,\gamma,\eta,\mathcal{M}\,)}_{\text{Likelihood}}\;\underbrace{P(\,\mathbf{W},\mathbf{b}\mid\lambda,\gamma,\eta,\mathcal{M}\,)}_{\text{Prior}}. \tag{A.14}$$

The proof contains two parts. We first establish that maximizing the likelihood of the data corresponds to minimizing the first term in (7.2), and subsequently show that maximizing the prior over the parameters corresponds to minimizing the two last terms in the objective of (7.2).

First, assume that the likelihood of the data is independent of γ and η, assume that the probability of $\mathbf{x}_p$ is independent of $\mathcal{M}$, and constant, and assume that the data point couples $(\mathbf{x}_p,\mathbf{y}_p)$ are independent as well.

$$\begin{aligned}
P(\,\mathcal{D}\mid\mathbf{W},\mathbf{b},\lambda,\gamma,\eta,\mathcal{M}\,) &= \prod_{p=1}^{P} P(\,\mathbf{x}_p,\mathbf{y}_p\mid\mathbf{W},\mathbf{b},\lambda,\mathcal{M}\,)\\
&= \prod_{p=1}^{P} P(\,\mathbf{y}_p\mid\mathbf{x}_p,\mathbf{W},\mathbf{b},\lambda,\mathcal{M}\,)\,P(\,\mathbf{x}_p\mid\mathbf{W},\mathbf{b},\lambda,\mathcal{M}\,)\\
&= \prod_{p=1}^{P} P(\,\mathbf{y}_p\mid\mathbf{x}_p,\mathbf{W},\mathbf{b},\lambda,\mathcal{M}\,)\,P(\,\mathbf{x}_p\,)\\
&\propto \prod_{p=1}^{P} P(\,\mathbf{y}_p\mid\mathbf{x}_p,\mathbf{W},\mathbf{b},\lambda,\mathcal{M}\,)
\end{aligned} \tag{A.15}$$

Suppose the errors $\mathbf{e}_p=\mathbf{y}_p-\mathbf{W}^\top\varphi(\mathbf{x}_p)-\mathbf{b}$ are then normally distributed with zero mean and variance $1/\lambda$ in each dimension, we obtain the relation

$$P(\,\mathbf{y}_p\mid\mathbf{x}_p,\mathbf{W},\mathbf{b},\lambda,\mathcal{M}\,)\;\propto\;\exp\Big[-\frac{\lambda}{2}\|\,\mathbf{y}_p-\mathbf{W}^\top\varphi(\mathbf{x}_p)-\mathbf{b}\,\|_2^2\Big]. \tag{A.16}$$

We now have all the elements to conclude the first part of the proof since maximizing the likelihood equals minimizing the negative log-likelihood:

$$
\begin{aligned}
& \underset{\mathbf{W},\mathbf{b}}{\text{maximize}} \quad P(\, \mathcal{D} \mid \mathbf{W}, \mathbf{b}, \lambda, \gamma, \eta, \mathcal{M} \,), \\
\equiv \quad & \underset{\mathbf{W},\mathbf{b}}{\text{maximize}} \quad \prod_{p=1}^{P} P(\, \mathbf{y}_p \mid \mathbf{x}_p, \mathbf{W}, \mathbf{b}, \lambda, \mathcal{M} \,), \\
\equiv \quad & \underset{\mathbf{W},\mathbf{b}}{\text{minimize}} \quad -\log \left[\prod_{p=1}^{P} \exp \left[-\frac{\lambda}{2} \| \, \mathbf{y}_p - \mathbf{W}^\top \varphi(\mathbf{x}_p) - \mathbf{b} \, \|_2^2 \right] \right], \\
\equiv \quad & \underset{\mathbf{W},\mathbf{b}}{\text{minimize}} \quad \frac{\lambda}{2} \sum_{p=1}^{P} \| \, \mathbf{y}_p - \mathbf{W}^\top \varphi(\mathbf{x}_p) - \mathbf{b} \, \|_2^2, \\
\equiv \quad & \underset{\mathbf{W},\mathbf{b},\mathbf{e}_p}{\text{minimize}} \quad \frac{\lambda}{2} \sum_{p=1}^{P} \| \, \mathbf{e}_p \, \|_2^2 \\
& \text{subject to} \quad \mathbf{e}_p \;=\; \mathbf{y}_p - \mathbf{W}^\top \varphi(\mathbf{x}_p) - \mathbf{b} \\
& \qquad\qquad\; 1 \leq p \leq P.
\end{aligned}
$$

To continue, assume that the parameters $\mathbf{W}$ and $\mathbf{b}$ are independent, assume that the M columns $\mathbf{w}_m$ of matrix $\mathbf{W}$ are independent as well, and assume that the prior distributions over the parameters are given by

$$
\begin{aligned}
\mathbf{w}_m \;&\sim\; \mathcal{N}\!\left(\mathbf{0}_{N_\mathcal{F}}, \left[\gamma \sum_{p=1}^{P} \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}_p) \, \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}_p)^\top \;+\; \eta \, \mathbf{I}_{N_\mathcal{F}} \right]^{-1} \right), \\
\mathbf{b} \;&\sim\; \mathcal{N}\!\left(\mathbf{0}_M, \; \sigma_b^2 \, \mathbf{I}_M \right) \text{ with } \sigma_b \to \infty.
\end{aligned}
$$

In (A.14), the prior distribution over both parameters then rewrites to

$$
\begin{aligned}
P(\, \mathbf{W}, \mathbf{b} \mid \lambda, \gamma, \eta, \mathcal{M} \,) \;&=\; \prod_{m=1}^{M} P(\, \mathbf{w}_m \mid \gamma, \eta, \mathcal{M} \,) \, P(\, \mathbf{b} \mid \gamma, \eta, \mathcal{M} \,) \\
&\propto \prod_{m=1}^{M} \exp \left[-\frac{1}{2} \mathbf{w}_m^\top \left[\gamma \sum_{p=1}^{P} \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}_p) \, \frac{\partial \varphi}{\partial \mathbf{x}}(\mathbf{x}_p)^\top \;+\; \eta \, \mathbf{I}_{N_\mathcal{F}} \right] \mathbf{w}_m \right].
\end{aligned}
\tag{A.17}
$$

We have all the elements to conclude the second part of the proof since maximizing the prior is equivalent to minimizing the negative log-prior:

$$\underset{\mathbf{W},\mathbf{b}}{\text{maximize}} \quad P(\ \mathbf{W},\mathbf{b} \mid \lambda,\gamma,\eta,\mathcal{M}\ ),$$

$$\equiv \quad \underset{\mathbf{w}_m,\mathbf{b}}{\text{maximize}} \quad \prod_{m=1}^{M} P(\ \mathbf{w}_m \mid \gamma,\eta,\mathcal{M}\ )\ P(\ \mathbf{b} \mid \gamma,\eta,\mathcal{M}\ ),$$

$$\equiv \quad \underset{\mathbf{w}_m,\mathbf{b}}{\text{minimize}} \quad -\log\left[\ \prod_{m=1}^{M}\ \exp\left[\ -\frac{1}{2}\mathbf{w}_m^\top\left[\ \gamma\ \sum_{p=1}^{P}\ \frac{\partial\varphi}{\partial\mathbf{x}}(\mathbf{x}_p)\ \frac{\partial\varphi}{\partial\mathbf{x}}(\mathbf{x}_p)^\top\ +\ \eta\ \mathbf{I}_{N_\mathcal{F}}\ \right]\mathbf{w}_m\ \right]\ \right],$$

$$\equiv \quad \underset{\mathbf{w}_m,\mathbf{b}}{\text{minimize}} \quad \frac{\gamma}{2}\ \sum_{p=1}^{P}\sum_{m=1}^{M}\ \mathbf{w}_m^\top\frac{\partial\varphi}{\partial\mathbf{x}}(\mathbf{x}_p)\ \frac{\partial\varphi}{\partial\mathbf{x}}(\mathbf{x}_p)^\top\mathbf{w}_m\ +\ \frac{\eta}{2}\ \sum_{m=1}^{M}\ \mathbf{w}_m^\top\mathbf{w}_m,$$

$$\begin{aligned}\equiv \quad & \underset{\mathbf{w}_m,\mathbf{b},\mathbf{J}_p}{\text{minimize}} && \frac{\gamma}{2}\ \sum_{p=1}^{P}\ \|\ \mathbf{J}_p\ \|_\mathrm{F}^2\ +\ \frac{\eta}{2}\ \sum_{m=1}^{M}\ \|\ \mathbf{w}_m\ \|_2^2 \\ & \text{subject to} && \mathbf{J}_p\ =\ \mathbf{W}^\top\frac{\partial\varphi}{\partial\mathbf{x}}(\mathbf{x}_p) \\ & && 1 \leq p \leq P.\end{aligned}$$

To conclude the proof, let us combine the results obtained in both parts:

$$\underset{\mathbf{W},\mathbf{b}}{\text{maximize}} \quad P(\ \mathcal{D} \mid \mathbf{W},\mathbf{b},\lambda,\gamma,\eta,\mathcal{M}\ )\ P(\ \mathbf{W},\mathbf{b} \mid \lambda,\gamma,\eta,\mathcal{M}\ ),$$

$$\equiv \quad \underset{\mathbf{W},\mathbf{b}}{\text{minimize}} \quad -\log\Big[\ P(\ \mathcal{D} \mid \mathbf{W},\mathbf{b},\lambda,\gamma,\eta,\mathcal{M}\ )\ P(\ \mathbf{W},\mathbf{b} \mid \lambda,\gamma,\eta,\mathcal{M}\ )\ \Big],$$

$$\begin{aligned}\equiv \quad & \underset{\mathbf{w}_m,\mathbf{b},\mathbf{e}_p,\mathbf{J}_p}{\text{minimize}} && \frac{\lambda}{2}\sum_{p=1}^{P}\ \|\ \mathbf{e}_p\ \|_2^2\ +\ \frac{\gamma}{2}\sum_{p=1}^{P}\ \|\ \mathbf{J}_p\ \|_\mathrm{F}^2\ +\ \frac{\eta}{2}\sum_{m=1}^{M}\ \|\ \mathbf{w}_m\ \|_2^2 \\ & \text{subject to} && \mathbf{e}_p = \mathbf{y}_p - \mathbf{W}^\top\varphi(\mathbf{x}_p) - \mathbf{b} \\ & && \mathbf{J}_p = \mathbf{W}^\top\frac{\partial\varphi}{\partial\mathbf{x}}(\mathbf{x}_p) \\ & && 1 \leq p \leq P.\end{aligned}$$

■

Appendix B

Dynamical systems

A dynamical system describes the movement of a point with a function. Further, we denote this point $\mathbf{x} \in \mathbb{R}^N$ and the function $f(\cdot) : \mathbb{R}^N \to \mathbb{R}^N$.

The update equation

Depending on the measure for time, the system is continuous or discrete.

Continuous dynamical systems

A continuous dynamical system moves the point $\mathbf{x}$ in a continuous way. It consists of a differential equation subjected to an initial condition.

$$\left[\begin{array}{ll} \dfrac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}(t) & = f(\ \mathbf{x}(t)\ ) \\ \mathbf{x}(0) & = \mathbf{c} \end{array}\right. \tag{B.1}$$

Thus in continuous time, $f(\cdot)$ models the instantaneous change of $\mathbf{x}(t)$.

Discrete dynamical systems

A discrete dynamical system moves the point $\mathbf{x}$ along discrete steps. It consists of a difference equation subjected to an initial condition.

$$\left[\begin{array}{ll} \mathbf{x}^{(k+1)} & = f(\ \mathbf{x}^{(k)}\ ) \\ \mathbf{x}^{(0)} & = \mathbf{c} \end{array}\right. \tag{B.2}$$

Thus in discrete time, $f(\cdot)$ directly generates the next position after $\mathbf{x}^{(k)}$.

Since this thesis studies discrete dynamical systems, we concentrate on the theory of discrete dynamical systems in the remainder of this chapter.

The state space

The state space is the space in which the point—or state—$\mathbf{x}$ moves. Above we denoted this space $\mathbb{R}^N$. Visualizing the movement of $\mathbf{x}$ through the state space reveals important characteristics of the dynamical system.

For instance, consider a one dimensional system, called the logistic map

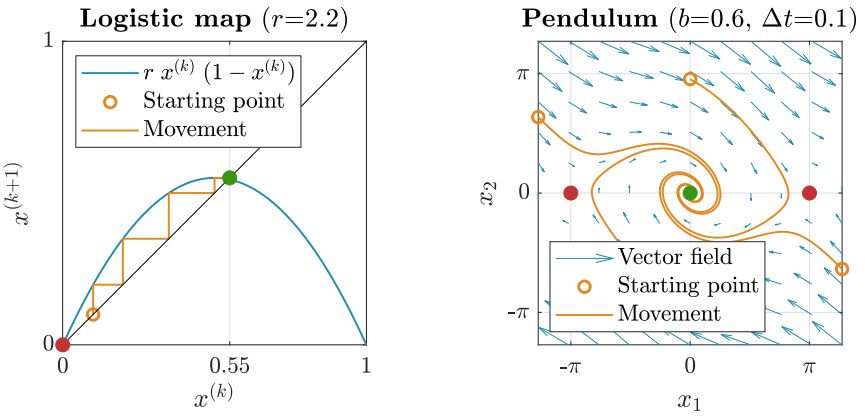
$$x^{(k+1)} = r\, x^{(k)}\, (1 - x^{(k)}) \tag{B.3}$$

The Cobweb in figure B.1a then suggests that for the parameter $r = 2.2$, the state x is repelled by the value 0, but is attracted to the value 0.55.

Likewise, consider a first-order discretization of the damped pendulum

$$\begin{bmatrix} x_1^{(k+1)} &= x_1^{(k)} + \Delta t\, x_2^{(k)} \\ x_2^{(k+1)} &= -\Delta t\, \sin(x_1^{(k)}) + (1 - b\,\Delta t)\, x_2^{(k)}, \end{bmatrix} \tag{B.4}$$

where x_1 is the angle with the vertical axis, and x_2 is the speed wherewith this angle changes. The vector field in figure B.1b then indicates that for $b = 0.6$ and $\Delta t = 0.1$, $(\pm\pi, 0)$ (pendulum at rest standing up) repels the state $\mathbf{x}$, whereas $(0, 0)$ (pendulum at rest hanging down) attracts it.



(a) Movement in a 1D state space **(b)** Movement in a 2D state space

Figure B.1: By visualizing the movement of the state through the state space we discover important characteristics of the dynamical system.

Equilibria

The green and red points in figure B.1 are the equilibria of both systems. Such equilibria are fixed points of the update equation, i.e. they satisfy

$$\mathbf{x}^* \;=\; f(\mathbf{x}^*). \tag{B.5}$$

As we observed earlier, an equilibrium either attracts or repels the state.

Locally stable equilibria

A locally stable equilibrium attracts all the states in its neighbourhood. These are the green points in figure B.1. In such an equilibrium the N eigenvalues λ_n of the Jacobian $\partial f/\partial \mathbf{x}$ lie within the complex unit circle:

$$\left| \lambda_n \left(\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}^*) \right) \right| \;<\; 1 \qquad 1 \leq n \leq N \tag{B.6}$$

Locally unstable equilibria

A locally unstable equilibrium repels all the states in its neighbourhood. These are the red points in figure B.1. In such an equilibrium at least one eigenvalue of the Jacobian is located outside the complex unit circle.

Conclusion

Overall, dynamical systems are fascinating objects with rich behaviors. We refer to Strogatz' book for a great introduction to the topic [92].

Appendix C

The Lagrange method for constrained optimization

The Lagrange method is an optimization strategy for finding a local optimum of a function subject to equality and inequality constraints.

Consider the constrained optimization problem, assumed to be convex,

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & g(\mathbf{x}) \;=\; \mathbf{0}_P \\ & h(\mathbf{x}) \;\geq\; \mathbf{0}_Q \end{aligned} \tag{C.1}$$

with objective $f(\cdot) : \mathbb{R}^N \to \mathbb{R}$, equality constraint $g(\cdot) : \mathbb{R}^N \to \mathbb{R}^P$, inequality constraint $h(\cdot) : \mathbb{R}^N \to \mathbb{R}^Q$, and optimal solution $\mathbf{x}^* \in \mathbb{R}^N$. Furthermore, denote the ensemble of points $\mathbf{x}$ that satisfy the constraints

$$\Omega \;=\; \{\, \mathbf{x} \in \mathbb{R}^N \;\mid\; g(\mathbf{x}) = \mathbf{0}_P \;\wedge\; h(\mathbf{x}) \geq \mathbf{0}_Q \,\}.$$

The Lagrange method then computes a solution of (C.1) in three steps.

1. Compute the Lagrange function

The Lagrange function $\mathcal{L}(\cdot,\cdot,\cdot) : \mathbb{R}^N \times \mathbb{R}^P \times \mathbb{R}^Q \to \mathbb{R}$ is constructed as

$$\mathcal{L}(\,\mathbf{x},\,\mathbf{l},\,\mathbf{m}\,) \;=\; f(\mathbf{x}) \;-\; \mathbf{l}^\top g(\mathbf{x}) \;-\; \mathbf{m}^\top h(\mathbf{x}). \tag{C.2}$$

The Lagrange multipliers $\mathbf{l} \in \mathbb{R}^P$ and $\mathbf{m} \in \mathbb{R}^Q$, also called dual variables, represent the sensitivity of the objective value $f(\mathbf{x}^*)$ to the constraints. For instance, if $\mathbf{m} = \mathbf{0}_Q$, $f(\mathbf{x}^*)$ does not change by slightly moving $h(\cdot)$. If $\mathbf{m} > \mathbf{0}_Q$ on the other hand, $f(\mathbf{x}^*)$ does change by slightly moving $h(\cdot)$. Besides, the dual variable $\mathbf{m}$ should be be non-negative to ensure that

$$\forall\, \mathbf{x} \in \Omega \;\;:\;\; \mathcal{L}(\,\mathbf{x},\,\mathbf{l},\,\mathbf{m}\,) \;=\; f(\mathbf{x}) \;-\; \mathbf{l}^\top \underbrace{g(\mathbf{x})}_{=\,\mathbf{0}} \;-\; \underbrace{\mathbf{m}^\top}_{\geq\,\mathbf{0}} \underbrace{h(\mathbf{x})}_{\geq\,\mathbf{0}} \;\leq\; f(\mathbf{x}).$$

The Lagrange function thus forms a lower bound for the objective function. To find the solution $\mathbf{x}^*$ that minimizes $f(\cdot)$, the Lagrange method then minimizes the lower bound for the objective that is as tight as possible:

$$\underset{\mathbf{l},\mathbf{m}}{\text{maximize}} \quad \underset{\mathbf{x}}{\text{minimize}} \quad \mathcal{L}(\,\mathbf{x},\,\mathbf{l},\,\mathbf{m}\,). \tag{C.3}$$

2. Minimize the Lagrange function

The dual function $\mathcal{Q}(\cdot,\cdot) : \mathbb{R}^P \times \mathbb{R}^Q \to \mathbb{R}$ bounds the Lagrange function as

$$\mathcal{Q}(\,\mathbf{l},\,\mathbf{m}\,) \;=\; \underset{\mathbf{x}}{\text{infimum}} \quad \mathcal{L}(\,\mathbf{x},\,\mathbf{l},\,\mathbf{m}\,). \tag{C.4}$$

3. Maximize the dual function

We maximize the dual function with the following optimization problem:

$$\begin{aligned} &\underset{\mathbf{l},\mathbf{m}}{\text{maximize}} && \mathcal{Q}(\,\mathbf{l},\,\mathbf{m}\,) \\ &\text{subject to} && \mathbf{m} \;\geq\; \mathbf{0}_Q \end{aligned} \tag{C.5}$$

After solving this optimization problem, we compute $\mathbf{x}^*$ from $\mathbf{l}^*$ and $\mathbf{m}^*$.

(C.1) is the *primal* problem as it is expressed in the primal variable $\mathbf{x}$. (C.5) is the *dual* problem as it is expressed in the dual variables $\mathbf{l}$ and $\mathbf{m}$.

Duality implies that a constrained optimization problem may be viewed from two angles: the primal problem (C.1) or the dual problem (C.5). For the convex optimization problems in this thesis, both are equivalent. Thus, we can find the optimal solution $\mathbf{x}^*$ either by solving (C.1) directly, or by solving (C.5) and then computing $\mathbf{x}^*$ from the solutions $\mathbf{l}^*$ and $\mathbf{m}^*$.

Overall, the Lagrange method is an optimization strategy to find a local optimum of a function subject to equality and inequality constraints by constructing two representations of the original optimization problem. For a more thorough discussion of the topic, we refer to the book of Nocedal and Wright [66], or the book of Boyd and Vandenberghe [13].

Appendix D

Deep feedforward neural networks

A deep feedforward neural network is a structure of connected nodes, called neurons, that maps an input space $\mathbb{R}^N$ onto an output space $\mathbb{R}^M$. Such an (artificial) neural network consists of three nested components: neurons are combined into layers, and layers are stacked into networks.

The neuron

A neuron is the building block of a deep feedforward neural network. This computational unit maps an input $\mathbf{x} \in \mathbb{R}^N$ onto an output $y \in \mathbb{R}$:

$$\begin{aligned} y &= a(\, w_1\, x_1 \,+\, \cdots \,+\, w_N\, x_N \,+\, b\,) \\ &= a(\, \mathbf{w}^\top \mathbf{x} \,+\, b\,), \end{aligned} \tag{D.1}$$

with weights $\mathbf{w} \in \mathbb{R}^N$, bias $b \in \mathbb{R}$, and activation function $a(\cdot) : \mathbb{R} \to \mathbb{R}$. This activation function ensures that the output y is nonlinear in $\mathbf{x}$, such as

$$\begin{aligned} a(z) &= \text{sign}(z) && \text{(sign activation)} \\ a(z) &= \tanh(z) && \text{(hyperbolic tangent activation)} \\ a(z) &= \max(0, z) && \text{(rectifier activation)} \end{aligned}$$

A single neuron can represent relatively few mappings, all rather simple. To obtain more powerful models, we aggregate several neurons into layers.

The layer

A layer consists of M neurons that process the same input vector $\mathbf{x} \in \mathbb{R}^N$, each with different parameters $\mathbf{w}_m$ and b_m. So the m^{th} neuron outputs

$$y_m \;=\; a(\; \mathbf{w}_m^\top \mathbf{x} \;+\; b_m \;). \tag{D.2}$$

The combined outputs of the M neurons form an output vector $\mathbf{y} \in \mathbb{R}^M$

$$\mathbf{y} \;=\; a(\; \mathbf{W}^\top \mathbf{x} \;+\; \mathbf{b} \;), \tag{D.3}$$

where $\mathbf{W} \;=\; [\; \mathbf{w}_1 \;\; \cdots \;\; \mathbf{w}_M \;] \in \mathbb{R}^{N \times M}$ and $\mathbf{b} \;=\; [\; b_1 \;\; \cdots \;\; b_M \;]^\top \in \mathbb{R}^M$, and the activation function $a(\cdot) : \mathbb{R}^M \to \mathbb{R}^M$ is taken element by element.

The *width* of a layer indicates the number of neurons inside that layer. So *narrow* layers have few neurons, while *wide* layers have many neurons.

The network

A network consists of L layers that process an input vector $\mathbf{x}$ sequentially, i.e. the output vector from layer l serves as input vector for layer $l+1$. A *feedforward* network has no feedback connections between its layers. The first layer thus receives the input $\mathbf{x} \in \mathbb{R}^N$, and outputs $\mathbf{h}_{(1)} \in \mathbb{R}^{M_{(1)}}$

$$\mathbf{h}_{(1)} \;=\; a_{(1)}(\; \mathbf{W}_{(1)}^\top \mathbf{x} \;+\; \mathbf{b}_{(1)} \;), \tag{D.4}$$

the l^{th} layer receives as input $\mathbf{h}_{(l-1)} \in \mathbb{R}^{M_{(l-1)}}$, and outputs $\mathbf{h}_{(l)} \in \mathbb{R}^{M_{(l)}}$

$$\begin{aligned} \mathbf{h}_{(l)} \;&=\; a_{(l)}(\; \mathbf{W}_{(l)}^\top \mathbf{h}_{(l-1)} \;+\; \mathbf{b}_{(l)} \;) \\ &=\; a_{(l)}(\; \mathbf{W}_{(l)}^\top \;(\; \cdots\; a_{(1)}(\; \mathbf{W}_{(1)}^\top \mathbf{x} \;+\; \mathbf{b}_{(1)} \;)\; \cdots\;) \;+\; \mathbf{b}_{(l)} \;), \end{aligned} \tag{D.5}$$

and the last layer receives $\mathbf{h}_{(L-1)} \in \mathbb{R}^{M_{(L-1)}}$, and outputs $\mathbf{y} \in \mathbb{R}^M$

$$\begin{aligned} \mathbf{y} \;&=\; a_{(L)}(\; \mathbf{W}_{(L)}^\top \mathbf{h}_{(L-1)} \;+\; \mathbf{b}_{(L)} \;) \\ &=\; a_{(L)}(\; \mathbf{W}_{(L)}^\top \;(\; \cdots\; a_{(1)}(\; \mathbf{W}_{(1)}^\top \mathbf{x} \;+\; \mathbf{b}_{(1)} \;)\; \cdots\;) \;+\; \mathbf{b}_{(L)} \;). \end{aligned} \tag{D.6}$$

Each intermediate variable $\{\; \mathbf{h}_{(l)} \;\}_{l=1}^{L}$ is called the *activation* of layer l.

The *depth* of a network indicates the number of layers inside that network. So *shallow* networks have few layers, whereas *deep* networks have many.

Figure D.1 depicts the construction of a deep feedforward neural network.

Even though a two-layered network is a universal approximator [46], networks that are used today contain tens or hundreds of layers [59], because deeper networks can compactly represent more complex functions. Note the importance of a nonlinearity in each layer. If $a(\cdot)$ were linear, then (D.6) would collapse into a simple linear mapping from $\mathbf{x}$ to $\mathbf{y}$. Consequently, the deep network would behave like a one-layer network.

In conclusion, a deep feedforward neural network essentially consists of an elaborate nonlinear mapping from an input space to an output space. This appendix briefly introduced the components of a deep neural network. For a thorough discussion on neural networks and deep learning in general, we refer to the books of Goodfellow *et al.* [36], and Patterson *et al.* [71].

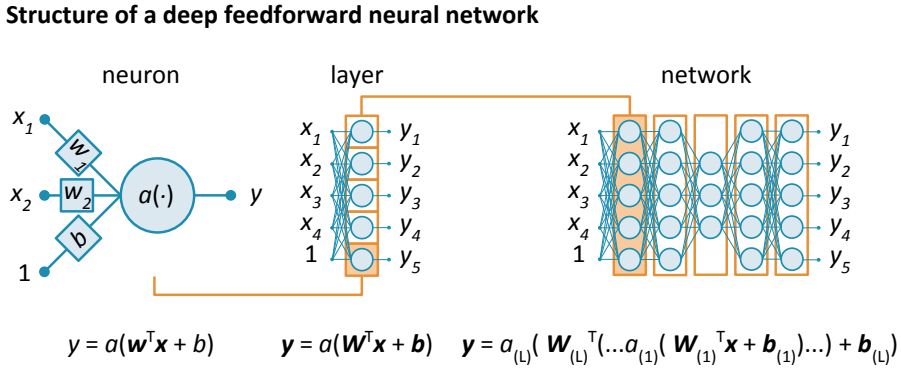


Figure D.1: A neural network consists of three nested components: neurons are combined into layers, and layers are stacked into networks.

Bibliography

[1] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A Learning algorithm for Boltzmann Machines. *Cognitive Science*, 9:147–169, 1985.

[2] C. Alzate and J. A. K. Suykens. A weighted kernel PCA formulation with out-of-sample extensions for spectral clustering methods. *Proceedings of the International Joint Conference on Neural Networks*, pages 138–144, 2006.

[3] D. J. Amit. *Modeling Brain Function: The World of Attractor Neural Networks*. Cambridge University Press, 1992.

[4] D. J. Amit, H. Gutfreund, and H. Sompolinsky. Statistical mechanics of neural networks near saturation. *Annals of Physics*, 137(1):30–67, 1987.

[5] J. R. Anderson. *Learning and memory: An integrated approach*. John Wiley & Sons Inc, 2000.

[6] M. Atencia, G. Joya, and F. Sandoval. Spurious minima and basins of attraction in higher-order hopfield networks. *International Work-Conference on Artificial Neural Networks*, pages 350–357, 2003.

[7] S. Bartunov, J. W. Rae, S. Osindero, and T. Lillicrap. Meatlearning deep energy-based memory models. *Advances in Neural Information Processing Systems*, 2019.

[8] R. D. Beer and J. C. Gallagher. Evolving Dynamical Neural Networks for Adaptive Behavior. In *Adaptive Behaviour*, volume 1, pages 91–122. 1992.

[9] A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik. Support Vector Clustering. *Journal of Machine Learning Research*, 2:125–137, 2001.

[10] Y. Bengio, L. Yao, G. Alain, and P. Vincent. Generalized denoising auto-encoders as generative models. In C. J. C. Burges,

L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 899–907. 2013.

[11] W. Bialek, I. Nemenman, and N. Tishby. Predictability, complexity, and learning. page 24092463, 2001.

[12] B. E. Boser, I. M. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. *Proceedings of the COLT Fifth Annual Workshop on Computational Learning Theory*, 1992.

[13] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[14] M. A. Carreira-Perpinan and W. Wang. Distributed optimization of deeply nested systems. *Proceedings of the 17th international Conference on Artificial Intelligence and Statistics*, 33:10–18, 2014.

[15] D. Casali, G. Costantini, R. Perfetti, and E. Ricci. Associative memory design using support vector machines. *IEEE Transactions on Neural Networks*, 17(5):1165–1174, 2006.

[16] D. Casali, G. Costantini, and M. Todisco. An SVM based method for associative memories. In N. Mastorakis, V. Mladenov, Z. Bojkovic, and D. Simian, editors, *Latest trends on Computers*, volume 2, pages 574–578. 2010.

[17] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, page 388394, 2000.

[18] R. Chaudhuri and I. Fiete. Computational principles of memory. *Nature Neuroscience*, 19(3):394–403, 2016.

[19] R. Chaudhuri and I. Fiete. Bipartite expander Hopfield networks as self-decoding high-capacity error correcting codes. *Advances in Neural Information Processing Systems*, 32:7686–7697, 2019.

[20] H.-D. Chiang and L. F. C. Alberto. *Stability Regions of Nonlinear Dynamical Systems: Theory, Estimation, and Applications*. Cambridge University Press, 2015.

[21] T.-D. Chiueh and R. M. Goodman. Recurrent correlation associative memories. *IEEE Transactions on Neural Networks*, 2(2):275–284, 1991.

[22] K. Cho. Boltzmann Machines and Denoising Autoencoders for Image Denoising. 2013. arXiv:1301.3468.

[23] Y. Cho and L. K. Saul. Kernel methods for deep learning. pages 342–350, 2009.

[24] J. D. Cohen, W. M. Perlstein, T. S. Braver, L. E. Nystrom, D. C. Noll, J. Jonides, and E. E. Smith. Temporal dynamics of brain activation during a working memory task. *Nature*, 386(6625):604–608, 1997.

[25] C. Cortes and V. Vapnik. Support Vector Networks. *Machine Learning*, 20:273–297, 1995.

[26] K. De Brabanter, K. Pelckmans, J. De Brabanter, M. Debruyne, J. A. K. Suykens, M. Hubert, and B. De Moor. Robustness of kernel based regression: a comparison of iterative weighting schemes. *Proceedings of the 19th International Conference on Artificial Neural Networks*, pages 100–110, 2009.

[27] M. Debruyne, A. Christmann, M. Hubert, and J. A. K. Suykens. Robustness of reweighted least squares kernel based regression. *Journal of Multivariate Analysis*, 101(2):447–463, 2010.

[28] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber. Fast direct multiple shooting algorithms for optimal robot control. *Fast Motions in Biomechanics and Robotics*, pages 65–93, 2005.

[29] C. Doersch. Tutorial on variational autoencoders. 2016. arXiv:1606.05908.

[30] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. 2018. arXiv:1802.01561.

[31] M. Espinoza, J. A. K. Suykens, and B. De Moor. Fixed-size least squares support vector machines: a large scale application in electrical load forecasting. *Computational Management Science*, 3(2):113–129, 2006.

[32] Y. Feng, Y. Yang, X. Huang, S. Mehrkanoon, and J. A. K. Suykens. Robust support vector machines for classification with nonconvex and smooth losses. *Neural Computation*, 28:1–31, 2016.

[33] C. Florensa, D. Held, X. Geng, and P. Abbeel. Automatic goal generation for reinforcement learning agents. 2017. arXiv:1705.06366.

[34] C. Garcia and J. A. Moreno. The hopfield associative memory network: Improving performance with the kernel trick. *Ibero-American Conference on AI*, 9:971–880, 2004.

[35] L. Gondara. Medical image denoising using convolutional denoising autoencoders. *IEEE International Conference on Data Mining Workshops*, 16:241–246, 2016.

[36] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[37] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. 2014. arXiv:1406.2661.

[38] K. Grace, J. Salvatier, A. Dafoe, B. Zhang, and O. Evans. When Will AI Exceed Human Performance? Evidence from AI Experts. 2017. arXiv:1705.08807.

[39] E. R. Hancock and M. Pelillo. A bayesian interpretation for the exponential correlation associative memory. *Pattern Recognition Letters*, 19:149–159, 1998.

[40] D. O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Wiley, 1949.

[41] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18, 2006.

[42] G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313:504–507, 2006.

[43] G. E. Hinton and R. S. Zemel. Autoencoders, minimum description length, and Helmholtz free energy. *Conference on Neural Information Processing Systems*, 1994.

[44] J. Hooper and D. Teresi. *The Three-Pound Universe*. Laurel, 1986.

[45] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79:2554–2558, 1982.

[46] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[47] P. Kanerva. *Sparse distributed memory*. MIT press, 1988.

[48] I. Kanter and H. Sompolinsky. Associative recall of memory without errors. *Physical Review A: General Physics*, 35(1):380–392, 1987.

[49] I. Karandashev, B. Kryzhanovsky, and L. Litinskii. Weighted patterns as a tool for improving the hopfield model. *Physical Review E*, 85(4):041925, 2012.

[50] M. Kiehl. Parallel multiple shooting for the solution of initial value problems. *Parallel Computing*, (3):275–295, 1994.

[51] D. Killock. AI outperforms radiologists in mammographic screening. *Nature Reviews Clinical Oncology*, 17(3):1759–4782, 2020.

[52] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014.

[53] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *International Conference on Learning Representations*, 2014.

[54] A. Krizhevsky and G. E. Hinton. Using very deep autoencoders for content-based image retrieval. *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2011.

[55] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum. Deep convolutional inverse graphics network. *Advances in Neural Information Processing Systems*, 28:2539–2547, 2015.

[56] R. Langone, M. Agudelo, B. De Moor, and J. A. K. Suykens. Incremental kernel spectral clustering for online learning of non-stationary data. *Neurocomputing*, 139:246–260, 2014.

[57] LawGeek. Comparing the Performance of Artificial Intelligence to Human Lawyers in the Review of Standard Business Contracts. *Internal report*, 2018.

[58] Y. LeCun. A theoretical framework for back-propagation. *Proceedings of the 1988 Connectionist Models Summer School*, pages 21–28, 1988.

[59] Y. LeCun, Y. Bengio, and G. E. Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[60] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[61] D.-H. Lee, S. Zhang, A. Fischer, and Y. Bengio. Difference Target Propagation. In *Machine Learning and Knowledge Discovery in Databases*, pages 498–515. Springer International Publishing, 2015.

[62] W. Lee, B. J. Ko, S. Wang, C. Liu, and K. K. Leung. Exact incremental and decremental learning for LS-SVM. *IEEE International Conference on Image Processing*, pages 2334–2338, 2019.

[63] L. Lina, P. Witold, Q. Ting, and L. Zhiwu. Fuzzy associative memories with autoencoding mechanisms. *Knowledge-Based Systems*, 191(5):105090, 2020.

[64] S. Loyka. On singular value inequalities for the sum of two matrices. 2015. arXiv:1507.06630.

[65] R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh. The capacity of the hopfield associative memory. *IEEE Transactions on Information Theory*, 33(4):461–482, 1987.

[66] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, 2 edition, 2006.

[67] D. Nowicki and O. Dekhtyarenko. Kernel-based associative memory. *IEEE International Joint Conference on Neural Networks*, 1:741–744, 2004.

[68] D. Nowicki and H. Siegelmann. Flexible kernel memory. *PLOS One*, 5(6), 2010. e10955.

[69] D. Ormoneit and S. Sen. Kernel-Based Reinforcement Learning. *Machine Learning*, 49:161–178, 2002.

[70] G. Parisi. A memory which forgets. *Journal of Physics A: Mathematical and General*, 19(10):L617–L620, 1986.

[71] J. Patterson and A. Gibson. *Deep learning: A practitioner's approach*. O'Reilly, 2017.

[72] R. Perfetti and E. Ricci. Recurrent correlation associative memories: A feature space perspective. *IEEE Transactions on Neural Networks*, 19(2):333–345, 2008.

[73] L. Personnaz, I. Guyon, and G. Dreyfus. Information storage and retrieval in spin-glass like neural networks. *Journal de Physique Lettres*, 46(8):359–365, 1985.

[74] K. B. Petersen and M. S. Pedersen. *The Matrix Cookbook*. Technical University of Denmark, 2012.

[75] S. Rait. *DRAAM: Deep (Recursive Auto-Associative Memory) And Applied eMbeddings*. Brandeis University, 2018.

[76] A. Razavi, A. van den Oord, and O. Vinyals. Generating Diverse High-Fidelity Images with VQ-VAE-2, 2019. arXiv:1906.00446.

[77] R. A. Rescorla and A. R. Wagner. A theory of Pavlovian Conditioning: Variations in the effectiveness of Reinforcement and Nonreinforcement. In A. H. Black and W. F. Prokasy, editors, *Classical conditioning II: current research and theory*, pages 64–99. 1972.

[78] S. Rifai, Y. Bengio, Y. N. Dauphin, and P. Vincent. A generative process for sampling contractive auto-encoders. *Proceedings of the International Conference on Machine Learning*, 29, 2012.

[79] S. Rifai, G. Mesnil, P. Vincent, X. Muller, Y. Bengio, Y. Dauphin, and X. Glorot. Higher order contractive auto-encoder. *ECML PKDD*, 2011.

[80] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. *Proceedings of the 28th International Conference on Machine Learning*, page 833840, 2011.

[81] E. T. Rolls. Attractor networks. *WIREs Cognitive Science*, 1:119–134, 2010.

[82] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors, 1986.

[83] R. R. Salakhutdinov and G. E. Hinton. Deep Boltzmann Machines. *Journal of Machine Learning Research*, 5:448 – 455, 2009.

[84] R. R. Salakhutdinov, A. Mnih, and G. E. Hinton. Restricted Boltzmann machines for collaborative filtering. *Proceedings of the International Conference on Machine Learning*, 24:791–798, 2007.

[85] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-Learning with Memory-Augmented Neural Networks. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 1842–1850, 2016.

[86] B. Schölkopf, A. Smola, and K.-R. Müller. Kernel principal component analysis. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *Artificial Neural Networks — ICANN'97*, pages 583–588. Springer Berlin Heidelberg, 1997.

[87] B. Schölkopf and A. J. Smola. Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural Computation*, 10:1299–1319, 1998.

[88] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. v. d. Driessche, T. Graepel, and D. Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[89] H. Sompolinsky. Statistical mechanics of neural networks. *Physics today*, pages 70–80, 1988.

[90] V. Srivastava, S. Sampath, and D. J. Parker. Overcoming Catastrophic Interference in Connectionist Networks Using Gram-Schmidt Orthogonalization. *PLoS ONE*, 9(9):e105619, 2014.

[91] A. J. Storkey. Increasing the capacity of a hopfield network without sacrificing functionality. *International Conference on Artificial Neural Networks*, pages 451–456, 1997.

[92] S. H. Strogatz. *Nonlinear dynamics and chaos*. Westview Press, 2 edition, 2015.

[93] J. A. K. Suykens. Deep restricted kernel machines using conjugate feature duality. *Neural Computation*, 29:2123–2163, 2017.

[94] J. A. K. Suykens, J. De Brabanter, L. Lukas, and J. Vandewalle. Weighted least squares support vector machines: robustness and sparse approximation. *Neurocomputing*, 48:85–105, 2002.

[95] J. A. K. Suykens, B. De Moor, and J. Vandewalle. NL_q Theory: A Neural Control Framework with Global Asymptotic Stability Criteria. *Neural networks*, 10(4):615–637, 1997.

[96] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Squares Support Vector Machines*. World Scientific, 2002.

[97] J. A. K. Suykens and J. Vandewalle. Least Squares Support Vector Machine Classifiers. *Neural Processing Letters*, 9:293–300, 1999.

[98] J. A. K. Suykens and J. Vandewalle. Recurrent Least Squares Support Vector Machines. *IEEE Transactions on Circuits and Systems*, 47(7):1109–1114, 2000.

[99] C. Tetzlaff, C. Kolodziejski, I. Markelic, and F. Wrgtter. Time scales of memory, learning and plasticity. *Biological Cybernetics*, 106:715–726, 2012.

[100] T. Van Gestel, J. A. K. Suykens, G. Lanckriet, A. Lambrechts, B. De Moor, and J. Vandewalle. Bayesian framework for least squares support vector machine classifiers, gaussian processes and kernel fisher discriminant analysis. *Neural Computation*, 15(5):1115–1148, 2002.

[101] V. Vapnik. The support vector method of function estimation. *Nonlinear Modeling: Advanced Black-Box techniques*, pages 55–85, 1998.

[102] V. Vapnik and A. Y. Chervonenkis. A class of algorithms for pattern recognition learning. *Avtomat. i Telemekh.*, 25(6):937945, 1964.

[103] M. Vidyasagar. *A Theory of Learning and Generalization: With Applications to Neural Networks and Control Systems*. Springer-Verlag, 1997.

[104] P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol. Extracting and Composing Robust Features with Denoising Autoencoders. *Proceedings of the 25th International Conference on Machine Learning*, pages 1096–1103, 2008.

[105] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11.

[106] G. Weisbuch and F. Fogelman-Soulie. Scaling laws for the attractors of hopfield networks. *Journal de Physique Lettres*, 46(14):623–630, 1985.

[107] M. A. Wierig and L. R. B. Schomaker. Multi-layer support vector machines. In J. A. K. Suykens, M. Signoretto, and A. Argyriou, editors, *Regularization, Optimization, Kernels, and Support Vector Machines*, pages 457–476. Chapman & Hall, 2014.

[108] D. Willshaw, O. Buneman, and H. Longuet-Higgins. Non-Holographic Associative Memory. *Nature*, 222:960–962, 1969.

[109] S. J. Wright. Coordinate descent algorithms, 2015. arXiv:1502.04759.

[110] H.-R. Zhang and X.-D. Wang. Incremental and online learning algorithm for regression least squares support vector machine. *Jisuanji Xuebao/Chinese Journal of Computers*, 29:400–406, 2006.

[111] S. Zheng, A. Trott, S. Srinivasa, N. Naik, M. Gruesbeck, D. C. Parkes, and R. Socher. The AI Economist: Improving Equality and Productivity with AI-Driven Tax Policies. 2020. arXiv:2004.13332.